Configuration Management of Distributed Systems over Unreliable and Hostile Networks

Karvinen, Tero

https://doi.org/10.34737/w7vvz

Configuration Management of Distributed Systems over Unreliable and Hostile Networks

Tero Karvinen

A thesis submitted in partial fulfilment of the requirements of the University of Westminster for the degree of Doctor of Philosophy

November 2023

Abstract

Economic incentives of large criminal profits and the threat of legal consequences have pushed criminals to continuously improve their malware, especially command and control channels. This thesis applied concepts from successful malware command and control to explore the survivability and resilience of benign configuration management systems.

This work expands on existing stage models of malware life cycle to contribute a new model for identifying malware concepts applicable to benign configuration management. The Hidden Master architecture is a contribution to master-agent network communication. In the Hidden Master architecture, communication between master and agent is asynchronous and can operate trough intermediate nodes. This protects the master secret key, which gives full control of all computers participating in configuration management. Multiple improvements to idempotent configuration were proposed, including the definition of the minimal base resource dependency model, simplified resource revalidation and the use of imperative general purpose language for defining idempotent configuration.

Following the constructive research approach, the improvements to configuration management were designed into two prototypes. This allowed validation in laboratory testing, in two case studies and in expert interviews. In laboratory testing, the Hidden Master prototype was more resilient than leading configuration management tools in high load and low memory conditions, and against packet loss and corruption. Only the research prototype was adaptable to a network without stable topology due to the asynchronous nature of the Hidden Master architecture.

The main case study used the research prototype in a complex environment to deploy a multi-room, authenticated audiovisual system for a client of an organization deploying the configuration. The case studies indicated that imperative general purpose language can be used for idempotent configuration in real life, for defining new configurations in unexpected situations using the base resources, and abstracting those using standard language features; and that such a system seems easy to learn.

Potential business benefits were identified and evaluated using individual semistructured expert interviews. Respondents agreed that the models and the Hidden Master architecture could reduce costs and risks, improve developer productivity and allow faster time-to-market. Protection of master secret keys and the reduced need for incident response were seen as key drivers for improved security. Low-cost geographic scaling and leveraging file serving capabilities of commodity servers were seen to improve scaling and resiliency. Respondents identified jurisdictional legal limitations to encryption and requirements for cloud operator auditing as factors potentially limiting the full use of some concepts.

Keywords: configuration management, malware, resilience, command and control, asynchronous

Contents

List of Tables

List of Figures

Accompanying Material

The source code is included for transparency and repeatability. Reading or viewing the source code is not required for understanding this thesis.

The advanced prototype implements the contributed improvements to configuration management. They include the Hidden Master architecture, simplified resource revalidation, the use of imperative general purpose language for idempotent configuration and base resource dependency model. The prototype was used in laboratory testing, both case studies and in the demonstration in expert interviews. Source code is over four thousand lines of Go language and requires practical programming skills to understand.

Advanced prototype source code https://TeroKarvinen.com/conftero

The source code will be published under Free license when this thesis is published. Before that, the code is available on request for internal use.

Acknowledgments

I would like to thank the people who helped me on my PhD journey.

- Shuliang Li, University of Westminster
- Fefie Dotsika, University of Westminster
- Farjam Eshraghian, University of Westminster
- Nathan Clarke, University of Plymouth
- Kimmo Karvinen, Core Factory
- Ville Valtokari, Core Factory
- Ilari Ali-Vehmas, Core Factory
- Lasse West, Aalto University
- Minna Kivihalme, Haaga-Helia UAS
- Taina Lintilä, Haaga-Helia UAS
- Antonius Camara, Haaga-Helia UAS
- Salla Huttunen, Haaga-Helia UAS
- Mikko Vainio, Elisa
- Mika Rautio, Poplatek
- Juuso Heljaste, CGI
- Samuli Vaittinen, NordCloud
- Arttu Uskali, Upcloud
- Niklas Särökaari, KONE
- Will Sillitoe, University of Essex
- My family

Declaration

I declare that all material contained in this thesis is my own work.

Tero Karvinen

List of Publications

This dissertation contains material from one paper. The rights have been granted by the publisher to use the work as a part of this thesis.

Karvinen, T. and Li, S., 2017, April. Investigating survivability of configuration management tools in unreliable and hostile networks. In Information Management (ICIM), 2017 3rd International Conference on (pp. 327-331). IEEE.

Abbreviations

AI: artificial intelligence

AP: access point

API: application programming interface

APT: advanced persistent threat

apt: advanced packaging tool

AV: antivirus; audio visual

AWS: Amazon Web Services

bit: a zero or one.

B: byte, eight bits.

BSD: Berkeley Software Distribution

CA: certificate authority

CAPEC: Common Attack Pattern Enumeration and Classification

CC: command and control

CD: continuous delivery

CDN: content delivery network

CI: continous integration

CM: configuration management

CMS: Configuration management system

DDOS: distributed denial of service

DGA: domain generation algorithm

DevOps: developer operations

DNS: domain name system

DOS: denial of service

DSC: PowerShell Desired State Configuration

DSL: domain specific language

EDR: endpoint detection and response

FSF: Free Software Foundation

FTP: file transfer protocol

GB: gigabyte, 1e9 bytes

GDPR: general data protection regulation (of the European Union)

gcc: GNU Compiler Collection

GNU: "GNU is not UNIX!" (a recursive acronym)

GPL: General Purpose Language, GNU General Purpose License

HM: hidden master

HSM: hardware security module

HTTP: http: hypertext transfer protocol

HTTPS: https: hypertext transfer protocol secure

IaC: infrastructure as code

ICMP: Internet Control Message Protocol

IDE: integrated development environment

IDS: intrusion detection system

IETF: Internet Engineering Task Force

IO: Input-Output

IoT: Internet of things

IP: Internet protocol

IPS: intrusion prevention system

ISP: Internet service provider

JSON: JavaScript object notation

kB: kilobyte, 1 000 bytes

LAN: local area network

LISA: Large Installation System Administration Conference

MA: MITRE ATT&CK

MAC: Mandatory Access Control

MB: megabyte, 1e6 bytes

ML: Machine learning

MPA: MITRE PRE-ATT&CK

ms: millisecond

NAT: Network address translation

NDA: Non-disclosure agreement

OBJ: objective

OODA: Observe, orient, decide, act

OOM: out of memory

OS: operating system

P2P: peer to peer

PHP: " PHP: Hypertext Preprocessor" (a recursive acronym)

PII: personally identifiable information

PoC: proof of concept

PR: public relations

RAM: random access memory

RFC: Request for Comments

RQ: research question

RTT: round trip time

SCADA: supervisory control and data acquisition

SDN: software defined networking

SOC: security operations center

SSH: secure shell

TCP/IP: Transmission Control Protocol / Internet Protocol

TOFU: trust on first use

UPS: uninterruptible power supply

URL: universal resource locator

USGCB: United States Government Configuration Baseline

VCS: version control system

VM: virtual machine

VPN: virtual private network

WLAN: wireless local area network

YAML: yet another markup language

XML: Extensible Markup Language

1 Introduction

Configuration management (CM) is an approach to automate the control of numerous computers. CM can extend life, reduce cost, reduce risk and even correct defects (Perera, 2016). Configuration management is used by many well known organizations, and both academic and industry intrest in it has grown (Rahman, Mahdavi-Hezaveh and Williams, 2019). Because configuration management system has full access to all controlled computers, an error can compromize the entire system (Hastings and Kazanciyan, 2016). This makes configuration management system a valuable target for attackers.

During the writing of this work, attacks on Solarwinds (Marelli, 2022) and Salt Stack (F-Secure, 2020) compromised thousands of computers in different organizations. Despite its value as a target, there is little research on the security of configuration management systems (Rahman, Mahdavi-Hezaveh and Williams, 2019; Xu and Russello, 2022), encryption and network performance (Xu and Russello, 2022). On the other hand, criminal malware has to solve similar security and reliability problems in even more hostile conditions. Malware is evolving fast (Silva et al., 2013; Lemay et al., 2018) and there is a growing number of publications available on malware. Malware command and control techniques to improve resiliency in unreliable and hostile networks could be identified and applied to benign enterprise configuration management.

Research firm Gartner predicts that 70% of companies will use continuous infrastructure automation by 2025 - a significant increase from less than 20% in 2021 (Bhat et al., 2022). Configuration management is used by well known organizations such as LinkedIn (CFEngine, 2016), Github, Netflix (Rahman, Mahdavi-Hezaveh and Williams, 2019), Facebook (Tang et al., 2015), CERN (Andrade et al., 2012) and parts of the US government (NIST, 2016).

Researchers have identified some areas of configuration management as candidates for future research, challenging or lacking in some aspect of research. Considering this work, security and networking are the main areas of intrest. Many authors encouraged more research on the security of configuration management systems (Marsa-Maestre et al., 2019; Rahman, Mahdavi-Hezaveh and Williams, 2019; Kumara et al., 2021; Rajapakse et al., 2022; Rong et al., 2022; Xu and Russello, 2022; Ullah et al., 2023). In configuration management of IoT (Internet of Things), Silva et al. (2019) found security, scalability and reliability to be an open challenges. Kumara et al. (2021) specifically point out secrets management and Xu and Russello (2022) call for research on integrating encryption mechanisms. Network performance has been pointed out by Xu and Russello (2022) for generic CM and by Silva et al. (2019) for IoT. Different

aspects of domain specific languages (DSL) are seen as requiring more research by Xu and Russello (2022) (policy languages, intent translation) and Delaet, Joosen and Van Brabant (2010) (better abstractions, becoming more declarative). In their work to improve Puppet DSL, Fu et al. (2017) critize that DLSs have "evolved in an ad hoc fashion, resulting in a design with numerous features, some of which are complex, hard to understand, and difficult to use correctly". This work will propose improvements in multiple areas pointed out as gaps by other authors in these areas.

Based on published systematic literature reviews, the field of configuration management research is not yet very large. In their gray literature review, Kumara et al. (2021) claim that little academic literature on infrastructure as code (IaC) exists. In their systematic mapping study, Rahman (et al. 2019) identified just 31 IaC publications matching their criteria, and most of them focused on tools and frameworks. Hintsch (et al. 2016) found 159 publications in their systematic literature review, of which only 36 were journal articles. Both reviews found the number of IaC publications growing.

Malware offers practical examples of solving challenges related to security, networking and scale. Criminal malware evolves quickly due to incentives and pressures related to cyber criminal business. Already in 2007 criminals using Zeus botnet took control of 3.6 million computers to steal money from an e-bank, and in 2008 Conficker reached the size of 10.5 million nodes (Silva et al., 2013). Nation state actors turn to cyber espionage, operating in environments defended by equally powerful defenders (Lemay et al., 2018). Criminal and other offensive actors continuously improve their methods. The fast evolution of command and control channels is pushed by an arms race with defenders (Anderson et al., 2021). Criminals need the ability to control their botnets while hiding their identity, avoiding detection and being able to recover from serious damage done to their infrastructure (Huang, Siegel and Madnick, 2018).

Normal, legal enterprises need to control their own computers. Configuration management systems have risen as an answer to the challenges of growing sizes of computer networks, heterogeneous networks and stricter requirements for auditability, controllability and risk management (Hintsch, Görling and Turowski, 2016). Heterogenity and multivendor environments are common in all parts of networks. Heterogenity is common in networking hardware (Xu and Russello, 2022), Internet of Things (IoT) devices (Silva et al., 2019) and common computers, such as laptops, servers and computers virtualized to cloud (Delaet, Joosen and Van Brabant, 2010).

Configuration management systems allow operators to control large number of computers over the network. One computer, the master, delivers instructions

to controlled nodes called slaves, agents or minions. Agent nodes blindly apply these instructions. A large system can have thousands of nodes configured by a single master (Fu et al., 2017). This makes the master node a highly valuable target for criminals and other threat actors.

When deciding on security investment, managers should evaluate vulnerabilities and potential losses (Huang, Hu and Behara, 2008). Compromising the master would compromise all of the agent computers and all data stored in those computers. This makes the master one of the most valuable computers among those managed by the configuration management system, and the secret key of the master is one of the most valuable files in this network. Lateral movement - access to new parts of network - is a key part of successful targeted attack (Noureddine et al., 2016). Security of configuration management has been identified as an open challenge and top priority in IoT (Silva et al., 2019).

Direct connection between master and agent is implied or stated by multiple authors. In their survey on system configuration tools Delaet, Joosen and Van Brabant (2010) categorize all configuration management tools as either agents phoning home to master (pull) or master contacting the agents (push).

Choosing between push and pull architectures is a trade off that must consider security, tool support and the type of nodes to be configured. When using push architecture, the master node is the server, so it must reside in a known address and open a listening port visible to agents. For configuring moving nodes, such as laptops and IoT devices, it would often have to be visible to the Internet to achieve this. In pull architecture, the server is placed on agents. While allowing better protection for master, this makes it impossible to control agents that are in unknown addresses or unreachable parts of networks. Having a highly valuable computer constantly visible to attackers introduces a dangerous time gap to incident response. A gap in incident response potentially increases the costs and damage caused by an attacker (Noureddine et al., 2016).

Modern configuration management defines the desired state as plain text. Defining the desired state of the system allows configuration management system to control computers from an arbitrary start state (Hummer et al., 2013; Fu et al., 2017). Thus, applying the configuration multiple times results in the same result, or a result nearer to the desired state. This quality is known as idempotence. Writing the code as plain text is termed infrastructure as code (IaC). IaC allows the use of many of the same techniques used by programmers, such as version control systems (Rahman, Mahdavi-Hezaveh and Williams, 2019).

Domain specific languages are widely used in configuration management. Config-

uration languages are significantly different from general purpose programming languages (Anderson and Cheney, 2012).

A more competitive environment for software and process development requires a shorter time span of features from the accepted idea to the production code delivered to customers - without sacrificing quality. This has led to DevOps movement, a trend to tightly integrate software development (Dev) and operations (Ops, keeping the systems running in production). DevOps practices combine configuration management systems, automated testing, continuous integration and even the continuous delivery of automatically compiled software to production systems and end users. (Brunnert et al., 2015)

Criminals and spies attempt to maintain operational security, and keep their methods secret. Nevertheless, information about tactics, tools and procedures of botnet operators is widely available from industry sources and academic literature (Lemay et al., 2018).

Configuration management systems are a key component to keep development, testing and production environments highly similar in relevant aspects while allowing for obvious differences between development workstations and production servers.

The proliferation of cloud systems has made it possible to build systems of multiple servers quickly and with agility. Scaling to demand makes it possible to only buy capacity that is needed to serve paying customers. To leverage this agility, automatic systems are needed to provision new systems, install necessary software and configure it to meet the specific business need of each case. (Hintsch, Görling and Turowski, 2016) Effective orchestration of resources is needed to meet acceptable quality levels (Tomarchio, Calcaterra and Di Modica, 2020).

For example, a company might have Windows workstations, Linux VPS and Android cell phones. Each operating system could be present in multiple versions. These heterogeneous systems have raised the need to abstract away the differences between operating systems, versions and platforms. Popular CMS:es have their own resource abstraction layers to deal with this challenge.

As computer systems store private data, business information and automate real life processes, the information stored in the systems must be protected. For organizations with a more serious threat environment, protecting against advanced persistent threats (APT) eavesdropping on communications with the help of computers is a risk. Configuration management systems protect against configuration mistakes that would be inevitable if humans were to manually configure large networks. Configuration management tools can automatically

check for conformance and automatically fix and report anomalies. For example, the US Government publishes its own security baseline for Linux and Windows systems (NIST, 2016).

1.1 Aim and Objectives

The aim of this thesis is to explore the survivability of configuration management systems.

This is done by identifying and adapting methods successfully used by the command and control (CC) systems of criminal malware.

In order to achieve the main aim, the literature for the communication architectures used in both criminal malware and benign enterprise CMS will be reviewed. As malware works in more heterogeneous and hostile environments than regular system administration, it could provide some ideas and architectures to be applied in configuration management systems. I will build a conceptual model for comparing and analyzing these tools. In the empirical part, I will construct a software prototype to experiment some of these technologies. I will evaluate the construct both theoretically and in an unreliable, hostile simulated environment. Finally, I will propose ways to improve configuration management systems. The objectives of this thesis and their linking to research questions are in table 1.

This work analyses and improves upon modern configuration management systems that are idempotent and versionable (text based, infrastructure as code). To be able to effectively find out how existing configuration management systems work, the work concentrates on free software, i.e. open source CMS.

Table 1: Objectives and research questions

OBJ	Objective	Research question
OBJ1	Review the literature to identify key malware techniques and key configuration management tools and techniques	Which malware resiliency techniques are applicable to benign configuration management systems? (RQ2)
OJB2	Develop a stage model for comparing malware and configuration management systems	How can existing stage models be adapted for comparing malware and configuration management systems? (RQ1)
OJB3	Develop key concepts for novel configuration management system	How can defining idempotent agent configuration be simplified? (RQ3)
OJB4	Design an advanced research prototype implementing the novel concepts	How can these techniques and concepts be implemented in a functional prototype? (RQ4)
OJB5	Validate the concepts by testing the prototype in laboratory environment	Based on load simulation, faulty network emulation and attack tree analysis, how does the resiliency of the configuration management software prototype - implementing some techniques adapted from malware - compare to a leading industry solution? (RQ5)
OJB6	Validate the technical benefits and potential business benefits in two case studies	What utility do the models and the research prototype provide when run in a field environment with business requirements? (RQ6)
OJB7	Identify and validate the potential business benefits in expert interviews	What potential do business benefits experts see for the models and the research prototype? (RQ7)

1.2 Research Questions

To answer RQ1, malware and penetration testing stage models were reviewed in the literature. An adapted model for comparing malware CC to configuration management was then built. Armed with this model, a review of successful malware campaigns and techniques and evolution was analyzed to select promising resiliency techniques for adapting to CM, thus answering RQ2. As the purpose of CM agents is to configure the agent system, idempotent agent configuration definitions were considered to answer RQ3.

In chapter "Designing Hidden Master Architecture", a research prototype implementing the features found in answers to RQ1, RQ2 and RQ3 was designed and built. First, a trivial prototype implementing the hidden master architecture for network communication was built and briefly tested. Next, a full and useful prototype implementing multiple resiliency techniques and a simplified system idempotent configuration definition was created. This system had multiple additional features to facilitate field testing in realistic environments, such as dependency free static compilation, full command line interface, the capability to upgrade agents over the wire, and single binary installation with built in key installation. This provided the answer to RQ4.

Finally, in chapter "Evaluating and Validating the Hidden Master Architecture", the research prototype was validated using multiple methods. To see the impact of the reduced attack surface, attack tree analysis was performed. Basic functionality was tested using fully emulated computers and load testing was done using simulated loads.

To compare and contrast research prototype to existing solutions, a network emulating real world problems was created. This network allowed error conditions to be generated, such as packet loss, jitter, latency and data corruption. Multiple scenarios were used to test and evaluate the two research prototypes. Further scenarios compared the main research prototype to a leading solution in the industry. These tests provided answer to RQ5.

Case studies in realistic field contexts were used to validate the utility of the solution and its ability to meet real life business requirements. First, a smaller case was performed using a heterogeneous network of computers (n=23) running multiple different (but similar) distributions, with each computer installed and managed by a different administrator.

The main case study was being performed in the systems of a company that controls and monitors Internet of Things (IoT) devices in its customers' premises. This case validated the utility of the concepts found in the literature research, model development and implementation. These case studies provided the

answer to RQ6.

Expert interviews were conducted to identify the potential business benefits of the proposed models and the research prototype. This was done to allow for broader perspectives than would be possible in case studies. Expert interviews provided an answer to RQ7.

Experts could consider possibilities that would be prohibitively expensive or risky to perform as case studies. Deploying a configuration management tool and possibly replacing an existing tool would be a large project for a company. Considering that a configuration management tool harbors the most valuable piece of data in the network that it controls, namely the secret keys of the master, a tool widely deployed in production would have to be mature, of high code quality and audited. This is further discussed in [chapter "Configuration management system as a target"]. It is self evident that the first deployments of a tool coded by a single person and never audited does not meet these criteria. This challenge was overcome by limiting the risk, scope and cost case studies, and answering the broader questions with expert interviews.

2 Literature Review and Related Work

2.1 Configuration Management

This chapter provides a review of configuration management systems, malware and relevant models for conceptualizing cyber attacks. The emphasis with configuration management systems and malware is on the network operation of these technologies, but the activities of the payloads are also examined. The literature review consists of three parts: configuration management, malware and conceptualizing attacks.

Malware is evolving fast (Silva et al., 2013; Lemay et al., 2018). This could be due to economic pressures in the criminal market. To understand the evolutionary development of the tools, tactics and procedures in malware versus configuration management systems, the literature review is ordered chronologically.

This literature review completes objective 1 "Review the literature to identify key malware techniques and key configuration management tools and techniques" to answer research question 2 "Which malware resiliency techniques are applicable to benign configuration management systems?". The look on stage models of cyber attacks forms the basis for developing a novel stage model for comparing malware and configuration management systems in chapter 4 "Designing Hidden Master Architecture Prototype".

2.1.1 Definition and Qualities of Configuration Management Systems

Configuration management systems (CMS) automate the definition and the deployment of configurations to computer systems. This approach is usually implied when talking about actual software configuration management tools. Poat, Lauret and Betts (2015) compare "three configuration management tools, Chef, Puppet, and CFEngine". I will use this narrow definition when talking about CMS. The CERN toolchain model (based on Google toolchain model), defines multiple activities based on DevOps principles (Andrade et al., 2012). In this model, my working definition best fits inside control and provisioning activities.

Layman texts and CMS documentation sometimes use the concepts of CMS and software configuration management interchangeably. Standards and academic texts, such as Scott and Nisse (2001) and Tripp et al. (1998), use the concept of software configuration management in a broader meaning that, on one hand, it includes project organizations and management practices, but on the other

hand does not require the use of CM tools to deploy changes on the target computers. Inside this broader definition, software configuration management tools could be seen as an automated subcomponent of configuration control. Perera (2016) takes this approach and uses the term "configuration deployment" similarly to my working definition of CM tools.

CMS are a critical part of large computer installations. According to Perera (2016), they can "extend life, reduce cost, reduce risk, and even correct defects".

Modern configuration tools are versionable and idempotent. In practice, the configuration manifests describing the target state of the network must be plain text to take advantage of version control systems. This "Infrastructure as Code" approach allows administrators to use software engineering methodology to control their network (Sharma, Fragkoulis and Spinellis, 2016).

Idempotence means that an operation can be applied multiple times without changing the result beyond the initial application. The concept of idempotence originates from algebra. Some consider idempotency to be one of the defining qualities of modern configuration management systems (the other being infrastructure as code). In practical CMS, idempotence is often achieved by describing the target state of the system and letting a CMS tool make changes only in case there are deviations from the target state.

Convergence is a concept similar to idempotence. Burgess, the author of CFEngine, uses the concept of convergence in his "Computer Immunology" paper (Burgess, 1998) to refer to a concept similar to idempotence.

To support a clear view of the network of computers, help operators manage large networks and allow consistent orchestration between hosts, CMS should be able to provide a single source of truth.

2.1.2 Evolution of Software Configuration Management

As pointed out in "Definition and Qualities of Configuration Management Systems", we examined configuration management systems using the definition of controlling and configuring a whole network of computers. As more modern technologies have been built on the ideas suggested earlier, I will review literature using a chronological approach, but sometimes point out the modern use of some early ideas.

Some problems of modern configuration management have been known from the early stages of computer science. As early as in the 1940s, von Neumann pointed out challenges of increasing systemic complexity of large systems. His lectures were later compiled as a book (Von Neumann and Burks, 1966) describing self replicating machines, von Neumann universal constructors. He designed this

system and pointed out the challenges without the use of a computer. In this work, he already describes concepts of survivability and systemic complexity:

> "[Computing] machines are designed to stop when a single error occurs. The fault must be located and corrected by the engineer, and it is very difficult for him to localize a fault if there are several of them. [...] The ability of a natural organism to survive in spite of a high incidence of error (which our artificial automata are incapable of) probably requires a very high flexibility and ability of the automaton to watch itself and reorganize itself. And this probably requires a very considerable autonomy of parts."

The same concepts and even the same biology metaphor have been used in later research. Burgess (1998) wrote about computer immunology, and later, in 1993, wrote a popular configuration management tool, CFEngine, that implemented some of these ideas. Burgess has since written many other widely cited papers improving on these ideas.

In 1983, project Athena was started at the Massachusetts Institute of Technology (MIT) (Treese, 1988). Project Athena was a key step in configuration management and created one of the biggest educational centrally managed computer networks in 1990, serving more than 10 000 customers (Champine, Geer and Ruh, 1990). In addition to more general learning from the project, researchers also developed some very successful software: the Kerberos authentication system and the X Window System (Arfman and Roden, 1992). Both are still widely deployed solutions, with the X Window System being a part of most Linux desktop distributions and Kerberos being a key part of Microsoft Windows Active Directory authentication. In addition to project Athena's results, its approach of running systems with actual clients while developing the systems and researching them is noteworthy.

Steiner and Geer Jr (1988) published the results of using their own configuration management system in production. Even though they don not use the terms and language of modern configuration management systems, it is evident from their descriptions that the concepts are the same. This might also be the reason why their work is sometimes missed or omitted. In their systematic literature review, Hintsch, Görling and Turowski (2016) used names and terms from modern configuration management, so the earliest publications they found were conference papers from the 12th Large Installation System Administration Conference (LISA) in 1998. Steiner (et al) published their work ten years earlier.

The system described by Steiner and Geer Jr (1988) consists of a master

configuration database (SCM), a push based networking layer with a master (DCM) and slave (Update Server) daemons, and finally a slave daemon for applying the configuration (Update Server). In their work, some words, such as "server", hold multiple meanings, including master (master-slave), host (including slaves) and daemon. An explanation of the terms they used are listed in table 2. The structure of the network is similar to modern configuration management systems used in the industry.

Their system shows many qualities of modern configuration management. They describe the target state of computers, striving to be idempotent. Despite the heterogeneity in their network, they control the system mostly from a single database (SCM), following the concept of single source of truth. Surprisingly, they do not write infrastructure as code, and it can be assumed that their system is not versionable. Instead, the data was stored in a relational database - RTI Ingres. The data took 13 megabytes of storage, which seems quite large. In comparison, the US Government Security Baseline Puppet configuration for Linux is about 0.7 MB (NIST, 2016). It is possible that the data in the database contains some other information in addition to the idempotent configuration. The reason for a lack of interest in plain text, infrastructure as code and versionability could simply be the state of version control systems in the 1980s.

Table 2: Components of Steiner and Geer Jr (1988) configuration management system

Component	Abbr.	Purpose
Service Management System	SCM	database of slave configuration
Data Control Manager	DCM	master daemon for distributing slave configuration over network
Update Server	-	slave daemon, applies configuration on slave

Steiner and Geer Jr (1988) list challenges not yet addressed by their system: providing TCP/IP network configuration for slaves and updating their software. It is somewhat surprising that these challenges are easily solved by today's tools. Dynamic Host Configuration Protocol (DHCP) automatically configures a computer that is connected to the network, providing the network specific information such as IP address, network mask, default gateway and domain name server address. Updating software in Athena 1988 required physical staff to visit the computer. Nowadays typical Linux distributions (similar to the Unix systems used in Athena) include advanced package managers that largely automate software updates and can run completely unattended. Popular

distributions include these tools by default: Red Hat and CentOS use YUM package manager (yum);and Debian and Ubuntu use Advanced Packaging Tool (apt). On the other hand, user and site specific configuration is still today an area of manual work and competing tools and paradigms.

In the same year, Harrison, Schaefer and Yoo (1988) described their solution to the problem of compiling and deploying software for multiple platforms with rtools. Even though their work contains many parts that are similar to a configuration management system, the goal is fundamentally different. Rtools is used for compiling and distributing application binaries, but does not really tackle their configuration. Even though their system could have been a solution filling the gap pointed out by Steiner and Geer Jr (1988), in hindsight it feels complex for stock software. In the 2010s, binary software was compiled by operating system distributors (in Linux) or by software vendors (Windows, OS X). The end user organization was left the smaller responsibility of choosing software to install and configuring it. However, the approach taken by rtools (Harrison, Schaefer and Yoo, 1988) is an area of interest for software developed internally.

DevOps is the idea of integrating software development (Dev) and operations (Ops). The term DevOps was only coined in 2008 and has gained popularity in recent years. On one hand, there are pressing business needs to keep adapting to the market, which in practice means developing new features and deploying them to production. On the other hand, operations need to keep the software stable and running. Practical methods of DevOps are automating builds and deployment (continuous integration, continuous deployment) supported by infrastructure as code, automation through deep modeling of systems and monitoring. (Brunnert et al., 2015)

Rtool provides automatic build and deployment (Harrison, Schaefer and Yoo, 1988), so it could be seen to include features of an early DevOps tool. However, it does not provide many important support functions, such as automated testing of the artifacts to be deployed or the features of modern configuration management. Also, rtools is not integrated to a version control system, so it provides no clear path to integrate the work of multiple programmers.

In -Hagemark (1990), Hagemark proposed a plain text language for configuring "many computers as one computing site". Even though Hagemark emphasizes other aspects, it is noteworthy as the first article I found to propose plain text language, which some modern articles consider a key aspect of modern configuration management, "infrastructure as code". This is also the first article found using the concepts of push and pull architectures in the context of CM. The plain text language has qualities of modern CM domain specific languages

(DSL). Hagemark's language is declarative (idempotent) and abstracts lower levels, similar to resource abstraction layers provided by modern CM tools. He also puts the responsibility on computer vendors to provide hooks to their systems. In 2012, Microsoft introduced PowerShell Desired State Configuration (DSC) to provide idempotent hooks for CM (Coulter et al., 2017). To improve scalability, the system should do "as much processing as possible on the client side". The same principle is used in some leading CM tools in 2018. SaltStack calls this the "no freeloaders" principle. Hagemark touches the concept of orchestration by providing examples where single change affects multiple hosts on a network. Similar to a lot of research coming later, Hagemark's interest is concentrated on what happens in slaves, and he did not address network security or authentication at all.

The developer of CFEngine, one of the first widely used configuration management tools, published his "Computer Immunology" in 1998. Burgess (1998) calls for more fault tolerant computer systems. He emphasized features common to modern CM tools, especially idempotency. Concepts of single source of truth and resource abstraction are mentioned briefly. The use of infrastructure as code approach is evident from the fact that configuration is plain text. The list of features he considered unique in CFEngine is in table 3. His concept of fault tolerance is practically the same as survivability used in this thesis: the ability to "cope with and recover from errors automatically". He emphasizes feedback mechanisms, and proposes CFEngine for this purpose. According to Burgess (1998), one of the key difficulties for the "immune system" (CM tool) is to know the history and current state of the system due to a lack of existing software to do this. This idea is somewhat similar to the concept of continuous monitoring in DevOps. Nowadays, there is a lot of software for collecting detailed information about system state and operations, such as auditd and host based intrusion detection software, but it remains to be seen if such tools will be integrated with CM tools.

Burgess (1998) names convergence as key point of CFEngine and computer immunology: "i.e., one describes what a system should look like, and when the system has been brought to that state, CFEngine becomes inert." His definition of convergence is the same as idempotency in this thesis and many modern CM tools. Even though Burgess has been an influential, cited author, and his tool CFEngine has been a leading tool for some time, the features have been mentioned in earlier papers. It seems that more unique was the bringing together of these features to create a practical, general purpose tool that is in use in multiple, unrelated organizations. It is worth noting that even though modern tools use different names for the concepts, they are very similar to those described by Burges and used in his CFEngine.

Table 3: Unique features of CFEngine according to Burgess (1998)

Feature (Burges)	Modern description
Non-procedural programming, descriptive language	Declarative DSL
Convergence	Idempotency
Abstract classes	Resource abstraction
Help administrators communicate, prevent conflicting work	Single source of truth

2.1.3 Existing Literature Reviews

Multiple literature reviews have been written on configuration management systems. This look on existing literature reviews looks both the results and their methods. In the results, especially key themes and identified gaps and areas for future research are interesting. Methods of conducting the research help guide the systematic search and review performed in this work.

Rajapakse et al. (2022) conducted a systematic literature review on 54 peer-reviewed studies, then used thematic analysis on the extracted data. The focus of the work was DevSecOps, so only parts of the review are of interest here.

They searched IEEE Xplore and ACM Digital Library for terms related to DevSecOps, finding 283 non-duplicate papers. Their inclusion criteria was being on topic, having English full text available, peer reviewed, not a review and longer than five pages. Quality assessment was limited to a meeting two of the three items on a yes-no scale: context and solution described; and design or method suits the aims. Forward and backward snowballing found six more papers.

Thematic analysis by Rajapakse et al. (2022) identified main themes of people, practices, tools and infrastructure. Only some of the 21 challenges identified in these categories are within the scope of this work.

Rajapakse et al. (2022) had multiple conclusions relevant to this thesis. New technologies are needed to support rapid development cycles. Developers should minimize tool-related security issues. More solutions should be empirically validated. (Rajapakse et al., 2022)

Rahman, Mahdavi-Hezaveh and Williams (2019) performed a systematic mapping study to find gaps in IaC research. Inclusion criteria was peer-reviewed IaC publications since 2000 in English, with full text available for download. Quality of each article was evaluated using nine point quality questionnaire by

Kitchenham et al. (2012). Rahman, Mahdavi-Hezaveh and Williams (2019) found 31 publications related to IaC. Most of the publications (52%) were related to tools. As their conclusion, they "observe the need for research studies that will study defects and security flaws for IaC".

The search approach taken by Rahman, Mahdavi-Hezaveh and Williams (2019) seemed laboursome. Their search criteria matched nearly ten thousand titles, which they manually filtered to 31 IaC-related publications. This meant manually discarding 1-31/9840 = 99.7 % of matches. Most articles were manually discarded based on title only. For the systematic search performed in this thesis, some of this work could be avoided by carefully developed search criteria and by using automation.

Kumara et al. (2021) reviewed gray literature on infrastructure as code (IaC). Gray literature review meant that they used Google to search for reports, blog posts, white papers and official documentation of IaC languages (domain specific languages). Their research questions were definition and classification of IaC; and good and bad IaC practices. Their inclusion criteria included English full text articles without paywall matching the focus of the study, published within the last three years. Quality assessment included reputation of author and organization.

Kumara et al. (2021) found 67 suitable sources. Writers found multiple areas for more attention and further research. They found that "IaC patterns and anti-patterns/smells" need further research; "software maintenance, evolution, and security of IaC is in its infancy and deserves further attention"; and that "several best practices exist, but they mostly concern the complexities inherent within IaC".

Wurster et al. (2020) compared DSLs of 56 configuration management tools. They searched for open source IaC deployment technologies using ACM Digital Library, IEEE Xplore and Google. Techniques were ranked using Google search volumes, top rankings going to Puppet, Chef, Ansible and Kubernetes. These techniques were categorized to general purpose, provider specific and platform specific. Wurster et al. (2020) suggest their own model, "The essential deployment metamodel", for comparing DSLs. Wurster et al. (2020) don't identify gaps in research, but they suggest future research could use their model.

The work of Wurster et al. (2020) is part of the tool centric tradition in IaC research. The scope of the problem and related tools is partially different from this thesis. Both are looking at open source, general purpose IaC tools that directly configure hosts. Wurster et al. (2020) also consider provider- and platform specific tools; containerization and virtualization tools that only have

IaC as a minor part; and tools that only rent or provision machines but don't configure them. The need to use a model to map different concepts and words to be able to compare DSLs seems to emphasize the extra conceptual burden caused by these single purpose languages, a challenge tackled by this thesis in chapter "Defining Configuration".

Kosar, Bohra and Mernik (2016) conducted a systematic mapping study of domain specific languages (DSL). They concluded that research on this area has multiple gaps. Considering this dissertation, interesting gaps in DSL research included lack of evaluation research; DSL integration with other software engineering practices; DSL phases of domain analysis, validation and maintenance; and lack of formal methods.

A systematic literature review on practical configuration management systems has been conducted by Hintsch, Görling and Turowski (2016). They state that their work is the first literature survey on CMS tools. They searched twelve online databases with a list of tool names and the string "configuration management", a search similar to "puppet AND configuration management". The details of their search is in table 4. After filtering, they identified 159 relevant articles. Only 36 of those were journal articles. The conference with most papers was LISA.

Table 4: Search used by Hintsch, Görling and Turowski (2016) in their literature review.

Search	Time frame	Publication type
(BCFG2 OR Cfengine OR Chef OR Puppet OR LCFG OR "Bladelogic Server Automation" OR NSM OR "Tivoli System Automation for Multiplatforms" OR SCCM OR "HP Server Automation" OR "Netomata Config Generator") AND "configuration management"	Before or on Feb 2015	journal OR conference paper OR transactions

The earliest publications Hintsch, Görling and Turowski (2016) have found are either from 1998 or 1994 (they present both years as the first in different

parts of the review). As we have seen, this is approximately ten years after many key concepts of configuration management have been proposed and even used in production by Steiner and Geer Jr (1988), Harrison, Schaefer and Yoo (1988) and Hagemark (1990). Their review has also missed Burgess (1998), even though they have included two of his newer publications. As early writers have used different names for the concepts or simply have not given them specific names, any approach relying only on searching a list of words is likely similarly limited.

Hintsch, Görling and Turowski (2016) have inductively sorted these articles into 17 categories. Security, the most interesting category for my research, contains only 13 publications. Some of those publications look at various aspects of security not related to protecting the network part of the system. For example, Adesemowo and Thompson (2013) considers problems of IT asset disposal; Anderson and Cheney (2012) examines lack of access control and the semantics of configuration languages. The small number of relevant articles in the literature review by Hintsch, Görling and Turowski (2016) show that there could be a research gap in the survivable networking of configuration management systems.

The references of the found articles (backward search) or articles referencing them (forward search) were not considered by Hintsch, Görling and Turowski (2016). Only articles mentioning one of the 11 CMS tools were considered, which has left out articles not mentioning any tools by name. Also the list of tools is taken from Delaet, Joosen and Van Brabant (2010), which only includes tools that existed in 2010, and leaves out newer tools such as Salt. However, one would assume that works on newer tools would at least mention older ones in the full text.

2.1.4 Related Work on Protecting and Scaling the Master

Research builds on existing work, and tries to add to existing gaps. To this end, both identified gaps and earlier research on the key areas of this work was looked at. Identified gaps were collected from areas recognized for future research or gaps by key articles based on manual literature review. To make sure that manual search did not miss key articles, searches on key areas were performed on four literature databases.

The questions for searches were created from key areas of this work, related to the contributions built in the rest of this work. The focus was on protecting master secrets and using asynchronous networking in configuration management. Key contributions of this work, including the Hidden Master architecture, are on these areas.

In their article on designing literature studies in software engineering, Kuhrmann, Fernández and Daneva (2017) list library databases they consider standard for systematic literature reviews and systematic mapping studies. They recommend choosing those six databases or a subset of those for search. For this review, four of those databases were chosen:

- IEEE Digital Library (Xplore)
- ACM Digital Library
- ScienceDirect (Elsevier)
- SpringerLink

For choosing publications for further review, exclusion and inclusion criteria was created, using Kuhrmann, Fernández and Daneva (2017) as an inspiration. Inclusion and exclusion criteria is in table 5. Articles were considered to be on the main topic when the focus was on configuration management systems as defined by this work. Articles based on alternate definitions such as release engineering or very broad meanings including software project organizations and management practices. Articles discussing only automated system provisioning without configuring software inside them, just containerization or software defined networking were excluded. Articles concentrated on non-configuration management aspects of DevOps, such as continuous integration and continuous delivery pipelines were excluded. To answer each specific question, inclusion criteria 8 required that article relates to the specific topic of the question, such as "protecting master secret keys". When an article was ambigous on criteria 8, the full text was consulted to make a decission.

Table 5: Inclusion (I) and exclusion (E) criteria.

No.	Criterion
1	I: Title, keywords and abstract make explicit that paper is on the main topic, configuration management
2	I: Presents contributions related to the main topic
3	I: in English
4	I: Peer reviewed (journal article or conference paper)
5	I: Full text available for download
7	E: Main topic of the article is release engineering, different and conflicting definition of CM, software defined networks or non-CM aspect of DevOps.
7	E: Paper is already in result set
8	I: Title, keyword and abstract make it clear that the article is on the specific question topic to question

Literature databases were searched for research articles on protecting master

secrets or using asynchronous networking in configuration management (the hidden master architecture). The initial versions of the query were created in parts, for example to include articles mentioning “infrastructure as code” and “asynchronous”. These queries were quite broad. These initial queries were combined to a single query using boolean logic.

Search query: (“infrastructure as code” OR “configuration management system”) AND (“resilient” OR “resiliency” OR “public key” OR “hidden master” OR “intermediate node” OR “async” OR asynchronous)

Search was adapted to each literature database to make use of special features of each tool. For example, inclusion criteria (table 5) required that articles are in English and full text is available for download, and these requirements could be checked as search filters in most databases. The extra filters for each database were

- IEEE Explore: “Subscribed Content”
- ScienceDirect: “Subscribed Journals”; Article type: “Review article” or “Research article” (this limitation only excluded 2 articles); Subject areas: “Computer Science” or “Engineering”.
- ACM Digital Library: Search was limited to words in abstract to make it similar to search in other literacy databases. As ACM would search full text by default, a search query optimized for metadata would produce large number of false positives.
- SpringerLink: “English”. “Include Preview-Only content”: no. Discipline: “Computer Science”. Discipline “Engineering” had three results, all of which were irrelevant. To limit to peer reviewd publications (an inclusion criteria), separate searches for “Article” (33 hits) and “Conference Paper” (11 hits) were performed and the results combined.

Table 6: Search results from literature databases.

Database	Hits
IEEE Explore	7
ScienceDirect	69
ACM Digital library	3
SpringerLink	44
—	—
Total	123

table 6 shows the result of search in the database.

Search results from each literature database were evaluated against inclusion

and exclusion criteria (table 5). After this, 48 articles were identified for further processing and their full text was downloaded. These articles covered a large time span, from 1988 to 2023. Even though the number of articles could have been reduced with a recency filter (e.g. last 5 years), this could have missed important content. As discussed earlier in this literature review in "Evolution of Software Configuration Management", multiple inventions in this field are both invented and deployed to field much earlier than often thought.

Second round of filtering concentrated on criteria 8, applicability of the item to the specific topic, "protecting master secrets or using asynchronous networking in configuration management". For the second filtering round, full text was available. Reasons for filtering out articles at this point were that they were not concentrated on configuration management or agent-master communication. Filtering for being on topic identified 17 items.

The items are listed in table 7. Most of the identified items, 12, were journal articles. The rest 5 were conference papers. The metadata provided by library databases misidentified some conference papers as book sections. One of the conference papers was authored Tero Karvinen, the author of this thesis, with professor Shuliang Li as the second author. The oldest item was published in 2004 and the newest in 2023.

Table 7: Included publications

Publication
Balis et al. (2018)
Caballer et al. (2023)
Colarik, Thomborson and Janczewski (2004)
Karvinen and Li (2017)
Kinkelin et al. (2019)
Kos, Milutinović and Čehovin (2015)
Mansouri, Prokhorenko and Babar (2020)
Marsa-Maestre et al. (2019)
Pasquale et al. (2009)
Rahman, Mahdavi-Hezaveh and Williams (2019)
Rong et al. (2022)
Sherman et al. (2005)
Siqueira et al. (2006)
Stocker et al. (2022)
Toffetti et al. (2017)
Ullah et al. (2023)

Publication
Zhao and Guo (2018)

Ullah et al. (2023) gathered research on IoT configuration management, proposed a taxonomy of those systems and reviewed both concept-only and production configuration management systems on this area. Their definition of "orchestration" is similar to the definition of configuration management as used in this thesis, and the definition of Cloud-to-Things Orchastration Solution (CoTOS) is similar to configuration management system (CMS). Similarly, they define cloud-to-edge computing continuum (also cloud-to-things continuum) as extension of cloud with "energy-efficient and low-latency devices closer to the data sources located at the network edge". This is quite similar to common definition of IoT, the Internet of Things.

Ullah et al. (2023) identify multiple challenges relevant to this thesis. The devices is are heterogeneous, having different operating systems, architectures and computational capabilities. Network is volatile, as the nodes can go down, lose connectivity and their locations may change. End nodes need to be monitored. Software needs to be deployed and reconfigured based on a policy. Massive scale of networks require scaling. System must guarantee security of the whole system against different attack scenarios.

To compare different Cloud-to-Things Orchastration Solutions, Ullah et al. (2023) create a taxonomy. They consider such facets as design of the network, supported application types and providers and security; all of which have multiple subcategories. Some areas in this taxonomy are of specific interest in this thesis.

In environment category, where Ullah et al. (2023) contrast single cloud operation to multi- or cross cloud operation, which allow avoidance of vendor lock-in. In connectivity, their taxonomy notes that [slave] nodes can automatically register and reconnect. It seems that automatic reconnectivity would be a hard requirement for any IoT system, so it's surprising that it is part of taxonomy. As automatic registration requires establishing trust between nodes, it seems more challenging. They define a similar concept, "edge resource authentication", in their security handling category. This work proposes a novel solution to some of these challenges, the Hidden Master architecture.

In their review of 10 concept-only orchestration systems and 9 research project based orchestration systems, Ullah et al. (2023) found only one multi- or cross cloud tool supporting both automatic registration and automatic reconnectivity, PrEstoCloud. According to whitepapers provided by the PrEstoCloud project,

their security over untrusted network is based synchronous VPN network. Different PrEstoCloud papers consider different VPN solutions, including IpSec and OpenVPN (Jacquemart and Urvoy-Keller, 2018); and fully meshed peer-to-peer cjdns (Ledakis et al., 2018).

Considering the actual configuration of software on the slave node, Ullah et al. (2023) define multiple criteria for comparison. Software can static or user defined way of reacting or procatively making changes. Security handling is identified as challenge due to heterogeneity and low powered devices. They identify security areas such as configuring application security settings and inter-component communication. Finally, they call for higher level of abstraction in defining configuration. It should be noted that the use general purpose programming language for configuration management proposed chapter "Defining Configuration" provides a novel and open ended answer to some of these challenge.

The six key areas for lacking current solutions identified by Ullah et al. (2023) include proactive run-time reconfiguration, decentralized architectures and security management.

Caballer et al. (2023) have developed their own Infrastructure Manager application. The application takes instructions in TOSCA, and configure computers using Ansible. OASIS TOSCA (Topology and Orchestration Specification for Cloud Applications) is a YAML based domain specific language for defining and managing computers, services and their relationships. (Lipton and Lauwers, 2019). Ansible is one of the leading configuration management tools, further discussed in Leading Configuration Management Tools.

As Ansible requires the slave nodes to be in known addresses and running a server visible to master, it is most suited for configuring servers and less useful for moving nodes such as laptops and IoT devices. To bypass this challenge, Caballer et al. (2023) use reverse SSH tunnels and SSH bastion hosts. They don't give very detailed explanations how this is done.

Based on common knowledge on using reverse tunnels in SSH, it could be assumed that the master is running an SSH server and is in a known address. Then slaves could connect server using SSH and form a reverse tunnel. As Ansible uses SSH, another SSH connection trough this tunnel could then be established. Based on my experiments on this approach in different context, the solution requires manual setup and care. Some issues are re-establishing lost connection, keeping client (in this case, slave node) address stable and managing which ports are used on the server side. Some tools such as 'autossh' help automate some parts of this setup, but without further clarification and

tests shown by Caballer et al. (2023), it's not easy to see how this solution could be robusts, resilient or scalable. The use of SSH proxy / bastion host in this setup is not obvious as the authors don't clarify it further.

For deploying the nodes (e.g. renting a new host from a cloud provider) Caballer et al. (2023) have developed their own tool. Abstraction of different cloud providers seems convenient, but multiple existing tools could reduce the workload of creating these connectors, such as 'salt-cloud' or Terraform, which is also mentioned by the authors. Similar to this thesis, Caballer et al. (2023) performed both laboratory testing and validated in their work in a real use case.

In their article on smart grids, Stocker et al. (2022) used WireGuard VPN to secure their configuration management system over untrusted network. They note that in this overlay VPN network, "only the coordinating host be visible externally". For scaling purposes, they mention the possibility to enable peer-to-peer communication with WireGuard. This synchronous setup seems to leave the machine with master private key always visible to network, a challenge tackled by the Hidden Master architecture in this thesis. For the actual configuration management, they used Ansible, which uses it's own DSL and push architecture, connecting to slave nodes using SSH. Stocker et al. (2022) developed an inexpensive laboratory test bed. It consisted of three Raspberry Pi (inexpensive small Linux computers) with both real (physical) and emulated energy related hardware.

Rong et al. (2022) suggest that different organizations could securely provide services to each other using federated identities. Decentralized identities would be provided by blockchain and smart contracts. They call their architecture OpenIaC with a catchy phrase "network is my computer". Even though some standards are mentioned and the work has some diagrams, the details of the model are left as an exercise for the reader. Rong et al. (2022) identify multiple challenges of interest. Service orchestration (provisioning and configuration management) should be highly reliable and scalable. Infrastructure as code (used as a near synonym for configuration management) has multiple challenges that should be solved. Languages should find a balance between declarative and imperative expressions - a challenge tackled in chapter "Defining Configuration". Rong et al. (2022) highlight the challenge of integrating different systems on multiple levels. The complexity of infrastructure, provisioning and configuration management tools is emphasized when production infrastructure is scaled out geographically and in load. Vertical integration from network to applications and services forms another layer. Finally, managing the organizations and people who manage these systems is equally complex.

Rong et al. (2022) call for redesigning secure networking. Network protocols such as Ethernet are old and built on assumptions that no longer hold. New networking solutions should be built for pervasively hostile network environment, and be based on zero trust networking architecture. Open challenges include performance, resilience, adaptability to application requirements and supporting open and decentralized services. It should be possible to scale and move processing geographically, across organization and jurisdictions and trough the continuum from edge to cloud. (Rong et al., 2022)

Even though many of the challenges noted by Rong et al. (2022) are similar to those identified by other works, some of their solutions are not as easy to agree with. They suggest replacing Ethernet with some other, yet undefined protocol, and list the lack of internet protocol kind of ports as a downside. The TCP/IP stack is a layer model, where different features are provided by a different layer. For example, ports are provided by TCP in the transport layer. Verifying endpoints is usually done in the application layer. Considering the amount of networking hardware existing, their call to redesign "the layer 2 network architecture (hardware, software and protocols) seems like a major and thankless task.

Mansouri, Prokhorenko and Babar (2020) scaled databases by automatically extending private cloud to public cloud. Their focus was on testing if the scaling works, considering the latency and bandwidth limitations caused by the network distance and similar factors. The configuration management system to provision, configure and control nodes was based on WireGuard VPN and Terraform (predecessor of OpenTofu) provisioning tool. The approach to protect master-slave network traffic using VPN is similar to that used by Stocker et al. (2022) discussed earlier. Even the choice of VPN tool, WireGuard, was the same. In their search for related studies, all studies listed by Mansouri, Prokhorenko and Babar (2020) using a network connection use VPN: public VPN, IPsec tunnel or WireGuard. Their results regargind cloud bursting highlight that performance is dependent on the design and implementation of the actual tools and procols. They found that cloud bursting is efficient with MongoDB and MySQL; but not efficient with Cassandra, Riak, Redis and Couchdb. Mansouri, Prokhorenko and Babar (2020) list multiple challenges with hybrid cloud: security, resilience, scalability and reliability. Resilience depends on cloud infrastructure (guaranteed by providers) and network connection between clouds. Scalability and reliability could be enhanced by having data in multiple clouds. Connections can be costly, so secure, resilient and cost-free solution such as WireGuard could provide required features in cost-effective manner. (Mansouri, Prokhorenko and Babar, 2020)

Rahman, Mahdavi-Hezaveh and Williams (2019) performed a systematic mapping study on infrastructure as code. They found that the field of IaC research is small, but the interest in the area is growing fast. They found only 32 publications matching their criteria, including one where the author of this thesis is the first author. There are multiple gaps and potential research avenues: anti-patterns, defect analysis, security, knowledge and training; and industry best practices. There is little research on the security of configuration management systems. For the IaC scripts, they highlight the high risk of defects as these script change the configuration of production systems in large scale, with the risk of causing large outages. They find that IaC scripts themselves should be studied for defects and anti-patterns. IaC scripts should be analyzed using common weakness enumeration (CWE), a method usually used with regular computer programs. (Rahman, Mahdavi-Hezaveh and Williams, 2019).

Marsa-Maestre et al. (2019) propose REACT model, where a network of computers is reconfigured in reaction to anomalous behavior indicating potential compromise. Zero day attacks are new attacks that are not detected by signature based antivirus and similar technologies, and the related vulnerabilities are not yet patched in software. REACT detects potential compromises based on zero day attacks by using existing intrusion detection system's (IDS) anomalous behavior detection. Based on a multilayer graph of the network, REACT finds high risk nodes and reconfigures them. Reconfiguration means both removing and provisioning containers and machines, and changing network connections between them. High risk nodes are those that are network reachable by compromised hosts (even trough multiple hops) and those that have similar configuration. They ran a Python simulation on these reconfigurations. For future research, they suggest network resilience under multiple simultaneous attacks. (Marsa-Maestre et al., 2019)

The approach taken by Marsa-Maestre et al. (2019) is interesting, but they don't provide implemention of their system, so they have not performed emulated testing. As IDS anomaly detection systems produce many false positives, many network reconfigurations could pose a challenge to continuous operation of production systems. As Marsa-Maestre et al. (2019) consider nodes to be connected if there is a network path between them or similarity in configuration, alerts could affect large parts of the network. Configuring services and deciding network connectivity between them is often a decision mandated by multiple factors. Changing a working configuration and running installations without human intervention sounds risky. The connected operation of many systems require establishing trust between systems, usually implemented by accepting cryptographic keys. If the same keys are redeployed to keep existing trust, these keys could already be compromised. If new keys are generated, there is a

challenge of securely re-establishing trust.

The REACT system proposed by Marsa-Maestre et al. (2019) is a configuration management system, using master-slave architecture where master has full control over all slave nodes - all computers in the network. This makes the REACT system itself the most valuable target in the network. Due to reactive nature of this system, the master (with the cryptographic keys to control slaves) must always be online. Compromising those keys would allow attacker to gain full control of all layers of the system. Also, there must be a network path that connects this master node to all other nodes.

Kinkelin et al. (2019) find the centralized power worn by a single administrator and his machine key security risk. They propose distribution the process of creating and approving configuration, both to multiple persons and to multiple devices. They note that such requirement for multiple reviews has already been in production in major source code repositories GitHub and GitLab, at least from 2019. In addition to just requiring multiple approvals, Kinkelin et al. (2019) discuss possible methods of conflict resolution in case multiple admins create conflicting configuration. They have developed their own solution, TANCS, based on blockchain and smart contracts.

The actual solution, TANCS, is described in a conference paper by Kinkelin et al. (2018). The approval operations happen in Hyperledger Fabric, a private blockchain. Popular configuration management tool Ansible is run locally by each slave node. It seems slave nodes run a client that retrieves approved configuration from the blockchain. The focus of the work is how configuration are approved. The requirements they set for the system are multi-party authorization, accountability and tractability and tamper resistance.

The idea of applying multi-party authorization to configuration management system by Kinkelin et al. (2018) seems useful, as configuration management system is such a critical part of system security. Similar to the blockchain approach proposed by Rong et al. (2022), one could ask if blockchain is needed here, or if simple signatures with public keys would be sufficient. Configurations are themselves information to be protected, and can often contain secrets. For master-slave network communication, Kinkelin et al. (2018) simply state that slaves retrieve configuration from the blockchain. Their requirements, and the guarantees they identify in their technology, concentrate on tamper-resistance. Compared to the well known CIA triad, the system accounts for integrity, but little attention is paid for confidentiality and availability. As the system is decentralized, the direct connections between peers should be looked at from the point of view of revealing structure of the network, especially the computers allowed to approve new configuration.

Zhao and Guo (2018) propose a simple server-side web application to manage network infrastructure. A single master server, the "Network Configuration Management Server" provides a web interface for administrators. The same server holds the secrets to control and monitor network devices trough telnet, SSH, SNMP and similar protocols, and keeps a database of current state of slave nodes. Web server administrator interface provides authentication and authorization. (Zhao and Guo, 2018)

The solution proposed by Zhao and Guo (2018) is simple and thus easy to understand. However, the single server with all the secrets creates a single point of failure to the network, and highly tempting target for attackers. Having a large amount of different services and protocols in the same system creates a large attack surface to protect. The unencrypted and unauthenticated master-slave network connections allow eavesdropping and tampering of configurations from master to slave, and monitoring data from slaves to master. It should be noted that the use of old and unprotected protocols might be in some cases mandated by the limits of old networking equipment. For monitoring purposes, the connections are likely to happen automatically. In addition to just tampering downstream instruction to slaves, an adversary on the same network could use a man-in-the-middle attack by pretending to be a slave node and attacking the master. If data from slaves is presented in web interface, typical web attacks such as cross site scripting or cross site request forgery could be one approach. The requirement to use a web interface to upload and download configuration seems to be against principles of IaC, even though this could be mitigated by creating APIs and related clients.

Balis et al. (2018) propose a system to optimize performance of environmental sensors. During normal operation, features such as cost of operation and energy consumption (expected system lifetime) should be preferred. During a disaster, preference should be given to quality of service, including data measurement, processing and transmit interval. Balis et al. (2018) have created a model to optimize operation to these conflicting requirements. In their call for optimizing the whole system and not just individual nodes, the approve the need for higher levels of abstraction. Their work is focused on environmental threats (as opposed to human adversaries), such as flooding. They leave encryption of master-slave network communication for future research, with a plan to use symmetric AES encryption.

The holistic approach taken by Balis et al. (2018) is useful from business perspective. If there are no human decission makers available, why spend limited resources to collect data that's not used? The optimization and the use of different operating modes provided could be useful part of configuration

management and monitoring systems. However, more attention could be paid to protecting the master-slave network communication. Without encryption, it suffers from similar threats as the solution created by Zhao and Guo (2018). Even their future plan to protect master-slave network communication with symmetric encryption has limited use against human adversary. If the physical sensors deployed on the field have the same key, adversary can obtain any sensor, extract the key and then eavesdrop and tamper communication with all sensors, falsifying both downstream and upstream communication and cause incorrect decisions to be made. If each node has different key, key management will be a major challenge as each key must be copied to both master and slave using a secure channel.

Toffetti et al. (2017) developed a system where slave nodes automatically elect a leader to control the network in case one machine goes down. Key-value store Etcd holds both the instructions to slaves and current known states of slaves. If current leader etcd node goes down, an automatic vote for new leader is done using Raft protocol. Configuration is defined using a custom Etcd directory structure as custom DSL. This hierarchical graph can be represented as an indented text file. The actual configuration on slaves is done with CoreOS Fleet. The system includes multiple additional components, such as ElasticSearch-Logstash-Kibana monitoring and log consolidation stack.

In the system proposed by Toffetti et al. (2017), automatic change of master node could be expected to improve availability. As Etcd directory server compares known actual state of slaves to the target given by administrators, it's implied that each of these servers have the secret keys to control the whole network. Instead of single tempting target for attackers, this seems to risk compromising the whole system in case one participating node is compromised. By moving the work done in the directory servers to slave nodes, directory servers could simply serve copies of the instructions from the administrators, thus avoiding not only the vote but the storing of secret keys to machines in such vulnerable positions. This is indeed what is done in the Hidden Master architecture proposed by this thesis.

Kos, Milutinović and Čehovin (2015) propose nodewatcher, a server to generate configured firmware images for Open Wireless Network of Slovenia and monitor them. Wlan slovenija is a 400 node open community wireless network. Nodewatcher provides a web server for image generation and monitoring, written in Python Django framework. Node (wireless access point) owners log into it to generate firmware images to their access point, bundled with related configuration. Schema registration uses DSL based on Django object-relational mapper. Node owners are responsible for flashing firmware images and their

configuration. Node details are saved to central server for automatic monitoring. Generated images contain an agent that collects metrics and sends them to central server using JSON files served over HTTPS connection, either by publishing them on a local web server (pull) or making periodic HTTPS POST requests to central server. Monitoring data is consolidated and visualized on the central server. Nodewatcher is made more interesting by the fact that it's actually deployed in production network, and successfully used by node administrators not employed by a single organization.

Considering security, Kos, Milutinović and Čehovin (2015) especially consider challenges in availability and integrity. It's implied that confidentiality is handled using TLS with HTTP. They state that redundancy might be a future solution to availability, but that is not important because operation of nodes and routing protocols does not rely on nodewatcher. After initial, partially manual provisioning, nodewatcher performs only monitoring function. For security, they agree that centralized network management installation might be a key target for an attack, and note that configurations in nodewatcher might contain secrets such as passwords. So, they suggest the use of public key authentication [in node configurations] where possible. Communication between the central server and nodes can be protected by mutual authentication with TLS. (Kos, Milutinović and Čehovin, 2015)

What comes to encryption and network communication, Nodewatcher proposed by Kos, Milutinović and Čehovin (2015) has similar architecture to "Network Configuration Management Server" by Zhao and Guo (2018). On the one hand, having a single web server manage everything provides for an easy to understand architecture and all the convenience of server side web programming. On the other hand, web server holds the keys to the kingdom, is always on and reachable from the Internet.

Nodewatcher server is more valuable as a target than implied by the Kos, Milutinović and Čehovin (2015). Compromizing the nodewatcher server eventually provides access to all nodes, as malware or extra keys can be installed to updated firmware images. The use of regular TLS communication depends on trusting all certificate authorities accepted by the browser, which brings in many trusted parties from multiple jurisdictions. This is in contrast with self-generated certificates (key pairs) used by many other provisioning and configuration management systems. The article does not detail how the keys are managed. It would be interesting to know how TLS keys related to master-slave communication are generated and updated.

Pasquale et al. (2009) propose publishing configuration changes across organization boundaries. Local changes are discovered by an agent installed on

all systems, consolidated on a web server inside organization and published as an Atom feed of XML documents. Configuration management systems can react to these changes, even across organizational borders. To facilitate this, an aggregation server can collect data from multiple organizations.

As Pasquale et al. (2009) don't consider security implications, it's not easy to see how this model could be applied to production networks. The article does not discuss encryption, and it does not define a threat model. The actual work of making changes (versus just detecting the need for them) is not discussed. It could be expected that making changes to configuration management of production systems based on automatically detected changes on systems of another organization would be a risky undertaking. As this article is from 2009, it could reflect more idealistic and less adversarial view of networks connected to the Internet.

Siqueira et al. (2006) considers automatically managing networks. The goal is for agents to automatically discover master in local network, and retrieve policy from them using web technology. Masters in different networks communicate with each other. (Siqueira et al., 2006). The paper does not go to details how the communication could happen, does not define threat model or discuss encryption. From the architecture it seems that masters in each network ("Autonomic Policy Server") is highly valuable target for attack, having full control of each network ("Ambient network") and information about other nearby networks.

Sherman et al. (2005) have developed a configuration management system ACMS for controlling over 15 thousand servers in Akamai Network. ACMS uses web servers to asynchronously to deliver configuration over Akamai's proprietary content delivery network (CDN). The system includes acceptance algorithm, but unlike some other systems, it's not about the content of configuration but instead to more reliably store a copy of the file in storage storage points. The system has multiple layers: publishers, any of which can create new configuration, storage points and Akamai CDN as a scaling mechanism; and receivers on slave nodes. (Sherman et al., 2005)

According to Sherman et al. (2005), the ACMS system is in production on "over 15 000 servers deployed in 1200+ different ISP networks in 60+ countries". This is both a considerable proof of the system working, an indication that their approach scales well and an indication that HTTP protocol can be used in highly scaling configuration management. The scaling is achieved in part using their proprietary CDN and proprietary error reporting infrastructure, so further research would be needed to see if the approach works with other systems.

Sherman et al. (2005) make no claims of the security of their system. They don't define threat model, key management plan or discuss encryption in any way. In fact, they state that "The techniques that we use to accomplish [security of configuration updates] are standard, and we do not discuss them further in this document.

Colarik, Thomborson and Janczewski (2004) searched patent literature for interesting update and patch management systems and created a taxonomy of the findings. The taxonomy they created could inspire other taxonomies and chain models, but some concepts would need to be clarified. For example, the authors assume that master is always placed on server. As we can see in chapters "pull architecture" and "push architecture" in this thesis, master (of master-slave architecture) can be placed on server or client (in client-server architecture).

Key research gaps relevant to this work are summarized in table 8.

Table 8: Research gaps in configuration management systems relevant to this work

Gap	More research is needed on
Security	Developers should minimize tool-related security issues (Rajapakse et al., 2022). We "observe the need for research studies that will study defects and security flaws for IaC" (Rahman, Mahdavi-Hezaveh and Williams, 2019). "[Software] maintenance, evolution, and security of IaC is in its infancy and deserves further attention" (Kumara et al., 2021). System must guarantee security of the whole system against different attack scenarios (Ullah et al., 2023). Decentralized architectures and security managment are among the six key areas lacking current solutions (Ullah et al., 2023). Secure networking should be redesigned, based on zero trust networking architecture, for pervasively hostile network (Rong et al., 2022). Security is a challenge in hybrid cloud (Mansouri, Prokhorenko and Babar, 2020). There is little research on the security of configuration management systems (Rahman, Mahdavi-Hezaveh and Williams, 2019). Future research on network resilience under multiple simultaneous attacks is needed (Marsa-Maestre et al., 2019).

Gap	More research is needed on
Scaling	Massive scale of networks require scaling (Ullah et al., 2023). Service orhestration (provisioning and configuration management) should be highly reliable and scalable (Rong et al., 2022). The complexity of infrastructure, provisioning and configuration management tools is emphasized when production infrastructure is scaled out geographically and in load (Rong et al., 2022). It should be possible to scale and move processing geographically, across organization and juristictions and trough the continuum from edge to cloud (Rong et al., 2022). Scalability and reliability could be enhanced by having data in multiple clouds (Mansouri, Prokhorenko and Babar, 2020).
Defining configuration	"IaC patterns and anti-patterns/smells" need further research (Kumara et al., 2021). "[Several] best practices exist, but they mostly concern the complexities inherent within IaC" (Kumara et al., 2021). DSL phases of domain analysis is an area for future research (Kosar, Bohra and Mernik, 2016). Configuration should aim for higher level of abstraction (Ullah et al., 2023). Languages should find a balance between declarative and imperative expressions (Rong et al., 2022). The whole system should be optimized instead of just individual nodes (Balis et al., 2018).
Empirical Validation	More solutions should be empirically validated (Rajapakse et al., 2022). DSL research included lack of evaluation research (Kosar, Bohra and Mernik, 2016). There is a clear "lack of evaluation research, in particular controlled experiments" (Kosar, Bohra and Mernik, 2016).

Gap	More research is needed on
Integration	DSL integration with other software engineering practices Kosar, Bohra and Mernik (2016). Challenge of integrating different systems on multiple levels, adaptability to application requirements and supporting open and decentralized services (Rong et al., 2022).

2.1.5 Network Architectures of Configuration Management Tools

Configuration management systems follow master-slave architecture. The master computer will issue configurations, and the slave computers apply these configurations.

Even though master-slave is the correct term for this architecture, some other word is sometimes used in place of the slave to avoid confusing non-technical audiences. Such alternative terms include agent and minion. This thesis uses the words agent and slave interchangeably when referring to a slave node in a configuration management system.

Internet communication uses client-server architecture. The server is constantly waiting in a known address. At the time of it's choosing, client initiates a connection.

As the client can keep the connection open permanently, additional layers can be built on top the connection opened. In master-slave architecture, master can be placed either on client or server, leading to push and pull architectures.

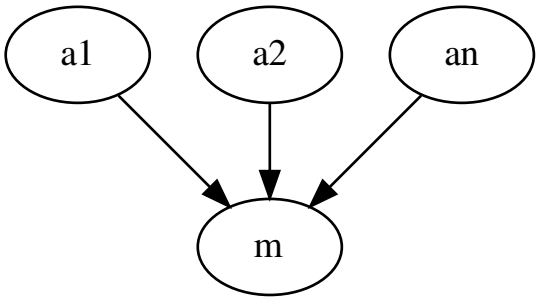

Figure 1: Pull architecture

2.1.5.1 Pull architecture In the most common configuration, master is the server, and slave nodes connect to this server. This pull architecture is shown in fig. 1. In the client-server model on the Internet protocol suite (TCP/IP), the server opens a listening port and waits indefinitely. The client connects to the server at a suitable moment.

Making the master the server in typical configuration management systems solves many practical problems. As typical workstations cannot publish a server, the master cannot connect to them as a client. Also many slaves, such as laptops and mobile devices, are not constantly powered on, so they cannot always be reached when the master attempts to contact them.

However, having the master as a server creates new risks and problems. The visibility of the master on the public Internet makes it vulnerable to most common attack methods. At the same time, the master server is a very valuable target on the network. This is due to combining two tasks, which I will later claim to be unrelated: having the signing keys for slave catalogs; and distributing those catalogs.

The Internet and modern local area networks are based on TCP/IP. Because the IPv4 address space of 254**4 addresses is running out, many networks use network address translation (NAT) to allow many workstations to browse the Internet from the same public IP address. NAT makes it impractical to have servers on the workstations behind NAT.

2.1.5.2 Push architecture In push architecture, slaves run servers and the master connects to these servers. Push architecture requires slaves to be able to publish a server that is visible on the internet, which is not possible in many NAT:ted, firewalled and mobile networks. This makes push architecture more suitable for managing servers than laptops. Push architecture is shown in fig. 2.

The slave server does not need to be an agent specific to the configuration management tool, but instead a general purpose remote control system such as SSH can be used. This reduces the amount of software that needs to be updated and avoids agent-master versioning problems. Agent and master software versions are often linked to each other, but different OS and distributions update their packaged software at a different pace.

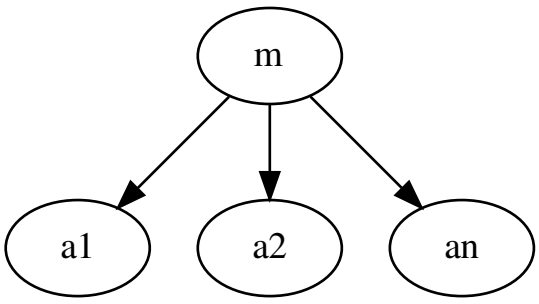

Figure 2: Push architecture

2.1.5.3 Implying Direct Master-Slave Connection

Many articles on configuration management systems assume that slaves must at some point directly contact the master. This assumption can be either explicit or, more often, implicit. As we will later see when examining successful botnets, such contact is not required.

In their comparison of open source CMS, Delaet, Joosen and Van Brabant (2010) considers only two possible network architectures: push and pull. They explicitly state: "In all approaches, each managed device contains a deployment agent that can be push or pull based".

Vanbrabant (2014) categorizes "deployment architectures" of configuration management tools to pull or push, implying a direct network connection between master and slave computers. This is further emphasized by examples. Poat, Lauret and Betts (2015) have selected popular configuration management tools (Chef, Puppet, CFEngine) for comparison, all of which require direct connection between master and slaves to configure multiple computers. In their paper on orchestration ("model driven Cloud-management") Wettinger et al. (2013) imply the direct connection between master and server in their choice of tools and in the options they use for the orchestration system to deploy the catalogs. Even though there are multiple peer-reviewed works on applying the configuration, less interest is paid to the secure transport of these configuration instructions. Święcicki (2016) describes a novel tool "Overlord", but bypasses the transfer by stating that the "program then could be transported to the target machine".

2.1.5.4 Proposed Improvements to Push and Pull Architectures

The secret key of the master allows full control of all slave nodes. This makes master's secret key the most valuable file in the network, as it grants access to all other information in all nodes. When considering improvements to the security of master-slave network architecture, it should take into account how well this architecture protects master's secret keys.

Virtual private networks (VPN) have been proposed and used in research prototypes for ensuring the security of master-slave network communication Mansouri, Prokhorenko and Babar (2020). A VPN establishes another, encryption protected network over an untrusted network such as the Internet. This is often shown as a virtual network interface inside the nodes participating the network.

As long as all nodes remain intact, a VPN can protect their network communications. Adversaries often can't immediately compromise high value targets such as Active Directory domain controller or configuration management master

node. They resort to compromising less protected nodes on the network and laterally moving towards more valuable nodes.

When master and slave node are connected with a VPN, an attacker controlling a slave can immediately proceed attacking the master. A constant VPN connection means that these attacks can be performed when timely expert human response is unlikely, such as at night or during holidays. Thus, even though a VPN protects can protect against network eavesdropping or man-in-the-middle attack, it does little to protect master against compromized slave nodes. Protection against man-in-the-middle attack is dependent on key management and the bootstrapping process used for initially establishing trust.

As Internet protocols, including VPN solutions, use the client-server architecture, a VPN based solution can lead to master being visible to the Internet. If the server is on the slave nodes, configuration management can't work with moving nodes such as laptops, as being a server requires opening a listening port in a known Internet address. That's why the VPN server can be on the master, requiring this valuable machine to be visible on the network. In their solution using WireGuard VPN, Stocker et al. (2022) note that "only the coordinating host be visible externally".

The use of secure shell (SSH) reverse tunnels has been proposed as an alternative to VPNs (Caballer et al., 2023). The challenges are similar to those of VPN, but it's not clear how a solution based on SSH reverse tunnels could scale or be reliable. The possibility of using pseudonymous peer-to-peer networks - such as i2p, Tor and Freenet - is considered in "Designing Hidden Master Architecture" section "Protecting the Master's Private Key in the Hidden Master Architecture".

Blockchains and smart contracts have been proposed for use in configuration management systems Rong et al. (2022). The power of single administrator could be limited by requiring multiple persons to authorize changes (Kinkelin et al., 2019). The solution by Kinkelin et al. (2018) uses Hyperledger Fabric, a private blockchain. However, multiple authorization could be implemented by requiring multiple signatures with keys from a pre-approved set, and Kinkelin et al. (2019) themselves note that the requirement of multiple approvals is already used without blockchains. Instead of smart contracts, the approval logic could be distributed to slave nodes. After all, it's the software running as root on slave nodes that must actually run the configuration, either by validating the signatures - or by reading blockchain and accepting the result of the smart contract. The distribution of large configuration catalogs must also be solved, as blockchains can usually only store limited amounts of data. The confidentiality of the catalogs and related key management must also be

solved, as the concept of blockchain concentrates on integrity of the data in the chain. For blockchains to be useful in configuration management, it would be interesting to see them compared to simple commonplace solutions providing the same features.

2.1.6 Configuration Management System as a Target

A successful attack on configuration management system results in the full compromise of every controlled computer. This makes it a very tempting target for attack.

In many attacks, the attacker must gradually increase his foothold on the target system. For example, an attacker might take over a poorly protected, old, low value computer to bypass a firewall on the edge of an enterprise network. This foothold is then used for launching an attack from the internal network. When regular user accounts are compromised, the same credentials are used for lateral movement. A privilege escalation is needed to get administrative rights on systems. This way, the attacker must own the network step by step.

In contrast, a compromise of a configuration management system means a single step can compromise every computer and system in the network. This can make CMS the most critical system. After a compromise, nothing else is required of the attacker. The attacker can disable or misconfigure the intrusion detection, antivirus systems and audit logging. All files can be read and altered, and the fact that these changes happened can be hidden. Cameras and microphones can be used for eavesdropping on communications. To extend the attack to external systems, user passwords can be collected with key logging.

If the attacker succeeds in compromising CMS, he can then install hostile remote administration tools (e.g. meterpreter) or simply use the existing CMS to carry out criminal activities. Using the existing CMS for attack could make the attack more reliable (less likely to crash target systems) and harder to detect by external network intrusion detection systems (IDS) (Hastings and Kazanciyan, 2016).

Because CMS is such a critical part of system management and security, it should be able to withstand internal and external faults, challenges caused both by environment and human malice. Existing off the shelf CMS have insufficient fault tolerance features (Duplyakin, Haney and Tufo, 2015).

2.1.6.1 On Attacks Against Configuration Management In their article on analyzing attacks against large networks, Hariri et al. (2003, pp 1) emphasize two common types of attacks: DOS attacks against servers and

routers. It should be noted that the architecture proposed in this paper can reduce the effect of both types of attack.

In practice, the master has full permissions on all slave computers, and slaves blindly follow instructions from the master. In Linux, BSD and Unix, the master has the equal status of root or sudo on slaves. In Windows, master has the equal status of Administrator or SYSTEM access. This makes the master computer a very valuable target for attacks. Compromise of the master results in the almost immediate compromise of all computers on the target network. As an added benefit, there is already a working configuration management system in place, so the attacker does not need to deploy his own remote control tools to all nodes.

In both legal penetration testing and hostile network attacks, attacks against servers are a common approach. When starting the active reconnaissance phase, the attacker will perform a network ping sweep, DNS zone transfer or a DNS scan, enumerate services using tools such as Metasploit db_nmap and perform vulnerability scanning with tools such as Nessus or OpenVAS. Popular books on penetration testing devote most of their pages to attacks against servers. By definition, public servers are visible to the public, and thus create an easily reachable attack surface.

Many CM systems provide their own transport system for catalogs. Compared to some popular public facing servers, such as OpenSSH and Apache web server, these CM master servers are relatively young. CM master servers are also developed with limited resources (compared to the popular public facing servers), as they are developed by the same teams that create the CM systems.

If the server is only visible in the company local area network, it cannot configure laptops and mobile devices being used outside that LAN. For example, internal security updates and mitigations could not be provided in a timely manner. Even if the access to the master server was through a virtual private network (VPN), all clients must have automated access to the master. This means that the compromise of any client allows direct connection to the master server. If multiple masters are created for different security levels, there is no longer a single source of truth for the network, leading to complicated configuration and the slowing down of the pace of updates.

Web servers are commonly used with a vast number of users, and they have a long track record of surviving these loads. For example, the Apache web server has existed since 1995 and has been the most popular web server since 1996. No configuration management system known to the writer has been tested anywhere near this scale.

In this work, I will research an approach that completely removes the attack surface of the public facing server and thus all attacks upon it. I will argue that using a regular web server as one of the platforms to deploy the catalogs to slaves, CM systems can capitalize on their well known scalability.

This approach could completely prevent attacks against public servers (as there would be none), leaving only client side attacks. Many approaches to client side attack have a social engineering component in them, allowing the operator to greatly reduce the attack surface by limiting his/her use of the master computer.

2.1.7 Leading Configuration Management Tools

Many well known organizations behind large computer systems have publicly named the CM tools they are using, or published the fact that they are using modern, infrastructure as code (IaC) configuration management tools and techniques.

Rahman, Mahdavi-Hezaveh and Williams (2019) list Ambit Energy, GitHub, Mozilla and Netflix as examples of users of IaC based techniques to manage their systems. Puppet configuration management tool users include the US Government, Wikimedia, CERN and Google. The US Government uses Puppet for security baseline for Windows and Linux (NIST, 2016). Wikimedia configures its servers with Puppet. CERN configures user managed images with Puppet. Google uses Puppet for managing laptops and desktops.

Business social media site LinkedIn configures its 40 000 servers with CFEngine (CFEngine, 2016). CFEngine is one the earlier modern configuration management tools. Mark Burgess, the author of CFEngine, has written many influential papers on configuration management, and consequently, CFEngine was the first tool to implement some of these ideas in production.

Social media giant Facebook uses well known open source tool Chef to manage OS settings and software deployment. To configure the applications developed by the company, a homegrown solution is used. (Tang et al., 2015)

In the context of this work, the interest is on modern, free software configuration management systems. To consider software modern, it should allow idempotent configuration. To allow versioning, auditability and efficient group work, the configurations should be described as text, following the infrastructure as code approach. For ease of integrating the hidden master architecture and to be easily analyzed, the software should be Free software as defined by Free Software Foundation (2015). Tools should be actively developed, e.g. have an active bug list, fresh commits from multiple users, packages in some distributions default

repositories and not have very old unfixed serious security vulnerabilities. To see if a tool is actually used in the field, we can consider such proxy indicators as use cases of large scale deployments, news articles and activity in popular forums.

The amount of practical, well known and production ready tools is limited. For example, Delaet, Joosen and Van Brabant (2010), name 11 tools for comparison, six of which are open source and thus in the focus of this thesis: BCFG2, Cfengine3, Chef, Netomata, Puppet and LCFG. Because this work is from 2010, it does not include some well known tools that are published later than that. For example, the initial release of Salt configuration management system is from 2011, and it usually takes some time after release before a tool is considered production ready. Ansible was published even later in 2012.

Popular, modern, free CMS tools - inclusion criteria

- Modern (idempotent, infrastructure as code)
- Free (FSF definition)
- Actively developed (fresh commits, multiple developers, packaged, no old unfixed vulnerabilities, stable release in the last two years)
- Popular (news, forums)
- Large deployments

To avoid missing the latest tools, numerous candidate tools were collected from various academic and non-academic sources. This list was then filtered using the inclusion criteria. Collecting the candidates was done using articles referenced by this thesis. For example, Delaet, Joosen and Van Brabant (2010) lists 11 tools, six of which are free software; Fox et al. (2015) lists 21 tools; Wikipedia article “Comparison of open-source configuration management software” lists 23 tools.

To filter the results, dead projects were excluded. This step was based on information from their homepages, and in some cases installation packages, Twitter channels, GitHub and SourceForge pages. Dead projects were those which had no stable releases across two years (2015-2017) or their homepages were down or not updated for a long period of time. Excluded dead projects were SmartFrog, ISconf, Netomata, PIKT, Radmind and STAF. One project, LCFG, did not support any of the commonly used operating systems and was excluded. LCFG does not have production quality (in vendors’ opinions) agents for Linux, Windows or Macintosh OS X. Even though Fox et al. (2015) mentions many DevOps tools, many of them are not free software (e.g. AWS OpsWorks, Cisco Intelligent Automation for Cloud among others) or work on other parts of DevOps than general purpose configuration management

(e.g. including Cobbler, Kubernetes, Docker CloudMesh).

Some software was mentioned in peer-reviewed articles (as referenced earlier in this article) and was included: Ansible, Puppet, Salt, Chef, BCFG2 and CFEngine. For example, Fox et al. (2015) calls Puppet, Chef, Ansible and SaltStack the "leading configuration managers" - a claim that was found to be supported by both academic references and regular search volumes. BCFG2 is mentioned in multiple peer-reviewed articles published in the last two years (2015-2017). CFEngine has by far the best coverage in the academic world, with about a hundred articles published in the last two years.

Compared to Delaet, Joosen and Van Brabant (2010), two tools are not interesting to this project: Netomata is now a dead project and LCFG was excluded for its lack of OS support. Two new popular tools have emerged after Delaet et al.'s paper: Salt and Ansible.

This has left some lesser known tools: Rex, Rudder, Juju, cdist and Capistrano, OCS Inventory NG with GLPI, synctool, NOC, Spacewalk and Quattor. These tools were not mentioned in peer reviewed articles used for collecting candidate applications.

The Linux distributions package is popular and well-established software so that it can be installed easily. Being packaged in the main repository of some popular distribution is then a positive signal for a program. The most popular server, desktop and laptop distribution families are Debian (apt, dpkg, deb) and Red Hat (yum, rpm).

To evaluate the packaging status of the best known tools, their packaging versions were sampled for one distribution from each family: Ubuntu 16.04 (Debian family) and CentOS 7 (Red Hat family) (CentOS project, 2016). The tools evaluated were Ansible, Puppet, Salt, Chef, BCFG2 and CFEngine. Ubuntu 16.04, the current long term support release, packages all the main tools. Puppet was packaged in the "main" component, so it will receive more attention from the distribution core maintainers. Others were packaged in "universe", a component for third party maintainers. CentOS 7 did not package any of these tools.

Metrics can be used to estimate interest in a tool or a topic, even without analyzing each mention for tone and content. In the context of configuration management, a similar calculation of metrics has been used by Rahman, Mahdavi-Hezaveh and Williams (2019) as a small part of their systematic mapping study of IaC research. Hintsch, Görling and Turowski (2016) list counts of papers naming a tool and those focusing on it before diving into the articles, the content of which they analyze. Metrics cannot replace a regular

literature review, and thus they are used here only to support a more detailed analysis of the literature.

To estimate academic interest to main CMS, Google Scholar search was performed with the tool name and the string "configuration management", e.g. ' "puppet" "configuration management" '. The search was limited by excluding patents and citations and showing only articles from the start of 2015 to November 2016. This is similar to the search performed in the literature review for CMS articles published before February 2015 by Hintsch, Görling and Turowski (2016), but targets fresher articles.

Table 9: References to configuration management tools in new peer reviewed texts

CMS	Hits
Puppet	381
Chef	362
Fabric	224
Ansible	173
Salt	167
CFEngine	77
Juju	45
Rex	40
Rudder	29
Capistrano	16
BCFG2	9
Quattor	5
Spacewalk	4
OCS Inventory	4
Cdist	1
synctool	0
NOC	na

As shown in table 9, Puppet and Chef got most academic hits, with Ansible and Salt the second most. CFEngine, the oldest of these systems, got slightly less new hits. BCFG2 got very few hits, only nine, and this might make it less interesting for further study. Compared to the literature review by Hintsch, Görling and Turowski (2016), I found more mentions of tools and some tools that were missed by Hintsch et al. The list of most mentioned tools is in table 10. The larger number of mentions could only be partially explained by filtering done by Hintsch et al, as they only discarded 167 publications and included 159.

More tools were probably found because I used a broader method for collecting candidate tool names and Hintsch et al. used a list of tools from 2010.

Table 10: CM tools with at least 10 references according to literature review by Hintsch, Görling and Turowski (2016)

CMS	Named	In focus
CFEngine	89	15
Puppet	82	11
LCFG	50	7
Chef	46	9
BCFG2	17	3

For lesser known CMS, Rudder hits include many unrelated hits using the word "rudder" in its everyday meaning. Cdist did not match any academic articles, the only hit being from a non-academic book. NOC is a common abbreviation for network operations center, which makes it more difficult when undertaking a search; of the 72 results, the first page did not contain any references to NOC, the configuration management system. Spacewalk had 4 hits, two of which were on subject. Even though some of Fabric's 224 hits were false positives, adding "python" as a keyword still brings 57 hits. However, the commands Fabric uses are not idempotent, and thus Fabric does not qualify as a modern configuration management system.

Google generic search data, the searches from Google.com front page, was queried using the Google Trends interface for the CMS systems with most academic hits. The search was the tool and the string "configuration management" without quotes, e.g. "puppet configuration management". Fabric, CFEngine, Juju, Rudder, Rex and BCFG2 had zero average searches per day. The search volumes of CM tools are in table 11.

Table 11: Search volumes of configuration management systems

CMS	Searches per day
Puppet	28
Chef	23
Ansible	13
Salt	12

Many articles compare these four configuration management systems. Kostromin (2020) names Puppet, Chef, Ansible and SaltStack (Salt) as the most popular software configuration management tools. A Google Scholar search for articles mentioning all four brings 75 hits in the two-year period from 2015 to 2017. A regular Google search with all these words brings 125 000 hits, which is quite a number considering the level of technical skill required to work with these tools.

As all proxy metrics used give the same kind of result, it can be concluded that the leading configuration management tools are Puppet, Chef, Ansible and Salt.

2.1.7.1 Applying and Transferring Catalogs Slaves can receive their catalogs in multiple ways. Slaves can act as clients and pull the catalogs from the master server. The master can contact slaves as a client and push the catalogs, with either SSH daemon on the slaves or with a custom slave server on each slave. Or the catalogs (or their source code) can be transferred by a method unrelated to the CMS, then applied locally on the slave.

In any network containing moving slaves such as laptops, pull architecture is an obvious choice. Most networks use NAT or firewall that prevents slaves from publishing a server on the Internet, making pull the only feasible architecture.

If the requirement of NAT bypass is dropped, architecture can be simplified by using an existing tool, such as SSH, to push configuration.

Table 12: Network architectures in leading CMS

CMS	Pull	Local	Push
Puppet	yes	yes	difficult
Chef	yes	yes	possible?
Salt	yes	yes	yes
Ansible	possible	yes	yes

All four popular tools support all of pull, push and local architectures, but strongly recommend either pull or push. The supported network architectures are listed in table 12.

Puppet, Salt and Chef are originally meant to be used with pull architecture, requiring extra components and setup for pull operation: mcollective for Puppet, Chef push job for Chef and Salt-SSH for Salt.

Ansible is meant for push over ssh, but can use ansible-pull to retrieve configuration from a Git version control repository. By using ssh client and a server

for transfer, Ansible does not need its own agent. Ansible marketing material calls this "agentless" operation, but obviously some software is needed on the slave (agent) side to receive and apply configuration, namely, the SSH daemon.

2.1.7.2 Protection of Transfer Ansible uses SSH (usually OpenSSH) and public keys. If normal SSH security routines are followed, this is very secure. As slaves cannot be reached when they are not in their home network, they could go a long time without updates, including critical security mitigations.

Puppet uses https transport without any external certificate authorities. TLS used with web servers is quite complicated which makes it less transparent for administrators.

Salt uses ZeroMQ, which is much less used protocol than ssh or https. It could be assumed that ZeroMQ and its implementation in Salt has received less scrutiny than more established https and SSH. On the other hand, there might be less interest and attack tools for a smaller target.

The principles of information security are the CIA's triad of confidentiality, integrity and availability. All of the default implementations of these popular tools aim to provide confidentiality and integrity by encrypting traffic and performing two-way authentication. However, they do little for availability, as the master server is a single point of failure in the network. Even if the master server is coupled, the requirement to trust the master server means that these machines must be physically protected, and this greatly reduces the possibilities to distribute them to different networks and geographical locations.

2.1.7.3 Protection Against Compromised Slave Agent systems should not know everything about other systems. Models of attacker tactics show that attackers have to spend considerable time performing active network reconnaissance to discover the configuration of hosts Mitre (2019), and do more active reconnaissance when performing lateral movement (Pols, 2017).

If all agents know all the secrets of the network then all the different security domains require separate systems. But if systems are separated, there is no longer a single source of truth. In addition to practical hurdles, this means that orchestration (making different computers act together) becomes tedious.

Puppet compiles instructions for the slave as a slave specific catalog only containing instructions for that specific slave, with the minor exception being non-template files. Puppet advertises and clearly documents this feature. For the purpose of this thesis, this was briefly tested by extracting a compiled, slave specific catalog. It did not seem to contain information other than that which was made available to the target slave.

Salt emphases “no freeloaders” in its documentation and advertising. In practice, this means Salt attempts to improve scalability by delegating as much work to slaves as possible. By default, all configuration (contents of /srv/salt/) is available to all slaves. Only specific secrets such as passwords and secret keys are stored separately using the “pillars” mechanism (/srv/pillar/). Pillars are only available to named slave computers. For this thesis, it was briefly tested if configuration could be compartmentalized by placing all configuration into a pillar. In a very brief test, only the target slave could receive this state. Other slaves only received a default dummy value. Using pillars in this way is undocumented and thus its usability in production is questionable.

The capability of different systems to limit agent visibility is in table 13.

Table 13: Protection against compromised agent in leading configuration management systems

CMS	Limit Agent Visibility
Puppet	yes
Chef	not tested
Salt	undocumented for normal configuration
Ansible	not tested

2.2 Malware

2.2.1 Malware Command and Control Networks

Command and control channels have seen significant improvements over time. I will first briefly overview some key concepts, then look at malware evolution with emphasis on command and control.

Botnet is a network of compromised machines under the control of an attacker (Binsalleeh et al. 2010). The largest botnets have had millions of slave computers in hostile, heterogeneous environments. As these botnets are extensively documented, they can provide insight into the possibilities of novel network architectures for CM tools.

Technically, any tool could be used for any purpose. For example, a legal penetration tester could use botnet non-maliciously to test an intrusion detection system. And this work will use botnet software to develop benign configuration management tools. However, as a working definition I will categorize botnet tools as malicious according to their original, intended purpose.

Malicious botnets have both similarities and differences to benign configuration management tools. Both aim to provide scalable, resilient and timely updates

to slave configuration. But only botnets require secrecy at each point: slave, network and possible CC server. Botnets might also require the capability of a reduced forensic footprint. Botnets are sometimes used for extracting data from victim systems, and this puts additional strain on secrecy requirements.

Additional challenges to botnets are created by varying operating systems on victim computers, changing network conditions, virus scanners and intrusion detection systems and legal attacks against their command and control channels. Once the CC analysis has been published, interested parties can create new IDS rules and antivirus detection routines (Binsalleeh et al., 2010).

2.2.2 Evolution of Malware Command and Control

Malware employs wide variety of command and control channels. Many developments in malware command and control predate development of configuration management systems.

The first malware command and control (CC) channel was introduced by Shoch and Hupp in their worm titled "worm". "Worm" also contained a kill switch, a feature allowing its controllers to disable and delete all running versions of the worm. (Chen and Robert, 2004) The existence of a kill switch has later become an important feature of malware to prevent collateral damage, prevent capture of malware samples by the defenders and for avoiding unwanted attention related to excessive propagation.

Some early botnets used Internet relay chat (IRC) as a control channel. Eggdrop was one of the first to use IRC based command and control in 1993 (Barford and Yegneswaran, 2007). Binsalleeh mentions Agabot, SDBot and SpyBot as examples of malware using IRC CC. Network operators can attempt to automatically detect botnet activity, making uncommon protocols hard to hide. This has made HTTP a tempting choice for bots such as BlackEnergy, Rustock and clickbot (Binsalleeh et al., 2010).

Gu, Zhang and Lee (2008) consider IRC a push architecture, because commands are immediately sent to slaves as the botmaster sends them. It should be noted that in the client-server architecture, the slaves (bots) are still clients when they connect to the IRC server. This way, they can access the master through NAT and firewall. Gu, like the author, considers HTTP a pull architecture, as slaves periodically download a set of instructions from the server.

For a truly distributed operation, some bots have adopted peer-to-peer (P2P) architecture. In P2P, slaves connect to each other without a central server. In the TCP/IP sense, each node can act as either server or client, as dictated by network conditions. Dittrich and Dietrich (2008) name Peacomm and Nugache

as examples of P2P botnets. In addition to the lack of server as a single point of failure, he mentions the small network footprint and unpredictable traffic patterns as additional benefits.

Naz was the first bot using social network as a CC channel. It uses steganography to hide control messages on Twitter, pretending to be a human user. Other social networks and third party services could be used as a CC channel by new botnets. (Kartaltepe et al., 2010)

CryptoLocker and similar software encrypt user files and extort for money. The ransom must be paid in Bitcoin to receive the key to decrypt the files. Modern encryption extortion can even work without a traditional CC network, as the password is passed across for the human victim to type in after the criminals have received payment.

The arms race between criminal botnet operators and their human adversaries is driving botnet architectures forward. The heterogeneity of victim computers and target networks puts additional demands on botnet CC development. The sheer size of most successful botnets (including millions of slaves) dwarfs the most benign configuration management systems.

Compared to benign configuration management, many developments in malware CC have happened much earlier. The first known worm existed before the Internet, when the Creeper system spread through Arpanet in 1971, the predecessor of the Internet (Chen and Robert, 2004). The first configuration management systems were in production in the 1980s, almost a decade later.

Fast development of malware CC can also be seen in modularized toolkit approach. Modularization was introduced to malware in the second wave during 1990's (Chen and Robert, 2004). Toolkits, such as Virus Creation Lab, allowed individuals with low technical skills to create tailored malware (Chen and Robert, 2004). Toolkit based offensive approaches are sold and used both in the criminal underground (Alkhateeb, 2016), law enforcement and intelligence operatives (Lemay et al., 2018) and penetration testers.

Modularization is a continuing theme in malware. It allows the separation of concerns, reduces footprint and loss of tools on capture. The cycle from new vulnerability to abuse and patching is becoming shorter. Modularisation helps to adapt to this cycle, as payload (including command and control channel) can be combined with new infection vector as soon as old ones are patched against.

As many advanced botnets have been reverse engineered and described in publicly available articles, it will be interesting to try applying this information to normal enterprise configuration management tools.

2.3 Conceptualizing Attacks

2.3.1 Survivability

Survivability is the capability of a system to fulfill its mission - in short, its business purpose - in the face of challenges. These challenges or faults include attacks by human adversaries, system failures, accidents and failures of large parts of the network infrastructure (Mead et al., 2000, p4; Fung et al., 2005, p1; Sterbenz et al., 2010, p1247–1248). These challenges or faults are often meant to cover all possible damaging events to the system (Mead et al., 2000, p4).

The business purpose of the system is essential when evaluating survivability. Even if individual components stay functional, the system has failed if it fails to provide the service it was designed for (Ellison et al., 1999, p55). Sometimes the concept of survivability can be used in a narrower sense. For example, according to Sterbenz et al. (2010, p1247) survivability can be classed under the umbrella terms robustness and challenge tolerance. In this work, I use the term survivability according to its wider definition of surviving any external or internal faults to fulfill the mission of the system.

The identification of essential services is the key concept of survivability (Ellison et al., 1999, p58). According to Ellison et al. (1999), essential systems are those that either are required for meeting the mission requirements or those whose failures threaten the system. A configuration management system obviously meets these criteria: the compromise of the configuration management system results in the full compromise of all controlled systems. Also, without the configuration management system it becomes difficult or impossible to react to a changing environment and threats in a timely manner.

A configuration management system could be further divided into essential and non-essential parts. To fulfill its mission, a configuration management system must be able to define configuration, transfer it to agents and apply configuration there. It could be argued that some features are useful but non-essential. Help screens, testing and debugging capabilities might be considered such non-essential functionality.

Survivability analysis is the process of identifying components susceptible to attacks, then quantifying the capability for surviving these attacks (Fung et al., 2005, p1). One method of identifying suitable targets for attack is the attack tree method.

2.3.2 Attack Tree

The attack tree is a method of systematically categorizing all methods by which the system can be attacked (Schneier, 1999; Mauw and Oostdijk, 2005). Attack trees were originally introduced in an article by Schneier (1999; Mauw and Oostdijk, 2005).

Attack tree takes the view of an attacker, and starts with the goal of compromising the target system. This is the root node of the tree. This node is then divided into subnodes by splitting the problem area. Each subnode is further split until all attacks have been enumerated. Each branch can consist of parts of an attack, all parts of which must be achieved (AND), or consist of alternative avenues of attacks (OR). (Moore, Ellison and Linger, 2001, p4–6)

For example, consider an attacker with the goal of penetrating a web server. The attacker could attempt to attack the server directly or perform a client side attack by targeting the administrator's workstation. A direct server side assault could attack the logic web application or the underlying software stack. To compromise the web application, one of the OWASP 10 (Phil et al., 2014) attacks could be used. Splitting these attacks into finer detail, all avenues of attack against the server can be listed.

Even though attack tree is a useful tool for enumerating possible avenues of attack, it does have some limitations. Attack trees do not necessarily directly lead to an enumerated set of hardening options (Dewri et al., 2012, p1). Dewri et al. (2012, pp 2) suggests adding security controls to protect against these attacks, but points out that this does not usually protect against unknown vulnerabilities. According to Hariri et al. (2003), using attack trees can miss or mark impossible some vulnerabilities that are yet to be discovered. However, even though a practical implementation of a vulnerability has yet to be found, the user of an attack tree could still identify a component as a target of attack or even the general category of attack.

In this work the preference is to re-evaluate and improve configuration management system network architecture to reduce the attack surface, thus removing a whole category of attacks instead of introducing a new layer of security controls. In case this is not possible, new security controls are a second choice.

A configuration management system is a highly valuable target. Compromising the master would grant full, unlimited control of all agents. In organizations that have fully adopted configuration management systems, this would mean full control of all computers and all the data held by them. In addition to providing this access, a configuration management tool by its nature provides an efficient tool to perform a wide range of activities on those systems without

installing additional tools.

2.3.3 Stage Models for Attacking Computer Systems

2.3.3.1 Cyber Kill Chain Cyber kill chain lists the main phases of a cyber attack, from reconnaissance to reaching the objectives. The phases are shown in table 14. As the attack is a chain, disrupting any phase will stop the attack. Because even advanced attackers must work economically, they are pushed to re-using parts of previous attacks, creating indicators defenders can use. By taking a threat focused approach and intercepting attackers at the earliest possible state of the cyber kill chain, defenders can regain advantage in this adversarial game. (Hutchins, Cloppert and Amin, 2011)

An original paper by Hutchins, Cloppert and Amin (2011) calls this phase based model "intrusion kill chain", but their employer, Lockheed Martin, later trademarked the same model as cyber kill chain. For each step, Hutchins et al. also take the defender view, pointing out steps to detect, deny, disrupt, degrade, deceive and destroy intrusion attempts. They heavily critique what they saw as conventional wisdom at the time: "The conventional incident response process initiates after our exploit phase, illustrating the self-fulfilling prophecy that defenders are inherently disadvantaged and inevitably too late."

Table 14: Cyber Kill Chain (Hutchins, Cloppert and Amin, 2011)

Phase	Explanation
Reconnaissance	Target selection, passive reconnaissance, active reconnaissance
Weaponization	Combine exploit with backdoor (payload) to create weapon
Delivery	Send the weapon (exploit+backdoor) to target
Exploitation	Exploit a vulnerability
Installation	Install backdoor
C2	Typically compromised host connects to controller over the Internet, "hands on keyboard" access to attackers
Actions on Objectives	The raison d'etre for the whole operation: violate confidentiality, integrity or availability of breached system; or move laterally

2.3.3.2 Critique and Expansion of Cyber Kill Chain

In my view, Hutchins' (Hutchins, Cloppert and Amin (2011)) cyber kill chain seems to be heavily influenced by spear phishing as an attack vector. In spear phishing, attackers send carefully written malware emails to small group of targets. All of the three examples in the paper use spear phishing, and journal editor also introduces the article as being about "malicious emails". Even though the model is useful for any common computer intrusion, some common attacks do not quite fit into the steps.

Consider an attack against server software using Metasploit, a popular cyber weaponization and penetration testing tool. Typically, an attacker selects an attack ("use exploit/windows/smb/psexec"), target host ("set RHOST 10.0.0.1"), and a backdoor ("set payload windows/shell/reverse_tcp"). This concludes the "weaponization" phase. When the attacker commands ("exploit"), Metasploit connects to the target, exploits and installs the backdoor - all in a single step. Thus, a single command would contain phases the "Delivery", "Exploitation" and "Installation".

The active, intelligence driven security model is suitable for organizations large enough to justify the effort, but its usefulness for individuals and small organizations seems limited by the amount of work it requires. To collect and compare indicators and catch attackers in an early phase, continuous monitoring and analysis effort is required. As the threat model, APT, is described as a manual attack by a proficient and well funded adversary, a small organization might avoid being targeted when attackers focus on more valuable victims.

2.3.4 MITRE ATT&CK

MITRE ATT&CK (MA) is a post-exploit model for cyber attacker tactics, techniques and procedures (TTP). MA defines 11 tactics as the interim goals attackers must reach, such as persistence or lateral movement. The tactics are listed in table 15. A large number of techniques represent how attackers reach these goals, e.g. creating a scheduled task or using Windows Administrative shares. Use of these techniques are linked to groups and the malware they use. Strom et al. (2017) Widely used techniques are listed under each tactic to create an ATT&CK Matrix (Mitre, 2019).

MA is based on the real life use of these techniques, and multiple examples of real life attack campaigns are listed for each technique (Mitre, 2019). The content of MA keeps changing based on public threat intelligence. After the initial release, there have been modifications to all major aspects of MA, including tactics, techniques, platforms and technology domains. MA does not attempt to enumerate all attack vectors, but MITRE recommends their

CAPEC and CWE for that. (Strom et al., 2018) The 2019 version of the matrix starts from the exploit phase ("Initial access"), and the earlier phases have been published as MITRE PRE-ATT&CK.

Table 15: ATT&CK tactics and examples (Strom et al., 2017)

Tactic	Examples
Initial Access	Drive by compromise, spear phishing
Execution	Malicious files (pdf, doc vulnerabilities), trojan
Persistence	Scheduled task, browser extensions, startup items
Privilege escalation	File systems permissions, UAC, setuid and sudo bypasses
Defense Evasion	Disable antivirus and firewall, deceptive filenames
Credential Access	Keyboard logging, brute forcing
Discovery	Network sniffing, enumerating local accounts
Lateral movement	Windows Administrative shares, pass the hash
Collection	Video capture, collect documents from file system
Command and Control	Custom encryption inside https, domain generation algorithm
Exfiltration	Encrypt exfiltrated files, schedule to business hours
Impact	Overwrite hard drive, DOS attack over network

Even though ATT&CK and its updated versions have been published in non-peer reviewed technical reports and white papers by MITRE, a non-profit corporation, it has been used as the basis of peer-reviewed conference papers. Maymi et al. (2017) developed a semantic adversary model for machine learning, but found it easy to map their operational graph to ATT&CK. Al-Shaer, Ahmed and Al-Shaer (2019) performed statistical analysis using ATT&CK database and taxonomy. Farooq and Otaibi (2018) propose using machine learning to reduce false positives. They have chosen MA and Common Attack Pattern Enumeration and Classification (CAPEC) as their main taxonomies. Farooq and Otaibi (2018) see cyber kill chain as too generic for actual attack classification, a view shared by the authors of ATT&CK. The industry has exclusive access to most primary sources of information such as incident response data and large databases of historic samples, and they often have capabilities not available to even academic research groups (Lemay et al., 2018). In his survey of advanced persistent threats, Lemay states that "there is no alternative to using industry sources", even though he points out the obvious downsides of using sources not

validated by the peer review process.

2.3.5 Comparing Cyber Kill Chain Model with MITRE ATT&CK

The cyber kill chain model in its many layman adaptations is established by analyzing cyber attacks and penetration tests. Even though this provides a simple model that helps to understand the general stages of an attack, it does not provide enough detail to categorize attacks or plan penetration tests or attacks. The steps listed in cyber kill chain (CKC) should be present in most attacks. For example, leaving out reconnaissance or exploitation would not be likely to result in a successful and stealthy attack.

MITRE ATT&CK is post-exploitation, based on an "assume breach" mentality. It originally expanded the last three stages of the cyber kill chain (installation, command and control; and actions on objective) to ten tactics. (Strom et al., 2017). The latest MITRE ATT&CK framework (MA) contains the last four stages of the cyber kill chain, as the exploit phase is expanded "initial access" and "execution". (Mitre, 2019) MITRE has also published MITRE PRE-ATT&CK (MPA) that covers the pre-exploitation part. In the cyber kill chain, the pre-exploitation consists of reconnaissance, weaponization and delivery. MITRE PRE-ATT&CK also covers areas that are not in scope of the cyber kill chain, such as continuous development of the adversarial groups' capabilities and infrastructure. Similarly to the cyber kill chain, MITRE PRE-ATT&CK also provides detection and mitigation steps for each adversarial tactic. The weaponization phase of the cyber kill chain seems difficult to detect, even though authors state that each stage can be defended against. One take on combining these popular stage models is Pols (2017) 18 step Unified Kill Chain. Even though its detailed steps could help to analyze attacks, many of those would not map well for searching for similarities in malware and configuration management systems.

In table 16, I propose one way of mapping the seven phases of the cyber kill chain to MITRE ATT&CK and PRE-ATT&CK stages. Some MITRE ATT&CK stages could apply to multiple cyber kill chain stages, such as defense evasion. This table is based on a version of ATT&CK framework before "Reconnaissance" and "Resource Development" were moved to MITRE Enterprise ATT&CK Matrix and PRE-ATT&CK was deprecated in October 2020.

Table 16: Mapping Cyber Kill Chain to MITRE ATT&CK (MA) & PRE-ATT&CK (MPA) model tactics

Cyber Kill Chain	MITRE
vn/a	MPA: Adversary OPSEC, establish & maintain infrastructure, persona development
1. Reconnaissance	MPA: Priority definition planning & direction; target selection; technical, people and organizational - information gathering and weakness identification
2. Weaponization	MPA: Build capabilities
3. Delivery	MPA: Stage capabilities
4. Exploitation	MA: Initial access, execution
5. Installation	MA: Persistence, privilege escalation, defense evasion
6. C2	MA: Command and Control
7. Actions on Objectives	MA: Collection, exfiltration, impact, credential access, discovery, lateral movement

This thesis utilizes methods used by criminals and applies them to benign enterprise configuration management tools. Earlier in "Evolution of Malware", I examined the methods used by specific popular malware and the generic categories made by employment of these methods. In this chapter, I will integrate the established cyber kill chain model with the MITRE ATT&CK model. Even though both of these models are created with hostile activity in mind, I will take my comparison of malware and CM goals and apply it to both cyber kill chain and ATT&CK. As ATT&CK details specific methods, the results of these steps could be implemented in test programs. The existence of analyzed malware case examples allows us to evaluate the likelihood of the success of these techniques without costly implementation, testing and simulation. Thus, only the best candidate techniques will be chosen for further implementation and testing.

2.4 Conclusion

With modern configuration management tools, administrator can define infrastructure as code. The target state is defined, and the configuration management tool only makes changes if needed. This quality is called idempotency. Configuration management system can provide single source of truth of the state of all controlled computers.

The key concepts of configuration management have been developed earlier than commonly thought. The challenges of increasing systemic complexity in

networked computing systems have been identified already in 1940s. In 1980s, some concepts of modern configuration management systems were already present in project Athena in 1980s, and papers identifying the concepts of modern configuration management were published. The earlier works use their own names for the key concepts, which might be a reason why these are sometimes omitted in literature reviews. As a research field, configuration management is still small.

Literature identifies multiple gaps and areas for future research that are of interest to this work: security, scaling, defining configuration, empirical validation and integration. Configuration management system allows full and unlimited control of the slave nodes. Thus, administrator access to configuration management system grants full access to anything else of value among all nodes participating the system. It's surprising that there is so little research on security of configuration management systems. Lot of research has concentrated on tools and using a domain specific language for configuring the slave node. Some of the complications of DSL seem to rise from challenges inherent to the DSLs themselves. Solutions should be empirically validated, and many authors call for more empirically validated research.

Real life configuration management tools must usually communicate over hostile and untrusted network, the Internet. It's surprising that this area has not been the focus of more studies. A direct connection between master and slave is proposed or implied in many publications. This leads to identifying exactly two architectures - push and pull - depending on which side, master or slave, is running the server. The same approach is taken by industry leading and most cited configuration management tools Puppet, Chef, Fabric, Ansible, Salt and CFEngine. To further protect this network communication, the use of virtual private networks (VPN) and blockchains have been proposed. As soon as attacker compromises one slave node, he can use VPN connections to identify and directly attack master. Compromise of master leads to compromise of all nodes in the system. These attacks can be timed so that a timely expert human response is unlikely, such as nights and holidays. Blockchain based solutions considered here did not have the focus of protecting master secret key, did not detail all relevant aspects of key management, and did not extensively show the benefit of using blockchains and smart contracts compared to simpler approaches.

Criminal malware command and control (C2) has evolved very fast due to pressures in the field. Continuous arms race between defenders and attackers has resulted in focus on security and many improvements and alternative solutions for secure network communications. Improvements in malware command and

control predate similar improvements in configuration management systems. The lack of empirical research has been identifed as a gap in configuration management systems research. But for malware, there are many examples of huge networks controlling millions of heterogeneous nodes. This provides field tested examples that could be applied to configuration management systems. Concepts used in the field for malware C2 include internet relay chat (IRC), direct socket connections, HTTP, HTTPS, use of innocent third party and peer-to-peer communications.

Cyber attacks are well conceptualized. The cyber kill chain and ATT&CK framework are established stage models. They can be used for both planning attacks and defences. Stage models identify the steps any attacker must take to successfully reach business goals of the operation. For example, attacker must survey the target to find a working method to exploit the system. These models have similar stages, allowing linking between the models.

3 Methodology

The aim of this thesis is to explore survivability of configuration management systems. This is done by identifying and adapting methods successfully used by malware. Based on design science and a constructive research approach, two research prototypes were developed. These artifacts were then validated using multiple methods in two case studies and testing in emulated and simulated environments. These technical tests in simulated and emulated environments shed light on the research prototypes themselves, but also compared and contrasted them to a leading solution in the industry. Case studies validated the research prototype in the field. Business benefits stemming from technical advances were evaluated with semistructured expert interviews.

This chapter presents a constructive methodology that will be used to answer the research question RQ4 in chapter "Designing Hidden Master Architecture"; and RQ5 and RQ6 in "Evaluating and Validating the Hidden Master Architecture". These research questions of interest are listed in table 17. The theoretical foundations for creating the research prototypes were laid in the literature review by creating a new stage model (RQ1), choosing promising resiliency techniques (RQ2) and examining the possibilities of simplifying idempotent configuration (RQ3).

Table 17: Research questions of interest in "Methodology" chapter.

RQ	Explanation
RQ4	How can these techniques and concepts be implemented in a functional prototype?
RQ5	Based on load simulation, faulty network emulation and attack tree analysis, how does the resiliency of the configuration management software prototype - implementing some of the techniques adapted from malware - compare to a leading industry solution?
RQ6	What utility do the models and the research prototype provide when run in field environment with business requirements?

3.1 Research Philosophy

Constructive research means building an artifact (construct) to solve a domain specific problem to create knowledge of how this problem can be solved in principle. The results can have both practical and theoretical relevance. Constructive research methods are fundamental to engineering and the sciences

when working with formation, modeling and using artifacts. (Crnkovic, 2010)

A design science paradigm is compatible with a constructive approach in its main features. Both approaches seek to create and evaluate constructs. Constructive research could even be seen as a subset of design science with a clear requirement for instantiations, such as computer programs. (Piirainen and Gonzalez, 2013)

Design science is a problem solving process. It consists of building an artifact and evaluating it. Design science is, by its nature, iterative. The two processes of building and evaluating artifacts can be seen as a cycle. Both process and artifact are part of the design. The artifacts are constructs, models, methods and instantiations (working prototypes). Constructing a working instantiation, such as a software prototype, proves that such instantiation can be built. This "proof by construction" demonstrates the feasibility of both process and the product. (Hevner et al., 2004)

Epistemology is the study of knowledge, determining what is true and how the truth can be acquired. Use of design theory and a constructive approach do not strictly dictate epistemology. Piirainen and Gonzalez (2013) note the use of design theory does not have to depend on any particular epistemology, and could even be based just on the utility of the construct. They state that in their view research can contribute to theory while providing utility. According to Hevner et al. (2004), truth and utility are "two sides of the same coin". They consider practical implications to be an important criterion for validating science.

In constructive epistemology, researchers construct new scientific knowledge. This is different from the positivist idea of finding out existing truth from the world. (Crnkovic, 2010) Design science can be seen as proactively creating new technology, in contrast to behavioral science paradigm's reactive approach (Hevner et al., 2004). Design science literature has a pragmatist leaning (Piirainen and Gonzalez, 2013).

3.2 Rationale

In design science, a constructive approach is suitable both for acquiring knowledge of the complex challenge of resiliency, and also to tie the technical results to their business utility. Resiliency in the face of both human hostility and environmental unreliability is a complicated matter. Design choices carry trade-offs, and might close out other options. Some design choices might work better together than others. Even though theoretical thinking has its value, validation of an actual instantiation (artifact, construct) can provide insight

not reachable with just models. Case study in a realistic field context will also validate the utility against real life business requirements. This makes design science or constructive research approach suitable here.

Design science applies existing knowledge when solving problems by designing artifacts (Peffers et al., 2007; Hevner and Chatterjee, 2010; Piirainen and Gonzalez, 2013). The theoretical basis for the work was built in chapter "Literature Review and Related Work", where a model for comparing malware and configuration management was proposed (RQ1), potential resiliency techniques were listed (RQ2) and some background on idempotent configuration was sought (RQ3).

The design science approach advances knowledge by creating and validating artifacts Crnkovic (2010). Following design science, using a constructive approach, a research prototype was developed. In the context of the approach, this could be called a construct, artifact, or more specifically, instantiation. As both the process of design and the artifact are part of the design (Hevner et al., 2004), they can both contribute to new knowledge. In chapter "Designing Hidden Master Architecture", the two prototypes were designed. That design considered the tradeoffs of different architectural decisions, the subsystems and components of the program, and protocols and interfaces between the parts. Simplified idempotent configuration (RQ3), initially considered in literature review, was fully developed there. The implementation chapter answered RQ3 and RQ4, and provided the construct that was validated using mixed methods.

When design science is applied to information systems, evaluation can consider either the technical performance of the system, the overall usefulness to the end user, or both (Hevner and Chatterjee, 2010). In this thesis, I evaluated both technical qualities (RQ5) and end user utility in a business context (RQ6). The evaluation of technical aspects requires different tools and methods compared to evaluation of business utility. Due to this, the study used mixed methods.

Technical aspects of the two research prototypes were evaluated using simulated and emulated networks, with scenarios emulating different use cases and error conditions. Business utility was evaluated in a case study in a company managing and monitoring IoT devices in remote locations controlled by its customers. A smaller case study was created to support and prepare for the main case study.

Technical evaluation uses simulation and emulation, with qualitative analysis of the metrics collected. Case studies combine quantitative analysis of the technical environment and qualitative study of expert interviews.

3.3 Research Design

Utilising the pragmatism of design science, constructive research used multiple methods to evaluate the construct in order to find out how configuration management system resiliency and survivability can be improved. Table 18 shows the activities which were undertaken to answer the relevant research questions. RQ1, RQ2 and RQ3 have been answered in literature review.

Table 18: Mapping empirical research questions to methods

RQ	Explanation	Method
RQ4	How can these techniques and concepts be implemented in a functional prototype?	Constructing two research prototypes
RQ5	Based on load simulation, faulty network emulation and attack tree analysis, how does the resiliency of the configuration management software prototype - implementing some of the techniques adapted from malware - compare to a leading industry solution?	Simulation and emulation testing in virtualized environments, quantitative analysis. Multiple tests in different environments.
RQ6	What utility do the models and the research prototype provide when run in field environment with business requirements?	Two case studies, qualitative and quantitative analysis
RQ7	What potential business benefits experts see for the models and the research prototype?	Semistructured expert interviews

3.4 Design and Construction of the Prototype

Constructing the two research prototypes provides the constructs for validation in the next phases, but it also describes the process and reasoning for design choices. The first simple prototype was a set of scripts under 100 lines of code. It used an existing, leading CM tool Puppet, but replaced the transport mechanism with an implementation of the Hidden Master.

An advanced prototype was also developed. It was a fully self-contained, dependency free tool that can be distributed as a single binary. This prototype

is over 4000 source lines of Go. As it is pure Go, it can be statically linked for multiple operating systems and CPU architectures, even though it is currently only for Linux amd64. The binary contains builtin support for Python-based configuration language, cryptographic functions and networking. This advanced prototype was used in the two case studies.

Prototype design and implementation is in chapter "Designing Hidden Master Architecture".

3.5 Technical Evaluation with Simulation and Emulation

Technical evaluation was used for measuring the capabilities of the software prototypes, and to compare and contrast them against a leading configuration management tool.

Software developers use different types of testing throughout the development process, and developing the advanced prototype was no different. Unit testing is used by programmers to test the correct operation of functions and classes of software, and these tests are usually written in the course of programming. For the advanced prototype, over 700 lines of code were unit tests. In this work, unit testing is considered part of the implementation and not part of the evaluation.

The software prototypes implementing the Hidden Master architecture were tested in both single machine tests and in an emulated network. For key tests, comparison was performed against one of the leading configuration management tools. Because my threat model for configuration management involves an unreliable and untrusted network, many tests were network based.

The difference between simulation and emulation is not always completely clear. Keti and Askar (2015) call Mininet both emulator and simulator in the same paper. When computers are pretending to be other computers, it is difficult to draw the line between which models are different enough from real life to be called simulation. With more hands on physical phenomena, the difference is easier to define. In his paper on material handling, Brooks, Davidson and Gregor (2014) defines simulation tools as those that merely approximate the real world, trying to make savings in computations and time; and emulation is that which just runs the real logic without real equipment.

The gradient nature of emulation versus simulation can be shown with examples from tests taken in this work. To measure the capability to withstand load, the agents were simulated by a single load-testing application that generated many network requests, which could be called simulation. Another test ran full operating systems with emulated hardware using VirtualBox - an emulated test.

Tests running on containers, such as Docker, could be considered emulation as they ran the real software, even if the software had to sometimes use uncommon settings to adapt to a lighter environment.

The specific tests started from evaluating the feasibility of the HM concept, to undertaking simple research and finally arriving at the advanced research prototype. The advanced research prototype and some industry leading software were then tested individually in various adverse conditions. The software chosen for comparison were researched in chapter "Leading Configuration Management Tools".

Industry leading tools require direct connection between master and agents, and thus require master secret keys to be constantly on network connected machines. As the master secret key allows immediate and complete access to all agents, this is highly valuable. HM breaks the assumption of direct master-agent connection, providing much better protection for CMS and the controlled machines. This aspect was analysed using attack tree analysis.

3.5.1 Tests Performed in Emulated and Simulated Environments

- Proof of Concept (PoC)
- Golden Path
- Load Test

Effect of Network Faults

- Virtual environment with fault injection
- Effects of Adverse Network Conditions to Salt
- Effects of Adverse Network Conditions to wget (HTTP)
- Effects of Adverse Network Conditions to SSH
- Comparing the Results of Adverse Network Conditions
- Comparing Memory Consumption and Resiliency Under Load

The Hidden Master architecture allows us to discard the requirement of direct, near real time two-way connection between master and agent. The asynchronous and file based nature make it possible to swap the transport layer. It was tested if the advanced research prototype could survive when the upstream internet was disabled, the network was partitioned and did not have a clear, pre-planned topology. For this, the transfer layer was changed to Syncthing peer-to-peer (P2P) communication. Another experiment tested swapping the transport layer from physical medium transferring encrypted catalogs to an air gapped network.

- P2P Operation in Shattered Network
- Air Gapped Operation

3.5.2 Requirements for the Simulation Environment

To give useful and realistic evaluation, the simulation environment should be able to run actual code in each node, generate regular network traffic and route it between nodes. Multiple experiments include problems in the network. The simulation environment must be able to simulate packet loss and varying latency (jitter) in a controlled way.

Scaling to larger environments brings with it new problems that do not exist in smaller, simpler networks (Gyarmati et al., 2013). The simulation environment should be able to simulate at least tens of machines on regular commodity hardware. If a test with hundreds of machines were possible, it would be beneficial. As some leading CM tools implement their own transfer protocols, creating a test load of multiple agents might require the actual running of these agents. This is in contrast with the use of plain HTTP file transfers in the research prototype, where standard web load testing tools could be used.

Even if the simulation of very large number of machines was made feasible by using very light containers, RAM requirements of the configuration management tool being tested might still limit the maximum number of test nodes. To achieve greater efficiency, for example lower use of memory, light containers make optimizations which make the containers less like real computers. As the leading configuration management tools have a large number of dependencies, this could make it difficult to install them when containers are very light.

For questions requiring a numerical answer, tools for gathering and processing the measurements had to be created. As far as possible, this was done by combining and applying existing tools and libraries. Such questions included the level of adverse conditions tolerated, such as maximum level of tolerated jitter or packet loss. Also the scalability of the systems was measured, such as the number of agents served by a single master under limited memory.

The requirements for the simulation and emulation environments are in table 19.

Table 19: Requirements for Simulation Environment

Requirement	Scenarios
Network	All but PoC
Fault injection (packet corruption, loss...)	Adverse network conditions tests
Command time measurement	Adverse network conditions tests
Traffic generation	Load test (for the research prototype)

Requirement	Scenarios
Light containers	Load test (for industry leading CM tools)

Proof of concept testing was performed using a single Linux computer, using local directories to simulate different computers. Planned testing environments for different scenarios are listed in table 20.

Small networks with only a couple of nodes and simple topology are easy to create with full virtual hosts. Common choices for testing are VMware, QEMU and VirtualBox. Virtualization solutions for production, such as Xen and KVM, were not considered. Both QEMU and VirtualBox have Free (by FSF definition) licenses. VirtualBox was chosen because it performed well on target OSes and was easy to automate with IaC tool Vagrant. Automating the creation of a virtual environment made the tests repeatable. As each operating system runs its own kernel, has its own RAM and emulated hardware, only a limited amount of hosts can run on a single physical computer. In our tests, the performance with full emulation depends on the required RAM memory, but seriously degrades after 5-10 nodes.

Larger number of nodes can be simulated using containers. Containers, such as Docker, are similar to full virtual machines in that they create the illusion of a separate computer. However, many resources, such as kernel, are shared between different containers. This makes containers much lighter, thus allowing a larger network and more nodes on the same computers. Popular container technologies include Docker, VServer, OpenVZ and LXC.

Mininet was considered and initially tested as it provides software defined network (SDN) capabilities with Python API, making it easy to create different network topologies, and add packet loss and jitter. In the initial tests, without any CM agents, Mininet (mn) was able to run 500 nodes on a commodity Linux desktop computer when no software was constantly running on the nodes. The initial tests ignored many practical considerations, which were left to the evaluation phase. Such considerations included separation of the container process space, separation of the container file systems and the RAM requirements of the actual software being run. Even though most of the challenges with MiniNet light containers could be solved, ultimately the industry leading tools did not run reliably enough in MiniNet containers. To create a fair test without undue influence from the testing environment, Mininet was not used for the tests.

Docker (also known as Moby) is a popular container technology. Containers

are defined as Dockerfiles, using Docker's own DSL. Thanks to pre-built initial images and caching, containers are faster to build and start than full VMs. Docker DSL is built around the idea of running one main application, which requires some changes to how the tested applications are run. However, these minor challenges were solved, and Docker proved capable of running the tested applications reliably for load testing. Docker was chosen as the container platform for load testing.

Table 20: Tools for different experiments

Tools	Scenarios
Vagrant, VirtualBox	PoC, Golden Path, Load Test, Fault Injection, P2P
Docker	Load test
Apache Bench (ab)	Load test
Netem	Adverse network effect tests
SSH, bash	All tests
Bash scripts	Adverse network effects tests, load test
Python and libraries matplotlib, pyplot json, pandas, logging, argparse	Adverse network effects tests, load test
Debian 11 Bullseye Linux	All tests
Fedora IoT 33 Linux, Ubuntu 18.04 LTS Linux	Preparation for case studies

3.6 Field Evaluation in Case Studies

Field validation connects the ideas and their instantiations to business utility. The advanced research prototype was tested in two case studies. In a smaller case study, the prototype was used to monitor and evaluate 23 manual system installations. In a larger case study, the prototype was tested in a realistic company environment to evaluate its business utility.

Validation of the construct is an essential part of the constructive method (Hevner et al., 2004; Crnkovic, 2010; Piirainen and Gonzalez, 2013). In both design science research and the constructive research approach, a construct (often called "artifact" in design science) is created based on real life problems and then evaluated. Theoretical contributions are identified and communicated. (Piirainen and Gonzalez, 2013)

The Hidden Master architecture and other novel ideas were implemented in a research prototype. Field validation aimed to answer RQ6: "What utility do the

models and the research prototype provide when run in field environment with business requirements?". Case studies were also expected to provide additional or supporting information for other areas. Indeed, case studies were expected to show if the design and implementation studied in the laboratory testing as part of RQ5 also adapt to complex and unexpected environments dictated by business requirements. Together with laboratory testing, case studies tested the features - the potential business benefits of which were later identified and evaluated in expert interviews.

3.6.1 Smaller Case Study

When the advanced prototype had only been tested in the laboratory, an intermediate step was needed before the main case study. Deploying the software in a production network, asking third parties to develop an advanced configuration and to operate it would not only be laborious, but might also pose unexpected risks. Thus, a smaller case study was performed in a smaller environment.

As a laboratory testing was by its nature is limited, it would also need to be tested if the system could adapt to varying real life conditions. Different operating system installations run by different operators could be faulty or in unexpected states. Network conditions could differ, and it was later found that networks in this smaller case study had multiple complications, such as multiple network address translations.

A course on Linux and Free software was partly evaluated by a multi-hour hands-on exercise. Due to the corona pandemic, this exercise could not be done in computer laboratories, which created a challenge for monitoring progress and evaluating the final result. The participants gave their informed consent to join the study. The results are presented so that the details of individuals are not visible. Remote monitoring and evaluation was done using Conftero, the advanced research prototype.

Based on a hypothetical business case, the candidates installed Linux in a virtualized environment, then proceeded to create users, install server and client software and write simple applications to demonstrate that components interoperated correctly. Participation in the research was voluntary. A fallback system for collecting partial system information was created to make sure that possible software bugs in Conftero would not risk evaluation.

Procedure for the case study

- Researcher sets up an intermediate courier/drop server in a rented cloud VPS (virtual private server). This server does not need to be trusted, so

inexpensive cloud services can be used.

- Researcher generates a single binary installer that includes the campaign key to join as an agent. This is a standard feature of the advanced research prototype.
- Candidates download and run the installer. The authenticity of the installer is verified by using a previously known address with a valid HTTPS certificate. Certificate validity is verified by the lock icon on browser address bar.
- Agents start automatically and periodically synchronizing with the courier/drop.
- Researcher sends new tasks using the advanced research prototype. As part of master syncs, reports from agents are automatically collected.
- Researcher sends a configuration that collects additional data from agents.
- Candidates are instructed to remove their exercise operating systems when they are ready. Due to the Hidden Master architecture, the last state of agents is already stored in courier and receivable by the master later.
- Research synchronizes master with the courier/drop. Courier/drop infrastructure is destroyed when the exercise ends.
- Researcher analyses the data offline, on the master node. Agents and agent node operating systems and the intermediate server and all data they contain is deleted as they are no longer needed.

The information for this test was collected using the builtin facilities of Conftero. This was possible because backchannel (agent to master) and reporting capabilities were built earlier for the company X case study.

3.6.2 Deployed to Production System by Company X

The advanced prototype Conftero was tested in Company X. The company develops embedded systems for audiovisual applications and cold chains, with pre-emptive tuning using artificial intelligence. The goal of the study was to see how Conftero adapts to real life applications in their test network. This larger case study was part of answering RQ6: “What utility do the models and the research prototype provide when run in field environment with business requirements?”.

Earlier parts of this study were essential to perform this case study. The concepts to be tested, such as the Hidden Master architecture, were developed by using the stage model to compare malware and configuration management. These were designed and implemented into two prototypes. The advanced prototype was tested in a smaller case study. This larger case study took the

advanced prototype and applied it to the business requirements of a larger case. The environment in this case was more complex both technically and businesswise. The final step of this process, the identification and evaluation of business benefits, used the previous phases to present the qualities and features of which the business possibilities were then analyzed by experts.

The details of this case study was planned together with Company X management. The case study happened in multiple phases:

- Scoping and planning
- (Feature implementation)
- Requirements gathering
- Deployment
- Wrap up

The main participants for the phases varied. All phases also had minor involvement from other parties, such as programmers commenting on the work or an individual providing some technical details of a system. The scoping and planning phase in September 2020 was done with Company X management. The scoping and planning phase also identified some features that were part of the design but not as yet implemented, such as the backchannel from agents to the master. Where a suitable case was provided to gain real life experience on those features, they were implemented. Feature implementation was done by the researcher during autumn 2020. Requirements gathering for the specific case was done with the CTO. The CTO provided the major requirements and explained the customer's view. A junior administrator helped with the company's standard requirements and systems. Deployment was performed with the junior administrator during July 2021 using the advanced prototype, with the research observing and participating undertaken by answering questions on the prototype. This consisted of over 20 hours of observation and collection of technical data with the prototype. CTO provided some higher level input for the deployment phase. The wrap up meeting was in August 2021 with the CTO and junior administrator.

Scoping and planning was performed to find a suitable business case for testing, as well as to establish ground rules and a legal framework for testing and finding agreement on project goals. Multiple business cases were considered before choosing the multiroom AV system and related services. The business case needed to be complicated enough to provide a chance to validate in a complex environment, but simple enough to be limited in scale. As testing involved running a very new code, the system needed to be such that they did not cause risk to human health (such as cold chains) or large financial risk (sales logistics). The case needed to be mature enough to allow for the possibility to

move from test to production environment. Because translating configuration from one language to another is a simpler and a different task for developing configuration, a case where new configuration was required was chosen.

Company X had strict requirements for security, reliability for client facing services and confidentiality. This required agreements, and sometimes imposed limits to the technical tests that were performed. After the research prototype worked well in simulated environments and a detailed contingency plan was created, Company X and their client allowed deployment of the advanced prototype in their production systems.

Requirements gathering collected both low and high level requirements. Clients (of Company X) views were relayed by the technology lead, and he also explained the high level requirements. Junior system operator provided the details of standard systems and their integrations.

The deployment phase involved participant observation of a junior system operator developing, deploying and operating the code. This involved over 20 hours of participant observation. In wrap up, experiences and views were collected.

It was originally expected that some of Company X's embedded Linux systems would be too light to run one of the leading configuration management tools, but they had recently upgraded to more powerful IoT hardware.

Based on discussion on the requirements and the business case, multiple possible test environments were recognized. One option was that the key exchange for an IoT VPN system could be done with Conftero in order to remove a tedious manual step for key verification. Another option was replacing and automating remote management of a new brand of IoT devices in the customer's premises, to both make control and monitoring more reliable as well as reducing manual work and errors.

A major feature, the back channel for reports from agents to the master, was developed on the request of the case company.

The results of this case study are reported in Findings and Analysis.

3.7 Expert Interviews

Expert interviews were conducted to link technical contributions to their potential business impacts, and to gain a view on linking the technological aspects configuration management to issues faced by practitioners in the industry.

Thematic analysis with live coding was used for identifying key concepts from interviews. The main steps of the analyses were similar to those suggested by

Braun and Clarke (2006) in their highly cited article on thematic analysis: familiarization with data, generating initial codes, searching for themes, reviewing themes, naming themes and producing the report. Even though transcription is often considered a standard practice, it may be unnecessary or unwanted in some cases (McMullin, 2023). Live coding, marking codes directly on video or audio recording, may eliminate the need for transcription (Parameswaran, Ozawa-Kirk and Latendresse, 2020). Live coding was used in this work.

Initial codes were generated from interview notes, and initial theme map was created. Based on the initial codes and initial theme maps, the actual coding was performed by tagging interview videos. For video tagging, VideoLAN Client was chosen because it was able to play audio at various speeds without stutter, and had filters to fix and normalize low quality or quiet audio. Interview videos were coded using VLC video player and permanent bookmarks plugin, except for where usable audio track was not available and coding was done on interview notes. Coding was done using the codes improved from the initial coding, with new codes identified on the second coding round added as they emerged. Based on the main coding phase, the main theme map emerged. Finally, a report was written with quotes to support the themes identified. The quotes were translated from Finnish by the author of this thesis.

The contributions improving idempotent configuration and network architecture can be readily implemented in software, and it is possible to make assumptions about the expected business benefits of these contributions. These contributions are the Hidden Master architecture, the use of imperative general purpose language for idempotent IaC configuration, the resource revalidation model and the simplified resource dependency models.

The technical validity of these concepts (OBJ3) was validated by designing and constructing two prototypes (RQ4, OBJ4) and evaluating them in emulated and simulated environments (RQ5, OBJ5). Applicability for realistic field use was validated in a field test in a production environment in two case studies (RQ6, OBJ6). In the main case study, interviews and observation also gave indications of possible business benefits. The research questions and objectives are listed in table 1.

Potential business benefits were evaluated in semistructured practitioner interviews. This provided the answer to RQ7: “What potential business benefits experts see for the models and the research prototype?”(#researchQuestions) (OBJ7). The technical qualities had been verified in earlier phases of the research. As the research prototype was new code, it is not yet suitable for large scale use in production environments where mistakes or programming errors could cause huge losses or danger to human health. Thus, for some

benefits, the business impact had to be evaluated using interviews.

Interview questions and prompts were based on the expected benefits of the key contributions, implemented in the two software prototypes. In addition to interviews and observation performed in the course of the main case study, the interview phase targeted 4-8 interviewees.

Selection of the interviewees was a challenge due to the requirements for their background, stemming from the research question. The interviewees were recruited using networks of the researcher. All interviewees were required to have multi-year experience in at least two of the following areas:

- Configuration Management Systems
- Management (in the organizational sense)
- Cyber security

As the main case study already provided a focused view on Company X and their client, the interviewees were chosen outside organizations involved in the main case study. To gain a broad view, each interviewee was from a different organization.

The interviews were individual interviews, performed over video conference connection. First, a brief background discussion and questions were handled. Then the interviewee was shown the operation of the main software prototype, offered a short description of the already completed main case study, and provided a chance to ask clarifying questions. Interviewees could also ask demonstration of specific capabilities. Then the interviewee was prompted to evaluate claims on the business impact of the concepts and demonstration. Even though specific questions were offered, the emphasis was on open ended, free form commentary. Interviewee was also given a change to freely comment on the area outside questions and prompts.

The prompts, questionnaire for the semi-structured expert interviews, is included as Appendix: Questionaire for Semi-Structured Interview. The questions are formed based on the expected business benefits arising from the qualities of the concepts themselves. It should be noted that these claims were used as prompts to open ended evaluation, to encourage participants to consider multiple areas of and aspects in the limited time given.

Generalizing the interviews is limited by the small number of participants (six) and the fact that they all are working in a single country, Finland. While larger interviews can of course be performed in future studies, the possibility to perform larger case studies in production networks should be considered as the software matures.

The interviews were performed over a video conference in Finland between May and August 2022. The interviews were conducted as individual interviews to allow for questions and open-ended answers. Six experts were interviewed and, as seen in the findings, a saturation point seems to have been reached. The background of the experts and their companies is discussed with the results. The results of the interviews are in Findings and Analysis: Interviews to Evaluate Business Benefits.

3.7.1 Ethics in the Interviews

Multiple ethical considerations had to be taken into account in the interviews, even when the experts were speaking in their professional capacity and the issues were not highly personal.

A key reason for identifying and evaluating potential business benefits using expert interviews - instead of simply running production on a large scale and measuring the business benefits - is partly dictated by ethics. Even if the prototype includes novel contributions improving security, it is at a prototype level and has not had third party security auditing. Considering that a CM tool has full access (root / Administrator / SYSTEM) to all systems, it is clear that a research prototype implementation cannot meet the maturity level required. Despite the quality of the concepts, security would be limited by the quality and testing of the implementation of the prototype. Thus, the choice of the interviews themselves as a method was dictated by ethics in addition to practical concerns.

CM systems are critical to security, and - as discussed earlier - the secret key of the master is the most valuable file in the whole network controlled by the CM tool. Security tools related to CM - such as EDR, IPS and IDS - often require some secrecy to provide maximum benefits. The interview questions also ask about pain points in current systems, often an area that cannot be publicly discussed so as not to present companies in a poor light.

To protect the interviewees and their current and former employers, multiple steps were taken. The issues required that the individual comments should not be easily traceable to the interviewee, or at least this connection should be plausibly deniable. Also it was offered to the experts that if they mistakenly revealed some confidential fact about their employer's systems, it would be taken out of the material at their request. The experts could choose if they wanted their name and company to be included in the acknowledgments.

One of the experts asked minor technical facts to be removed from the interview. These facts were irrelevant to the research, and they were removed. One expert provided background information that was omitted as irrelevant to the

research and potentially sensitive. Multiple respondents indicated that they had preconditions to participating, but found that the protections planned for the interviews already met their demands, and no single person special arrangements were needed.

3.8 Conclusion

Constructive research means building an artifact (construct) to solve a domain specific problem to create knowledge of how this problem can be solved in principle. This artifact is validated. The results can have both practical and theoretical relevance. In literature review, we saw the lack of empirical validation as one of the gaps in configuration management research. The use of constructive research approach fits this gap well.

The stages started from theory and finished in business aspects. This work created a theoretical model to compare malware command and control to configuration management systems. The model helped to bring these widely field tested concepts from malware to configuration mangement systems. Improvements to configuration management systems were combined into coherent design.

To create the artifact to be validated, two software prototypes were created. The simple one helped to better understand the problem, and to provides a toy system that can be quickly understood also on the source code level. The advanced prototype, a four thousand line Go program, can be used in the field for configuring production networks.

The advanced prototype was first validated in laboratory environment and compared to industry leading configuration management systems. To test applicability in the field, multiple case studies were performed in progressively more challenging environments. Field tests also validated the approach in face of business requirements and human aspects, such as learning and usability. The main case study involved observing the use of the system in a production network of a client of the studied organization. During this test, the protype was operated by a junior system operator in the organization studied. Obviously, extra precautions were observed when validating immature software on production systems.

Finally, expert interviews were performed to identify potential effects to business. These interviews were informed by demonstration of the prototype and the results of previous validation phases. Interviews were thematically analyzed.

4 Designing Hidden Master Architecture

Expanding on foundation created in literature review, in this chapter I will build a novel stage model for comparing malware and configuration management systems (RQ1), identify suitable resiliency techniques (RQ2) and simplify idempotent configuration on agents (RQ3).

Following the constructive research approach introduced in chapter 3 "Methodology", this chapter describes the design of the configuration management system. Both the process and the product (the prototype) are part of the design and can help to build knowledge (Hevner et al., 2004). The system utilizes the answers to research questions 1-3 mentioned above. Instead of just looking at individual improvements, this chapter takes on the challenge of building a complete and coherent configuration management system. Different choices and tradeoffs are analyzed, and two functional prototypes are designed and implemented. This answers RQ4 "How can these techniques and concepts be implemented in a functional prototype?".

The research questions answered in this chapter are listed in table 21.

Table 21: Research questions answered in the Design chapter

RQ	Explanation
RQ1	How can existing stage models be adapted for comparing malware and configuration management systems?
RQ2	Which malware resiliency techniques are applicable to benign configuration management systems?
RQ3	How can defining idempotent agent configuration be simplified?
RQ4	How can these techniques and concepts be implemented in a functional prototype?

4.1 A Novel Phase Based Model for Comparing Malware and CM

The main aim of this thesis is to explore survivability and resiliency of configuration management systems based on concepts used successfully by malware. To focus search and categorize the results, a model was needed for mapping areas of similarities and differences between the systems. Developing this model was a contribution of this work, and the answer to RQ1. The leading attack and malware related phase models were examined and their phases mapped

to each other, and then to stages of configuration management systems. Analyzing and categorizing steps from malware is a huge undertaking, but using MITRE ATT&CK as one model provided a large number of successful attack and malware campaigns already categorized to this model. Using the model combining malware and configuration management stages, it was then possible to map those attack techniques to CM related categories. This allowed malware resiliency techniques be considered for use in improving the resiliency of CM, the aim of this thesis.

Malware has to survive in an environment that is more hostile and less reliable than a common enterprise network. When successful, the criminal activities can bring great economic rewards. This combination has led to the fast evolution of malware and the development of interesting command and control architectures. Real world production malware uses push, pull, IRC, social media, P2P, HTTP and HTTPS transports, with and without intermediate hosts. They provide an interesting menu of battle proven techniques to test on benign enterprise configuration management.

To compare these tactics, techniques, procedures (TTP) both in this work and continuously in the future, a model for mapping malware TTP to those suitable for benign enterprise CM could be useful.

Phase-based models have been used for mapping hostile activity. As discussed in "Models for Attacking Computer Systems", cyber kill chain (Hutchins, Cloppert and Amin, 2011) and MITRE ATT&CK (Strom et al., 2018) are phase-based models for understanding and mitigating computer intrusions. These models themselves leverage existing phase based models in multiple fields. In the paper introducing cyber kill chain Hutchins, Cloppert and Amin (2011) recognize multiple such models, mostly from the US military, but also from the areas of computer security and insider threat prevention.

In table 16, I proposed one way of combining the cyber kill chain into two MITRE ATT&CK and MITRE PRE-ATT&CK (MPA) frameworks. By combining cyber kill chain and the latest MITRE Enterprise ATT&CK matrix, a framework for searching useful features tested in malware could be developed. As the key steps of MPA have been moved to Enterprise ATT&CK matrix and the old MPA deprecated in October 2020, this model will use the newer matrix.

When considering these stage models as a tool for finding useful examples to be applied to managing our own computers and networks, the wording of roles of different actors and systems should not be taken literally. To take the first technique of the first tactic of 2021 ATT&CK matrix as an example, Reconnaissance: Active Scanning, here the description talks about adversaries,

victims and detection. When managing our own computers, the target machines are not victims, and we are obviously not our own adversary. Scanning your own networks is a sensible security practice, which can be used for both asset discovery and verifying that there is no 'shadow IT' or other unexpected servers. Thus, this tactic can be applied to configuration systems. As there is ready-made ATT&CK based analysis provided for existing attacks, and it is a detailed framework, it provides a good starting point for analysis. table 22 maps Cyber Kill Chain phases and MITRE ATT&CK tactics to their configuration management counterparts. The seven Cyber Kill Chain phases were numbered from CKC1 to CKC7 in order to refer to them in this study. MITRE ATT&CK tactics use the IDs given to them by MITRE.

Differences in the approaches of malware and configuration management should also be considered. Even though there do not seem to be popular models of configuration management tool qualities, there is some agreement on the key features of modern configuration management: idempotent, infrastructure as code configuration and single source of truth. The offensive stage models considered here do not list any of these features or qualities. As seen in the chapter on malware evolution, a single source of truth is a common feature in botnets. Malware configuration is often text based, but the author is not aware of research that would indicate if malware authors use version control or idempotent configuration. These features should be included in a model on configuration management systems. The potential benefit of these features to malware is left for future research.

Table 22: Examples of attack stage model techniques with configuration management counterpart.

Cyber Kill Chain and ATT&CK	ATT&CK Techniques applicable to CM
n/a, TA0042 Resource development	Acquire infrastructure
CKC1 Reconnaissance, TA0043 Reconnaissance	Active scanning...
CKC2 Weaponization	n/a
CKC3 Delivery, TA0001 Initial access	Replication through removable...
CKC4 Exploitation, TA0002 Execution	Trusted relationship, valid accounts...
CKC5 Installation, TA0003 Persistence, TA0004 privilege escalation, TA0005 defense evasion	Scheduled task...
CKC6 C2, TA0011 Command and Control	Application layer protocol...

Cyber Kill Chain and ATT&CK	ATT&CK Techniques applicable to CM
CKC7 Actions on Objectives, TA0009 Collection, TA0010 exfiltration, TA0040 impact, credential access, TA0007 discovery, TA0008 lateral movement	Data transfer file size limits

To map the steps in these attack models to configuration management, the stages of configuration management should be considered. Modern configuration management tools use infrastructure as code (IaC) approach to create idempotent configuration Hummer et al. (2013).

They use master-slave architecture, so that the master sends configuration to slaves (sometimes called agents), which they always and automatically apply. As decisions are made by the master, it can be seen as the single source of truth in the system.

These instructions are created by the administrator of the systems, then passed from master to agents over an untrusted and unreliable network, namely, the Internet. For visibility of the network, the agents report their state back to the master. Because the network is untrusted, the instructions must be cryptographically protected. The cryptographic protection for instructions includes encryption, signing and versioning. For constantly held connections, two-way authentication can be used in place of signing.

The stages of defining, transferring and applying configuration relates to the continued operation of a CM system. Even though this is the main part of the CM life cycle, the system must first be installed and trust established between participants before the operation can start. When this stage is added, we can start modeling the configuration management tool life cycle.

Some common corrective steps are required to keep the system operating safely, but they are not needed continuously. When the master or agents change their network configuration, such as the IP address, network communication must be re-established. It depends on the configuration management system's network architecture how this is done. When there are updates to the configuration management tool itself, master and agents must be updated. This step can be mandatory if the updates are related to security. These steps are collected as their own stages under the heading "Special actions".

Before initial installation of master and agents can begin, the administrator should have suitable infrastructure in place. Demands for the infrastructure are

dependent on architecture and business requirements. In many cases, server management can use push architecture and thus requires less CM infrastructure than managing desktops, laptops and IoT devices. To install agent daemon to controlled assets, the assets must first be identified. Leading CM tools leave this as the responsibility of the administrator.

Based on the mapping of malware stage models table 22 and the stages of configuration management discussed above, it is possible to look at which malware stage model stages can provide suitable techniques. This comparison is in the configuration management stage model summarized in table 23. As explained earlier, in ATT&CK the "tactic" is the goal the attacker is trying to reach, such as "persistence" or "initial access". What attacker does to reach the goal is called "technique" in ATT&CK. For example, techniques to achive persistence include T1098 "account manipulation". Multiple projects (including MITRE ATT&CK itself) provide techniques extracted from successful malware campaigns (called "procedures" in ATT&CK lingo).

It should be noted that the mapping of these stages is based on the contents of ATT&CK stages and the techniques they contain, instead of the names of the tactics. The names used in offensive phase models don't always map well to the configuration management world. For example, an administrator managing his own servers does not need to evade his own defenses, but TA0005 "Defense evasion" contains techniques such as T1542 "Pre-OS Boot". Pre-OS boot could be useful in the main case organization to protect remote IoT devices from becoming unresponsive after applying invalid configuration.

When considering a case of a partially compromised agent system, techniques that allow a configuration management system to control a system on a lower level (lower protection ring) could allow faster recovery and forensic collection of data, such as remotely dumping memory. Similarly, TA0004 "Privilege Escalation" contains multiple techniques that are useful in a configuration management context, such as T1547.006 "Boot or Logon Autostart Execution: Kernel Modules and Extensions".

Table 23: Stage model for configuration management operation

Stage	Explanation
T0	**Prerequisites** (TA0042 Resource development, TA0043 Recon, TA0006 Credential access, TA0007 Discovery)
T0.1	Set up infrastructure
T0.2	Discover assets to be controlled

Stage	Explanation
T1	**Initial installation and establishing trust** (TA0001 Initial access, TA0002 Execution, TA0003 Persistence, TA0011 CC, TA0004 Privilege Escalation, TA0005 Defense evasion)
T1.1	Master and agent: Obtain trusted copy of CM tool
T1.2	Master and agent: Key exchange
T1.3	Master and agent: Persistence
T2	**Continued operation** (TA0002 Execution, TA0009 Collection, TA0010 Exfiltration, TA0011 Command and Control, TA0040 Impact)
T2.1	Administrator: Define idempotent configuration
T2.2	Master or agent: Securely transfer configuration to agents
T2.3	Agent: Apply configuration on agent, generate report
T2.4	Agent: Securely report results from agent to master
T2.5	Administrator: View summary of system state
T3	**Special actions** (TA0011 Command and Control, TA0008 Lateral movement)
T3.1	Master or agent: Re-establish master-agent networking
T3.2	Master and agents: Update master and agent software

The 215 techniques in MITRE ATT&CK Enterprise Matrix (Mitre, 2019) were read and those most promising to applied for configuration management were picked for further categorization and study. This initial filtering found suitable techniques in all tactics (categories) and resulted in 89 techniques.

The most interesting techniques were chosen. As criteria, those techniques chosen were to be new in the context of enterprise configuration management systems, not causing excessive risk and possible to implement in the research prototype. They should also concentrate on the areas in the focus of this work, secure network communication and idempotent agent configuration.

Multiple techniques of interest were recognized. T1059.006 “Command and Scripting Interpreter: Python” is used by 28 case examples listed. Malware of particular interest include tools that allow the attacker to execute Python scripts on targets such as Dragonfly 2.0, Cobalt Strike (adversary simulation), CoinTicker, and Keydnap. Many of these tools have to install Python interpreter, either as a separate installation (DragonFly 2.0), or one packaged with Python packager (Machette, DropBook). (Mitre, 2019) This technique could be further improved if we could run a limited subset of Python without packaging the whole installer. It was later found in this thesis that it’s indeed possible to create such a limited subset, and even embed it into a single, statically linked binary.

It was surprising that using dependency-free, static binary was not a technique found in MITRE ATT&CK, even though it is used by malware. As malware has to survive in a heterogeneous environment, one could expect not depending on software and libraries on target would make it more resilient. The opposite of static linking is dynamic linking. In dynamic linking, software depends on libraries either provided by the operating system or shipped with the software. ATT&CK lists dynamic linking, scripting language and depending on a scripting language not provided with an OS under "TA0002 Execution". Despite ATT&CK, dependency-free, statically linked single binary is a technique planned for the main research prototype.

Hidden master architecture, a key feature already in the initial plan, could be seen an example of TA0011 Command And Control: T1102 Web Service. According to Mitre (2019), this technique has been successfully used by multiple malware campaigns.

Threat groups successfully using web servers as part of their command and control network include Carbon, DropBook, Fox Kitten, Inception, Ngrok, Rocke, SharpStage and Turla. Other threat actors have used web service to infect targets in water holing attacks or to host malware stages, but that use is quite different from the hidden master architecture. For a lot of malware, the web server has been a well known third party web service. Carbon used Pastebin; Dropbook has used SimpleNote, DropBox and Facebook; Fox Kitten has used Amazon Web Services. (Mitre, 2019)

From the point of view of threat actor, the third party services they use can be considered untrusted, even hostile. It is likely that the owner of a service such as Facebook is taking active steps to prevent malware from being hosted on its servers. Thus, these malware campaigns can provide examples of techniques used to communicate over untrusted web servers, and real life case examples of how those techniques and procedures have succeeded.

T1092 Communication Through Removable Media allow communication or data exfiltration through removable media, such as USB mass storage devices. T1091 Replication Through Removable Media is mostly used by malware that leverages Windows autorun to infect machines. For T1091, malware DustySky scans removable media. (Mitre, 2019)

As the Hidden Master Architecture dictates asynchronous operation, it raises the question of whether an Internet connection - or any network connection - is really required. As the Hidden Master architecture does not depend on an authenticated connection, but instead individually authenticates the encrypted catalogs, removable media could be used. This would allow secure updates of

devices that are distant, beyond network or lack networking capability. An example of distant device would be a device deployed outside cell phone coverage. Some embedded devices can access USB mass storage, but lack networking capabilities. USB mass storage devices should be considered untrusted for the purpose of running privileged commands, but so is the main communication channel - third party web servers. HM agents could scan removable media to see if they contain encrypted catalogs that have the correct cryptographic signatures.

Persistence (TA0003) is the ability keep running or run again after initial installation. Many persistence mechanisms are listed, including T1547 Boot or Logon Autostart Execution (14 sub-techniques) and T1037 Boot or Logon Initialization Scripts (5 sub-techniques). (Mitre, 2019)

The agent systems controlled might be in hard to reach places, such as remote locations behind NAT, firewall and dynamic address. As the main mechanism to control and gain visibility to these systems, it's critical that CM tool can keep running or gets woken up periodically. If target systems are heterogeneous, multiple persistence techniques might be needed so that at least one matches those available. For example, Fedora IoT Linux only provides scheduled tasks through systemd, unlike many other distributions that have cron pre-installed.

4.2 Design Goals

The aim of this work is to explore survivability of configuration management systems. The resiliency of CM can be divided into resiliency against natural unreliability and resiliency against active hostility in the network. Based on my earlier examination of how malware CC achieves these goals and the attack tree analyses used to point out categories of vulnerabilities in CM tools, I can elaborate this broad aim into specific design goals. Design goals for the research prototype are listed in table 24.

Table 24: Design goals

Design goal	Benefit
Protect master secret key	Prevent full system compromise
Improve network and geographical dispersion	Radically reduce costs and security implications of dispersion

Design goal	Benefit
Decouple subsystems	Change transfer method according to network conditions. Allow statically linked slave daemon.

The master secret key is the single most important file in the whole network of controlled computers because it allows unlimited access to all data and resources on slaves. Compromise of the master secret key would thus result in the full compromise of the whole network.

The computer containing the master secret key should be highly secured. In systems common in the industry, the key is stored in the computer serving catalogs to slaves, thus being continuously connected to the Internet. Securing the computer could require keeping the computer within the organization's own premises and having a 24/7 security response team ready to react to attacks. These stringent security requirements for the master in traditional CM solutions make it very expensive or cost prohibitive to create additional master computers around the globe in different networks.

Improving network and geographic distribution requires reduces the security requirement from hosts directly serving the catalogs to slaves.

The decoupling of subsystems is a common design goal. In HM, decoupling could allow the changing transfer method to match different requirements. HTTPS could be used in favorable network conditions, and P2P techniques and pseudonymous networks employed when conditions deteriorate.

CM should be able to repair systems (Burgess, 1998). Due to dynamic linking (relying on installed libraries), the very same damage causing the need for repair might make CM inoperable. As popular CM solutions are implemented in high level languages (e.g. Python and Ruby) and rely on a lot of libraries, they could be especially vulnerable to this kind of problem. Using a statically linked daemon to apply catalogs on the slave, the surface for this problem would be greatly reduced.

4.3 Protecting the Master's Private Key in the Hidden Master Architecture

As I have noted, combining catalog signing keys (root access to slaves) with catalog distribution causes security risks and other problems. *Hidden master* architecture avoids this problem by keeping the signing keys in a computer

that only connects to the Internet update catalogs on an intermediate server (fig. 3).

These intermediate catalog distribution servers do not need to be secure. Thus, they can be commodity web servers in networks not controlled by the organization using them as intermediate nodes in configuration management.

Security of the system is dependent on encryption and key management, which is detailed later in this chapter. Even though the first transport protocol implemented is HTTP, the separation of encryption from transport allows other transport schemes to be used. The hidden master was also used with peer-to-peer communications in this work.

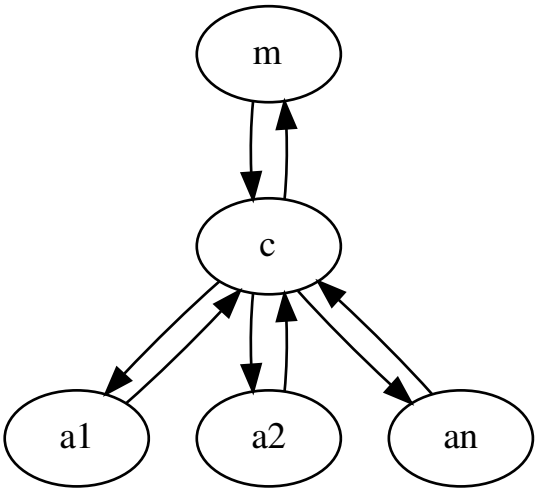

Figure 3: Hidden master architecture network structure

The purpose of the hidden master architecture is securely delivering files to slaves. The key difference to other configuration management system architectures is hiding the master, so that the master does not need to be available to the Internet during the slave configuration.

Using untrusted intermediate hosts such as commodity web servers improves survivability, as the instructions can be provided with a large number of hosts. To further slow down adversaries, all slaves do not have to know all the possible locations of encrypted catalogs.

Many networks limit access to http port 80/tcp and https port 443/tcp. As normal web browsing uses these ports, they also provide a large amount of normal traffic to cover the control management system.

Some networks use deep package inspection in addition to filtering ports. In deep package inspection, the firewall examines the contents of the traffic to decide what goes through. One way communication by downloading encrypted files provides very good protection against deep package inspection, as it would

be difficult to measure the difference of downloading an encrypted catalog file inside encrypted TLS tunnel (https) versus downloading another file.

In the rare case of needing to pass unencrypted communication with deep package inspection, such as a network firewall performing a man-in-the-middle attack against TLS as a requirement for network access, the files could be obfuscated. Security through obscurity is obviously not useful if the obfuscation technique is known to an adversary. Thus, the obfuscation layer should be implemented on a case by case basis. Obvious methods include obfuscating the encryption headers with symmetric encryption (same key for all slaves) and impersonating some other file format by changing the file suffix and magic sequence at the beginning of the file.

As the instructions are simply encrypted files and the intermediate hosts need not be trusted, other transports could be easily implemented. Some interesting choices would be regular peer-to-peer networking or some anonymised network.

Anonymous and pseudonymous peer-to-peer networks include TOR, i2p and Freenet. The main benefit of using an anonymous network is the ability to hide the structure of the network and provide a host that is survivable against censorship. The anonymized nature of these networks can be both a curse and a blessing. For the defender, they could hide the structure of the configuration management network, for example from the owners of third party networks visited by mobile agent nodes. For the attackers they could provide an obvious and already configured way to hide and bypass network perimeter defenses. Due to the security concerns related to pseudonymous networks, it could be expected that they are blocked by many firewalls and identified by network monitoring and intrusion detection tools.

When choosing an anonymous network for transport, it should be noted that the hidden server architecture can ignore one of the main balancing acts of these networks: exchanging anonymity for latency. As a typical CMS only applies the catalogs every 15-30 minutes, the latency of the transport can be large ignored.

4.4 Compromizing Configuration Management Attack Tree

For this attack tree, the goal is to fully compromise the configuration management system, and thus all computers participating in it. This is the root node of the attack tree. This attack tree considers technical attack, and excludes attacks on human elements, such as coercion, bribery and infiltration. Categories of attack removed by the Hidden Master architecture are marked with "(HM)".

- Attack Master
  - through hosting (HM)
    - Attack cloud operator (HM)
    - Compromize physical premises of the machine serving agents (HM)
  - Over public network (HM)
    - Attack master daemon (HM)
    - Attack other software on master (HM)
  - Compromize agent
    - Attack master through CM control channel (HM)
    - Inject mallicious code to report
- Attack operator workstation
  - Lateral movement
    - File shares
    - Authentication
      - Centralized authentication attacks
      - Password reuse
      - Other accounts (local administrator, backup...)
- Poison supply chain
  - Infect any software, library or recursively included dependency
    - Master
    - Operator workstation
  - Infect CM tool supply chain
  - Infect hardware supply chain, e.g. add bugged keyboard
- Attack CM control channel in transit
  - Mathematically break encryption
  - Use flaws in CM tool cryptographic implementation
  - Use operator errors

To better understand different tradeoffs when protecting against the risk shown in the attack tree, we could examine the configuration management system architectures described in the literature and used by industry leading configuration management systems. They are divided into pull and push, both of

which have different risks and benefits.

It should be noted that these attack tree analyses consider architectures and other abstract concepts. In actual cases of real life security, many practical issues have a large impact. Software has to be reviewed, tested and audited. Software can only really mature when it is in large scale production use. Thus, it should be understood that the software prototypes related to this work are not mature enough to be suitable for large scale production. And, despite noting any possible architectural and other improvements, industry leading configuration management tools, such as Puppet or Salt, are mature products used by huge production installations every day.

In this analysis, two different types of network architectures are talked about: client-server and master-agent. Client-server is the architecture used in typical network communication. The server is on, the listening port open, waiting is in a known address, and it is visible to the network. The client connects at the time of its choosing. Master-agent (master-slave) architecture means that the master sends instructions to agents at a time it chooses, and the agent applies these instructions. Typically, configuration management systems use both of these architectures, forming a master-agent on top of a client-server.

In push architecture, the master connects agents at a time chosen by the master. Seen through the lens of client-server architecture, the master is the client, and the agent is the server. The server must be in a known location, have a port open and be visible to network. This requirement rules out moving agent nodes, such as laptops and many categories of IoT devices, as they move to unpredictable network locations, which are often behind firewalls and non-routable IP addresses due to NAT (network address translation). Push architecture does mitigate risk to master. However, it requires every agent node to expose a port to network. Due to these requirements, push architecture is best suited for controlling servers, and thus it still requires a solution for non-server nodes. Ansible is one of the industry leading tools that specializes in push architecture.

In pull architecture, agents phone home. Here, the master is the server, and it must be in a known, network visible location. Pull architecture has multiple benefits, the most obvious being the ability to control moving agents, such as laptops and IoT devices. It also allows for agents that are only on at unpredictable times to report and receive the latest instructions. A major downside is the master being visible to the network and thus vulnerable to attack.

The risk of exposing the master as a server to the Internet became evident during

the writing of this thesis when one of the leading configuration management systems, Salt, had critical vulnerabilities (CVE-2020-11652, 2020, 2020). They allowed attackers to remotely take over the Salt master (and thus all agents) over the Internet. These attacks would have been eliminated by the use of the Hidden Master architecture I published with Li three years earlier (Karvinen and Li, 2017). Salt now recommends limiting access to ports required by agents, not exposing Salt master to the Internet and using a VPN (SaltStack Inc., 2022). These steps seem to be a sensible mitigation to the serious risk posed (and realized) by exposing a machine with the master's secret keys to the Internet, but limit the usefulness of the system in case of moving agents. The remaining fundamental problem still remains that compromising any agent allows direct attacks to the master through the VPN. The attacker would be wise to choose a time when active monitoring and swift response are not available.

One mitigation for the many challenges of push architecture could be using a virtual private network, such as Wireguard or OpenVPN. In addition to making the system more complicated to manage and analyze, it does not solve all challenges of push architecture. Many non-server nodes are not on all the time. When the lid of a laptop is closed, it cannot receive instructions from the master, and nor can it send its report back. This would require a human operator to run pushes at various times, hoping to catch all nodes. In some cases, reports from nodes could end up being very old. For push architecture, running inside a VPN does not remove the need for round the day, round the year incident response capability. Once an attacker has compromised any agent, he can use the VPN to attack the master. A smart attacker can make the attack at an inconvenient time, such as 4 am on Christmas Eve.

The Hidden Master architecture removes categories of attacks by reducing the attack surface. A key feature is protecting the master secret key by keeping it in a computer that is never contacted by agents. The intermediate machines contacted by agents are untrusted. This allows the use of regular web servers. Because the Hidden Master architecture is asynchronous, other network protocols can be used too. This work has experimented with air gapped file transfers and peer-to-peer operations without stable network infrastructure. Essentially, any untrusted method of transferring files could be used as a network backend.

The attack tree above shows how the Hidden Master architecture completely removes the possibility to directly attack the master. Unlike the use of industry leading solutions, the master cannot be attacked through the configuration management system control channel either. If we start with a sensible expectation that an attacker can compromise a single agent node, the attack that is

left is injecting malicious code into the report. Even if reports are not supposed to run code, an attacker could try to find a vulnerability in libraries using the reports. If such extreme security would be needed, the Hidden Master architecture allows for the use of two different computers to send instructions and receive reports, and some limited testing for this has been performed with the prototype.

Hidden Master architecture makes it impossible to attack the master when a response is not available. This is in stark contrast with those architectures described in the literature reviewed in this work and in industry leading solutions. When the master computer is closed and shut down, it cannot be attacked over the network. This means that all attacks against the master must happen when the operator is present and actively engaged with the system. As the encryption layer is based on OpenPGP standard, it is possible to further secure the system by storing the secret keys in a hardware security module carried by the operator. This was not implemented in the prototype.

Even when the Hidden Master is turned off, the network can keep operating. The agents will receive the latest instructions the next time they synchronize against a courier/drop node, and at the same time they provide their report. Due to campaign keys, even new nodes can be securely added while master is away. For example, IoT device initial image can contain an agent campaign key that allows agents to receive the latest campaign instructions. The design of the Hidden Master architecture is in "Designing and Implementing Hidden Master Architecture Prototype".

Potential business benefits of the Hidden Master architecture are identified and evaluated in "Interviews to Evaluate Business Benefits". Reduced attack surface might lead to reduced risk, simpler and lower cost operation, low cost geographic and inter-operator scaling.

4.5 Concept of Use

Use cases for HM are similar to use cases of CM in general. An attempt is made to allow scaling from beginner to a very large scale. Use cases and their challenges are listed in table 25.

This example workflow omits many details and alternatives to provide an overview of the concept of use.

1. Administrator writes configuration in established CM tool (e.g. Salt).
2. Administrator executes command to roll out instructions to slaves. The catalog of instructions is automatically encrypted, signed, and uploaded to untrusted web servers. Only this step requires connection to the Internet,

and the administrator can immediately disconnect if he wants.

3. Slaves periodically connect to mules. They download, decrypt, verify and apply catalogs.

Table 25: Challenges of different use cases and example networks.

Use case	Network	Challenges
Hobbyist	server, laptop, desktop	Securing master can be cost prohibitive. Double NAT bypass
Data center	1000 servers	
Enterprise	500 laptops, 500 desktops, 50 servers	Lot of computers. NAT bypass.
Huge	40k servers	

4.6 Tradeoff analysis

4.6.1 Timely vs Timeless

Timely communication between master and slaves is common, but produces multiple problems regarding the resiliency of CM. Most leading CM systems and scholarly articles imply a requirement for slaves to connect directly to the master. This is discussed in "Implying Direct Master-Slave Connection". This requirement obviously means that master and slave must be on the network at the same time, making the master vulnerable to attacks over the internet. If the master is running continuously, these attacks can be performed at a time when a human the response is less likely, such as during the night or holidays. The master also creates a single point of failure to the network. Some systems collect data with a high rate and minimal latency, but averaging data on display and human initiated queries forfeits this advantage. Dropping timeliness could reduce these problems, making CM more resilient. Such a system could be called timeless.

The OODA loop (observe-orient-decide-act), originally developed by Boyd of the US Air Force, is a model to gain situational awareness, take initiative and ultimately achieve victory in adversarial situations (Lenders, Tanner and Blarer, 2015). Situational awareness is not simply speed of observation and acting. In his seminal paper, Endsley (1988) categorizes situational awareness into three levels, where the highest level allows projections into the future. Similarly in the cyber domain, situational awareness is not simply about individual cyber events but how they improve overall understanding of the situation (Franke and Brynielsson, 2014; Lenders, Tanner and Blarer, 2015).

Timely communication reduces the reaction time to both natural and human made disruptions. To gain this benefit, there should be a human to perform these activities. Small and medium organizations might find it hard to justify the cost of having a professional security response team available for 24/7 response. This also gives economic incentives to concentrate on the higher level situational awareness instead of simply attempting to speed up the OODA loop.

To improve the resilience of these systems, the hidden master architecture chooses timelessness over timeliness. The master does not need to be available on the network at all times, and the instructions are carried to slaves on a best effort basis. When no new instructions can be received, slaves should keep operating indefinitely following the last instructions received.

Future versions of hidden master architecture could combine traditional direct, timely communication with more resilient timeless, indirect communication. In future development, a Conftero native back channel could provide feedback at the same rate as agents poll instructions in approximate intervals of 15 to 30 minutes. Direct communication could be used in situations where it provides the highest benefits: development and when a timely human response is available. Some practical computer systems already provide the ability for consolidated logging. Conftero events could thus be placed into regular logs (e.g. syslog, Event log) and shipped with other logs. In this way, any events produced by Conftero would be available in the same enrichment, search and summarization platform with other logs.

4.6.2 Encryption Method

Computationally secure, strong algorithms are widely available and included in the libraries of common programming languages. Unfortunately, many practical cryptographic systems are broken by attacking the implementation instead of mathematically breaking the encryption method. In a study on Middle East malware phylogeny, Moubarak, Chamoun and Filiol (2017) found that even malware attributed to nation states had made serious security mistakes when implementing command and control for their malware. When designing secure communication for a configuration management system, the encryption should withstand attacks for a very long time. This is in contrast with cyber attackers, who could instead reach their goals in a short time frame and then remove their malware. In some cases, attackers could allow the attack to be discovered in order to gain publicity.

4.6.3 Back Channel

Slaves might need to send information back to the master. In this work, the system for sending this information from slaves to the master is called the back channel.

Efficient control of slaves requires the information of slave states. In practice, this task is often done with multiple systems simultaneously. Multiple categories of software products make a principle task of gathering information about slaves and consolidating this for use by the system operators.

Monitoring and alerting systems watch the state of each slave node, combining external and internal checks. Examples of such systems are Nagios, Cacti and Zabbix. The simplest case of monitoring is external monitoring, such as simulating a client browsing the web by sending an HTTP request from a single vantage point. Other functionality requires implementing a secure channel between the slaves and the master. Watching the internal state of slaves, such as the load level, disk usage or use of swap space obviously requires secure access to slaves. But external monitoring also requires secured connections when requests need to be undertaken from multiple vantage points. For example, sending an HTTP request from one slave to another requires secure access to the slave sending the request.

Log consolidation systems, such as Elasticsearch-Logstash-Kibana (ELK), rsyslogd or Graylog, retrieve log events from multiple slaves and put them into a single database. Administrators can then perform queries or visualize this data to gain situational awareness of their network. System logs typically contain confidential data. Protecting personally identifiable information (PII) is not only ethical, but also often required by regulations, such as the European Union's general data protection regulation (GDPR). Log events can contain data that make it easier for attackers to compromise the system, such as errors, misconfigurations and specific software versions. Even though it is undesirable, system logs might contain plain text passwords, as users sometimes type their password in place of the user name; or a failed database connection includes a password in the quoted source code line. For these reasons, log consolidation systems require secure communication channels.

When monitoring and alerting solutions and when log consolidation systems implement their own channel to slaves over network, they need to address most of the same challenges that are met when implementing a CMS: strong encryption, two-way authentication and key management.

When simply reading the system state, the requirement is not as high as with CM. As CM has the highest privileges (root, Administrator, SYSTEM) on

the machine, a compromise of the CM control channel results in a complete system compromise. In contrast, a compromised monitoring or log consolidation channel does not need to result in a completely compromised system, as the monitoring agent could be given read only permissions on slave. As shown above, it is still high enough to mandate strong security measures. Implementing multiple channels makes the system architecture more complicated and adds new possible points of failure. Security requirements for back channels are collected to table 26.

Table 26: Back channel security requirements.

Goal	Examples	Security required
External check	Nagios HTTP request	Low
Alternative point	HTTP request from slave	Medium
Internal check	Zabbix CPU utilization	Medium
Log consolidation	rsyslogd syslog collection	High
CM collect results	Salt collects grains	High
CM modify slave	Puppet installs Apache	Highest for slave

Any practical system could easily end up with multiple control channels, each of which implements encryption, two-way authentication and key management. Initially, this might provide redundancy in case of a failed control channel. An operator could use SSH to fix the broken CM slave. But redundant systems create an additional attack surface, complicate the system and create a management burden.

4.6.4 Back Channel in the Hidden Master Architecture

A back channel would create some unique challenges in the hidden master. On one hand, a back channel would be useful to maintain situational awareness and see how far the actual system state is from the idempotent goal administrators have set with configuration management. On the other hand, slaves cannot simply send their results directly to the master, as a hidden master is not required to be present on the network at any specific time.

As the Hidden Master architecture already implements secure, asynchronous channel from the master to slaves, similar techniques can be used for implementing the back channel. To make the back channel mirror the architecture of a regular forward channel, slaves should send their reports to mules.

An alternative to a Hidden Master specific back channel would be picking and using an existing log monitoring or log consolidation system, writing slave

daemon events to local logs. The master could then read these logs from wherever they are consolidated. This would allow use of an existing tested solution for the back channel. But this would likely create a single point of failure in relation to the back channel, the very thing the Hidden Master architecture tries to avoid on the forward channel from master to slaves.

When the master is communicating to slaves using web servers, it uses web servers in the simplest and the most obvious way: one authenticated computer uploads files, and multiple unauthenticated computers download files. A back channel places an additional requirement on mules: they should accept file uploads from web clients. Even though this requirement is not uncommon for web services, it requires dynamic capabilities from the web server. A simple file upload form handler could be written in PHP or Python Flask.

One of the key features of the Hidden Master architecture is geographical, organizational and network dispersion of the critical control infrastructure with low costs. To keep this feature, only some intermediate nodes are required to implement the file upload required for the back channel. This way, static-only file sharing hosts can still be kept for downstream, master to agents communication. Downstream-only hosts have less functionality, and thus smaller attack surface. Using third party cloud services for distributing configuration is much easier when these hosts are used for downstream-only file sharing.

4.7 Layer Model of the Hidden Master Architecture

A layer model of HM architecture allows the separation of concerns, the pointing out of components and their responsibilities, and the defining of interfaces in the system. The interfaces enable components to be swapped without affecting other components or the overall structure of the system. Different testing scenarios might require the transfer layer to be swapped from HTTP to P2P while maintaining the same method for idemponently defining the target state of the system. Clearly defined architecture makes it possible to augment the system with additional components. For example, a physically secure key storage could be added to the master without any modifications to slaves or mules. Or slaves and mules could implement domain generation algorithms with minimal changes outside the transfer layer.

The key feature of the hidden master architecture is protection of the master secret key. Thus, all versions of this architecture must be able to configure slaves and keep distributing encrypted catalogs without the presence of the hidden master.

The architecture is divided into three layers: transfer, encryption and configu-

ration. Each layer provides the same function across all nodes in the network. Each layer can only communicate with layers directly above or below it. Only the transfer layer can communicate across nodes. Layer responsibilities are collected to table 27.

Table 27: Responsibilities of layers

Layer	Responsibility
Configuration	Defines and applies slave configuration.
Encryption	Encrypts, signs and verifies catalogs.
Transfer	Copies encrypted catalogs across nodes

Based on these responsibilities, the properties or qualities of each layer emerges. Layer properties can be found in table 28. In the configuration layer, instructions to slaves are defined (in master) and applied (in slaves). The configuration layer could either work standalone, or be integrated with an existing tool such as Salt, Puppet, Chef, Ansible or CFengine. The way for defining a slave configuration does not affect any other layer. As a trip to each slave can be prohibitively expensive, multiple methods could be implemented, with a modern and convenient idempotent system as the default system with a trivial command based system used as a fallback.

THe encryption layer is responsible for protecting the instructions (the encrypted catalog) while in transit. Using the principle of end-to-end encryption, the catalog is encrypted on the hidden master and only decrypted on the target slave. Thus, the encryption must protect against network eavesdropping (compromised mules) as the whole network path between master and slave becomes viewed as untrusted. Depending on the threat model, the mules could be partially compromised to begin with. To allow low cost network and geographical decentralization, the hidden master architecture uses low cost virtual private servers and even cheap web hosting. Not only are the operators of these systems not trusted, but also other clients on the same physical hardware could be able to break out of virtualization and read the memory of other users using published attacks against hardware vulnerabilities such as Rowhammer.

Table 28: Qualities of layers

Layer	Network	Trusted	Outside CM
Configuration	no	yes	yes
Encryption	no	yes	no

Layer	Network	Trusted	Outside CM
Transfer	yes	no	no

4.7.0.1 Subsystems A subsystem is a self-contained part of the system, serving an outside purpose. Other subsystems could consider it a black box that just implements the required interfaces. Each subsystem could be implemented with different programming language or technology. A high level language such as Python makes it easy to integrate the existing CM when it is written in the same language. A low level language, such as C or Go would make it possible to compile slave daemons with low resource consumption and - with static linking - minimal dependencies.

HM subsystems are the hidden master, mule and slave. Subsystems and their responsibilities are listed in table 29. The master subsystem creates and uploads encrypted catalogs to slaves. In the initial implementation, there is a single master. In the future, there could be multiple masters, either providing separate sets of instructions or coordinating with other masters, e.g. using a version control system. The mule subsystem consists of untrusted computers, e.g. cheap virtual private servers, hosted file services or static web hosting. The mule subsystem is responsible for passively receiving the encrypted catalogs from the hidden master, and keeping them available to slaves. Implicitly, mules allow slaves to be controlled when the master is disconnected, and provide geographic and network diversity for resilience. Because mules are not trusted, it is important that they do not contain any keys. The slave subsystem consists of all the slaves. The slave subsystem is responsible for downloading, decrypting, verifying and applying the encrypted catalogs.

Table 29: Subsystems

Subsystem	Responsibilities
Master	Create catalog; Encrypt and sign; Upload to mules
Mule	Receive encrypted catalogs; Serve encrypted catalogs
Slave	Download, decrypt, verify and apply encrypted catalog

4.7.1 Components

Components are tightly coupled parts of the system, that have limited use of their own. A matrix in table 30 of subsystems and layers shows the initial components.

Table 30: Matrix of subsystems and components

Layers/Subsystems	Master	Mule	Slave
Configuration	Define configuration	n/a	Apply configuration
Encryption	Encrypt & sign	n/a	Decrypt & verify
Transfer	Upload to mule	Serve enc. catalog	Download

The main components of HM are in the intersections of layers and subsystems. As the initial HM implementation is only concerned with configuration and not monitoring or log consolidation, the traffic only moves one way. Thus, there are seven components that are executed in the same order, as shown in table 31.

Table 31: Flow from master to slave.

Step	Subsystem	Action
1.	Master	Define configuration
2.	Master	Encrypt & sign
3.	Master	Upload to mule
4.	Mule	Serve encrypted catalog
5.	Slave	Download catalog
6.	Slave	Decrypt & verify encrypted catalog
7.	Slave	Apply configuration

4.7.2 Interfaces

"Beginners look at the boxes, professionals look at lines."

The main interfaces are formed between the interacting subsystems. They are

- Master-Mule interface
- Mule-Slave interface

Data transferred in both of these interfaces is the encrypted catalog. The whole Transfer layer can consider the encrypted catalog as a black box file. The encrypted catalog is a single PGP encrypted file whose name is derived from slave public key ID. Each set of instructions to each slave creates a separate encrypted catalog.

The minor interfaces are formed between each of the components. As the data flows one way from master to slave, we can examine these interfaces as the outputs of each component. This output is the input of the next component.

1. Master: Define configuration: Single compressed file with configuration. E.g. File overlay, Shell script, Puppet catalog or Salt-SSH tarball.
2. Master: Encrypt & sign: Single encrypted and signed file whose name is derived from slave public key ID. The file is encrypted according to PGP standard using, for example, gnupg or libraries for the chosen programming language. E.g. A9F4DEADBEEF.pgp
3. Master: Upload to mule. File transfer suitable for the mule. E.g. scp, rsync or ftp upload. FTP upload is possible, because the confidentiality and integrity of data is protected by the encryption layer.
4. Mule: Serve encrypted catalog. Mules should be chosen by price, availability and distribution. The simplest way of serving will work, such as static files on a web server or an FTP server.
5. Slave: Download catalog. Using a method suitable for chosen mule. E.g. regular HTTP or FTP download, using wget or Python requests.
6. Slave: Decrypt & verify encrypted catalog. Decryption is done with slave secret key, verification is done with master public key. As the encryption scheme uses PGP standard, it can be decrypted with different software than that which encrypted it. Verification is done using the master public key. If verification fails, the catalog is discarded and the failure recovery process started.
7. Slave: Apply configuration. A method suitable for the configuration format is applied. Both the master and slave use the same system for configuration, and HM does not implement every possible configuration method.

Now that each step of the flow from master to slave is detailed, it is evident what data is being transferred. The actions taken and the data being passed indicate suitable programming libraries in each state. These are detailed in table 32.

Table 32: Component data and library requirements

Component	Library feature examples	Data (in addition to output from previous component)
Master: Configuration	Salt, zip	List of slaves. User supplied configuration.
Master: Encrypt & Sign	pgp	Master secret key, slave public key
Master: Upload	scp	Mule addresses
Mule: Serve	Apache	-
Slave: Download	wget	Mule addresses

Component	Library feature examples	Data (in addition to output from previous component)
Slave: Decrypt & Verify	pgp	Slave secret key, master public key
Slave: Apply configuration	Salt, unzip	-

4.8 Key Management

Trust between nodes in configuration management system is established and maintained using asymmetric, public key cryptography. Thus, managing and verifying encryption keys is central to secure operation of configuration management system. Establishing trust between nodes without leaving a gap where keys are accepted without verification is a challenge also tackled in this chapter.

In configuration management system, acceptance of master public key signifies full control of slave node. This is in contrast with communication systems, where trust in key usually means trusting that the key refers to correct identity. Due to the different meanings of trust, the keys must be kept in a separate key ring from email and general purpose keys, or the trusted master key ID should be saved separately. To prevent unnecessary coupling of software components, the implementation of the Hidden Master architecture will use a separate key ring.

In master-slave architecture, the master sends commands to slave computers. Thus, the master needs a public key for each slave. Slaves must verify that the instructions really come from their master. Thus, all slaves need the master's public key. Mules, the intermediate hosts, are untrusted, so they must not have any keys. Key requirements in each operation in downstream flow (master to agent) is detailed in table 33.

Table 33: Use of keys in the Hidden Master Architecture downstream flow

Operation	Keys required
Slave: decrypt encrypted catalog	Slave's secret key
Slave: verify encrypted catalog	Master's public key
Mule: not allowed to read catalogs	No keys
Master: encrypt catalogs to slaves	Each slaves public key
Master: sign catalog	Master's secret key

If implementations to HM want to monitor slaves, collect or exfiltrate data, a back channel should be implemented. With a back channel, slaves use the master's public key to encrypt messages that the master decrypts with its secret key. Thus, the back channel does not require any new keys. The master is ultimately trusted in the HM architecture. Because the master can eventually obtain all data on all slaves, it does not matter if the master has keys it does not need, such as slave private keys. At the start of the project, implementation of the backchannel was not planned. When two case studies were found to benefit from a backchannel, it was implemented and successfully tested.

There might be a requirement to configure large and an initially undefined number of new slaves while the master is away from the network. For this purpose, campaign key pairs could be generated. With campaign keys, multiple slaves retrieve and apply catalogs encrypted with a single public key. When communication with these slaves continues, each slave could later have its own key in addition to a campaign key. Originally, it was planned to leave the campaign keys for future research. However, this feature emerged as beneficial and even required for case organizations, so it was designed and implemented.

Having the correct keys in correct parts of the system is required for secure operation. Key management is a key part of initial installation. Key management plan for downstream flow (master to agent) is in table 34.

Table 34: Key management plan for downstream flow

Node	Public keys	Secret keys
Master	Each slaves'	Master's
Mule	None	None
Slave	Master	Slave

4.9 Initial installation

The HM initial installation consists of steps after which it is no longer necessary to access slaves outside the HM to make them retrieve instructions. After the initial installation, the first encrypted catalog processing is the same as the continuous operation.

The initial installation is complete when the slave nodes periodically attempt to check one or more addresses for encrypted catalogs and possess the keys to decrypt those catalogs. Periodically checking the addresses requires a list of these addresses and a process with persistence. In the advanced prototype, persistence was achieved by scheduling the task with cron or systemd.

Initially installing the the system components can provide a dilemma: it would be easy to install HM using HM, which is, by definition, not yet installed. It is assumed that HM is installed in the same way as other configuration management tools, either as the last step of the operating system installation, using a previous configuration managment tool or manually.

Initial installation completion requirements:

- Slave periodically attempts to download instructions
  - Persistence
  - List of mule addresses
- Slave can decrypt and verify the catalogs
  - Master public key
  - Slave secret key

The sequence of initial installation is presented in fig. 4. The master is represented by m, courier and drop are on the same computer and are presented by c, and the single agent is presented by a. Depending on the threat model, there are multiple ways to deploy the master public key K(m,pub) with agents. Here, it is deployed through an alternative secure channel outside the hidden master architecture. Such a channel should be chosen according to environment and requirements. The deployment could be done during system provisioning, through ssh, by using an existing configuration management tool or by copying the file from a USB flash drive. Agent key K(a1,pub) is generated by the agent, and delivered back to the master with a regular Conftero backchannel report.

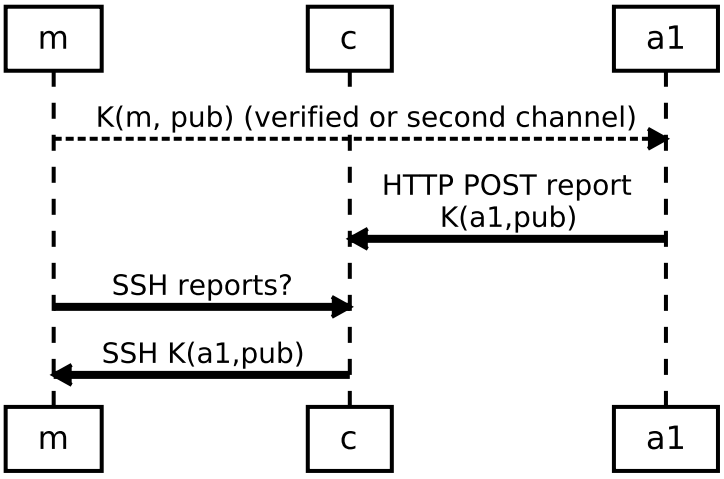

Figure 4: Initial key exchange sequence diagram

Persistence means that the slave daemon keeps running and is automatically started after reboots and other interruptions. Typically, leading CM tools register service manager scripts to get automatically started and then continuously keep running the slave daemon process. For example, Puppet and Salt on

Ubuntu Linux use Systemd to start a continuously running process. CFEngine sometimes uses a combination of continuously running process, service manager and a cron job (scheduled task) with each of those automatically repairing other methods if they fail. To periodically run a program, cron job would appear the obvious choice for this purpose. However, on large networks masters' resources could be overwhelmed if all slaves contacted them at the same time.

In malware, persistence is very common goal. MITRE ATT&CK lists multiple methods attackers use for obtaining persistence.

An initial list of mule addresses can be provided with the HM slave daemon binary. It can be either a separate file or bundled in the binary, depending on the implementation language. The first HM prototype implementation uses regular, static web servers as mules. Thus, these addresses are simply a list of http and https URLs.

Code: Mule addresses

```
http://example.com/somemanifests
https://lasermonkey234.info/lightningbanana/
```

Encrypted catalogs are designed to be useless without keys to decrypt them. To decrypt the catalog, a slave secret key is required on the slave; and a matching slave public key is required on the master. These keys can be generated on either the master or slave. If these keys are generated on the slave, the slave public key could be transferred over an untrusted network such as the Internet if the master could verify it. If keys are generated on the master, the slave secret key must be delivered to the slave through a protected channel.

As CM has unlimited privileges on slaves, it is also critical to verify catalog signatures before applying them.

4.9.1 Campaign keys

Campaign keys can solve the challenge of installing and configuring an unplanned number of future agent nodes. Such a requirement can arise when multiple devices (workstations, servers, laptops, IoT devices) must be delivered to internal or external customers; or when a cloud system rents and initializes multiple virtual servers in response to a rising load. Provisioning can be performed by using an initial hard disk image or running a scripted installation. Provisioning can make the agents phone home on first boot, but then the authenticity of the agent must still be verified - a tedious and error prone job for a human. This was the situation in the case organization studied here.

The campaign key is the initial public key *K(cam, pub)* installed during provi-

sioning. It allows agent *a1* to decrypt, verify and apply instructions meant for the group of agents it belongs to. To report back to the master *m* (through courier / drop intermediate nodes *c*), it uses the public key of the master *K(m, pub)* also installed during provisioning. As the HM agent first boots up, it generates agent keypair *K(a1,sec)* and *K(a1,pub)* and includes it in the first report to the master, thus allowing the master to send instructions to individual hosts.

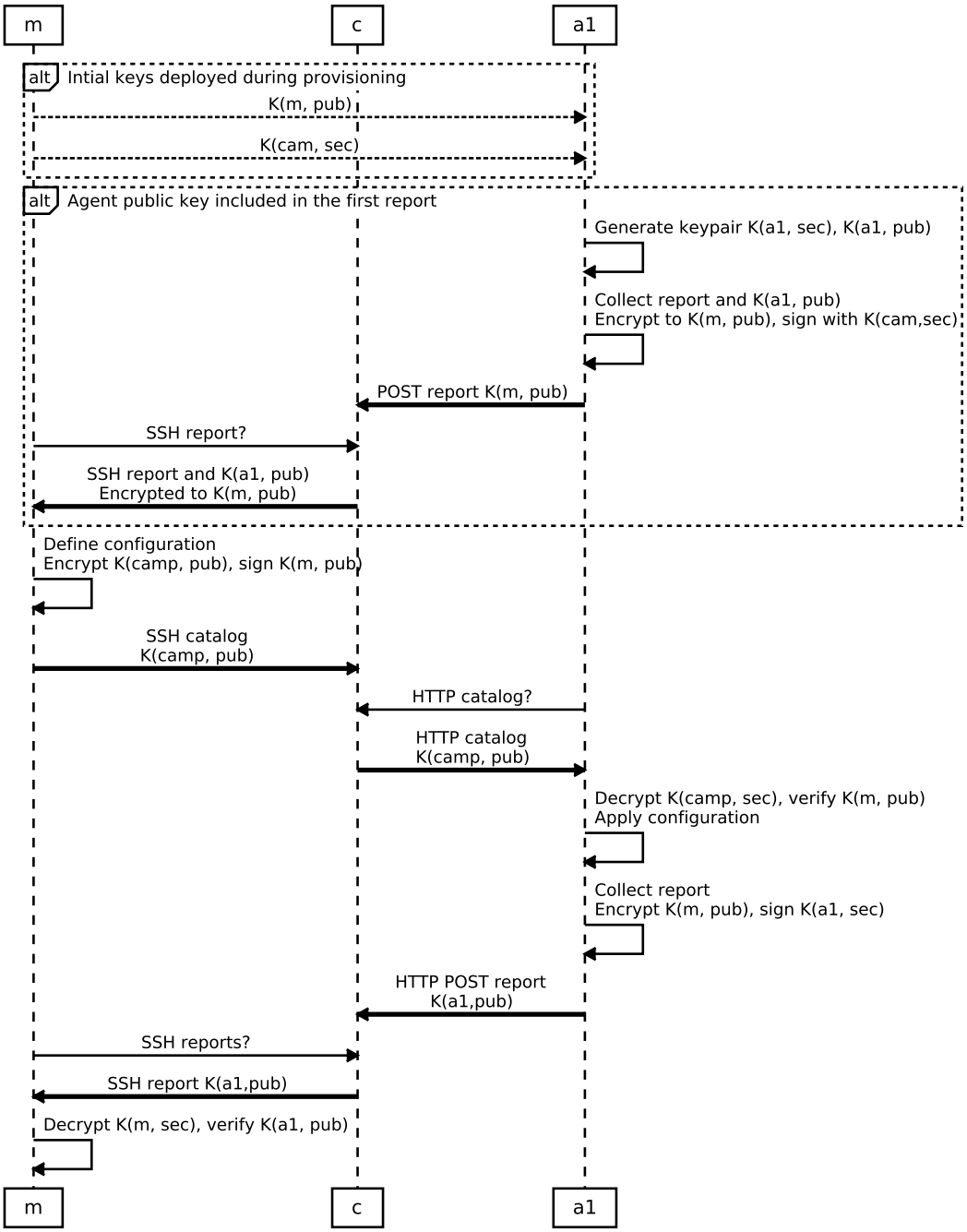

Figure 5: Key exchange for campaign keys

The full key exchange including the campaign key is shown in fig. 5. The initial key exchange is shown as two dashed rectangles. As the Hidden Master architecture is asynchronous, these initial steps might be spread over considerable

time or occur within a minute.

For the initial installation, master *m* generates an agent binary for this campaign *camp*. This step is shown as the first dashed rectangle - "Initial keys deployed during provisioning" - in fig. 5. This binary contains the master public key *K(m,pub)* and campaign key *K(cam, sec)*. This binary and keys are deployed to an agent using a secure secondary channel, such as during the operating system installation using a provisioning script or initial image. If agents are already provisioned but not yet controlled by this system, SSH can be used as the secondary channel if agents are reachable over the network.

It should be noted that even if master *m* or any of the courier/drop nodes *c* never connect to the network at this point, agent *a1* will still apply the version of configuration that was available at the time of building the agent binary.

The individual public key for agent *a1* is generated and reported on the first boot. This is shown in the second dashed rectangle - "Agent public key included in first report" - in fig. 5. As the agent *a1* first boots, it generates its own keypair *K(a1,sec)* and *K(a1,pub)*. As with every run of the agent, it applies configuration and generates a report. If configuration for camp is available at the time of generating an agent binary used here, this configuration is applied. To allow master m to securely control individual agents, the agent's public key *K(a1,pub)* is encrypted to the master public key *K(m,pub)*.

Public key of the agent and the whole report is signed with campaign's secret key *K(cam,sec)*. This frees operators from manually checking key fingerprints when accepting new agents, and the agents' reports can be accepted automatically. Future versions of the advanced prototype could sign the catalogs with all available keys, including the agent secret key *K(a1,sec)* and all campaign secret keys.

Normal operation starts after the initial key exchange is finished. In fig. 5, the normal operation is shown below the two dashed boxes. As all network operation (i.e. all operations in the Transfer layer) in HM is asynchronous, these steps can happen very quickly (within a minute) or take considerable time (e.g. a week). As implied by the name of the Hidden Master architecture, the master is not expected to be constantly present in the network.

Usually, the master is expected to perform sync operation which executes catalog encryption, signing, uploading; and report download, decryption and verification in seemingly one step for the user. This single sync command uses different keys for each distribution, that is, a separate key for each campaign and each agent. This prevents an adversary able to compromise a single agent from gaining access to the configuration of other campaigns and agents. Use

of the single sync command also minimizes exposure if security requirements demand the highest protection for the master. Here, each step of operation is examined separately. As the HM is an asynchronous architecture, these steps can happen at vastly different times for the agents and master; or during a short time frame.

At a whim, the operator can define configuration on the master. As all configuration management systems considered in this work define infrastructure as code (IaC), configuration is defined by typing code in the text files. If some nodes are already synced (sent a report through the Transport layer), configuration can be guided by the data in the reports. By using campaign keys, it is also possible to define configuration for hosts that do not yet exist. The master does not need to be connected to the Internet for defining a configuration or for analyzing the reports.

Master *m1* encrypts the catalog to each campaign's public key *K(camp,pub)* and signs it with its own secret key *K(m,sec)*. It then uploads this catalog to the intermediate server *c*. The server does not contain any encryption keys and does not participate in the encryption layer. It is expected that the adversary is able to compromise some of these intermediary hosts, as they must be accessible to the Internet or at least a large part of it. In case of rented cloud space, the owner of the cloud has access to all data in that cloud even without an attack. Making the Internet accessible hosts untrusted and especially moving master secret keys away from them offers a major security benefit over systems that do not make this separation.

File based PGP encryption protects the information both in transit and at rest on the intermediate hosts *c*. The transport layer can implement additional encryption and obfuscation methods to create costs to adversaries, to frustrate traffic analysis and to avoid over eager network filtering. In this example, the master-courier (*m*-*c*) traffic is protected by SSH Secure Shell. The courier-agent (*a*-*c*) does not use the additional protection of transfer, but transport layer security (TLS) could be easily added here by using HTTPS transport.

If the threat model includes nation states or other well funded adversaries, it should be noted that the HTTPS transport with system built-in certificate authority (CA) list cannot be trusted. As any CA can sign any certificate, a well funded adversary can simply buy a CA, or a state can order a CA to sign a certificate for it. The Hidden Master architecture relies on the OpenPGP cryptosystem, and uses other cryptography only as a non-essential addition as needed.

At predefined intervals, e.g. every 15 minutes, an agent *a1* fetches a new

configuration, decrypts it using its own private key or any of its campaign keys. Here, the campaign key *K(camp,sec)* is used for decrypting the catalog of encrypted configuration. Catalogs are always verified with the master's public key *K(m,pub)*. Additionally, catalogs contain a monotonously growing *counter*. Verification step checks that the catalog version is newer than the one installed. Catalogs failing verification are discarded.

A verified catalog is then applied on the agent *a1*. Novel contributions, design and implementation to defining and applying configuration are described in detail in chapter "Defining Configuration". The report of the result and the system state is generated. Agent *a1* encrypts this report for the master. Encryption is always performed using the master's public key K(m,pub). The catalog is signed using the agent *a1* secret key *K(a1,sec)*. Catalogs could also be encrypted with secret keys of the campaigns a1 participates in, such as *K(camp,sec)* here, and then verified on the master *m*. This additional campaign key signing is not implemented in the current prototypes.

The encrypted report is uploaded to the intermediate node *c*. The current main implementation uses PHP script with HTTP POST request. This implementation is expected to be deployable in many simple and cheap hosts and hosting services. As described in chapter "P2P Operation in Shattered Network", transports can be swapped without modifications to other HM layers. Again, as *c* does not possess any keys, it cannot decrypt or modify the messages, and does not need to be trusted.

Finally, the master downloads the encrypted reports using SSH. As the HM is asynchronous, this can happen at the whim of the operator. As the encrypted reports are already stored on the intermediate host *c*, the master *m* does not need to be on the network at the time that the agent node *a1* is connected or even running. This feature was used in the smaller case study where the reports analyzed for agents (and the OSes running them) were already deleted.

The master decrypts and verifies the reports. All reports are decrypted using the master secret key *K(m,sec)*. Signatures are verified using the agent key *K(a1,pub)*. Decrypted reports are stored on the master *m*, and can be analyzed without connection to the Internet or the intermediate host *c*.

4.10 Pseudo Code of Master and Agent Operation

Agent (slave)

- Download: Download any new files on the intermediate hosts
- Verify: Files whose cryptographic signatures verify against a set of trusted keys are processed further, while others are quarantined or ignored

- Apply: Verified files are extracted, and another CMS is used for applying the catalogs

Master

- Compile: target state of slaves is described using another CMS, and catalogs are compiled to each category of slave.
- Encrypt: each slave catalog is encrypted using the trusted private key of each slave.
- Publish: The encrypted catalogs are uploaded to the untrusted intermediate hosts.

During the master compilation stage, depending on the other CMS, it might require some setup if identical slave catalogs are to be combined.

To provide large files for slave nodes, the system could also add the dropping of external files to a category of slaves.

4.11 Sequence of Messages in Transfer Layer

The exchange of encrypted messages in the Hidden Master Transfer layer is presented in fig. 6. HM architecture uses end-to-end encryption of both catalogs (instructions to agents) and reports (from agents to master). Encryption is the responsibility of the Encryption layer, and the implementation in the research prototype uses PGP. Thus, the messages are encrypted both in transit and at rest in the intermediate courier / drop. The transfer layer may provide its own encryption, which is not depended upon. Here, master-courier communication uses SSH encryption, and courier-agent communication does not use additional encryption. Obviously, such encryption would be simple to add by using HTTPS (HTTP over TLS).

It should be noted that courier / drop (c) only exists in the Transfer layer. It does not possess any keys, and even a compromised courier has no access to the content of catalogs or reports. It also cannot impersonate other parties: neither master to agents nor agents to master.

fig. 6 shows a key difference to a leading CM, such as Salt or Puppet. The hidden master model (and the research prototype) operate without direct connection between master and agents. In fact, agents and master do not have to be connected to the network at the same time.

4.12 Defining Configuration

Modern configuration management is idempotent. This means that the target state of the system is described, and a CM tool only makes changes if the

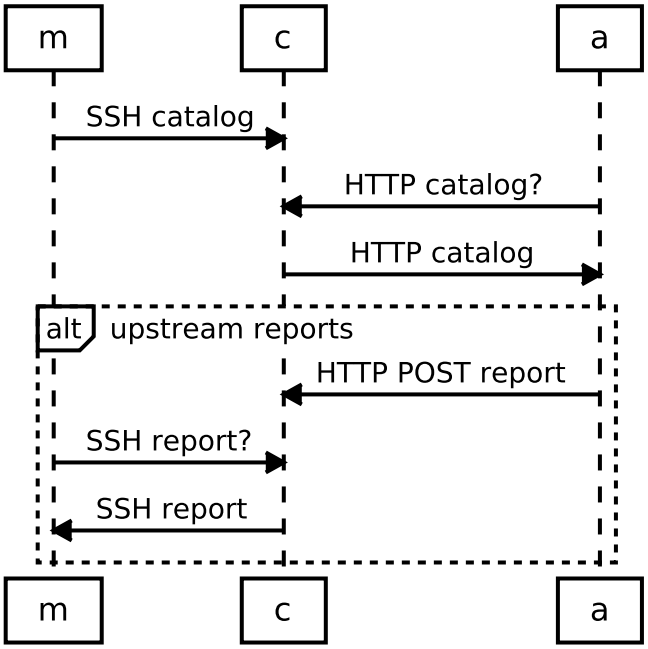

Figure 6: Encrypted message transfer in the Hidden Master Transfer layer

current, actual state differs from the target. If the CM tool is run periodically, it is expected that controlled systems end up in a stable state where changes are no longer needed.

In contrast to regular imperative programming, declarative programs are not run in a specific order. Instead, the order of actions to take is defined arbitrarily, unless dependencies between items require otherwise. Most general purpose programming languages are imperative, such as Python, Go, C and Java. Many document definition languages are declarative, such as HTML or Markdown.

Some researchers, such as Wurster et al. (2020), consider the concepts of declarative and idempotent to be equal. According to Wurster et al. (2020), the declarative approach is widely accepted in both industry and research, and used by many tools such as Chef, Puppet, AWS CloudFormation, Terraform and Kubernetes.

Leading configuration management tools expose their resources to users as domain specific language, DSL. Puppet uses its own declarative, non-imperative language in text files called manifests. Depending on settings, Puppet manifests are not necessarily evaluated top-down. Salt uses its own YAML (yet another markup language) based DSL in combination with code generation using the Jinja2 template engine. In addition to this main language, Salt provides multiple alternative languages for defining configuration. Even though some of these extra languages are Python, they use less than obvious control structures to define dependencies in the same way as the main DSL. Ansible provides yet another DSL of its own. As these DSL are not used anywhere else, it could be

assumed that there is little transfer effect even for experienced programmers.

In practice, many declarative configuration DSL:s (e.g. Salt, Puppet) implement idempotency as a simple if-then. When considering the state of a resource, they first compare it to the current state (if), then run a command to modify the state if it differs from the target state. Eventually, both Salt and Puppet have added features to control the run order of declarative configuration. One of these new run orders is top-down.

When declarative configuration is run top-down, it is not far from a similar imperative program consisting of if-then statements to modify the state. These if-else checks can be further abstracted as functions, creating concise imperative programs that are still idempotent. This approach is taken by Conftero configuration language.

In their systematic mapping study of infrastructure as code (IaC) Rahman, Mahdavi-Hezaveh and Williams (2019) categorically conclude that "IaC scripts use specific language (DSL)". Shambaugh, Weiss and Guha (2016) who they cite does not express this view as categorically, but they do mention numerous configuration management tools that use a DSL. Shambaugh, Weiss and Guha (2016) do not include any examples of idempotent CM that do not have a DSL.

Shambaugh, Weiss and Guha (2016) point out three main benefits of a DSL. Administrators are already familiar with the OS, the tools, and the techniques that these DSLs control. Many types of resources can be managed, such as packages, configuration files and users. He also points out that a DSL "provide[s] relatively low-level abstractions". (Shambaugh, Weiss and Guha, 2016). As a counterpoint, one could ask if "low-level abstraction" is useful at all, as the administrator would have to learn a DSL, but still gain very little reduction in complexity. As these DSLs simply run OS tools underneath, the administrator might end up writing as complex (and long) code in an unfamiliar DSL that simply ends up running the commands he would have already known.

Kosar, Bohra and Mernik (2016) performed a systematic mapping study on 390 publications about DSLs. They concluded that measuring the effectiveness of DSL approaches is lacking, and there is little evaluation research, particularly controlled experiments (Kosar, Bohra and Mernik, 2016). This is in stark contrast to the view taken by some proponents of DSL. Mernik, Heering and Sloane (2005) claim that "[Domain specific languages] offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application".

4.12.1 Size and Complexity of Some DSLs

DSLs can become highly complicated and large. I analyzed the DSL of Salt, one of the leading configuration management tools. To estimate the complexity of the language, I collected statistics from its manual. As Salt generates this manual from its source code at runtime, it can be expected to give a good view of the actual language in use. To perform these tests, I used Salt version 2017.7.4 "Nitrogen", as it was packaged and supported on the test distribution Ubuntu Linux 18.04 LTS "bionic". Details of this test are explained in Estimating the Size of Some Domain Specific Languages and the results are collected in table 35.

Salt DSL has 510 state functions allowing the administrator to control the state of agent computers. The documentation for these states is more than 20 000 lines, over 75 000 words - longer than a typical doctoral dissertation. This does not yet include control structures, such as branching (if-else) and loops (for). This functionality is provided by using Jinja2 templates to generate YAML code, which is then converted to internal Python structures and run. Using templates to generate code creates another level of low-level abstraction for the administrator to fathom.

Puppet, another leading CM tool, provides 113 functions out of the box. This does not include control structures. Puppet control structures are very different from those used in common programming languages such as C, Go or Python. Puppet relies on defining new resources and their relationships. For example, consider a case where the administrator would like to create users so that they all get similar settings. In a common programming language, a task often repeated could be represented as a function. Instead, Puppet defines its own concept for this purpose - virtual resources.

Table 35: Qualities of some domain specific languages

Tool	Functions	Control structures
Salt	510	Generate code with Jinja2 templates
Puppet	113	Own concepts, e.g. virtual resources

4.12.2 Use of DSL Functions in Case Configuration

Considering that leading CM tools have DSL with a large number of functions and detailed rules for defining relationships between these resources, it could be assumed that building one's own resource abstraction is a large undertaking. However, some program patterns are very common when defining configuration, such as the package-file-service pattern for setting up daemons. To evaluate the

scope of this task, I had a look at some publicly available third-party production configuration using Puppet DSL. I selected the United States Government Configuration Baseline (NIST, 2016) and Mozilla Release Engineering Puppet Manifests (Mitchell et al., 2020).

For Mozilla Engineering manifests, functions and control structures were gathered by searching manifests for the left curly bracket, duplicates were removed using Linux command line tools and finally percentages (of use count) were calculated with Python. One non-command right curly bracket was removed manually.

Table 36: Most used functions (F) and control structures (C) and internally defined (I) in Mozilla Release Engineering Puppet manifests.

Share	n	Type	Commands
20%	475	C	case
19%	429	F	file
10%	234	F	package
6%	136	F	exec
3%	75	F	service
3%	71	I	fw::rules
3%	67	I	registry_value
3%	66	C	if
3%	63	I	registry::value
3%	62	C	class
2%	52	C	anchor
2%	41	I	packages::pkgdmg

table 36 contains all commands where the usage of which comprises more than 1% of uses. They cover 87% of all commands and control structures, including the ones that are defined in these manifests. By categorizing these as functions (F), control structures (C) and calls to functions defined internally (I), we can see that very few functions are actually required.

The most used functions are those one could expect based on common program patterns used by the leading CM tools. Daemons are set up with package-file-service, and client applications simply use package-file. Many functions perform a simple system call in the background, and where this does not work, the user can call exec himself. When using exec, the user is responsible for making the state idempotent. Some common functions that did not reach over 1% of those used are user (0.8%) and group (0.1%).

Control structures used include “if” and its special case “case”. The challenges of DSL development are reflected by wide use of “class”, which Puppet uses in a meaning that is not related to its regular use in programming languages. Use of “anchor” could be described as a way of handling resource (function) dependencies inherent in Puppet DSL.

Similar analysis was performed on the United States Government Configuration Baseline (USGCB). One irrelevant hit was removed where a variable value contained the search string.

Table 37: USGCB use of commands separated to functions (F), internal (I), unrelated (U) and control structures (C).

Share	n	Type	Commands
22.5%	23	F	augeas
16.7%	17	F	file
14.7%	15	F	service
14.7%	15	F	exec
12.7%	13	F	package
8.8%	9	I	pam::changeparm
2.9%	3	U	node
1.0%	1	I	umask-replace
1.0%	1	I	set-mount-options
1.0%	1	C	if
1.0%	1	F	group
1.0%	1	U	filebucket
1.0%	1	F	cron

The results in table 37 show that the most common functions included package-file-service, exec and group. File manipulation can also be performed by modifying existing files instead of replacing them, and augeas was used for this. In fact, file manipulation using augeas was the most common function. User function was not used at all, which could indicate that users are controlled by some other centralized system outside CM. Timed tasks were created with cron state. As this configuration was created for Red Hat Enterprise Linux 5 only, cron state could easily be replaced by simply creating a file in /etc/cron.d/.

This analysis shows that even though leading CM provides a respectable number of functions to manipulate the system, a small number of functions seem to cover a large number of use cases in the example production configuration researched. This result is in line with the experiences gained by the author

from multiple years of Puppet and Salt configuration management courses in Haaga-Helia UAS.

4.12.3 Conftero Definition Language

Conftero could either provide its own method for defining configuration on thle agent, or integrate with an existing solution. Benefits and challenges of each approach are in table 38. This could remove the problem of integrating against the moving target of an external configuration management system. But it could also provide new benefits. As seen in the previous chapter, a small number of configuration functions are responsible for a large share of configuration. It could be expected that a smaller language is easier to master and could even reduce cognitive load while programming. While leading CM tools define their own DSL, using a dialect of a widely known language could make it simpler to use existing knowledge of programming with CM. A full featured programming language could also provide simpler and better defined control structures, avoiding kludges like Puppets anchor or Salt's use of code generation using the Jinja2 template engine.

The downsides of implementing one's own configuration system are the required efforts and the need for users to rewrite the possible existing configuration. It also remains to be seen if the required qualities such as idempotency can be reached with a full, general purpose language.

Table 38: Possible benefits and challenges of embedding existing language

Benefits	Challenges
Reduced integration effort	Effort for creating the system
Dialect of existing language faster to learn	Existing configuration must be rewritten
Well defined semantics and control flow	Control flow not tailored for CM

4.12.3.1 Dependencies Between Main Functions

This chapter analyses the dependencies of functions (resources) from the point of the developer of the configuration management system. In Conftero, deep understanding of these concepts is not required for the user. In fact, Conftero aims to use a single changes flag to free the user from managing the dependencies, and this user perspective is handled in the next chapter.

The most common functions include package-file-service, user, group and exec.

Common use cases include daemon and app configuration. In some cases, users and groups need to be created for technical reasons, even if human users are managed outside CM. Key functions are listed in table 39.

Table 39: Key functions for configuration management

Category	Functions
Daemon setup	package, file, service
App setup	package, file
User management	user, group
File manipulation	file, directory, symlink

The system should be idempotent. This means that changes are made only if the system is not already in the correct state. As a result, a correctly configured, periodically running system does not make any changes in a typical run. Idempotency can be achieved simply by a simple list of "if"-structures. For example, existence of the file is checked, and a file is created only if it does not exist. Contents of the file are checked and if they differ from the target, they are corrected. Differences from the target are logged, which tells the administrator if the system has reached a stable state.

Most system configuration functions end up running a native command on the agent. For example, package function runs 'apt-get' on Debian Linux. If resource abstraction is implemented, the same package resource can run 'yum' package manager on Red Hat Linux. In current leading CM tools, choosing the name of the package is still left as an exercise for the administrator, usually adding an "if" or "case" at the top of the code.

File is a very common thing to manage. In Linux, almost all configuration is stored in plain text files. In Windows, many programs (but not all) store their configuration in text files too. As files have many attributes (content, mode, owner...) to manage, it makes sense to implement the file function from scratch instead of executing an external program.

Many functions end up calling a program in the agent system. Many features of the system require that only one component handles this area, such as package manager or service (daemon) management. It would not result in a stable system if daemons were restarted by killing their processes, or if files controlled by package manager were overwritten by an outside application.

The next abstraction level can be built on service, package and file. For example, managing scheduled jobs with cron can simply use file resources. This means that a large number of functions could be implemented out of a small number

of base functions, or that the user could simply use the core functionality to make programs whose function is obvious and self contained.

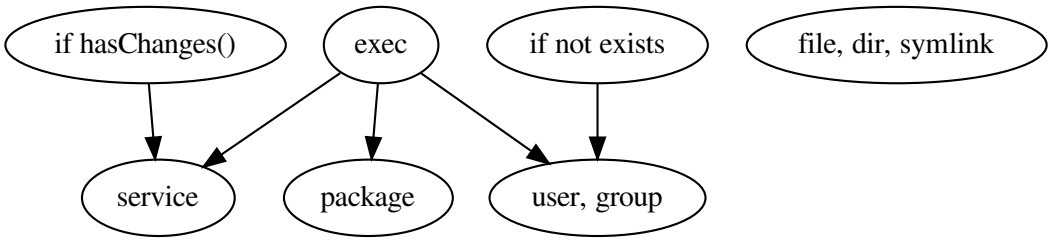

Figure 7: Dependencies in configuration management functions

A dependency graph of these features is shown in 7. The key functions are "exec" for calling other programs and "file" for creating files in the system. It is easy to make the file function idempotent with built in checks. For exec, making it idempotent is different depending on the program being called. Idempotency could check if a file created by the program exists. Some programs (e.g. 'apt-get install') are already idempotent, and any changes made could be detected from the output for logging purposes. Knowing whether changes happened is important to recognize if the system has reached a stable state.

Other important functions are created by calling file and exec. These are package-file-service, user and group. These functions can have built in checks, so the administrator does not need to write control logic himself.

From the internal dependency graph we can see that the system has very few top level functions. Only exec and file functions (file, directory, symlink) directly affect the agent system. Other resource functions can be built from those two basic facilities: package, file, service, user and group. Even higher level functions, such as cron management, could be further built using these facilities. As we can see from the actual production examples above, most configuration is built using basic facilities. As the typical workflow is to first perform an operation manually and then automate it, the administrator is likely to already know how to combine these basic facilities when the automation starts. It is also open to question if the added complexity and cognitive load negate the benefit of creating additional layers of abstraction.

4.12.3.2 Flow Control and Resource Dependencies This chapter looks at the dependencies from the point of the user of the system, i.e. the administrator who uses CM to configure agents. Conftero aims to free the user from thinking about resource (function) dependencies at all by using a single per-module changes flag instead of individual dependencies. The previous chapter handled the CM system developer's view of the dependencies inside the Conftero code.

Most daemons only load new configuration when the daemon is restarted. Thus, a restart must be performed when the configuration files change. On the other hand, services should not be restarted without a reason, as this would interfere with the proper operation of the system. If the web server is restarted, clients could be logged out and their shopping carts emptied. If SSH daemon is restarted, remote users could be logged out.

Control flow can easily become confusing when it is defined with CM DSL. This can result in the need for code generation (Salt), a need for kludgy program patterns (Puppet) or in code where it is difficult to predict the execution order. In Conftero, I attempt to create the control flow using the simple tools provided by a general purpose programming language.

Instead of a dependency graph between the resources managed, Conftero will use a top down flow with regular programming language constructs. I propose the use of a general changes flag to replace the resource dependency graph. If any of the functions end up modifying the agent system, the current code has changes and the function hasChanges() returns true. The following Python code for installing a web server and enabling user homepages is a very common example of a package-file-service pattern.

```
package("apache2")
symlink("/etc/apache2/mods-enabled/userdir.load",
        "../mods-available/userdir.load")
symlink("/etc/apache2/mods-enabled/userdir.conf",
        "../mods-available/userdir.conf")
if hasChanges():
    restartNow("apache2")
```

This code has an obvious control flow. Starting from the top, web server apache2 is installed or upgraded if needed. Configuration is modified by changing two files in /etc/. If any of these functions made any changes, hasChanges() is true, and the service is restarted, thus taking the new configuration into use.

In leading CM tools, a similar state would require creating multiple dependencies. As a single file often considers only a limited number of services, most functions end up on the dependency graph.

Let's consider a Salt state making the same web server setup as above.

```
apache2:
 pkg.installed
/etc/apache2/mods-enabled/userdir.conf:
 file.symlink:
   - target: ../mods-available/userdir.conf
```

```
/etc/apache2/mods-enabled/userdir.load:
 file.symlink:
   - target: ../mods-available/userdir.load
apache2service:
 service.running:
   - name: apache2
   - watch:
     - file: /etc/apache2/mods-enabled/userdir.conf
     - file: /etc/apache2/mods-enabled/userdir.load
     - pkg: apache2
```

Salt states (code files) are written in Salt DSL. Similar to other leading CM tools, the user must manage the dependency graph himself. As many changes require restarting the service, "watch" expressions are used for creating a dependency from service to all other functions in the state.

Salt configuration is expressed in YAML, a structured language similar to JSON and XML. YAML was originally created for storing configuration. Following about a hundred students who configure machines with YAML, common challenges seem to be both syntactical (two space indentation, colons) and semantic (which parts are lists and which parts are associative arrays). Examining the common challenges in reading and writing configuration could be an area for more detailed research.

Because user is made to manage the dependency graph, each thing must have a name for reference. Here, names include "apache2" and "apache2service". For most functions, the code is made shorter by implicitly using the name as the main parameter for the function. This feature, namevar, makes code shorter but also adds another layer of abstraction. Because each thing must have a different name, it is not possible to have service "apache2" and package "apache2". This is unfortunate, as a large portion of software packages are named after the daemon contained in them, resulting in names like "apache2service" and thus preventing the use of the namevar feature.

Another leading CM tool, Puppet, uses a different DSL. Consider this Puppet manifest implementing the same package-file-service pattern as above.

```
class apache {
    package {"apache2":
        ensure => "installed",
    }

    file {"/etc/apache2/mods-enabled/userdir.conf":
```

```
        content=>template("../mods-available/userdir.conf"),
        notify=>Service["apache2"],
    }
    file {"/etc/apache2/mods-enabled/userdir.load":
        ensure=>"symlink",
        target=>"../mods-available/userdir.load",
        notify=>Service["apache2"],
    }
    service {"apache2":
        ensure=>"running",
        enable=>"true",
        require=>Package["apache2"],
    }
}
```

Like Salt, Puppet makes user manage the dependency graph between functions, and here 15% of rows end up being dependencies. Puppet has multiple ways of creating dependencies that have different effects, and user must be able to choose the correct one and also know if a (daemon restarting) dependency is a side effect (e.g. "require").

Puppet uses terms known to all programmers, such as "class". Unfortunately, their definition in Puppet is completely different from the well known definition in programming. When combining functions ("resources") together, user learns concepts such as "defined resource types". While teaching Puppet techniques, these seem to create a lot of confusion, and this could be another area for further research.

Table 40: Comparing source line count when defining resource relationships for package-file-service

Tool	Lines	Dependency l.	Dependency %
Puppet	14	3	21%
Salt	16	4	25%
Conftero	5	1	20%

Source lines of code were calculated to compare the amount of dependency related code, and the results are listed in table 40. Empty lines and single character lines (e.g. '}') were ignored and line counts calculated with 'nl' and 'wc -l'. The leading CM tool DSL configuration examined was found to be approximately three times as long as the Conftero Python configuration. However, the portion of dependency related code stayed approximately the

same due to the denominator, the total line count, differing considerably. While this result points to interesting possibilities in developing a new configuration language, these comparisons should be considered only with other measures of code complexity and the effect on actual development. Source line count is merely a proxy indicator of code complexity and the number is dependent on the definition of source line.

The example of installing, configuring and keeping a daemon running is a simple task. Even though there are dependencies between the resources (package-file-service), it does not yet need actual control structures. Cases where multiple system users require similar configuration or cases where some configuration is repeated could make possible the comparison of more demanding control structures. Here, Salt uses code generation with templates, and Puppet uses its own concepts such as virtual resources. Even though one could expect traditional programming language to succeed well in this area, this comparison is left for future research.

Defining configuration in Conftero differs from typical solutions within research and industry. It uses general purpose language instead of a domain specific one. Due to this, Conftero uses common control structures, and competing solutions define their own unique methods for flow control. Conftero's resource model is simple, with less than ten state functions. Leading solutions have more than a hundred state functions. However, all CM tools use infrastructure as code methods and aim for idempotent configuration. These similarities and differences are listed in table 41.

Table 41: Comparing Conftero configuration definition to those common in industry and research

Aspect	Conftero	Typical tools
Language type	General Purpose Language	Domain Specific Language
Flow control	GPL: for, while, if-else	Templated code generation (salt), virtual resources and others (Puppet)
Run order	Imperative	Declarative (Puppet, Salt, Chef, Terraform...)
Idempotency	Idempotent	Idempotent
Pre-defined resources	Less than 10	Hundreds (Puppet 100+, Salt 500+)
User interface	Infrastructure as Code	Infrastructure as Code

4.13 Implementing the Main Prototype Conftero

The main research prototype was implemented in Go. Go is a statically typed systems programming language. It was chosen for its capability to cross compile multiplatform static binaries, a good standard library and strict error handling.

When Conftero was feature complete, it had approximately 4000 source lines of code, 700 of which were unit tests. During the work, the scope was extended to add a back channel from agents to the master. The Conftero configuration language was implemented using a Python dialect that is built into Conftero.

Some aspects of the implementation are interesting, but their novelty was not verified with literature review or they were more practical aspects:

- Statical linking (linking only against glibc) could make configuration management system more resistant to heterogenous environments and problems related to library updates on slave nodes.
- Including a complete Python dialect into the static binary further reduces dependency on software and libraries on slave node.
- Reduction of unnecessary abstraction seems to make some common operations faster, but this aspect was not investigated further. The speed of operations is only interesting when developing new configuration, but usually irrelevant when actually deploying configuration on slaves. For example, an industry leading tool Salt makes administrator define configuration in YAML-based DSL, then compiles this internally to Python-based structures, then calls idempotent state functions, which call imperative functions, which in turn often simply run a command in shell.
- Running Conftero slave software ('ccta') can be updated without reliance on outside tools such as package manager. This feature can be useful because configuration management system should be resilient to problems in package manager, and because slave and master software versions must match the whole lifetime of the system.
- The use of Go programming language allows cross-compilation to multiple processor architectures and environments. Some other programming languages of the same generation, such as Rust, could provide similar benefit.
- Multiple design and implementation decisions combined resulted in low resource use. Scheduled operation means that no resources are consumed when Conftero is not operating on slave. The choice of compiling to binary (and not bytecode or script) seemed to reduce resource consumption in practice. Low resource consumption is important when running on IoT devices or cloud servers, which are often priced according to RAM.

4.14 Novel Concepts in the Design

Multiple novel concepts were introduced in this chapter.

Security in configuration management is an area where many authors call for more research (Marsa-Maestre et al., 2019; Rahman, Mahdavi-Hezaveh and Williams, 2019; Kumara et al., 2021; Rajapakse et al., 2022; Rong et al., 2022; Xu and Russello, 2022; Ullah et al., 2023). In their systematic mapping study, Rahman, Mahdavi-Hezaveh and Williams (2019) did not find "any publication that focus on security issues", even though an error that violates security objectives "can compromise the entire system".

The Hidden Master architecture is a novel concept that radically increases security of configuration management system. It stems from the realization that master secret key is the most valuable file in the whole network. Not only does it provide full and unlimited access to all other files, it's a key to a system that makes this access fast and convenient. The Hidden Master architecture makes the master literally unreachable on the network when it's not issuing new configuration or receiving latest reports. This makes the window of attack very small, attack surface small and forces attackers to operate at a time when a human is actively observing the system. Literature review did not find any other solution that offers similar protection for master secret key, and in fact the value of this key does not seem to be highlighted in other publications.

Key management is critical for the security of Hidden Master. Hidden Master architecture also defines bootstrapping trust on an arbitrary number of new nodes. This is done with campaign keys for transmission of slave public key to master, removing the need for tedious and error prone verification of key fingerprints when accepting new slave nodes.

In the Hidden Master architecture, the combination of key management and asynchronous operation offers multiple benefits in addition to protecting the secret key of the master. The transfer layer becomes truly decoupled, and multiple examples of this were provided, including air-gapped and peer-to-peer operation. Many authors agree that configuration management systems should be scalable [Ullah et al. (2023); Rong et al. (2022); Rong et al. (2022); (Mansouri, Prokhorenko and Babar, 2020; Rong et al., 2022), a requirement helped by the Hidden Master architecture.

General purpose programming languages have been used to configure malware payloads, and this aspect is also identified as it's own ATT&CK framework technique. This work introduces general purpose language use to configuration management systems. Domain specific languages are widely used for configuration management systems. They introduce their own concepts for flow

control and abstraction. This work introduces two models for constructing a language for configuration management, base resource model and hasChanges revalidation model. Base resource model proposes the use five base resources, leaving higher level abstractions to the language. HasChages revalidation model removes the need for administrator to defined dependencies between resources in typical multi resource definitions.

A novel model for comparing concepts used in criminal malware and benign configuration management systems was created. As the new model includes mapping to MITRE ATT&CK, this readily allowed identifying existing concepts. This model could be used in future research to identify both existing and any emerging concepts for use in configuration management.

4.15 Conclusion

Security is critical in configuration management systems. When obvious steps are taken to protect network communications, the security lies in secret key of the master, and key management to correctly establish and maintain trust. The secret key of the master is the most valuable file in the whole network of computers controlled by the master. Compromise of master secret key allows attacker full and convenient access to all other data in all nodes. The Hidden Master architecture protects this key.

The Hidden Master architecture protects master's secret key. Due to asynchronous operation and key management, the master can be offline and even shut down when it's not sending slaves new configuration or receiving reports. Creation of the configuration and analysis of reports can be done offline. If the security situation is extremely difficult, it's possible for master to operate completely air-gapped and use another computer to analyse the reports from the slaves.

In the Hidden Master architecture, master signs and encrypts instructions to slaves. These encrypted catalogs can be stored into any untrusted transfer layer, for example untrusted web servers. The transfer layer is decoupled from other layers, allowing the use of peer-to-peer or air-gapped transfer layer if needed.

Improvements to configuration on the slave were proposed. The need to raise abstraction level and the challenges of domain specific languages can be helped by the use of general purpose language, such as Python, for configuration. The proposed base resource model simplifies the design of such definitions, both for creators of tools and to the users. General purpose language allows the use of regular flow control and abstraction methods, already familiar to anyone

knowing the basics of programming. The resource revalidation model proposed in this work further reduces typical configurations, making it often unnecesary to define dependencies between base resources.

To follow constructive approach, and to answer the call for more empirical research, two prototypes were implemented. In constructive research approach, theory guides building artifacts, which are then evaluated (Piirainen and Gonzalez, 2013). Here, the artifacts are the two software prototypes. These prototypes allow laboratory tests to compare the design to industry leading tools; and case studies to see if the design can conform to real life business requirements.

5 Findings and Analysis

The main aim of this thesis was to investigate the survivability of configuration management systems based on using and adapting concepts used successfully by malware. Using constructive approach, the theoretical basis was formed in the literature review (RQ1, RQ2, RQ3). This theoretical basis guided the designing and building of the two research prototypes (RQ3, RQ4). Concepts developed in the theoretical work and implemented in the prototypes needed to be validated. Validation is a key part of the constructive approach.

This chapter presents the results of the validation of the research prototype. Validation is done in simulated and emulated laboratory settings, case studies in realistic environments in the field and finally with expert interviews to evaluate potential business benefits. Research philosophy, rationale and research design were presented in chapter "Methodology". This chapter answers research questions RQ5, RQ6 and RQ7, which are shown in table 42.

Table 42: Research answered in "Findings and Analysis" chapter.

RQ	Explanation
RQ5	Based on load simulation, faulty network emulation and attack tree analysis, how does the resiliency of the configuration management software prototype - implementing some of the techniques adapted from malware - compare to a leading industry solution?
RQ6	Based on a case study in a realistic field context in a company, what utility does the research prototype provide in meeting the business requirements of the case?
RQ7	What potential do business benefits experts see for the models and the research prototype?

5.1 Empirical Validation in Emulated and Simulated Environment

Hidden master implementation was tested in an emulated environment. Emulation testing is part of the overall evaluation strategy laid out in Methodology chapter. This chapter both integrates and compares Conftero with leading configuration management tools, which were discussed in Literature review: Leading Configuration Management Tools.

5.1.1 Proof of Concept

Two functioning software prototypes were developed to test this project. A trivial proof of concept (PoC) prototype was developed using GNU Make, Bash shell script Puppet configuration management and the GnuPG encryption tool. Complete source code of proof of concept prototype is included in "Appendix: Hidden Server Architecture Encryption Demonstration". Operation of PoC is briefly explained as pseudocode in table 43.

This PoC took multiple shortcuts to test the feasibility of these ideas in the early stages of research. The goal of the PoC stage is to try the new and possibly difficult aspects, and leave obvious (but possibly laborious) aspects for later stages. All nodes were simulated with a single host, with directories representing nodes. Only one of each type of node was included: master, courier and slave. No consideration was given to the early stages, such as exchanging keys, establishing identities or installing software. Even though PoC was simple, it already showed the subsystems, components and interfaces that were discussed in "Layer Model of the Hidden Master Architecture".

Table 43: Pseudocode of PoC. M master, S slave, C courier.

Host	Action
S	Slave key pair is generated,and slave public key is delivered to the master
M	Master key pair is generated,and master public key is delivered to the slave
M	Administrator defines configuration in Puppet DSL (catalog)
M	Catalog is encrypted with the slave public key and signed with the master's public key
M-C	Master uploads encrypted catalog to the untrusted courier
C-S	Slave downloads encrypted catalog from the courier
S	Slave decrypts the encrypted catalog with the slave private key
S	Slave verifies the catalog integrity with the master public key
S	If decryption and verification is successful, the slave applies catalog using Puppet
S	A sample file is created, demonstrating a successful configuration cycle

5.1.1.1 PoC Execution A heavily abbreviated PoC execution output shows each stage. This is the result of executing the Makefile in appendix.

```
$ cat hello-slave.pp
```

```
file { "/tmp/helloTero.txt":
        ensure => "present",
        content=> "See you at TeroKarvinen.com!\n",
}
$ make
## Cleaning up... ##
## Generating key pairs... ##
## Key exchange... ##
## Publish encrypted catalog to untrusted server... ##
## Download and decrypt catalog to slave... ##
gpg: Good signature ...
puppet apply stage/slave/hello-slave.pp
$ cat /tmp/helloTero.txt
See you at TeroKarvinen.com!
```

I performed a similar test with Salt, another leading CM tool. Results showed that an included tool for host configuration over ssh, salt-ssh, could be modified to output simple files from Salt. Those files could be used by Conftero encryption and the file transfer layer. Unfortunately, any solution without source code modifications accepted upstream (in the official Salt project) would be tightly coupled with the salt version being used.

Leading CM tools (Puppet, Salt, Chef, Ansible) use resource abstraction. For example, "package" resource will use 'apt-get' on Debian, 'yum' on Red Hat and (depending on the configuration) Choco on Windows. If this resource abstraction is applied on the Master, it must have knowledge of the slave environment. The problem with resource abstraction was bypassed by sending slaves plain text, source code scripts instead of using external master daemon for compiling encrypted catalogs.

5.1.1.2 PoC Results The Proof of Concept test shows that generic encryption tools can be combined with at least one popular CM tool to the extent tested without modifying the source code of the CM tool. This is not an obvious result and not guaranteed for other tools or complex edge cases. Some tools use uncommon network transports, such as ZeroMQ in Salt. Others, such as Puppet which I used in PoC, applies resource abstraction on the master, and thus might need to know the details of the configured slave beforehand.

As the encrypted catalog can be dumped into a single, static file, it should be obvious that the network transport between subsystems can be easily changed while keeping other components the same. It seems that any method of transferring files could be used. Evaluation of the networking (file transfer)

layer is made much simpler as the security of transfer is not a responsibility of this layer. Using an established encryption standard (OpenPGP) and tool (GnuPG) instead of implementing a new system using low level encryption libraries, we can delegate many security challenges to this secure communication tool.

External configuration management tools, such as Puppet or Salt, were used in the PoC prototype and the early versions of the main prototype Conftero. As chosen development environment allowed creating static binaries without any dependencies, the downsides of using external CM tool were emphasized.

While the resource abstraction problem was bypassed, it was shown that all leading CM tools do not reveal a clear, file based interface that could be used. As such an interface is not clearly revealed or documented, relying on it would risk creating a tight coupling with a specific version of a CM tool.

5.1.2 Functional Prototype

To test the full design described in chapter "Designing Hidden Master Architecture", a functional prototype was developed. Go language was used to allow static linking, resulting in a single binary with no external dependencies. This makes it much more resilient against problems with runtime environment. Also, the pilot organization was expected to deploy to limited embedded hardware, where it was not expected to be difficult or inconvenient to install full Python or a Ruby environment required by some other CM tools.

5.1.3 Golden Path

Golden path scenarios test the main functionality of the application in typical or optimal working conditions. The purpose of Conftero is to deliver encrypted instructions to slave computers using untrusted intermediary hosts (Couriers).

Chapter "Methodology: Simulated scenarios" described the golden path test as follows : "HM transfers catalogs to two slaves over a network. No interference is generated". This experiment is part of answering RQ5: "Based on load simulation, faulty network emulation and attack tree analysis, how does the resiliency of the configuration management software prototype - implementing some of the techniques adapted from malware - compare to a leading industry solution?".

Network architecture consists of the hidden master (m), untrusted courier (c) and two agents (slaves) a1 and a2. The data flow is shown in fig. 8, and the details of the hosts are listed in table 44. After the initial installation, agents will only contact the untrusted courier host c to receive instructions.

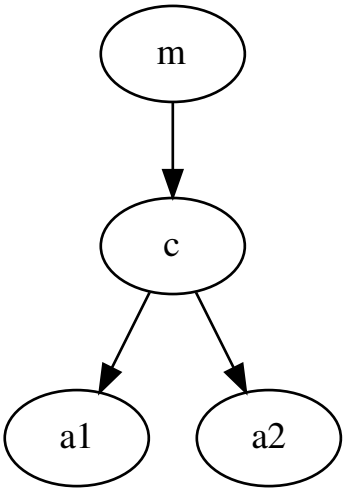

Figure 8: Golden path network structure

Table 44: Host attributes.

Host	Role	Address	Software
m	Master	192.168.12.1	cct, sshd
c	Courier	192.168.12.10	Apache2, sshd
a1	Agent	192.168.12.100	ccta, sshd
a2	Agent	192.168.12.101	ccta, sshd

This scenario considers the continued use of Conftero after installation. It is assumed that keys are generated on the master and Conftero is installed with OS provisioning.

Testing was done with Conftero (cct and ccta) 0.1.5-alpha and 0.1.15-alpha. The initial testing with 0.1.5-alpha pointed a need for minor user interface improvements not caught by the automated testing environment, which were fixed and the results of the test with the fixed version are shown below. The environment consisted of Ubuntu Linux physical and virtualized hosts running. All nodes were full operating systems with their own kernel and RAM memory. Software versions are described in table 45 and commands used to enumerate the testing environment are listed in table 46. The testing environment is interesting, because most leading CM tools aspire to be the single source of truth for their computing environment. This has led to the development of resource abstraction layers, but it might also show the challenges created by the limitations of the programming languages and their runtime environment.

Table 45: Testing environment

Purpose	Software	Version
OS	Ubuntu Linux	18.04.04
CPU architecture	amd64	
Virtualization	Virtualbox	5.2.34-dfsg-0~ubuntu18.04.1
Virtualization	Vagrant	1:2.2.1
CM	Conftero	0.1.5-alpha & 0.1.5-alpha

Table 46: Commands to enumerate testing environment

Purpose	Command
OS	grep DESC /etc/lsb-release
CPU architecture	uname -m
Virtualization	dpkg –list virtualbox vagrant
CM	cct version

At the starting point, all four machines were up. An SSH connection from the master to each of the hosts (c, a1 and a2) was set up with public key authentication. In a production version, encrypted catalogs would be uploaded (m->c) using SSH. Even though SSH connection to agents (a1 and a2) would not be needed in continuous use, it could be used for initial installation if the installation was not already performed during the operating system installation (provisioning phase). The homepage was available on untrusted courier c. This simulated a low cost untrusted web hosting, and would be used for publishing the encrypted catalogs.

The whole Conftero fits into a single binary file 'cct'. It was copied to the target system as /usr/local/bin/cct.

```
$ cct version
0.1.15-alpha, running on x86_64 linux ubuntu 18.04 virtual
```

Conftero has no dependencies, i.e. it is completely statically linked. This allows it to operate in limited environments such as embedded systems, and survive partial capabilities in situations where the runtime environment is damaged. The version tested is less than 20 MB, including the master's (cct) capability to generate all other required files such as the agent binary (ccta). To further verify that no special runtime environment is required, all tests were performed on freshly installed vagrant machines separate from the development environment.

```
$ ldd /usr/local/bin/cct
    not a dynamic executable
$ ls -sh cct
18M cct
```

A new hidden master repository was created. I chose "base" as the name. An unlimited number of repositories are allowed, with each in its own folder.

```
$ cct init base
Creating master directories...
Bundle files extacted.
Creating master OpenPGP keypair to 'base/keys'...
Creating settings file 'base/cctmaster.toml'...
Could not read counter file 'base/crypts/counter'. A new one will be created.
Tip: 'cd base' then run 'cct newagent foo' to create your first agent.
Success: Master initialized.
$ cd base/
```

The existing web space was added as a new courier. The upload SSH2 SFTP address and the download HTTP address were previously known to us as they are the addresses used for publishing homepages on untrusted courier c.

```
$ cct newcourier "sftp://192.168.12.10/~/public_html" \
    "http://192.168.12.10/~vagrant/"
Connecting as user '' SSH server '192.168.12.10' with timeout 5s...
Uploading file 'crypts/counter' to 'public_html/counter'...
SSH upload succesful. 'sftp://192.168.12.10/~/public_html/'
    is a valid SSH SFTP upload address.
Download - Downloading 'http://192.168.12.10/~vagrant/counter'...
Download succesful. 'http://192.168.12.10/~vagrant'
    is a valid download address.
Saving tested courier to settings file 'cctmaster.toml'
Tip: 'cct upload' makes your encrypted instructions available to agents.
Success: courier added.
```

Courier was automatically tested by generating *counter* file, uploading it to the server using SFTP and downloading it using HTTP. The counter file contains a monotonously growing number. Its main use is for agents to decide if they should download a new encrypted catalog.

A new agent was added on master.

```
$ cct newagent a1
Updating files for agent 'a1'...
Generating agent OpenPGP keypair...
```

```
Updating files for agent 'a1'...
Tip: Encrypt new instructions to agents 'cct crypt'
Success: Added new agent 'a1'.
```

This generated agent configuration on the master. To make these instructions available to the new agent, they were encrypted with agent a1 private PGP key and uploaded to untrusted courier c.

```
$ cct crypt
Encrypting catalogs for 1 agents...
Encrypting "agents/a1" to "crypts/8E4DF55002281230"...
Tip: Add some couriers to cctmaster.toml, then 'cct upload'
Success. Catalogs encrypted.
$ cct upload
Connecting as user '' SSH server '192.168.12.10' with timeout 5s...
Uploading plaintext counter to 'public_html/counter'...
Uploading 'crypts/8E4DF55002281230' to: '192.168.12.10',
    'public_html/8E4DF55002281230'...
Uploading 'crypts/counter' to: '192.168.12.10',
    'public_html/counter'...
Success: encrypted files uploaded.
```

The agent a1 must have agent software running to periodically retrieve new encrypted catalogs from courier c. Conftero agent application 'ccta' can be installed either on OS provisioning or later using SSH while the system is still directly accessible. I installed it using ssh. Privileged access required the use of 'sudo' with a password, which was queried interactively. Exit status 2 shows the result of idempotent configuration: the system was not in the requested configuration, thus changes were made, and no errors were met. Exit status would be named if –debug flag was used.

```
$ cct sshapply a1 sftp://192.168.12.100
Creating installer 'installers/ccta-a1-private'...
Connecting as user '' SSH server '192.168.12.100' with timeout 5s...
Uploading unencrypted agent (installer) to remote '/tmp/ccta'...
Attempting to run remote agent with sudo...
Remote sudo password:
Attempting remote sudo with password...
Tip: use --debug for more detailed output.
remote agent "a1": "Extracting embedded files...
a1: Copying ccta binary to '/opt/cctslave/ccta'... (will contain agent secret k
a1: Catalog succesfully downloaded, decrypted and verified from
    'http://192.168.12.10/~vagrant//'.
```

```
a1: 1 catalogs downloaded, decrypted and verified.
a1: [warning] Missing core.py, falling back to builtin.
a1: [warning] Missing core.py, falling back to builtin.
a1: 2
a1: "
Success: Installed ccta to ssh target and applied configuration.
```

With the agent installation now complete, direct connection between the hidden master m and the agent a1 is no longer needed. Agent systems can move behind ingress firewalls and reside in unknown and ever changing IP addresses. As long as they can access the web, they can receive the latest encrypted catalogs from the hidden master m.

For testing purposes, we examined the changes made by Conftero agent. On the master, a default module "managed" adds a file to the agent. Modules are defined using a built-in dialect of Python with idempotent functions. Here, file() declares that a file should exist in the given path, with given contents. As main Conftero functions are idempotent, action is only taken (and logged) if our declarations are not met. Thus, in most runs the agent system is in a correct state and no action needs to be taken by the file().

```
$ cat agents/a1/modules/managed/main.py
file("/etc/conftero-managed",
    "This system is managed by Conftero.\n")
```

This module has already run ("been applied") on the agent, first on the initial installation and later when the agent has run periodically. We can see this by running a command on the agent over SSH.

```
$ ssh 192.168.12.100 'cat /etc/conftero-managed'
This system is managed by Conftero.
```

The agent periodically pulls encrypted catalog from courier c. In my golden path test, this period is one minute. In a similar production system, a period of 15 to 30 minutes with splay time (randomization) should be used. To test this, I modified the module on the master.

```
$ cat agents/a1/modules/managed/main.py
file("/etc/conftero-managed",
    "Master did not need to contact agent for this change.\n")
```

And uploaded the encrypted catalog:

```
$ cct crypt
[...]
$ cct upload
```

```
[...]
```

After waiting for one minute, I used SSH to see the changes on the agent's file system.

```
$ ssh 192.168.12.100 'cat /etc/conftero-managed'
Master did not need to contact agent for this change.
```

We can see that agent a1 was successfully configured without a connection between master m and agent a1. The encrypted catalog was transferred using untrusted courier c.

A second agent was added using the same steps as the first agent. The output is similar to the commands above, so it is omitted here.

```
$ cct newagent a2
$ cct sshapply a2 sftp://192.168.12.101
$ cct crypt
$ cct upload
```

We used SSH to see from the agent a2 file system that it was successfully configured.

```
$ ssh 192.168.12.101 'cat /etc/conftero-managed'
This system is managed by Conftero.
```

5.1.3.1 Results of Golden Path Test Golden path test showed that Conftero main functionality works in a simple use case and optimal conditions. Minor user interface bugs not found by automatic testing were found and fixed in the golden path test. Agent installation was not originally planned as part of this experiment, but as it was already implemented it was used as the easiest way to set up the system. It was noted that agent installation using 'cct sshapply' prefers encrypted catalogs loaded over the network instead of checking the counter. In typical usage scenarios, it has little effect on use, and it was not fixed at this time. These results are summarized in 47.

Table 47: Results of the Golden Path test

Aspect	Result
Work from single, statically linked binary	OK
Define and test courier	OK
Encrypt, sign and upload catalogs	OK
Download, decrypt and verify catalogs	OK
Published changes automatically applied by agents	OK
Install agent over SSH	OK (not in test plan)

Aspect	Result
Other: User interface	minor improvements made
Other: sshapply configuration priority	minor bug, not fixed

5.1.4 Load Test

Chapter "Methodology: Simulated scenarios" described "Load": "a load of 10 000 slaves is estimated, then simulated with stress testing tools such as ab."

To perform load testing, the load level must first be estimated. I start by assuming that encrypted catalogs are downloaded every 15 minutes with splay time (randomization) that statistically distributes the load evenly over time. Even though Conftero architecture allows inexpensive, geographically and network distributed parallel couriers to distribute traffic, I commence with the most load-wise difficult scenario of having just a single courier. If we have 10 000 hosts each loading an encrypted catalog once, rounded to full download there will be

10 000 / 15 min = 667 / min = 11 / second.

For a web server, this is very low load.

The use of a public counter file is a key optimization. Agents check the version of the catalog by comparing the downloaded counter to the version they already have. If no new catalogs are available, the agent skips download, decryption, verification and extraction steps. This saves considerable work on both the intermediary server and on each client. It could be expected that most downloads will only download the counter. An agent only downloads the encrypted catalog if counter indicates it has new instructions. The counter is implemented as a plain text file with a single, monotonously growing integer, encoded as ASCII text.

More generically, we can calculate the load level per courier, the courier download frequency (f_c) from agent update interval (t_u) and the agent count (n).

$f_c = t_u / n$

It can be seen that the ability to distribute requests over time and the agent update interval (t_u) have a great effect on the server load. It should be noted that 0.1.15-alpha does not yet implement splay (distributing agent updates over time).

The size of the encrypted catalog has an effect on the courier load level. Large catalogs obviously transfer more data, but a high number of hosts combined with

a large catalog could result in a situation where previous catalogs are still loading while new ones are requested. However, the most obvious implementation of agents, also used in the research prototype, is blocking, and thus the same client will not start another download while the previous download is in progress.

Using multiple large catalogs on a server with very limited RAM memory may mean that some catalogs do not end up in disk cache (RAM). Thus, the disk bus (IO) load could be higher. If this becomes problematic, it could be mitigated by use of campaign key pairs.

Even though 10 000 agents is quite a large network, it might be interesting to design a scenario where the load level mandates the use of multiple couriers. In any case, multiple couriers are beneficial for the resiliency of the network.

5.1.4.1 10 000 Hosts with 15-Minute Update Interval The starting situation was a courier with a single encrypted catalog. The courier was a Linux virtual machine with 1 GB RAM. The test was performed over a local, virtualized network which has very low latency. The test setup is detailed in 48. The commands to gather this information are mentioned in the beginning of this chapter.

Table 48: Courier setup for load test

Software or feature	Comment
Apache 2.4.29-1ubuntu4.13	userdir enabled
Ubuntu Linux 18.04.1 LTS	
Virtualbox 5.2.34-dfsg-0~ubuntu18.04.1	
Vagrant 1:2.2.1	bento/ubuntu-18.04
RAM 1 GB	
Architecture x86_64	

Before starting the stress test, I tested that courier was working and examined the sample encrypted catalog. This is the same catalog that was created earlier in the Golden Path test. Even though the catalog could also be downloaded with any web browser such as Firefox, I used the command line browser 'curl' to get more exact and repeatable results.

```
$ curl -s http://192.168.12.10/~vagrant/1DC69D5A1873014D|head -4
-----BEGIN PGP MESSAGE-----

wcBMA6uQfcxjjLbKAQgAUPTWbqyeaVCv6K/1NupRniVgZBJFki8ltdlNe9lOOMF+
N2xh+cUyuWwbhacUNlBvpQ2TTf9eP86lnCCfrAp6uC+dCtLURn8XKT3VMuEyeOhW
```

Download worked as expected. The start of the catalog shows its PGP encrypted message in ASCII armor. The encryption is obviously required, but to save space, it could be stored in binary form. ASCII armor wastes space and thus increases the network load.

```
$ wget http://192.168.12.10/~vagrant/1DC69D5A1873014D
$ ls -sh 1DC69D5A1873014D
12K 1DC69D5A1873014D
```

The size of the catalog was 12 kilobytes.

I used Apache benchmarking tool 'ab' from apache2-utils against the URL verified in previous steps. It should be noted that while load testing tools such as ab are useful inside our own network against our own machines, using them in other networks or against hosts one does not own could result in a denial of the service attack (DOS).

Using the planned 10 000 agents with 100 concurrent requests, a load test was performed.

```
$ ab -n 10000 -c 100 http://192.168.12.10/~vagrant/1DC69D5A1873014D \
    |grep -P 'Failed| 100%|per second|#/sec|Time taken'
Time taken for tests:   2.013 seconds
Failed requests:        0
Requests per second:    4966.82 [#/sec] (mean)
 100%   1037 (longest request)
```

The whole network of 10 000 agents could retrieve their encrypted catalogs in two seconds. In the plan, a time of 15 minutes would have been acceptable. As the margin is about 45 000%, I conclude that for small catalogs, the courier load for even a large network with just a single courier is insignificant. On the one hand, web servers are usually highly optimized for efficiently serving static content. But on the other, the efficiency of configuration management was even higher than expected.

To get an idea how large the margin is, I performed another test with 10 times more agents: 100 000 agents with 500 agents performing the download at the same time.

```
$ ab -n 100000 -c 500 http://192.168.12.10/~vagrant/1DC69D5A1873014D \
    |grep -P 'Failed| 100%|per second|#/sec|Time taken'
Time taken for tests:   22.630 seconds
Failed requests:        0
Requests per second:    4418.84 [#/sec] (mean)
 100%  14176 (longest request)
```

As 100 000 agents could be served in 23 seconds, the courier web server is not a bottleneck. The rate of requests served, about 4500/s, is similar to the earlier test. The longest request took over 14 seconds. Even though this would be unacceptable for a human looking at a web page, it is insignificant compared to a 15-minute update interval. However, for very large networks seeing slow downloads it is possible that there are nodes that always time out. This could be mitigated by always randomizing the time waited, so that a different host goes last on each update round.

The test differs from the real life production environment in some aspects. In a real network, there is a latency, typically from a 20 ms wired network to even a 500 ms cell phone network. In the test setup, the latency was less than 1 ms. When Conftero is used, the agent uses built-in HTTP client to download the encrypted catalog. It will also perform two HTTP requests when the catalog has changed, even though the first download (counter) is very small. Here, 'ab' and its built in HTTP client was used. Despite these differences, the huge margin indicates that the hit frequency on courier is unlikely to become a bottleneck.

5.1.4.2 10 000 Hosts with Big Encrypted Catalog To test the effect of the catalog size on the courier load, a real life sample configuration was downloaded and processed with Conftero. RHEL 5 Puppet Modules from the United States Government Configuration Baseline (USGCB) (NIST, 2016) was compressed for size example only, and no effort was made to port it from Puppet to Conftero Python.

After downloading and uncompressing, the sample from USGCB was copied as a new module to agent a1 configuration on the master. This USGCB module did not have a main.py entry point, so it would simply be copied to the agent without any other action. As a result, the size of uncompressed, clear text configuration for agent a1 was approximately 750 kB.

```
$ du -hs agents/a1/
736K    agents/a1/
```

The catalog was compressed, signed and encrypted, then uploaded to an untrusted courier. Most of the output of these commands is omitted for brevity.

```
$ cct crypt
$ cct upload
```

Before starting load testing, I verified that download worked.

```
$ wget http://192.168.12.10/~vagrant/1DC69D5A1873014D
$ ls -sh 1DC69D5A1873014D
```

```
204K 1DC69D5A1873014D
```

Despite the fact that cct unnecessarily uses ASCII armor, the size is only approximately 200 kB. This is likely due to zip compression before encryption. After waiting for a minute, I could see that this catalog is indeed automatically downloaded to the agent.

```
$ ssh 192.168.12.100 'sudo du -hs /opt/cctslave/modules/usgcb'
692K    /opt/cctslave/modules/usgcb
```

A load test of 10 000 simulated agents with 100 concurrent connections was performed.

```
$ ab -n 10000 -c 100 http://192.168.12.10/~vagrant/1DC69D5A1873014D \
    |grep -P 'Failed| 100%|per second|#/sec|Time taken'
Time taken for tests:   10.931 seconds
Failed requests:        0
Requests per second:    914.81 [#/sec] (mean)
 100%   2673 (longest request)
```

As all 10 000 agents were served in under 11 seconds, it seems that plain text configuration can be served by even a single web server, even if there is a huge number of agents.

5.1.4.3 Courier Load when Distributing Large Binaries One possible use case for Conftero could be distributing large binaries. For example, a company could distribute proprietary software to its laptops and consider Conftero as a secure channel for this. Firefox 75 Linux amd64 installer was used as a size sample. In real life, Conftero can install Firefox using package manager, and the size of the encrypted catalog would be less than 10 kB.

```
$ ls -sh firefox-75.0.tar.bz2
65M firefox-75.0.tar.bz2
$ cct crypt
$ cct upload
```

Adding 65 MB Firefox to an existing configuration resulted in a catalog of about 90 MB.

```
$ wget http://192.168.12.10/~vagrant/1DC69D5A1873014D
$ ls -sh 1DC69D5A1873014D
87M 1DC69D5A1873014D
```

We simulated a case where master has published a new version of the encrypted catalog, and thus all agents must download a new version. Performing the test with the parameters we used earlier is quite fruitless:

```
$ ab -n 10000 -c 100 http://192.168.12.10/~vagrant/1DC69D5A1873014D \
    |grep -P 'Failed| 100%|per second|#/sec|Time taken'
```

The first set of concurrent downloads will download 90 MB * 100 = 9 GB of data, which would take considerable time. Testing with 1000 simulated agents:

```
$ ab -n 1000 -c 10 http://192.168.12.10/~vagrant/1DC69D5A1873014D \
    |grep -P 'Failed| 100%|per second|#/sec|Time taken'
Time taken for tests:   118.950 seconds
Failed requests:        0
Requests per second:    8.41 [#/sec] (mean)
 100%   2727 (longest request)
```

Serving 1000 agents took 2 minutes. Extrapolating from this, serving 10 000 agents would take 20 minutes. This is one third higher than the 15 minutes used in the earlier example, but lower than the budgeted upper limit of 30 minutes.

We can conclude that when serving large encrypted catalogs, such as those containing large binary files, a single small (1 GB RAM) courier cannot serve 10 000 agents with 15-minute update intervals. Many networks do not have such a large number of agents, and the problem is mitigated. But this raises the question of how one would handle a huge network.

For large networks with large files, the Hidden Master architecture in Conftero provides a solution. By using multiple couriers and allowing agents to randomly select between those, this problem is mitigated. This multi-courier architecture is presented in fig. 9. All couriers can serve the catalogs in high demand, and the load is evenly distributed between those couriers. As this load balancing is random and statistical in nature, no extra coordination is required from either the agents or couriers. Thus, this problem becomes embarrassingly parallel, and we can expect the capacity to scale linearly as couriers are added to the network.

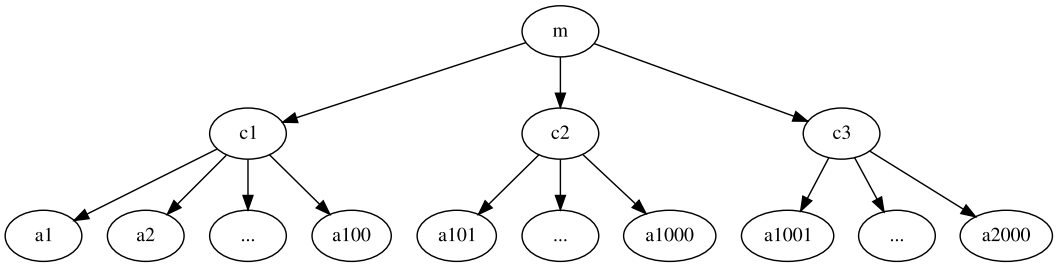

Figure 9: Multi-courier architecture makes downstream transfer embarrassingly parallel

This test was performed using small virtual machines in a virtual network.

This presents multiple sources of errors. The virtual network has insignificant latency and considerable speed, both of which could make the results look better than in real life. On the other hand, the single machine initiating the load test could be itself saturated. Due to large margins of error in the results, it seems that any common latency (20 ms - 500 ms) would not affect the results. The machine initiating the tests was not saturated in these tests, which could be seen from the lack of error messages and the speed of test runs. Network speed (transfer rate) could affect these numbers, and this was further evaluated in network tests simulating faulty and limited networks - something presented later in this work.

5.2 Effect of Network Faults

Network faults, such as lost packets, are common in real life networking environments. Common network protocols, such as TCP (transmission control protocol), already has built-in mitigations for network faults. Obviously, it is possible to only correct some network faults but not others. For example, it is possible to correct single bit error with a checksum but not 100% packet loss.

This experiment is part of answering RQ5: "Based on load simulation, faulty network emulation and attack tree analysis, how does the resiliency of the configuration management software prototype - implementing some of the techniques adapted from malware - compare to a leading industry solution?"

An emulated environment was created for this test. Fully virtualized computers were created as platforms for master and agent (slave) computers, and in the case of the research prototype, a third computer was employed to operate as a courier. As all network paths traveled through one computer in this simplified configuration, fault injection was performed on this computer's virtualized network interface.

5.2.1 Virtual environment with fault injection

Configuration management system is expected to configure computers over a network, usually the Internet. This means that the network is unreliable, and can be hostile. The test environment must thus include the ability to create multiple nodes (computers) and a network between them. To simulate unreliability, there should be an ability to simulate errors, such as packet loss, latency and jitter. Requirements for simulation environment are listed in Methodology: Requirements for Simulation Environment

To keep the environment as realistic as possible, fully virtualized computers were used for this experiment. Originally the idea was to use very light

containerization such as Docker or MiniNet, but in the initial feasibility testing of the environment, this seemed to create multiple problems only related to the test environment, thus obscuring the problems of interest.

The virtual machines are controlled over SSH, which establishes an interactive encrypted connection to shell over network. Adverse network conditions would quickly make interactive use of SSH highly inconvenient. To sidestep this problem, the virtual machines each had an additional network interface for control over SSH.

Emulation environment with fault injection is shown in fig. 10. The target computers, master m and agent (slave) s1 communicate through their own network. This network is connected to its own network interface (a virtual network card) eth1 in each m and s1. Faults are injected in master's (m) network interface eth1. As all test traffic passes through this interface, errors for the whole network can be emulated in a single interface, and no SDN is needed. Even if more agents are created in future tests, all network traffic passes through a single interface in many scenarios.

Monitoring computer (mon) controls and monitors the experiment. Control and monitoring of the test uses its own virtual network and separate network interfaces on m and s1. This way, error emulation does not affect control and monitoring.

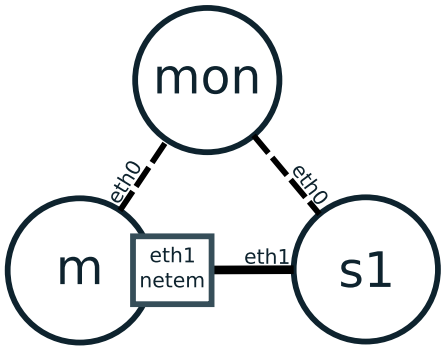

Figure 10: Emulation environment with fault injection

Fault injection was simplified by the fact that all traffic traveled through a single node in the experiment's simple architecture. For traditional configuration management systems, this was the master node. Conftero, the research prototype, uses the Hidden Master architecture, and slaves do not need to ever contact the master. By using a single courier node in this experiment, all traffic passed through that node. This allowed fault injection in the network stack of a single node, instead of adding complex software defined networking or virtual routers.

VirtualBox was used for emulating full machines. To keep experiments repeat-

able, VirtualBox was controlled by Vagrant, an IaC tool for controlling test VMs.

Fault injection was performed using the Netem kernel module. Netem uses pre-existing Quality of Service and Differentiated Services facilities in Linux kernel to emulate unwanted qualities of a wide area network in laboratory settings (Hemminger, 2005). Netem was controlled with the 'tc' user space program and qdisc queueing discipline. By default, netem only affects egress (outgoing) traffic. Ingress (incoming) fault injection was achieved by using Intermediate Functional Block pseudo device ifb. Packet loss as sampled by the 'ping' command was similar to the packet loss applied by this setup. One could expect the round trip packet loss to double when error is applied to both ingress (incoming) and egress (outgoing) traffic, but this was not the case. To facilitate testing, a simple shell script 'tem' was created to quickly enable and disable error injection with different settings.

Faults provided by the test environment were packet loss, latency, jitter, corruption and rate limiting. These network faults are listed in table 49. Typical example values are similar to those of a consumer grade broadband line in good conditions. The example values are listed as a convenience only, and are not based on extensive research or experimentation.

In the internet protocol suite, data is split into packets for transit. Packet loss is the portion of these packets that is not acknowledged as having been received at the other end.

Latency is the time it takes for even the smallest piece of data to travel through a network link. Latency can be present even on a high speed link, as it can take time to transfer the first byte even if the throughput was high. Latency is often measured with the 'ping' command round trip time (RTT), meaning the time it takes for a small piece of data to travel to the target computer and back. Jitter is the variation of latency and it means that some packets take a shorter time to arrive than others.

Corruption is the unwanted and unexpected change of data in transit. Here, the corruption percentage is the portion of packets that have any corruption. Duplication is the action of sending the same packet more than once.

Rate limiting is a limitation of throughput in the system, of how many bits can be transferred in a unit of time. For this system, throughput was measured in link layer, the lowest layer of the TCP/IP stack. The unit is in bits per second, which is often used to imply a measurement in link layer. As each layer in a stack model adds head or tail data to the packet, the payload transferred is often smaller. The transfer rate in the application layer (the top layer of

TCP/IP model) is usually in bytes per second. The number of bytes per second can be expected to be approximately 1/8 to 1/10 of the value in the link layer, as a byte has eight bits and each layer adds non-payload data to the packets. For example, the link layer transfer rate of 10 kbit/s (ten kilobits per second) would result in an application level transfer rate of little less than 1 kB/s (one kilobyte per second).

Table 49: Network faults

Fault	Unit	Example	Explanation
Packet loss	%	0 %	Portion of packets that do not reach their destination
Latency	ms	15 ms	Time for a packet to travel to the target and back
Jitter	ms	1 ms	Variation of latency
Corruption	%	0 %	Portion of packets that have changed bits
Rate limit	bit/s	100 Mbit/s	Maximum throughput
Duplication	%	0%	Portion of packets that were sent multiple times

For the test, a simple configuration to copy a 74 MB file to agent systems was created. The file contained installer for Firefox ESR 102 and instructions to copy the file to the agent system. No attempt to actually install the program was made, as this test was only interested in transferring the file. This was similar in size to the sample payload used in courier load testing with simulated load.

5.2.2 Effects of Adverse Network Conditions to Salt

Salt, one of leading IaC configuration management systems, was tested in adverse network conditions.

Virtual machine s1 was running salt-minion (the slave daemon), and m was running salt-master. As with most CM tools, both agent and master must run the same version, so both salt-master and salt-minion were 3002.6. Both machines s1 and m were running Debian Linux 11.4 Bullseye amd64 on VirtualBox controlled by Vagrant. Before the test, master-slave architecture was enabled and tested and keys exchanged. The agent (slave, minion) contacted master using the IP address 192.168.80.100. Using the pull architecture, the IP address of the agent was not relevant as the master was running the server (in the sense of client-server architecture).

The cache was cleared between tests using 'sudo salt "*" saltutil.clear_cache'. Configuration was applied and timed using 'time sudo salt "*" state.apply -l debug'. This used the 'time' command outside the measured tool for precise timing, but this depended on the tested tool's promise that configuration was delivered. To rule out the possibility of mistaken confirmation for delivery, the sample configuration instructed the payload to be copied to /tmp/ directory on agent, where it could be verified with 'sha256sum' cryptographic checksum. This verification was on some occations.

The results of the packet loss test are listed in table 50. Packet loss is round trip packet loss, e.g. what 'ping' could be expected to display. After a certain level of packet loss, applying consistently failed. Failure to apply or send configuration to the agent was interpreted from the repeated debug log message "Trying to connect ... retcode missing from client return".

Master failure was interpreted from the error message "The salt master could not be contacted. Is master running?" and from the fact that the master daemon had to be restarted after that to continue. Systemd status command 'sudo systemctl status salt-master' showed that the reason was the reservation of too much RAM: "failed (Result: oom-kill)". OOM killer is a Linux feature that shuts down and removes some software from memory when RAM is running out. The VM in question had 512 MB RAM. Apache2 web server running on the same machine was shut down to free more memory for tests, and the tests affected were repeated with the contaminated results discarded.

Where multiple attempts failed, the results show the failure rate (x % fail) and the mean time of successful attempts (with failed attempts removed from the mean). As Salt seemed to keep trying forever in face of failures, each test was terminated and ruled a failure after fifteen minutes. Based on the debug log, it

seemed that transfer did not start, so additional time would not have helped. Alternatively, a shorter cutoff time could be chosen for further tests in cases where it is clear that the program is making no progress. As CMS are often working non-interactively, a wait time of a couple of minutes at a time nobody is watching is not problematic, and thus it makes sense to set the cutoff value relatively high. Compared to an operation in optimal conditions, the time of the cutoff value is already large. The cutoff value is 900 s / 5 s = 180 = 1800 % of the time for a normal operation. As time measurements meeting a cutoff value are never finished, no mean time was calculated.

Table 50: Effects of packet loss to Salt

Packet loss	Failures	Mean time	Time measuremens
0 %	0%	4.7 s	4.0 s, 7.59 s, 3.54 s, 5.11 s, 3.39 s
3 %	0%	52.7 s	108.57 s, 77.56 s, 27.3 s, 30.85 s, 19.33 s
5 %	0%	65.2 s	39.55 s, 67.27 s, 46.49 s, 120.42 s, 52.18 s
10 %	0%	340.5 s	450.4 s, 249.65 s, 247.63 s, 383.43 s, 371.47 s
15 %	0%	761.9 s	719.35 s, 753.77 s, 719.15 s, 840.82 s, 776.46 s
18 %	100%	fail	900.01 s, 900.0 s, 900.01 s, 900.01 s, 900.01 s
20 %	100%	fail	900.0 s, 900.01 s, 900.0 s, 900.01 s, 900.01 s

The effects of packet loss to Salt is graphed as a scatter diagram in fig. 11. Each measurement is shown as a blue dot. Mean values of measurements are shown as a blue line. Because the vertical time axis is linear, the measurements at low packet loss are overlapping. In optimal network conditions and no packet loss, the transfer of the 74 MB catalog takes only a few seconds. As the packet loss increases, the time increases to minutes, ending in complete and

consistent failure at 18% packet loss. The detrimental effect of packet loss to Salt communications seems exponential.

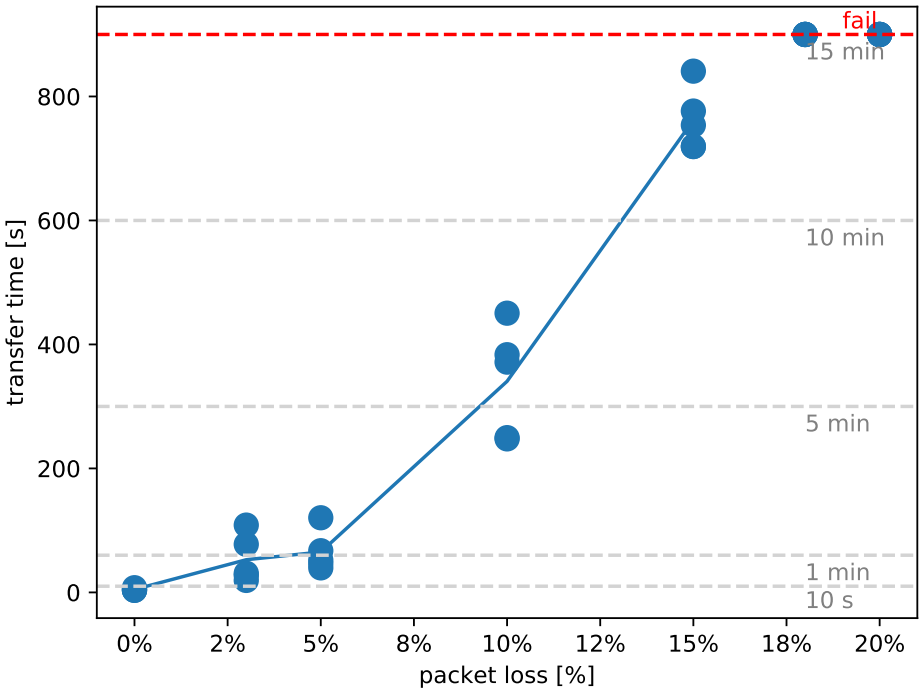

Figure 11: Effects of packet loss to Salt

5.2.3 Effects of Adverse Network Conditions to wget (HTTP)

To get a baseline of a well implemented traditional HTTP file transfer, I used wget to download the 74 MB sample file from an Apache 2 web server. Both Apache 2 and wget are established and mature tools packaged by multiple Linux distributions. Apache is the same web server used for load tests, advanced research prototype Conftero and the case study in Company X.

Using common tools for HTTP transfer was shown to be highly resilient to packet loss. Compared to Salt, it was necessary to test at much higher levels of packet loss to generate errors. As wget is usually used for downloading from web servers hosted by other organizations, it does not keep indefinitely retrying when meeting error conditions. Instead, it terminates after a few tries. It could be expected that increasing retries would make wget even more resilient in the face of network problems. Here, all tools compared were used with default settings.

Table 51: Effects of packet loss to Wget

Packet loss	Failure rate	Mean time	Time measuremens
0.0 %	0 %	33.6 s	22.57 s, 27.36 s, 34.36 s, 35.83 s, 47.65 s
3.0 %	0 %	55.6 s	14.93 s, 166.87 s, 26.97 s, 30.08 s, 39.23 s
5.0 %	0 %	26.7 s	14.82 s, 17.03 s, 29.52 s, 33.32 s, 38.85 s
15.0 %	0 %	39.3 s	21.16 s, 24.16 s, 45.59 s, 50.48 s, 55.3 s
18.0 %	0 %	74.3 s	118.0 s, 146.41 s, 16.22 s, 26.38 s, 64.46 s
25.0 %	0 %	29.3 s	21.41 s, 23.12 s, 23.12 s, 38.56 s, 40.18 s
30.0 %	0 %	47.4 s	29.78 s, 41.53 s, 42.84 s, 50.08 s, 72.62 s
35.0 %	0 %	53.6 s	38.76 s, 51.4 s, 57.96 s, 58.57 s, 61.22 s
40.0 %	0 %	48.2 s	33.33 s, 37.58 s, 45.53 s, 57.71 s, 66.98 s
50.0 %	0 %	121.6 s	125.69 s, 294.63 s, 41.47 s, 70.85 s, 75.23 s
55.0 %	0 %	122.5 s	123.54 s, 126.66 s, 217.32 s, 65.29 s, 79.85 s
60.0 %	0 %	212.6 s	119.76 s, 199.23 s, 318.38 s, 360.48 s, 65.25 s

Packet loss	Failure rate	Mean time	Time measuremens
65.0 %	20 %	126.4 s	141.4 s, 184.43 s, 85.22 s, 94.55 s, 900.0 s
70.0 %	20 %	424.0 s	213.52 s, 217.6 s, 374.83 s, 890.1 s, 900.0 s
80.0 %	100 %	fail	3.02 s, 3.04 s, 3.04 s, 3.05 s
90.0 %	100 %	fail	134.66 s, 2.94 s, 3.04 s, 3.04 s, 3.04 s
100.0 %	100 %	fail	3.01 s, 3.04 s, 3.04 s, 3.04 s, 3.06 s

The effects of packet loss are graphed in fig. 12.

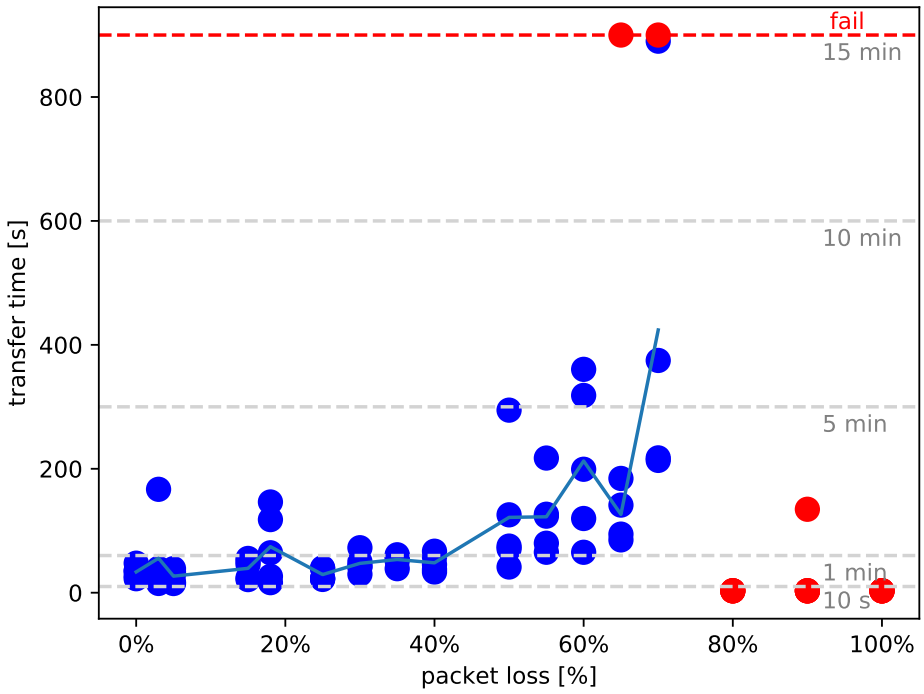

Figure 12: Effects of packet loss to wget

5.2.4 Effects of Adverse Network Conditions to SSH

SSH is a common way to remotely and interactively control servers. It establishes an encrypted, two-way authenticated connection to the server, allowing

the administrator to give shell commands on the command line. SSH also has a lot of other features, such as file transfer, secure tunnels and even VPN. SSH is used for some leading configuration management systems when using push-architecture, such as Ansible by default and Salt when using the 'salt-ssh' tool. Here, popular OpenSSH client and server packaged with Linux distribution (Debian 11 Bullseye) were used. Advanced research prototype Conftero uses SSH as a channel between the Hidden Master and Courier, but it implements its own SSH client instead of using OpenSSH.

Measuring the effect to SSH was required because one of the tested industry leading configuration management systems, Puppet, does not include tools for remote commands in the main application. For these tests, SSH was used for initiating pull on agents and verifying the result afterwards. To remove and estimate the effect of SSH on the measurements for Puppet, a separate test for SSH was performed.

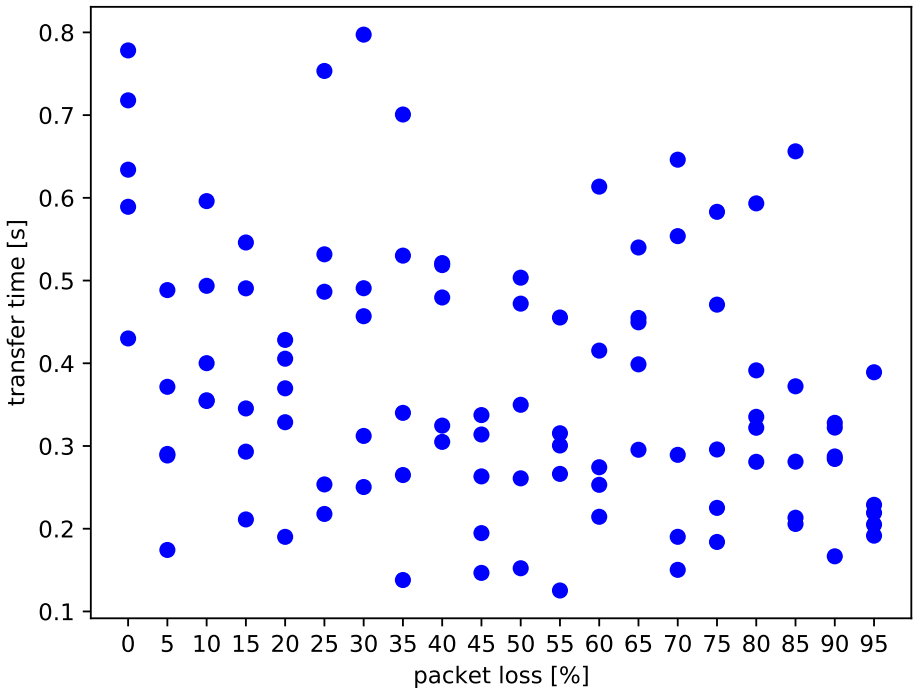

Figure 13: Effects of packet loss to short SSH commands

Because the traffic required for testing consists of short shell commands and their answers, the same type of test was performed here. This makes the SSH test performed here different from the large (74 MB) file transfer tested with Salt and wget, and those tests are not comparable. When preparing the test environment, it was noted that for some tools, transferring larger files is much more sensitive to packet loss than sending short commands. The command ran on master for this test was

```
$ ssh vagrant@192.168.80.101 "hostname --fqdn"|grep s1
```

The effects of packet loss to short SSH commands are very small and the effects of packet loss are graphed in fig. 13. The results show that even with 95% packet loss, the transfer time is less than one second. All of these communications were successful. In addition to checking the return value of the command, the actual results of the command were verified. There is no clear trend visible in the scatter plot. One possible reason is that the effect of packet loss is so small that other incidental factors affect the results more. The range, the difference between the largest and smallest observation, was only 700 ms.

Thus, SSH can be used for controlling Puppet and other tools in packet loss tests, as the time added by SSH is less than a second even on high packet loss.

5.2.5 Comparing the Results of Adverse Network Conditions

5.2.5.1 Packet Loss To compare the tools in adverse network conditions, each tool was used to download one megabyte file over an unreliable network. The file was filled with random contents, using cryptographic quality random data from Linux /dev/urandom. Random contents were required to prevent the automatic, unrealistic use of compression. An SHA256 cryptographic hash was used to verify the correctness of the transfer, using the 'sha256sum' tool.

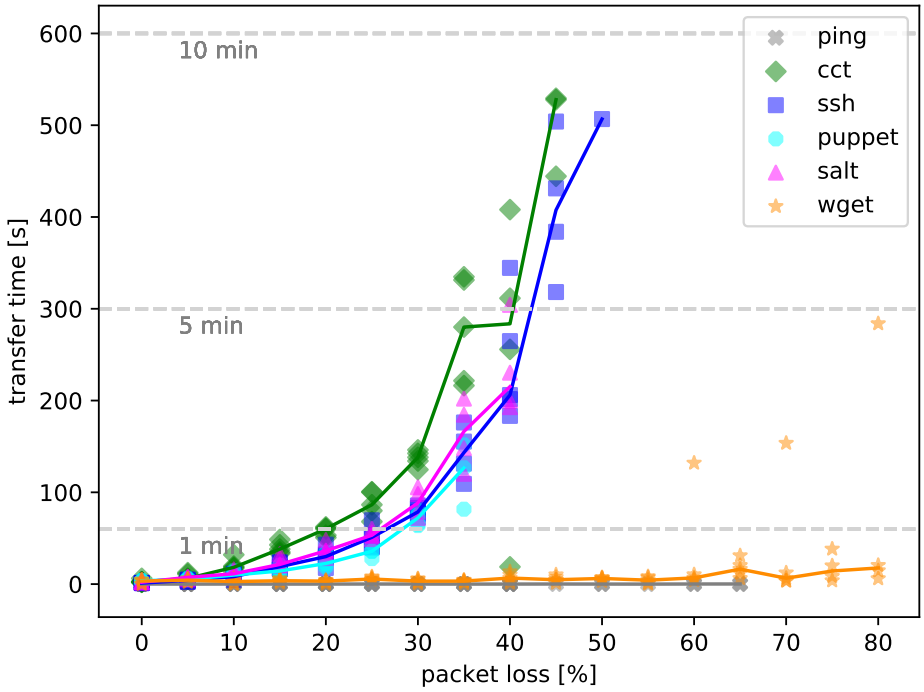

Figure 14: Effects of Packet Loss to Multiple Tools

Network errors were simulated with the Linux kernel module netem, and

controlled with ‘tc’ command and qdisc discipline. Machines were emulated with VirtualBox, controlled with Vagrant IaC tool. The test was run with GNU ‘make’. To facilitate running the test, some custom Python scripts were used: ‘tem’ to generate errors; ‘testero.py’ to run the tests; and ‘multigraphtero.py’ to combine test data and draw a graph using the Matplotlib Python library.

Tools were tested first in a fully working network (packet loss 0%), and then progressively worse conditions. For each level of packet loss, three tests were performed. When all three tests failed on a level of packet loss, no further tests were performed. Compared to some earlier tests, this saved a lot of time. After a failed level of packet loss, all the tests at a higher level can be expected to fail. Due to retrying, they would have only failed after timeout, resulting in most test time being spent on tests that produced no useful data. Thus, the unnecessary tests were skipped.

It could be argued that setting a higher timeout would eventually result in a successful file transfer, especially with tools that have unlimited retries. In real life, no human is likely to actively wait for configuration management to happen, except when developing the configuration on a test machine, so a long transfer time could be acceptable. In an earlier test, some tools were tested with a higher timeout. Examination of the transfer time median plot appears to show that the time could grow exponentially. The timeout used in this experiment is 600 seconds (10 min). The fastest transfer in the reference condition of 0% packet loss was 29 milliseconds. The timeout is over twenty thousand times the value at reference condition (600 s / 0.029 s = 20 689.7).

Industry leading CM tools Puppet and Salt were compared to the research prototype, Conftero. For comparison, an HTTP download tool ‘wget’ and remote control tool ‘ssh’ were used for downloading the same file as CM tools. Wget is a common tool for downloading files over HTTP and HTTPS, and it is included in multiple popular Linux distributions. Command ‘ssh’ from OpenSSH is a popular tool for remote shells, and it is included with many Linux and BSD distributions, and even ported to Windows by Microsoft. Ping was used for comparison only; it did not have a payload or retries similar to other tools. Instead, ping sent a single ICMP ECHO_REQUEST packet, and was considered success if ICMP ECHO_REPLY was received back.

The comparison of effects of packet loss is graphed in fig. 14. Each successful test is graphed as a symbol. The median of the results is graphed with a line. To make the plot more readable, failed transfers are not included.

Table 52: Worst packet loss conditions tolerated by each tool

Tool	Worst packet loss tolerated
Puppet	35 %
Salt	40 %
Conftero (cct)	45 %
SSH	50 %
(Ping)	65 %
Wget	80 %

The research prototype, Conftero, was more resilient to packet loss in this test than the leading industry tools Puppet and Salt. SSH survived better than Conftero. As seen in fig. 14, the reaction of the tools to packet loss is similar for every level for Conftero, Salt, Puppet and SSH. Even if the order stays the same on each level, the differences are small. Ping is listed only for reference, as it did not transfer a similar payload. The worst packet loss conditions tolerated are listed in table 52 and graphed as a bar plot in fig. 15.

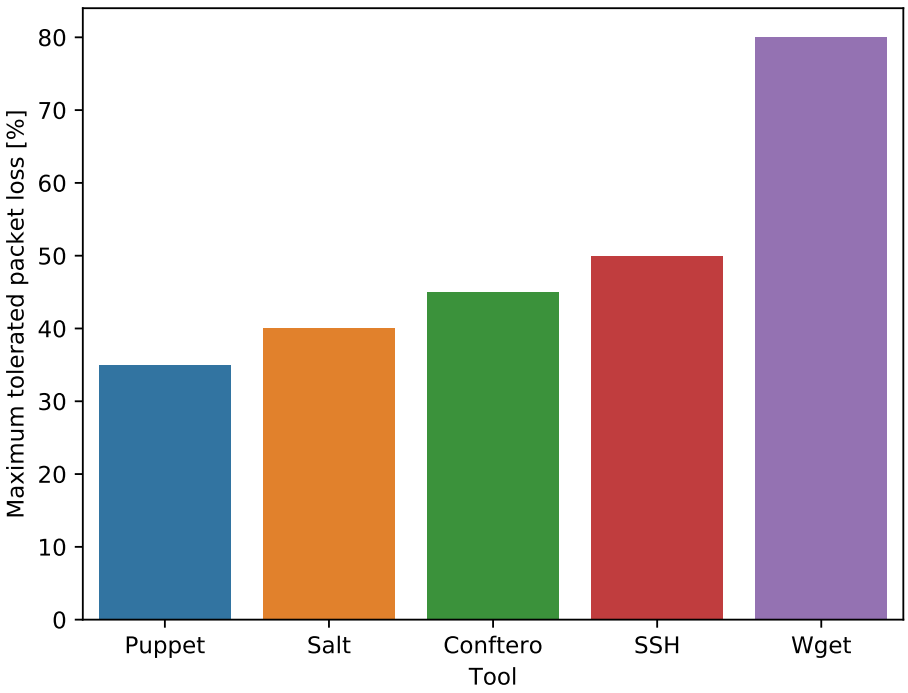

Figure 15: Maximum loss tolerated by each tool for 1 MB transfer

Wget, a popular HTTP download tool, was surprisingly resilient to packet loss. In the test, it succeeded even at 80% packet loss. Wget was better than the

other tools at all levels of packet loss. Compared to the CM tool Puppet, wget survived a 120% worse packet loss (80%/35%-100% = 120%). Figure 14 shows a major difference in transfer times between wget and other tools at all levels of packet loss (except 0% loss). For example, at 45% packet loss, Conftero took 528 s and wget took less than five seconds - a thousand percent difference. As Conftero was the most resilient CM tool in this test, the contrast with Puppet and Salt is larger. At 55%, wget's median transfer time was still under 5 s, while all other tools failed.

The results with wget seemed surprising, so they were manually verified. Multiple manual tests confirmed that wget is highly resistant to packet loss, even though it seemed to be reliable only at 70% packet loss in manual testing. Based on manual testing, the most common transfer failure occurred when opening the connection ("Connecting to..."). It is likely this could be changed by merely modifying the retry parameters.

Resilience of wget shows that HTTP can be highly resilient to network errors. Even though wget was much more resilient to packet loss than Conftero, the research prototype, this is most likely an implementation detail. As wget is a free, open source program, it is possible to simply replicate what it does.

The resilience of wget shows that the non-interactive transfer of files over HTTP is highly resilient against packet loss.

5.2.5.2 Packet Corruption Corruption is an adverse network condition where data is randomly changed before reaching its recipient. For the purposes of this experiment, the definition used by Netem kernel module was chosen. According to 'man netem', corruption is "the emulation of random noise introducing an error in a random position for a chosen percent of packets". The experiment was performed using the same environment and tooling as described above in "Packet Loss". For each level of corruption, the same 1 MB file was transferred.

The results of the packet corruption experiment are graphed in fig. 16. Maximum corruption tolerated by SSH, Conftero (cct) and Salt was 40%. Puppet tolerated 30% corruption. Similar to the packet loss experiment, wget could transfer the file at 80% corruption. Ping did not carry a similar payload, and it is only included as background information.

At all corruption levels (except 0%), wget was faster than CM tools or SSH. The speed advantage of wget was higher at worse network conditions.

Again, 'wget' was the most resilient against adverse network conditions by a significant margin. This indicates that HTTP transfer can be resilient in

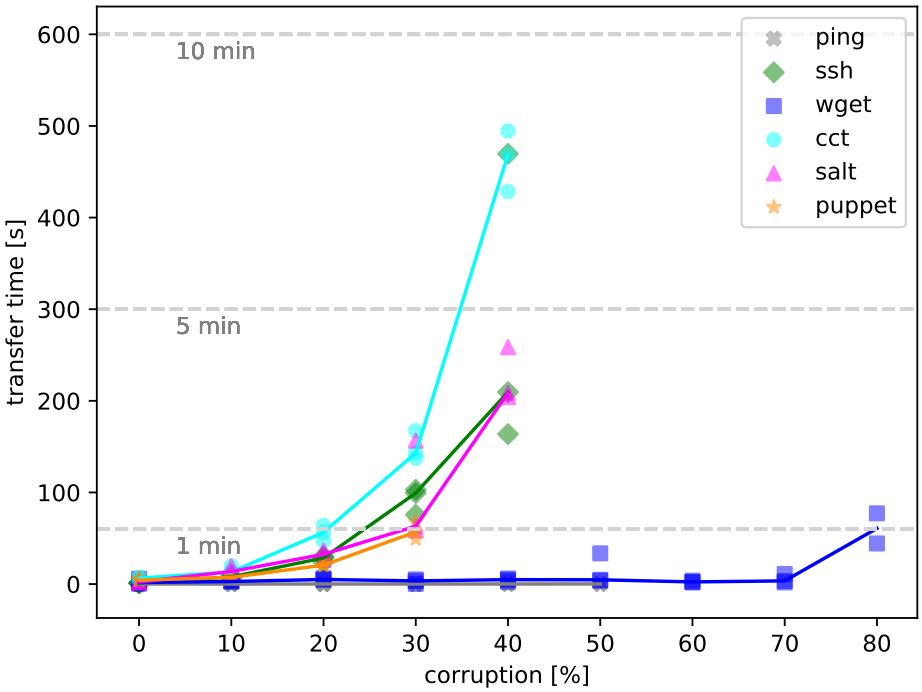

Figure 16: Effects of packet corruption to multiple tools

unreliable network conditions, and also that these gains could be brought to CM tools by watching what wget does.

5.2.5.3 Packet Duplication Packet duplication is an adverse network condition where some packets are sent twice. For this experiment, duplication percentage was the number of packets that are duplicated out of sent packets. The definition that came from netem module that was used for injecting the faults at the virtual network interface.

Many protocols number packets and automatically ignore duplicate packets. For example, TCP does this, and thus HTTP gains this feature as it works over TCP in the Internet protocol suite. Thus, it was expected that package duplication would have little effect on the tools.

The results were quite clear: package duplication had no effect on their operation. Even when all packets were sent twice (100% duplication level), all tools operated normally. The result is graphed in fig. 17.

5.2.5.4 Latency Latency is the delay before the first bit is transmitted to a recipient. Common wired Internet connections operating properly have low latency, for example around 20 ms RTT. In cell phone communications, latency of up to 500 ms is not uncommon.

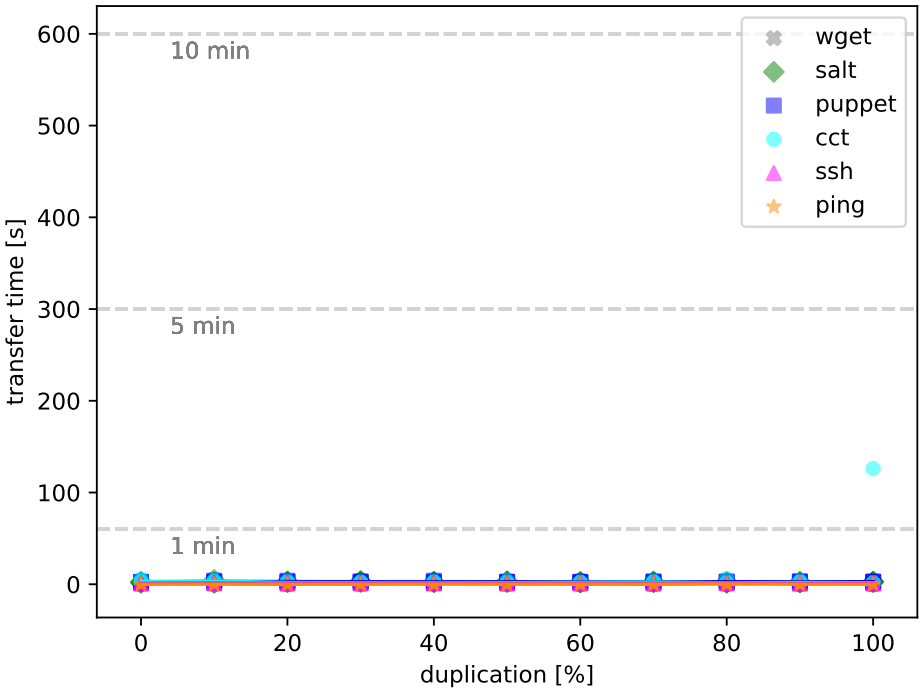

Figure 17: Effects of packet duplication to multiple tools

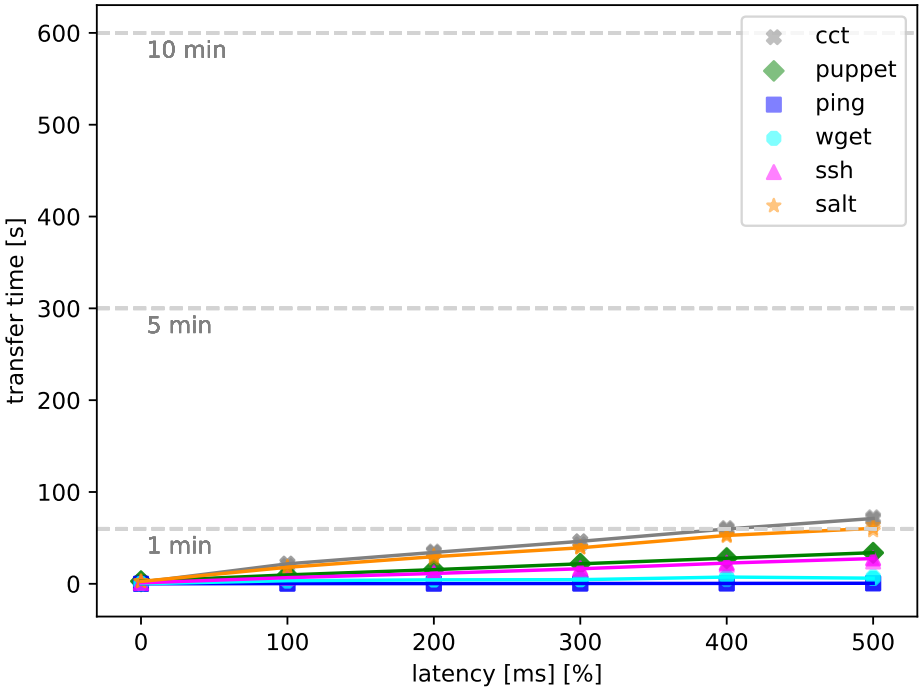

Figure 18: Effect of latency to multiple tools

Using a setup similar to previous experiments with packet loss, corruption and duplication, a progressively worse latency was created using Netem kernel module.

Latency does not need to correlate with throughput. It is possible to have a connection where it takes two seconds for the first bit to arrive, but the throughput is 10 MB/s. As CM usually operates non-interactively in production, it was expected that latency would not have a meaningful effect on the operation of these tools.

The effect of latency is presented in fig. 18. It shows that latency affects communications more than expected, but the effect is likely irrelevant for the non-interactive configuration of these systems. At all levels, Conftero (cct) and Salt performed the worst and - as with other tests - wget performed the best. As the worst transfer time was little over a minute, the only tools that would be affected would be those that are used interactively, namely Salt and SSH. Manual testing has revealed that these tools are more resilient to network errors when smaller payloads are used. Sending a single command (but not starting a remote session) results in smaller payloads.

Transfer of the file at reference condition (0 ms additional latency) took about 60 ms for wget and less than 3 seconds for other tools. An optimal tool might be expected to transfer the file below 600 ms even with 500 ms latency. The experiment shows that all tools performed worse. At 500 ms delay, transfer time was 7.6 s for wget (the best) and 71 s for Conftero (the worst). Thus, latency had a disproportionate effect on all the tools tested, even though this effect was insignificant for the purpose of these applications.

5.2.6 Comparing Memory Consumption and Resiliency Under Load

To compare and contrast capabilities of the Hidden Master and the research prototype to industry leading solutions, a new test was developed. Earlier in "Load test" chapter, the research prototype was tested by itself. This subchapter performs a comparison test between Conftero, the research prototype, and two industry leading tools - Salt and Puppet.

Load test indicated that the main research prototype is highly resilient to a high load. Even with a meager 1 GB of RAM on the intermediate courier / drop host, it could serve thousands of agents with a large (90 MB) encrypted catalog. These promising results could be achieved with just a single host, even though the hidden master architecture could be expected to scale near linearly by adding couriers and drops.

The use of standard HTTP as the transfer protocol in the last hop to agents

and the possibility to drop the requirement for near real time interactive communication between master and agent meant that standard web load testing tools, such as Apache Bench 'ab', could be used for load testing the Hidden Master architecture and the research prototype. In contrast, Salt uses ZeroMQ, a networking library that implements an asynchronous concurrency framework, and it is not obvious if a load of multiple nodes could be simulated with a single application.

To create a comparison, a combination of a full virtual machine (VM) and multiple light containers were used. The VM was a VirtualBox machine controlled by Vagrant IaC tool, similar to that used in earlier tests. It was expected that CM servers would be able to serve a large number of agents, and the RAM memory requirement of a full VM would pose too low a limit on the simulated load. For this purpose, light containers were required.

Initially, Mininet ('mn') was tested for this purpose. Mininet provides a combination of software defined networking, light virtualized hosts and a Python programming interface. Despite my efforts, I was not able to get industry leading agents to reliably run on Mininet containers, and it was not possible to separate real limitations of these tools from challenges caused by the testing environment.

Docker, a popular container environment, was used for containerization. A simple container based on (Debian 11) bullseye-slim was created with the required agent. To generate multiple similar agent environments, Docker images were spun with Bash. Compared to VMs, containers use much less RAM. They achieve this by sharing more with the host OS at the cost of creating a less separated environment. This approach made it easy to generate a large number (e.g. a hundred) containers on a regular off-the-shelf desktop computer.

The server machine was fully virtualized. Here, the server was the machine directly contacted by agents. In the case of Salt and Puppet, this was the master. In the case of HM, it was the courier/drop. To prevent running out of test load generation capacity and limit time required for testing, the RAM memory was limited to 512 MB. This was similar to a typical low end VPS in 2022.

The sample load test was the same as the one used with the research prototype earlier. It was a simple configuration with Firefox installer, a 65 MB binary. The use of an actual file instead of a dummy file consisting of zeroes or other repeatable content was to prevent unrealistic compression, as at least Conftero automatically compresses the payload before encryption.

Measuring RAM consumption is not as simple as it initially seems. Due to shared libraries, it is especially difficult to measure RAM taken by a specific application. Instead, a new server virtual machine was used for each test. To evaluate RAM usage, the amount of available RAM was measured with `free --mega`.

```
vagrant@m:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           472Mi        60Mi        90Mi       0.0Ki       322Mi       400Mi
Swap:            0B          0B          0B
```

Figure 19: Amount of available RAM memory in sample output of 'free -h'

Modern Linuxes attempt to make use of all available memory by using idle memory as disk cache to provide a possibility for large speed advantage if the same information is requested again. As this RAM disk cache is immediately discarded when memory is requested by programs, the amount of disk cache in use is irrelevant for RAM usage. Thus, free memory was estimated from "Available" memory reported by 'free'. Sample output of 'free -h' is presented in fig. 19.

The units reported by 'free -h' are 2-based units instead of 10-based units, e.g. mebibytes (1024^2) instead of megabytes (1000^2). Even though the difference between the units is just a few percent here and thus irrelevant in the context of these tests, the output was changed to 10-based system with '--mega' to make comparisons between tests easier.

During the tests, the amount of free RAM on the host OS was followed to avoid any unwanted effects on the tests. The host OS had available memory during all tests.

5.2.6.1 Salt Load and Memory Test The results on Salt test are in table 53. The test environment is summarized in table 54. For each level of agents, the number is the total, cumulative number of agents. For the test, the new agents were created on the host OS using Docker, accepted on Salt master, and sample payload was sent to them using 'sudo salt "*" state.apply'. The previously deployed payload was removed and agent caches cleared.

Table 53: Salt memory consumption and resiliency under load

Agents	Available RAM	CM RAM use	Comment
0	419 MB	0 MB	Initial state before installing CM tool
0	221 MB	198 MB	After installing salt-master
25	163 MB	256 MB	
50	139 MB	280 MB	Salt reported run time 83 s.
75	OOM-killed	OOM-killed	Failed, salt-master down.

The salt-master could handle 50 agents. At 75 agents, salt-master failed to apply changes. Salt command exited after 41 seconds displaying failures on multiple agents. After that, 'salt' commands could not reach the master. A examination of the logs showed that salt-master had consumed all available RAM and had requested more. Consequently, it was reaped by OOM-killer, the Linux out-of-memory killer.

```
$ sudo systemctl status salt-master|grep Active
     Active: failed (Result: oom-kill) since Sun 2022-09-11 [..]
```

We can conclude that on a machine with 512 MB RAM, the tested version of salt-master could serve 50 agents, but not 75 agents.

Use of swap would prevent immediate reaping by OOM-killer, but could be expected to make the system very slow. If working on a realistic time budget such as the 15 min budget used in load tests of the research prototype, swapping would not be likely to provide much improvement. In the case of trashing (continuous swapping and de-swapping processes), slowness could also make the system very hard to control without using external tools to reboot the whole server machine.

Table 54: Salt load and memory testing environment

Purpose	Software	Version
OS	Debian	11 Bullseye
CPU architecture	amd64	
Virtualization	Virtualbox	5.2.34-dfsg-0~ubuntu18.04.1

Purpose	Software	Version
Virtualization	Vagrant	1:2.2.1
CM	Salt-master	3002.6+dfsg1-4+deb11u1

5.2.6.2 Puppet Load and Memory Test Puppet is one of the industry leading CM tools.

Puppetserver (master daemon) failed to install in low-memory conditions. Puppet did not work in a low memory machine even when installing a newer version outside official Debian repositories and following advice from the installation guide. This was the case even though the low memory VM met the required minimum requirements of having 512 MB of memory. When performing the earlier tests on adverse network conditions, changing to a VM with 4 GB of RAM resolved the issue and started Puppetserver.

It is not impossible that Puppet could have a threshold memory requirement and it could serve many agents after that. But within the limits of this test, Puppet did not succeed.

5.2.6.3 Conftero Load and Memory Test The advanced research prototype Conftero was tested using the same setup as the Salt test above. The master and the courier/drop were on a virtual machine. The agents (slaves) were on Docker containers to be able to create more agents with limited resources.

First, a clean VM was set up. Then, Conftero master binary 'cct' version 0.1.47-alpha-20220807-1045 was copied, and a new Conftero repository was set up with './cct init .'. A new campaign for courier/drop called "cudo" was created and deployed to the server VM using './cct sshapply'. This installed Apache2 web server and PHP support to the server. Courier and drop were set up to this host and simultaneously verified using './cct newcourier' and './cct newdrop'. To simplify configuration, the master was also running on the server machine. Due to the HM architecture, this did not consume any memory (0 MB) when the administrator was not using Conftero to change or view the agent (slave) configuration.

The results are listed in table 55. Compared to Salt, memory usage of the idle server was very low. With no agent, Salt used 198 MB, and Conftero courier / drop used 13 MB. This is a low value in the absolute sense and represents only 7% of Salt memory usage. Because the value was so low, it was manually verified that the web server was indeed running and serving pages over HTTP.

Memory taken by serving Conftero agents was quickly reclaimed. This is in contrast with Salt, where memory used by serving agents seemed to be

permanently in use. This could be due to the different CM architectures of these tools. To find out the worst memory usage, Conftero courier/drop was monitored by running 'free' every 5 seconds in a loop.

To keep the load up, previous agents were deleted and all agents regenerated at each level. This made it simpler to make all agents contant courier/drop in a short time frame.

Table 55: Conftero courier/drop memory consumption and resiliency under load. Worst momentary values in parenthesis.

Agents	Available RAM	CM RAM Use	Comment
0	419 MB	0 MB	Initial state before installing CM tool
0	406 MB	13 MB	After installing courier/drop
25	403 MB (361 MB)	16 MB (58 MB)	
50	390 MB (396 MB)	18 MB (23 MB)	
75	402 MB (383 MB)	17 MB (36 MB)	Starting all agent containers took 6 min
100	399 MB (340 MB)	20 MB (79 MB)	From 100 agents, tests done with 'ab' and download only
500	403 MB (273 MB)	16 MB (146 MB)	
1000	404 MB (287 MB)	15 MB (132 MB)	Took 74 seconds
2000	395 MB (284 MB)	24 MB (135 MB)	Failed: connection reset by peer

Starting the Docker containers in a loop took some seconds per container. As the RAM usage of Conftero seemed low, it made sense to remove any sources of errors. Also, for high loads the capacity to create and simultaneously run Docker containers would run out.

HM architecture allows the use of standard file transfer protocols. Here, the use of HTTP allowed the use of standard HTTP load testing tools. To simulate higher loads, Apache Bench 'ab' was employed. Conftero was tested earlier with this tool using a larger VM in chapter "Load test". From 100 agents on, all tests were performed with ab. In ab tests, only a downstream (agent download) test was performed. For all ab tests, concurrency level (-c) was set to the same as the number of requests.

At 2000 simultaneous requests, the test failed: "[ab:] apr_socket_recv: Connection reset by peer (104)". However, Apache could serve requests immediately after that.

Conftero could serve 1000 simultaneous requests, over 10 times the amount of requests served by Salt. Tests over 100 simultaneous clients were performed with 'ab'. This only tests downstream (agent download), which could make Conftero appear better. Downstream has almost all data transferred (90 MB vs 2 kB), but agent uploads use dynamic script on the server. Ab is designed to send requests exactly at the same moment, creating higher momentary loads than other tools, and making Conftero look worse. Higher results could likely be achieved by using a high number of requests (-n) with lower concurrency (-c).

5.2.7 P2P Operation in Shattered Network

When a connection to the Internet is not available, individual hosts could still reach each other. This is easy if the hosts are in the same LAN. This becomes more difficult if the hosts are moving, and wirelessly connecting to different access points (AP) or ad hoc WLAN networks. An additional management challenge is created if this would require multiple separate backup servers to be configured in each additional network. In this work, I will call a network *shattered* when the nodes are not interconnected, but can only intermittently reach other nodes, possibly in an unpredictable manner with some nodes never able to reach some other nodes.

The Hidden Master architecture removes the requirement for near real-time interactive master-agent communication. This allows the master-agent transfer layer to be swapped cheaply. Layers and responsibilities of the HM architecture are listed in table 27 and further described in the chapter "Layer Model of the Hidden Master Architecture".

The use of peer to peer (P2P) architecture allows all nodes to distribute files. In typical implementations, the direction of file transfer is not mandated by the hosts working as the server in the client-server model (opening a listening port). This is mandated by widespread use of firewalls and NAT in typical

networks, which prevent many nodes from publishing a listening port, and thus from acting as the server in the client-server relationship. In P2P architectures, nodes can connect to each other in an opportunistic manner, transferring data when connection arises, and can avoid dependency from any one connection between nodes.

Leading configuration management tools, such as Puppet, Salt and Ansible, assume direct, two-way master-agent connection. They do not provide support for P2P architecture. Due to the connection requirement, swapping transfer protocols could be complicated. Leading configuration management tools are discussed in Literature review: Leading Configuration Management Tools.

5.2.7.1 Moving P2P Nodes without the Internet, Couriers or Drops

The scennario for the experiment is presented in fig. 20. Nodes are represented by circled letters: m for master, s1 and s2 for two agent nodes. The larger colored dashed circle around a node represents the range of node radio transmission. This P2P configuration does not need or have any intermediate courier or drop nodes. Instead, all nodes opportunistically transfer data when network situations permit this.

The scenario displays a simple shattered network. At time t1, represented by the box numbered 1. in fig. 20, master m and agent s1 are within radio range and can exchange data. Another agent, s2, is out of range of both m and s1, and cannot exchange information with any other node. Agent s2 could also be powered off.

At time t2, represented by box 2 in the diagram, agent s1 has physically moved. Now master m is out of range of both nodes. Agents s1 and s2 are within range of each other and can exchange information. It should be noted that master m and agent s2 are never in the range of each other in this scenario, and cannot directly exchange any information. The situation is the same even if m and s2 are never on at the same time.

The experiment will test if s2 can receive and apply configuration from the master m, without direct connection, without intermediate hosts (couriers, drops) and without a pre-planned network structure.

In addition to radio based scenarios presented above, multiple other example situations could require similar a network architecture. During power outage, many computers keep running on their own batteries, such as UPS, laptop batteries or backup power available in the local building. Wired upstream network connections and their active devices are obviously not powered by these local backup power solutions. Using the same P2P backend Syncthing, the hosts will automatically exchange information when there is an Internet

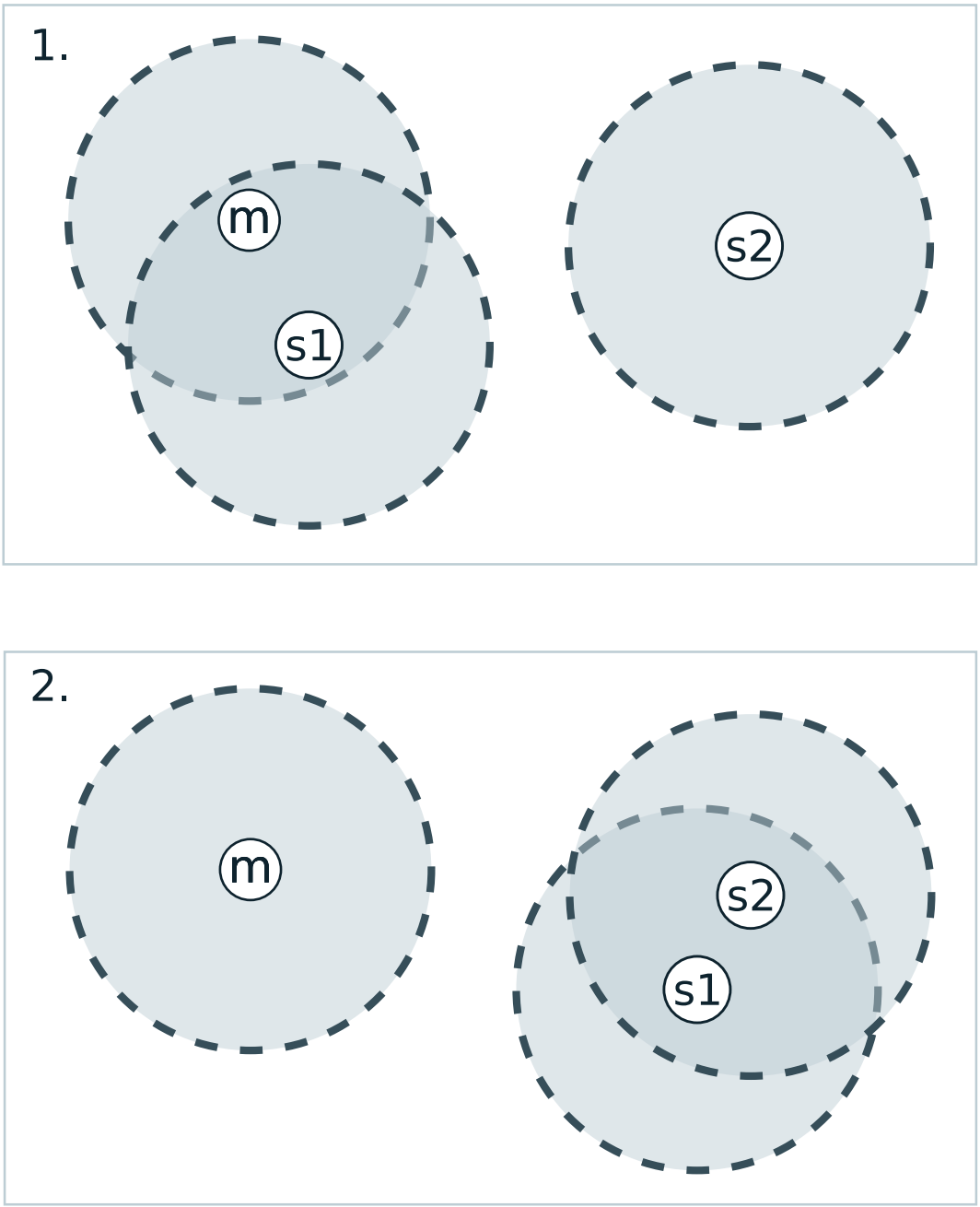

Figure 20: P2P transfer with moving agent

Protocol (IP) connection established between them.

Such a connection can be created by LAN switches, WiFi networks, and in many cases also by directly connecting the computers using a standard Ethernet cable without any active devices in between. Depending on the operating system, locally connected computers could automatically negotiate the linking of local addresses and establish a network between the computers (Cheshire, Aboba and Guttman, 2005). Such a local connection could be a regular Ethernet cable connecting the computers. Practical implementations are available for all major OSes: Avahi for BSD and Linux, systemd-resolvd for Linux, Bonjour and bootp for Apple and Winsock in Windows.

This scenario was only tested with Conftero, as the leading CM tools do not provide the required features for this.

5.2.7.1.1 Limitations The range of radio transmission between nodes is considered exact in this experiment. Nodes in range can transfer information, whereas nodes out of range cannot. In real life, the range of radio transmission is affected by multiple factors, such as terrain, multipath propagation, radio environment and in longer transmissions, radio weather. Also real radio transmissions degrade before disappearing to noise floor. These effects were not simulated in this experiment as the effects of adverse network conditions were tested and compared earlier.

Only downstream transfer of encrypted catalogs from the master to the agent was tested. Considering that both catalogs (master to agent) and reports (agent to master) are simply encrypted files, testing communication two ways would likely not bring much additional information.

The applications in the test environment were configured mostly manually. This was done both to simplify the experiment and to concentrate the test solely on swapping the transfer layer. Syncthing, the P2P backend, was allowed to exchange its own keys while all nodes were interconnected. For large real life networks, these steps, including the back end key exchange, should be automated using a CM tool such as Conftero.

5.2.7.2 Initial Configuration Three VirtualBox VM nodes (m, s1, s2) were defined using Vagrant IaC tool. Each node had only 512 MB of RAM, and ran Debian 11 Bullseye Linux.

Syncthing was manually installed with package manager (apt) and started ('nohup syncthing –gui-address=0.0.0.0:8384 &'). As the test nodes did not run a GUI, the web interface was used from the host OS using port forwarding, which made it necessary to bind Syncthing web GUI to serve all IP addresses (0.0.0.0).

Syncthing was configured manually, allowing all nodes to exchange Syncthing keys while they were still interconnected. Some Syncthing configuration was done to reduce the waiting for timers, and to prevent P2P backend from "cheating" by connecting to upstream Internet servers. To reduce waiting, folder rescan interval was set to 30 s (from one hour) and the reconnection interval was set to 10 s (from one minute). To prevent support from servers on the Internet, the following were disabled: relaying, global discovery and NAT traversal. All three nodes were paired through the Syncthing web interface (port forwarded to host OS), and the default folder was shared among them.

Conftero 'cct' was copied to master node m. This single binary can build all other components and generate all files required. It has no dependencies (apart from glibc), so no other software or libraries were installed. A new project was created using './cct init .'. Couriers were defined in master "cct.json" configuration. Usually couriers in HM are untrusted intermediate hosts running a web server. Instead, I defined couriers with file:// protocol. A new campaign "peered" was generated for s1 and s2 ('./cct newcampaign peered'). Agent installer binaries with campaign keys were generated ('./cct update -a'), copied to agent nodes s1 and s2 and installed on each agent node ('./ccta install'). During the install, Conftero automatically applied the configuration that was the most recent when installers were created. Conftero gained persistence using a cron job. Installation was verified by looking at the files created by the initial configuration ('/etc/conftero-managed'). To speed up future configuration, the installer was removed from the catalogs with './cct clean'.

5.2.7.3 t1 - Only Master m and Agent s1 Available At time t1, only master m and agent s1 were interconnected (fig. 20). Agent s2 was shut down ("vagrant halt s2"), so it was of course also not connected to any other node.

A new configuration on the master was created. This was done by modifying the target contents of /etc/conftero-managed, defined in module "agent" in campaign "peered". Both agents s1 and s2 are members of "peered" campaign. The modified campaign was uploaded to couriers with './cct sync'. This experiment used file:// protocol (instead of SSH and HTTP), so the encrypted catalog and counter file were simply copied (by the cct command) to the Syncthing default folder /home/vagrant/Sync/.

When taking a remote control connection to agent s1 ('vagrant ssh s1'), it could be seen that the contents of the test file /etc/conftero-managed were automatically modified as expected. This also showed that the transfer of the encrypted catalog using the swapped Syncthing P2P backend worked as expected in the easy case where both nodes could see each other on the network.

5.2.7.4 t2 - Only Agents s1 and s2 Available

At time t2, only agents s1 and s2 were on the network (fig. 20). Master m was no longer on the network. During this stage, we could see if the network was able to transfer instructions to agents during unpredictable changes in a shattered network.

To move to t2, master node m was shut down ('vagrant halt m'). After m was verified to be halted, the earlier powered off agent node s2 was started ('vagrant up s2'). As planned, Syncthing was started manually ('nohup syncthing –gui-address=0.0.0.0:8384 &').

Even with master m both unconnectable and powered off, agent s2 downloaded the encrypted catalog from another agent s1. The Conftero agent 'ccta' automatically read the instructions using the courier file protocol, reading the file directly from a directory on the local file system. The configuration was applied automatically. The result was verified by checking the contents of the test file /etc/conftero-managed. It contained the updated test string defined on the master at time t1: "This update was done on m when s2 was down".

In this experiment, Conftero successfully used a swapped P2P transfer and configured nodes in a shattered network.

5.2.7.5 Conclusions of Shattered Network Experiment

Unlike industry leading CM tools, Conftero and the HM architecture is resilient against network partitions. Conftero can be made to find a route around network problems and operate in unprepared and unpredictable network topologies.

The use of P2P backend is just one option to improve Conftero resiliency. The Hidden Master architecture gives up on the requirement of near real time direct two way connection between master and agents. This makes it possible to swap the HM transport layer, treating the transport tool as an untrusted black box. Basically, any method that can transfer files can be used. Purpose made tools, such as Syncthing for P2P communications, might be expected to solve their specific problem domain better than an addition coded as a side feature to a CM tool.

5.2.8 Air Gapped Operation

An air gapped network is not connected to other networks. At some point in time, some information must be brought to the air gapped network for the computers to have any OS and software to run. When software updates or new configuration is brought to these computers, it requires someone to be physically present. If trusted experts travel to each of the target computers, this can be expected to be more expensive than using less trusted and uneducated workers.

The Hidden Master architecture used in Conftero, the advanced research prototype, allows easy swapping of transport. As the transfer layer consists of encrypted files to be delivered over untrusted transport, any untrusted method of transferring files could be used as a black box.

The setup used was the same as in the P2P Operation in Shattered Network experiment. USB drive mounting was simulated by copying the files on the expected mount point location on agent node s2. The Syncthing daemon, required for the network transfer of Conftero catalogs, was stopped so that s2 could not communicate Conftero catalogs or reports with any other node.

5.2.8.1 Limitations Only downstream, master to agent, courier communication was tested. The transfer of encrypted reports upstream, from agent to master, could be expected to work in a similar fashion.

Automatic unmounting of the USB drive was not tested. Depending on the automounter used in the target system, the drive should be logically unmounted ('umount /media/usbDrive/') before physically disconnecting the drive.

Physical access to USB ports is lower privilege than having a user on the system. However, default settings in typical OSes are not enough to protect it. At least USB Human Interface Devices (USB HID) such as keyboards and mouses should be limited if the employee connecting the USB drive is not trusted to have an account. For advanced attackers gaining physical access to USB, a risk of performing memory reading attacks might need to be considered.

5.2.8.2 The steps of the experiment

- A new file:// protocol courier for a local USB flash drive was added on master m to cct.json.
- The operator of the system produced an encrypted catalog on the master node. A new configuration was created by modifying the target contents of sample file "/etc/conftero-managed". This created two files, the counter file ("counter") and the encrypted catalog ("55BD1DD5520BAE27"). With './cct sync', the files were copied to a USB flash drive.
- The drive could be given to an untrained employee. This was simulated by transferring the files to a target node s2 using a shared folder in the emulation environment.
- The untrained employee would travel to the air gapped computer (agent s2), connect the USB drive for two minutes, and then disconnect it. This was simulated by copying the files to the location where an automatic mounting system shows the contents of the USB drive, /media/usbDrive/.
- After less than a minute, Conftero agent ccta had read, verified, decrypted

and applied the instructions. This was verified by reading the sample file. Reading the file 'cat /etc/conftero-managed' revealed the test message "This update was brought to air-gapped m2 with a USB flash drive."

5.2.8.3 Conclusion of the Air Gapped Experiment This experiment showed that Conftero can deliver instructions to air gapped computers using movable media and minimal user interaction with the agent machine.

Together with P2P Operation in Shattered Network experiment, this also shows that as the Hidden Master architecture does not have the requirement of direct, synchronous agent-master connection, transfers can be easily swapped. The swapped transport can be considered simply as an untrusted black box.

5.3 Case Studies

These case studies were done to validate the model by testing the research prototype in the field against business requirements. In contrast to Empirical Validation in Emulated and Simulated Environment, these studies use the research prototype to reach some external goal. Instead of a laboratory environment, these case studies were run in real environments being used in production.

These case studies were done to complete objective 6: "Validate technical benefits and potential business benefits in two case studies". Completing the objective provided an answer to RQ6: "What utility do the models and the research prototype provide when run in field environment with business requirements?".

5.3.1 Conftero in Computer Exercise Evaluation

A final computer exercise was evaluated using conftero, the research prototype.

The master and agent version was detected with the Conftero builtin capabilities. This information was available in machine readable format, because Conftero performs resource abstraction similarly to other modern CM tools. A summary of the runtime environment on both master and agents is collected in table 56.

Conftero implements the Hidden Master architecture. This includes the capability to connect to the network only briefly, and then spend unlimited time offline to analyze the state of the agents and develop a new configuration. The new configuration can be uploaded to the courier at any time, including time when agents are offline. A more detailed explanation of the hidden master architecture is in Designing Hidden Master Architecture.

This allowed results to be analyzed afterwards, even after most of the agents (including their whole operating systems) had been removed. Both exercise scoring (for course evaluation) and analysis for this study were done afterwards, based on the reports stored by Conftero.

Stored results were analyzed using command line. Initial results were collected using Conftero cct master command for the master, and stored JSON reports for agents. To obtain the data summarized here, the results were cleaned using CLI pipelines using standard Linux commands including Python and grep. Conftero contains rudimentary CLI functions to display agent statistics, but they are more suitable for a dashboard-like overview than the exploratory analysis of agents.

Table 56: Qualities of master and agents

Quality	Value
Masters	1
Agents (n)	23
Master version (cct)	0.1.47-alpha-20210316-2009
Master runtime environment	Go 1.15.2 x86_64, Ubuntu Linux 18.04
Agent version (all agents)	0.1.47-alpha-20210316-1732
Agent Linux Distributions	Multiple Ubuntu and Debian versions
Agent OS and CPU Architecture	Linux amd64 (x86_64)

There were 23 agents, of which about 80% ran versions of Debian Linux and the rest ran Ubuntu. All hosts used amd64 processor architecture, which was expected as only amd64 binaries of Conftero were provided. Qualities of the master and agents are listed in table 56 and OS distribution is listed in table 57.

```
ls reports/|nl
./cct --versio
grep -ir CctVersionInfo reports/
```

Listing: Commands to evaluate test environment on agents and master

Table 57: Distribution of agent operating systems

Count	Percent	OS version
13	57 %	Debian Linux 10.8
6	26 %	Debian Linux 10.7
3	13 %	Ubuntu Linux 20.04
1	4 %	Ubuntu Linux 18.04

Count	Percent	OS version
———	———	———
23	100%	Total

Computer exercise evaluation emphasizes some challenges to the configuration management software. When exercises are scored, agents cannot be trusted. When reports are fetched from agents, their identity should be verified so that a dishonest system operator cannot overwrite reports from other agent machines. Due to the instability caused by novice system administrators working under pressure, it is not uncommon that machines crash or become unusable. In this regard, one machine became unusable near the end of the test.

Due to sudden changes caused by the corona pandemic, trusted Internet visible server infrastructure was not easily available for use in the test. The environment had multiple NAT in the path between most master-agent pairs. Both the master and agents were behind NAT, and the agent in the virtual machine was in some cases NAT:ed again by the host OS. The untrusted courier (master to agent) and drop (agent to master) were on the same computer, cheap commodity virtual private server. The virtual server was rented for 5 USD per month with no other costs and a possibility to cancel after the first month, and this was a typical price and agreement at the time of the test.

It was found that all but one candidate (22/23) could successfully install Conftero. One virtual machine was running 32 bit (x86) guest OS, for which Conftero did not have binaries available. Even though it is not difficult to compile 32 bit binaries, they would likely have very little use as 32 bit non-virtualized computers are uncommon. The situation was fixed by installing a 64 bit (amd64) guest OS.

One of the contributions of this work was the use of general purpose language and imperative instructions to perform idempotent infrastructure as code (IaC) configuration. Conftero had a builtin Python dialect in the static binary, which implemented the IaC using the novel resource dependency model presented in this work. In the exercise evaluation, it was found that this language could be adapted to opportunistically use software and tools already existing in the agent OS to collect data about the agent environment.

Conftero has an advanced key exchange, including campaign keys for pre-emptively distributing keys to an unknown number of machines provisioned in the future. However, when the agent binary installer was downloaded by the candidates, it was not protected by cryptography apart from the web server TLS and certificates. This might warrant further consideration for similar use

cases, even though reliable first key exchange probably requires active human participation at some point. This problem does not exist when the installer is included in OS provisioning, which is a more suitable method for enterprise environments.

Summary of findings

- Using the hidden master architecture, Conftero could operate in double NAT (master and agent) environment without trusted infrastructure. Multiple different networks and setups did not cause any problems.
- Conftero worked reliably in different versions of Debian and Ubuntu Linux
- Conftero did not work in 32 bit x86 architecture, because such binaries were not compiled
- Installing Conftero worked by downloading the binary and following three-step instructions. The authenticity of the binary had to trust limited external mechanisms.
- Report collection through the back channel worked
- The Python based imperative, idempotent configuration language could be used to define report collection.
- The language embedded in static Conftero binary could be adapted to opportunistically use the tools already installed on agent machines to collect data about the environment
- Reports could be analyzed without Internet connection, and when agents no longer existed.

5.3.2 Conftero Deployed to Company X Production Environment

To evaluate the business value in a challenging real-life environment, Conftero was used by Company X in the production environment of its client. This case study was undertaken to validate the model by testing the research prototype in the field against business requirements (RQ6).

As described in Methodology, the original goal was to perform limited testing in the Company X test network, separate from the production environment. After an initial evaluation of the test network, Conftero was seen as safe enough to run on production systems provided that a suitable client could be found and precautions to mitigate risks were developed. The nature of Conftero requires full (root) access to all aspects of the system, and it is designed to make major changes to multiple machines without human intervention. Multiple administrative and technical precautions were taken to control the risks, making all parties confident that the testing could be safely performed in the production systems. Thus, the goal of this case study was upgraded to perform testing in

client production systems.

Company X is a small smart building vendor established in its single country market. It has its own IoT architecture platform with AI prediction. Two of the largest grocery store chains (~80% of the market) in Finland use their cold chain or delivery solutions, and their AV solution is used in over 1500 rooms across multiple organizations.

A suitable client was found. Client Y is one of the largest universities in its country, employing thousands of personnel, teaching over 15 000 students and ranking under 200 in both the QS World University Rankings and Times Higher Education World University Rankings. They kindly allowed the test to run in a single location multiroom AV system.

Limitations of the case study include the number of organizations studied and the scale or configuration management deployed. Case studies were limited to two organizations. Ethics and business continuity considerations prevented running the research prototype on large scale production deployments, critical infrastructure or in systems responsible for human health. The first small case study was performed internally in the organization employing the researcher. The second main case study did not have this limitation.

5.3.2.1 How the Study was Conducted This case study used expert interviews, technical tests and observations of the deployment of the system in the environment of the two organizations. The main deployment phase with Client Y was performed in June 2021, with interviews, scoping and development performed earlier and a followup check done later.

Expert interviews were conducted with the leadership of Company X together with scoping the project. Interviews also included a senior technical expert, head of department and a junior system operator. Opportunistically, comments and questions were noted when programmers outside the target group interjected into the discussion when working physically in the client's premises.

Observation of system deployment was undertaken with a junior system administrator during the deployment phase, using video conference and ssh remote connection while developing the configuration for Client Y AV deployment.

Technical tests and analysis of the systems were performed by the author using a remote shell connection to the target system.

5.3.2.2 Description of the Environment Company X has developed its own IoT and AI platform. The parts used in this study were similar to typical modern server environments in most aspects. They ran amd64 Linux with

fast, continuous network connections with ample RAM. Compared to some other server deployments, the rate of software updates was high. The ability to install additional administrative software was limited both technically and due to security policies. The main technical limitation for software installation to host OS was heavy use of containerization for software deployment. Continuous security updates also affected initial deployment of the research prototype, but made it possible to see how assumptions of the research prototype were held in unseen environments. Company X's platform already had tools to administer, monitor and update their systems. To test the research prototype in this case study, some of these tools were not used, and Conftero was tested instead.

Heavy use of containerization was a surprise, and it made possible to test the non-DSL approach to develop infrastructure as code for systems that were not available when Conftero was designed. Stringent security requirements created additional integration points. On one hand this limited testing of some network transports, but on the other it showed that the hidden master architecture's freedom from real-time two-way connection makes it malleable to surprising network structures.

The level of detail in describing company X systems is limited by business secrets and the security requirements of the company and its clients.

Client Y ran a typical large, partitioned network with their own monitoring. They also set limits to what can be done in the network, mostly following regular security policies of Company X. Configured systems were in sole control of Company X. Apart from networking to target systems and policy decisions, Client Y computer architecture had little effect on the performance of the research prototype.

5.3.2.3 Scoping and Planning Phase Initial scoping meetings were held with C-level management, and they involved both the business and technical leadership of the company. Another technical meeting was held with just technical leadership.

During scoping, management pointed out the limits to this test. Tests should advance step by step, starting from nearly risk-free test environments and advancing towards greater risk. Plans to mitigate any risk to client systems should be done and accepted beforehand. Protection of client production systems was the main consideration. Company X was eager to test the research prototype and provide resources for testing. These resources included computers, access to the internal network, use of IoT devices and time from their personnel.

The legal framework for the testing was agreed upon. In addition to NDA and other typical project agreements, licensing was discussed. The current plan is

to release Conftero under a license meeting both the FSF Foundation's Free Software definition and Open Source Initiative's definition for open source, such as the GNU General Public License (GPL), but this was not decided on at the time. As the author of Conftero I confirmed that using it to configure, monitor or administer systems in a regular manner places no requirements on the licensing of these systems and no requirements on licensing the infrastructure as code configuration files. In the licensing phase, this could be done using a special exception with the GPL.

As software is a system product, all of its components must have compatible licenses if a derivative software product is created. Free licenses requiring strong reciprocity, such as the GPL, require derivative and combined work to retain the same license. This creates a challenge to programs whose main purpose is to create outputs that include parts of the original program. The GNU Compiler Collection (gcc) is one such example. As gcc compiles source code to executable binaries, such binaries can include code from the compiler. This is solved by a copyright holder giving a special exception to the GPL license of the program. (Välimäki, 2005) A similar approach could be taken to keep the licensing of Conftero clear.

5.3.2.4 Requirements for the Project Client Y's business requirement was the deployment of a multiroom AV system. User visible part contained multiple computers, audio and video outputs and inputs. The system required mobile or web authentication to use. For administration and security purposes, the system had remote control, monitoring, automatic updating and recovery features. These systems were provided by Company X's IoT platform. Thus, the requirements for the research prototype were to deploy, configure and control these systems securely and follow the policies of both Company X and Client Y.

Table 58: Requirements for Deployment

Req	Description
req0	Deploy multiroom AV system with authentication, monitoring, administration and remote control
req1	Control host Linux system to deploy containers
req2	Configure systems inside containers
req3	Orchestrate information between containers
req4	Set up and initialize the relational database
req5	Generate and manage multiple asymmetric keys
req6	Manage access control techniques, including the MAC security context

Req	Description
req7	Automatically deploy updated software versions
req8	Follow Company X's security policies (keys, networking...)

The requirements for this project are listed in table 58. The end product required by Client Y is listed as a single point req0, as many of these complex features are delivered by existing software. Other requirements req1-req8 deal with support functionality provided by Conftero, the research prototype. Many of these requirements were unexpected such that Conftero had never been tested with similar requirements.

Containers, such as Docker, Podman, LXC or Mininet, create a separate environment for a program. For example, a web server running in a container cannot usually access the host OS file system. For efficiency, containers share many resources with the host OS. The main shared resource is usually running the same kernel as the host OS. In this way, containers usually offer more efficiency but less isolation than full virtualization. Popular administration tools exists that are solely limited to containers (e.g. Kubernetes). Managing containers was not a requirement prior to this case study, and thus was not tried with Conftero before this. Requirements req1, req2 and req3 are related to containers.

Setting up a database (req4) is a common task in an system administration, and well supported by both Conftero and popular solutions in the industry. It usually follows a package-file-service pattern, with additional command run for loading the initial database from a text file containing SQL commands.

Generating and managing keys (req5) often differs depending on the tools to be used. Searches in popular search engines showed that it is easy to find incorrect and dangerous instructions for generating keys for use with CMS. For example, some instructions for creating Linux users result in md5crypt keys ("1") vulnerable to dictionary attack. In Conftero, generating keys was usually done running the same commands that the administrator would run when working manually, and then making these commands idempotent. This approach sidestepped the requirement to understand and research the details of cryptographic hashing techniques used by each targeted tool.

Modern Linux systems have added isolation techniques to adapt to the developing threat environment. A class of those tools provides mandatory access controls (MAC) and other access control related security policies. SELinux and AppArmor are the two leading tools in this area. SELinux is typically used with Red Hat family distributions (e.g. Red Hat, Rocky, Fedora), and

AppArmor in the Debian family (e.g. Debian, Ubuntu, Kali). These policies can limit access to directories, files and the network. They do not replace exiting isolation techniques such as users or capabilities, but instead further restrict access. Conftero had not been tested for controlling those access control systems before this case study. Controlling access control techniques is req6.

Company X kept its systems up to date, and had a high rate of updates. When they made updated versions of their software available, it was automatically deployed while minimizing negative effects on the continuous availability of the system. Some of these updates included container images. Automatically deploying detected updates for in-house custom software is listed as req7.

The IoT platform of company X is used in places where human health must be protected, such as cold chains for the two largest grocery chains in the country. Thus, they had multiple stringent security policies. Unfortunately, but for a good reason, some of these policies require network facing systems to minimize the attack surface and use mature software that is employed in production by thousands of companies. Obviously, a research prototype cannot meet these criteria. This requirement req8 was handled by using additional technical protection from the Company X platform, and adapting the network transfer of encrypted catalogs to this requirement. Additionally, the research prototype was not run in machines responsible for human health.

5.3.2.5 Precautions Even though I consider the security principles and standards of Conftero to be sound, the implementation is just a research prototype and cannot be on par with highly secure tools established in the industry. Mature tools, such as OpenSSH, OpenVPN and WireGuard have faced years of public scrutiny and penetration attempts in tens of thousands of production installations, and it is obvious that a new tool cannot meet that level of security. Thus, multiple precautions were taken to keep production networks safe and all parties informed.

Security precautions can be divided into scope, administrative and technical. The scope of the study was limited to production systems that could not cause risk for life or health, and did not contain classified or other high-risk information. Thus, control of cold chains for food was ruled out. As described in the technical precautions, the target systems did not have access to high risk information, such as large databases of PII or classified information. As with most projects with actual companies, systems did contain regular business information of a confidential nature.

Administrative approaches involved keeping all parties informed, making sure a skilled response was available and limiting the immediate effect of possible

incidents. All parties were informed, also in writing, and accepted the research. During the study, Company X personnel and the researcher (the dissertation author) were available to respond to any problems. The study was timed so that any downtime would happen when demand for these systems was low and alternative premises were likely to be available.

Technical precautions included using additional encryption and authentication systems, additional monitoring systems and containing systems in the network to limit possible lateral movement. Alternative transport for encrypted catalogs was used to benefit from the security of existing systems in the company. Initial feasibility tests were performed in a completely unrelated environment. Then key tests and practice was then performed on a separate but nearly identical test system. Finally, the study was undertaken in the production network.

The research prototype worked, and the study was completed without incident. At the end of the study, configuration was done with Conftero (the research prototype) and was left to run in the production network. The Conftero agent was removed from the target system, and keys with administrative access were invalidated.

5.3.2.6 Observation of Deployment Deployment was performed with a junior system administrator with some programming experience. Observation was performed over video conference during June 2021. This consisted of over 20 hours of participant observation, spanning 12 days. Participant observation allowed for the understanding of the thought process, goal setting and challenges related to installation, and made it possible to ask for clarification. When needed, the researcher provided additional information on any questions, noting the area of interest. Answering questions was also necessary in practice, as it was not possible to develop extended documentation or training material for the prototypes.

5.3.2.7 Results of Company X Field Study Based on observation, technical tests and expert interviews, the concepts underlying the research prototype could be validated. The research prototype Conftero performed well in this field study and was able to meet the requirements posed by the stakeholders.

The real life production environment provided an unexpected environment with multiple limitations. This is in stark contrast to the emulated test environments and field tests where the research prototype was the only application defining security policy and administrative practices. This had the greatest effect on network architecture, but also some challenges on integration points, namely initial deployment using SSH.

Working with system administration with no prior experience of either Conftero or other idempotent, infrastructure as code CMS put the non-DSL approach to the test. Conftero uses concepts that are novel in the context of CMS. The approach taken by Conftero is described in chapter Defining Configuration, and compared to a typical CMS approach in table 41.

As hoped for, the use of Python dialect made the system approachable and tempting. The system administrator could extrapolate from his prior Python experience. Especially notable was the lack of questions regarding flow control in CMS code. Some leading solutions, such as Puppet and Salt, require the system administrator to learn and adopt a completely new approach to flow control in the software. The non-DSL approach used in Conftero uses the same flow control structures as all common programming languages, such as Python, C or Java. The lack of questions on this area and the observed ease of use indicate that leveraging existing experience and conceptual knowledge might lower the bar for IaC configuration. It is possible that this simplicity would also reduce programming errors, but this was not tested in this case study and could be an area for further research. Interviews and the collection of opportunistic comments and questions showed that the use of Python dialect seemed quite obvious to the developers and system operators in this case study. It could be assumed that many fundamental structure questions were skipped because they were considered obvious in any Python dialect. This assumption is somewhat supported by observation of the use and implied understanding in more detailed questions. For example, all those interviewed seemed to (correctly) assume that the user of the system could employ conditionals and functions. Detailed questions included: “Can you use f-strings?”; “Can you import libraries?”; and “Can you read files?”. Some considered the lack of Python requirement surprising. Conftero achieved this by using Starlark library statically compiled into a Go binary.

Static linking showed both benefits and downsides in this case study. Static linking means that all dependencies of a program are bundled into it. This affected the choice of programming language, and thus was a major decision in the design and implementation of the system. Technically, the agent and master binaries had minimal dependencies to target stable OS APIs, as literal fully static linking (to unstable APIs) was expected to paradoxically result in less compatibility between systems. Static linking in Conftero followed this more practical and broad definition of static linking.

The freedom from dependencies turned out to be beneficial, as the system developed on Debian family Linux distributions (Ubuntu, Debian) worked out of the box on Red Hat family distributions. Due to the high rate of updates

and use of new versions of software, a large number of dependencies on libraries in target systems would most likely have reduced compatibility. By definition, a configuration management system is managing the configuration of slave machines. As this includes the versions of libraries included, it would make it difficult to use a CMS to recover from problems with libraries it itself is dependent on. There were no problems with installed libraries in the case study; this feature was not tested.

To consider the benefit gained from static linking, it could be compared and contrasted to a popular approach using dynamic linking. Conftero was developed in Go language, which can intrinsically create static binaries. Industry leading CMS software is coded in languages that - in practice - require dynamic linking. Puppet and Chef are coded in Ruby, and Ansible and Salt are coded in Python. Both Ruby and Python are evaluated at runtime, and could be considered scripting languages. They cannot produce real static binaries. This means that the CMS becomes tightly coupled and highly dependent on the environment it should be managing. The libraries included in Linux distributions change with time and also between distributions. During the work on this thesis it was observed that due to a lack of required libraries, one of the tools ceased to function. Some Python projects also get tightly coupled with specific minor versions of the programming language. An example (outside CMS) is the popular Python web framework Django, that requires Python 3.8 while some long term support distributions ship older versions of Python (e.g. Ubuntu 18.04 LTS ships Python 3.6.9).

A problem with a one of the libraries included in the research prototype was discovered. To make initial deployment easier, there is an optional feature to deploy the agent daemon directly to the slave machine through SSH. The included SSH library, golang.org/x/crypto/ssh, did not support the latest authentication algorithms, mandated by the latest OpenSSH server. This challenge was not shown in a slower moving environment based on the Debian stable. This challenge was independent of static or dynamic linking, as the bug was not fixed in the upstream library. A possible mitigation would be using the external 'ssh' command (OpenSSH client) for this. This would create a soft dependency for the command, as the lack of this command would not prevent use of any other features. Executing external commands is sometimes considered a less controlled and less stable approach than using an actual programming API. In this case, many features of any modern CMS are implemented by running system commands, so little additional risk would be created in this optional feature. As cryptography is an evolving field, this would make Conftero more future proof in case of any similar change. OpenSSH binary is available by default in popular Linux distributions in both Red Hat and Debian families,

and even in new versions of Windows.

Table 59: Findings in Company X case

Req	Benefits (+) and downsides (-)
Static linking	+ Support for different family distribution with much newer libraries.
	+ Works in environments with limited library availability
	- Requires working in more difficult language with worse libraries
	+ Easy to master different version campaigns from one master
Small non-DSL	+ Easy to start, Python experience applies
language	+ Flow control is familiar from other languages
	+ Easy to adapt to unexpected requirements
Non-realtime	+ Adapts to unexpected network architectures and security policies
communication	+ Easy to deploy Couriers (for downstream, master to agents)
	- Drops require a specific server environment (for upstream)

5.4 Expert Interviews

Business impacts of the key concepts were validated with a prototype demonstration and a related practitioner evaluation with six companies. After viewing a demonstration of the software and a presentation of the concepts behind it, practitioners answered a semi-structured questionnaire with both closed and open-ended questions. The questions can be found in Appendix: Questionaire for Semi-Structured Interview.

Expert interviews were undertaken to complete objective 7 “Identify and validate potential business benefits in expert interviews”. This provided an answer to RQ7 “What potential business benefits do experts see for the models and the research prototype?”. Other validation questions, RQ5 and RQ6, were answered earlier in this chapter “Findings and Analysis”.

Potential business benefits had to be analyzed indirectly using expert interviews to make the research possible. The wide deployment of a research prototype to real life production environments would be risky. Creating a production quality tool with high production requirements, then using it in a business environment and generating profit in excess of what established and mature tools could offer would also be unfeasible and out of scope of this study.

Interviews and related demonstrations were performed in Finland, in Finnish, using video conference and screen sharing. The interviews took place between May 2022 and August 2022. Six experts were interviewed, and it seems that a saturation point was reached.

To allow time for comments and questions, all expert interviews were performed individually. Each interview took a little over an hour. Interview methodology was described in the Methodology chapter. Methodology also includes formulation of the expected business benefits, that were used to create the questions for the semistructured interview (included as an appendix). Together with longer deployment case study, this evaluation helps understand the potential business impact in RQ6.

Configuration management systems, as discussed earlier, are a very high value asset to protect among the machines participating in configuration management. Questions about current challenges might also present employers in something other than the best light. For these reasons, the experts participated on the condition that individual answers could not be reliably linked back to each interviewee. To follow on this request, both the answers and the backgrounds of respondents are grouped by theme.

Companies where the interviewees worked included a large multinational IT consulting and software company; a market leading mobile operator in Finland; a cloud consultant owned by one of the world's largest IT companies; and payment a terminal operator owned by one of the largest digital payment operators in Europe. Current positions varied from the more technical (cloud security engineer, architect) to the more business orientated (security management consultant, cyber security operations expert, head of corporate services development, head of support). All but one had over 10 years of professional experience in IT.

Interviewees were chosen based on having a background suitable for this study, and recruited using the networks of the researcher. One of the interviewees was previously the team lead of a red team in Finland with a major security company. Red teaming meant simulating advanced adversaries against well secured and monitored networks in persistent campaigns spanning multiple months. Targets included critical infrastructure and financial companies. Another interviewee had protracted experience in protected and operated payment systems. One interviewee was lead support for a cloud provider, allowing unique visibility into the problems hundreds of clients meet when attempting to control and operate their systems. The cloud provider also offered API endpoints to make it easier for its clients to use CM on their cloud. All positions included the need to fit technology and development to business requirements. Multiple interviewees mentioned a lifetime computer hobby in addition to using computers at work.

Generalizing the results of the expert interviews is limited by the number and selection of the interviewees. The interviewees were all based in Finland and working in Finnish companies or in Finnish departments or subsidiaries of

multinational companies. To find experts with the required combination of experience (business, security, IaC CM), they were recruited using the networks of the researcher. There were just six interviewees, but smaller numbers are common in qualitative work. In this study, experts were allowed to express their views in free form during individual interviews which lasted over one hour each. This approach sought to obtain more in-depth and thought out expert views compared to group interviewing a large number of experts or performing a large number of short interviews. Where the experts' views seemed to be similar, it appears that a saturation point was reached.

Closed questions related to claims about the expected business benefits of the technical or architectural qualities of the same concepts that were tested in the research prototype. Closed questions took answers on a five point Likert scale: 1 "Strongly disagree"; 3 "Neither agree or disagree"; and 5 "Strongly agree". Despite this closed numeric answer format, interviewees commented on most questions, asked questions of their own and in some cases requested additional technical demonstrations of the software. The interviewees were allowed to provide as many additional comments as they wished, their questions were answered briefly and technical demonstrations were provided (within the limitations of the demonstration environment). These additional comments were recorded, and key comments are paraphrased below. One respondent would have liked to answer some questions with fractional or range answers (4-5 in three questions). Here the respondent was pushed to choose a single answer and provide possible additional free form commentary.

5.4.1 Thematic Analysis of the Interviews

To identify new and unexpected themes, a thematic analysis of the interviews was performed. This was done following the methodology described in "Methodology: Expert Interviews".

Initial codes were created based on interview notes. Based on these codes, initial theme maps were created. This resulted in two initial theme maps. The main initial theme map is shown in table 60. An alternative initial theme map was later discarded. It was based around main themes of "Higher level of abstraction", "Multiorganizational", "Change" and "Productivity".

Table 60: Initial theme map

Theme	Subthemes
Administrative and Organizational	Skill gap, complexity, managing risk, workforce, profit and loss

Theme	Subthemes
Legal and Contractual	Contracts and Laws limiting CM, Juristictions, Standards and certifications, Cross organizational work limited by policy
Technical	Challenge and potential future direction, Current advantange of CM...
Productivity	
Risk	

The main coding was performed using interview notes and inteview videos when available. After main coding and reviewing the interviews, final themes started to emerge. These themes were further improved to arrive on three main themes and their subthemes. The final theme map was based on three themes of the main initial theme map. The final theme map is shown in table 61.

Table 61: Final theme map

Theme	Subthemes
Adminstrative	Skill gap, effect on employees, emergent challenges, skill gap, risk, complexity
External factors	Contracts and Laws limiting CM, Juristictions, Standards and certifications, Cross organizational work limited by policy
Technology	Challenge and potential future direction, Current advantange of CM...

Below, each subtheme is illustrated with quotes from the experts interviewed. Quotes are shown as bullet points, with literally translated quotes in quotation marks, and paraphrased quotations without quotation marks. For literally translated quotes, changed and added words (such as expanded abbreviations) are marked with brackets "[]" and removed material is marked with ellipses in brackets "[...]" Translations from Finnish and paraphrasing was done by the author.

Some subthemes only apply to the Hidden Master architecture. On the one hand, the Hidden Master architecture, the prototype and its applications were the focus here. On the other hand, other products mentioned are mostly established, mature products, often in production use by the respondents' employees.

5.4.2 Theme one: Administrative

The "Administrative" theme included organizational, administrative and personnel matters. It did not include technical administration of computer systems, which is part of theme three, "Technology".

5.4.2.1 Raise in Productivity There was wide agreement that the use of configuration management tools and DevOps practices increased productivity, which was to be expected. The raise in productivity was mentioned by all interviews.

- "The advantage of current tools is - perhaps - we can - in practice - optimize our productivity. A single team can control [our] infrastructure, the software on top of it."
- "Compare this to history - if we remember - projects where [we] had to wait three months to get a rack. So we're living a different era now. We can move on our projects, do an MVP [minimum viable product]. Instantly. Because server capacity is available for us."
- "The speed of deployment - it's of direct magnitude [with configuration management]. Sounds like sales slides but it's true."

All respondents found that the Hidden Master and the models could further reduce costs. In addition to closed questions, cost reductions got many mentions in the free commentary.

- The security of Hidden Master architecture could reduce manual checking and layering, thus reducing cost. Reduction of manual steps and checking is likely to improve morale by allowing developers to concentration on more interesting tasks.
- We already have decentralization and redundancy built into our supply chain, but at relatively high cost. We connect to different operators internationally, run multiple server facilities in different geographical locations and have redundant providers, so this redundancy would not be new to us. HM could reduce these costs.
- The majority of costs come from operating the system, and provisioning costs are usually a minor part of life cycle expenses.
- Some of our CM networks are really small scale, so the scaling of a single network is not really a pain point for us. Of course, we like any savings.
- Scalability is useful on high demand spikes. In our networks, average loads can be very low compared to spikes. We prefer our private cloud, but find it economical to rent cloud capacity for high load situations.

5.4.2.2 Effect on Employees

Employee happiness and retention was seen to improve. Provisioning, managing and monitoring computer systems could include a lot of tedious, repetitive work. The use of configuration management systems not only reduced the amount of work, but also turned it into more abstract and advanced. Skillful employees liked this work more.

- "Developers don't get their kicks from repeating the details manually. Employee satisfaction ... You don't have to market this much."
- Configuration management improves morale by reducing tedious parts of the work.

5.4.2.3 Skill Gap

However, the use of more efficient, powerful, advanced and abstract tool brought new challenges that did not exist in previous approaches to operate these systems.

Skill gap and the need for suitable employees was seen as a challenge.

- "When you go to OPS organization [the operations department], you have a challenge to find the coding infra guy. The old school infra guy could want the DLS [domain specific language] configuration. But on the other hand [the general purpose language based configuration] seemed so simple that it could be a non-problem."
- You might need less employees, but now they have to be more skillful.

This work introduced a new way of defining configuration on slave, idempotent configuration using imperative general purpose language; base resource dependency model; and resource revalidation model. It was implemented in the prototype as a Python dialect and a library. "Easy to code" was mentioned by all interviewees. Inertia, a preference for current tools and models, was mentioned as a challenge for starting use of Conftero.

- Employees with operations background but no development or coding skill could find DSL more approachable as an idea.
- Only fundamental programming and logic skills are enough to write simple code.
- New employees should already know Python, which makes it easy to learn Conftero.
- Language seems simple. It will take a while to learn without a Python background.
- Python (dialect) will be major benefit to developers.
- This looks simple and fast to learn.
- If Kubernetes and related container technologies proliferate, will this change education to reduce the programming skills of operators (thus reducing the benefit of familiar language)?

- If you already use Puppet, changing to anything requires learning. It is a challenge to replace existing tools.
- Orientation is a minor part compared to other life cycle costs. The main costs are likely to be operational costs while the product is in use.
- Ease of starting a new tool and resistance to change depends on the background. Developers can be suspicious of security of any new tool or concept. Especially one that is not widely deployed.

5.4.2.4 Risk Configuration management can reduce risks, but also bring with them new risk and accelerate the speed in which they could materialize. Powerful tools require the ability to work on higher level of abstraction, and also makes it easier to make large scale errors.

- "You reduce the risk of errors that are common when doing configuration manually."
- It is very easy (and fast) to make errors that affect production. Code review could help here.

For hidden master, it was pointed out that technical security will not remove all risk

- There is clear security benefit [in the Hidden Master architecture], but one can never remove all risk. For example, human factors (infiltration, social engineering, coercion) is a possible avenue of attack here.

5.4.2.5 Complexity and Integration All respondents identified complexity as a challenge. It manifested in multiple levels, and this section concentrates on complexity on organizational and management level in an organization. Other sections discuss complexity inter-organizationally and on the technical level.

- Architectural challenges can expand to use all the available benefit provided by configuration management systems.
- CM allows control at scale where it is not possible to remember everything that is running. E.g. it is possible to remember what 10 computers are running, but very difficult for 1000-2000 computers.

Configuration management must work in practical enterprise networks. These networks can't be designed or changed in one day, but they evolve as the business evolves. This creates the challenge of integrating the systems. The lack of integration is often seen in losing an overview understanding of the whole system, and lack of visibility.

- Integration between CM and identity management tools is a challenge.

In particular, big organizations can end up with large number of separate CM tools, losing visibility and the big picture.
- Using multiple CM tools together might require a a defensive analyst to look at multiple screens.

Many tools have been created to reduce workload and complexity, but paradoxically can require even more tools to operate safely and efficiently. Especially container technologies were mentioned in this regard.

- Containers, such as Docker, are convenient but require a lot of tooling: e.g. CI/CD pipelines for security updates, vulnerability scanning. Many tools and providers help with CI/CD, such as GitHub (Actions), Travis, Jenkins. Docker containers use a DSL that's cumbersome.
- Our current tools do not allow continuous deployment, and it does not work with containers.

Having a clear, limited and well defined scope for a tool could simplify it's design, but naturally raises integration challenges and the problem of "looking at multiple screens", dispersion of key information and loss of big picture view.

- Some tools have a limited scope. E.g. Terraform makes it easy to provision hosts, but requires another tool to configure the applications the host was deployed for.
- Monitoring is hard, and there is no visibility to servers.
- Lack of visibility and a large scale can result in the appearance of an unknown attack surface.

5.4.2.6 Potential Markets for the Hidden Master Multiple potential markets for the Hidden Master were identified. IoT, rugged and embedded environments were mentioned in many comments. One respondent found the architecture to have potential in the Third World. One respondent pointed out that this could be adapted to offensive operations, providing some security benefits not available in the tools he had worked with.

- Conftero could have an interesting use case in IoT, edge computing and small devices.
- This could work in a rugged environment and with distributed data.
- IoT would be a good use case for this.
- This has huge potential in the Third World and developing markets. In my experience, poor network connections and underdeveloped infrastructure are very common in third world. When I briefly worked in an educational setting in a third world country, IT was used, but underdeveloped network infrastructure and poor networks were a challenge.

- From an offensive point of view, this could be a good redundant channel, if the more real time C2 channel (e.g. Cobalt) is caught and disabled by defenders.
- This model (HM architecture) will likely be copied to other established CMs (e.g. Ansible, Puppet, Salt). I hope it will.
- This tool would have offensive use. It could work as a redundant channel for hosts that are controlled by Cobalt strike by default.

5.4.3 Theme two: External factors

Like all tools, configuration management systems must work in the reality of legal and contractual frameworks of the business. Laws and contracts might make it difficult to reap all benefits that would be technically possible with configuration management systems. Organizations take there place in supply chains, renting services such as cloud, and providing services to their own clients. Especially for IT-services, configuration management must integrate into systems across organizational borders.

Many possibilities are provided by configuration management systems.

- Global geographical distribution becomes much easier.

5.4.3.1 Supply Chain and Cross-Organizational Challenges The question of outsourcing, or what to outsource, was mentioned many times. Outsourcing allows company to focus on its core business, but also means giving away control and the risk of lock-in. In lock-in, it becomes difficult to change a provider even if it no longer serves its purpose, or better options become available. Lock-in usually benefits the service provider, but is detrimental to buyer.

- "The thought that you take it from different operators, it brings management costs."
- "But it's not so easy to move - say - a group of machines from one AWS [Amazon Web Services] region to another - that it would succeed like [snaps fingers]. Even if you have the IaCs [infrastructure as code definitions of your systems]. It would mean some work: open the IPs, move IPs manually, maybe the region does not provide all the services. And the like [laughs]."
- Operations (the provisioning and administration of OS and the platform below applications) can sometimes be outsourced to cloud providers
- Configuration management should be more platform agnostic. Terraform (a popular CM tool) advertises this, but is it practical to move between two major cloud providers (e.g.from Google Cloud to AWS)?

- "I sometimes wish the tools were more system and platform agnostic. [..] Terraform - for example - keeps advertising it as a product like this. But when you try to move - say - an AWS [Amazon web services] Terraform to GPC [Google Cloud Platform] - it just doesn't happen."

One respondent noted that static compilation used in the prototype does not remove risk from supply chain.

- The whole software is part of the supply chain, even if it is statically compiled. This could include Go compiler and all statically linked libraries.

5.4.3.2 Laws and Contracts Contractual and policy issues may limit how much technical benefits materialize as benefits to business.

- "If we think what kind of an organization requires inter-operator and geographic spread, [..] we're talking medium to large organization. At this size, you have to look at juridical issues. [..] If you have some admin data, does it go outside the EU. Do we have DPA agreements [data processing agreements] in order. What SLA [service level agreement] do they offer. Because our own firm's security police mandates that we need a certain level SLA. [...] For a big [enterprise] it could be easiest to say that let's pick this large operator with activities on multiple geographic areas."
- We're bound by the certificates required to operate in our line of business.

The use of Hidden Master architecture could be affected by legal considerations. Two respondents noted that this level of security might raise political issues depending on the market and jurisdiction.

- "[Using the Hidden Master security model] you can take [slave nodes and courier nodes] to Russia and China, without having to worry about serious trouble. But in China, you could have some regulatory problems. China has its own laws on crypto. [..] So certain kind of data must use mandated crypto algorithms. So if you take the Hidden Master to a market like China, you might need to be able to change what crypto is provided."
- Local regulation (e.g. China) might forbid the use of the level of security provided by HM implementation.

5.4.4 Theme three: Technology

Using configuration management in support of business must also happen on practical level. Thus, it is not surprising that there were many comments on tools. As respondents were asked beforehand not to divulge information that

would undermine their security monitoring or incident response capabilities, it is likely that some tools or configurations are not mentioned here.

5.4.4.1 Challenges of Current Tools Tools made for specific platform or ecosystem were found to be convenient, as long as you worked within the confinement on that platform. As noted in the discussion on supply chain, lock-in is often a price paid for building operations around these tools. Regular configuration management tools can support offensive cyber operations.

- Using Microsoft tools simplifies the integration and user experience in Microsoft environment
- Terraform is high level and simple to start
- For offensive scenarios, popular tools such as Cobalt Strike and Empire greatly enhance attacker capabilities. Traditional tools, such as Terraform and Ansible, can be used for building redundant infrastructure for attacks.

Lack of features in current tool used by the companies of the interviewees could be seen as a practical matter. However, one could raise the question if these gaps are indicative of some larger challenge faced by the current level of configuration management tools. One respondent included offensive C2 tools as part of configuration management, and interestingly found a common challenge: skill gap.

- Verifying provisioning: should one trust it when CM tool claims configuration is done?
- It is very easy (and fast) to make errors that affect production. Code review could help here.
- For offensive operations, Cobalt and Empire are both fingerprinted in AV and EDR. Obfuscating them requires expertise from the attacking team. Some expensive products provide better evasion.
- Identifying manual (non-IaC) configuration on managed hosts is currently difficult.

5.4.4.2 Hidden Master Key Management and Cryptography Respondents found the Hidden Master architecture scalable, resilient and secure. Key management was found to be a crucial factor in the security of the model. One interviewee also verified that multiple couriers and drops can be used, and this was indeed the case.

- This will improve availability and resiliency.
- Only the master needs to be trusted here.
- This obviously scales really well.
- HM has minimal dependency on network connections.

- Security (of the architecture) is really impressive.
- Key management and key life cycle are very important in the security of this model.
- The choice of PGP could allow for dual control, requirement of cryptographic signature from two persons. Also, PGP keys could be placed in the HSM (hardware security module).
- When deploying IoT devices to clients, the devices must be identified and recorded, and keys exchanged. [Interviewee requested and got a recap of HM key management.] Key management used here reduces and automates this work, but some parts will require a decision from a human.

Security was found to be a key benefit of HM. One respondent emphasized the novelty of this model and found it a definitive security improvement. It could be assumed that when the respondent says an attack is "impossible", he implies that it is not feasible to attack the master using an approach similar to that used for attacking traditional configuration management infrastructure, as obviously no real life system is completely invulnerable. Based on respondents' comments on key management, it could be thought that the high level of protection for the master's secret key moved the main attack surface to other areas of the organization, such as personnel and key management.

- I've never considered HM architecture before, but it makes sense. It's practically impossible to attack the master.
- Key management, key life cycle, policies and training of personnel will affect the security on this system.
- Security of HM is really impressive.
- For small organizations without 24/7 monitoring and incident response capabilities, HM can create major benefits.

5.4.4.3 Comparisons and Analogies of the Hidden Master Conftero (the research prototype) and the Hidden Master architecture was compared to some other systems.

- There some similar challenges in payment terminal key management.
- Kubernetes and related container technologies could bring in a higher level of abstraction.
- This has similar structure to a C2 attack infrastructure, which is used in offensive operations for protecting the master. Owned machines beacon to the master through intermediate hosts. For active hosts, it might be 5-10 seconds. For backups, it might be one DNS query per month.
- This (HM) architecture differs from popular C2 tools in that there is no requirement for the master to stay online. Popular offensive C2 tools

(e.g. Cobal Strike, Empire) still require the master to stay online. There have been cases published in the media where real, criminal attackers have taken over the network using the tools deployed by pentesters or a red team.
- Predefined keys and phone-home addresses are used in Tinker Canary which is a honeypot that phones home through DNS.

5.4.4.4 Maturity of the Hidden Master prototype For wide production use in real life organizations, a mature product is needed. Even though it is obvious (and explicitly mentioned in the interview preamble) that a software prototype cannot meet this bar, some ideas for developing Conftero towards maturity were given.

- For real-life production, more documentation is needed.
- Integration and standardization of the demonstrated software is necessary for wide adoption.
- I hope this considers all other applications requiring Apache web server.
- Could hasChanges() be used by an attacker to create an unwanted operation? I cannot come up with a concrete idea right away.
- We use cloud provider specific tools a lot.
- This would not work if the slave was read-only.
- How to productize this, how to make this tool interesting for marketing?
- Will this be released as open source / Free software? Please inform me when you publish it.
- Areas I'd consider if auditing the security: code quality; key management; scenarios when master or an agent is taken over by the adversary.
- It might be difficult to replace existing, already deployed CM products.
- The code of the research prototype is not likely to meet the quality of established products.

5.4.4.5 Defining Configuration Most comments on defining configuration concentrated on the ease of learning, and are discussed under "Administration" theme. Technical comments on the syntax were

- Conftero syntax is so simple that Notepad (a simple plain text editor) would be enough. IDE features would not be needed.
- GPL is used for a limited scope configuration in AWS Cloud Development Kit.
- If hasChanges() really works, it is a big improvement compared to previous dependency models.

5.4.5 Likert Scale Questions

The semi-structured interviews were mostly qualitative in nature, and claims in the questionnaire worked mostly as prompts for free form answers. However, as also Likert scale answers were collected, some statistics can be calculated. The main emphasis should be given to thematic analyses above. The questionnaire included 14 questions using Likert scale, from 1 ("Not at all") to 5 ("Extremely well").

All interviewees found HM could reduce costs, improve security and improve resiliency in multiple ways. This is in line with comments respondents gave during and immediately after the demonstration - before seeing the closed questions.

Interviewees found multiple business benefits for HM. Interviewees agreed or strongly agreed on all eight claims on the HM business benefits, with a single exception of inter-operator scaling in one answer. Three respondents pointed out that they already use inter-operator or geographical redundancy or scaling to some extent. Two respondents said redundancy of networks and providers is an important part of their operations. All three found HM architecture to provide possible cost and security benefits for this redundancy. Summary numerical Likert scale answers to HM questions is table 62.

Inter-operator scaling means renting server capacity from multiple cloud providers, thus reducing supply chain risk from a single operator. A single cloud provider carries with it risks that do not need to affect the whole market, such as financial difficulties or suspending a customer for political or PR reasons. While other interviewees found that HM can improve resiliency and reduce cost with inter-operator scaling, one respondent found it dependent on the specifics of the company. In his case, company policy requires thorough evaluation, service level agreement, compatible security policies and other agreements to accept a new cloud provider for production systems.

Obviously one could claim that this level of scrutiny for the untrusted intermediate hosts is made redundant by the security model of HM. After all, the intermediate courier/drop servers are not trusted, do not have the keys read or modify messages on the HM network and do not even form a single point of failure as there can be redundant drops and couriers. Nevertheless, as these policies are partly the result of certificates mandated for those wishing to operate in the payment card industry, it is not necessarily trivial to change the resulting policies. Another respondent considered the use of redundant providers an essential part of ensuring availability of IT systems, and thus had it implemented on multiple levels. However, he found it likely that HM would

provide cost reductions.

Table 62: Recognized business benefits of the Hidden Master architecture

Benefit	1	2	3	4	5
- Reduce cost of protecting configuration management system				◆ ◆ ◆	◆ ◆ ◆
- Improve security by better protecting master secret keys				◆ ◆	◆ ◆ ◆ ◆
- Reduce cost of scaling by leveraging file serving capabilities of commodity web servers				◆ ◆ ◆	◆ ◆ ◆
- Improve resiliency against network problems with efficient geographic scaling				◆ ◆	◆ ◆ ◆ ◆
- Improve resiliency and reduce costs with inter-operator scaling		◆		◆	◆ ◆ ◆ ◆
- Reduce cost to both sending and receiving organization when deploying new machines and IoT devices				◆ ◆ ◆	◆ ◆ ◆
- Reduce cost and risk when preparing to deploy unknown number of machines or IoT devices			◆	◆ ◆	◆ ◆ ◆
- Reduce relative cost and risk of running configuration management system in small networks				◆ ◆	◆ ◆ ◆ ◆

All respondents found that the language and the models could improve time-to-market and increase developer productivity. Respondents mostly agreed that this could also save costs in multiple areas and reduce supply chain dependency. Respondents disagreed on whether these improvements could reduce resistance to DevOps.

Language and the dependency models were considered very simple and easy to understand. This was a design goal for using idempotent - imperative general purpose language. It was also a goal for a simplified dependency model between user defined dependencies. To make it feasible to implement the prototype and perform limited testing in production environments, another dependency model was created for implementing the language. It is possible that this has

contributed to the easiness of the language.

Conftero allows the operator to use a general purpose programming language to define configuration on agents. In the research prototype, configuration language is a dialect of Python. Idempotency is achieved by defining idempotent functions for essential configuration features, such as files, packages and users. This is in contrast with industry leading solutions, such as Puppet, Salt and Ansible, that require the user to learn and implement configuration in their own language that is not used anywhere else.

To further simplify the language, Conftero allows dependencies to be defined using a single dependency type, hasChanges(). To make the implementation of the language and the tool possible, key idempotent functions were identified and implemented. This is in contrast to industry leading tools, that provide their own DSL with multiple dependency models and sometimes even redefine programming concepts to have an unexpected meaning (e.g. a "class" in Puppet).

These improvements were expected to make the system easier to learn, use and debug. Further, it was expected that this would reduce risk, allow faster time to market and reduce costs.

All interviewees found these features likely to provide these benefits, and they agreed with all claims in this section. The recognized business benefits of idempotent general purpose language and simplified resource models are in table 63.

Table 63: Recognized business benefits of idempotent general purpose language and simplified resource models

Benefit	1	2	3	4	5
- Save costs in new employee orientation to systems.			◆	◆ ◆ ◆ ◆	◆
- Faster time to market by leveraging existing skills				◆ ◆	◆ ◆ ◆ ◆
- Reduce resistance to DevOps change by allowing programmers to use familiar languages and patterns		◆		◆ ◆ ◆	◆ ◆
- Save cost of errors when debugging dependencies			◆	◆ ◆	◆ ◆ ◆
- Increase developer productivity by leveraging existing tool support, such as syntax highlighting				◆	◆ ◆ ◆ ◆ ◆

Benefit	1	2	3	4	5
- Reduce risk from supply chain dependency			◆	◆ ◆	◆ ◆ ◆

To compare similarity of answers to each questions, correlation was calculated. For the purposes of correlation, Likert scale data was considered ordinal, as the responses have a clear rank order, but distance between different options is not necessarily the same. Thus, Spearman rank correlation was chosen as the correlation measure. The significance level was chosen to be $p < 0.05$.

The data was converted to numeric format, and Spearman correlations and p values were calculated for each question against all other questions. This was done using spearmanr function from scipy.stats Python library.

Correlation analyses indicated that many questions were correlated at chosen significance level. Total 91 pairs were compared. Out of those, 34 pairs were correlated at the chosen significance level. Correlations are listed in Appendix: Correlation Matrix for Likert Scale Interview Questions.

Questions with answers were clustered using hierarchical clustering. Clusters were built bottom-up, using agglomerative clustering based on distance map created using Spearman correlation. The number of clusters, three, was chosen by experimenting with small number of clusters and excluding one single item cluster. Clustering was done using from Scikit-learn library in Python, using AgglomerativeClustering function. Spearman correlation and distance matrix were calculated with functions from Python Pandas library.

Clustering ended up with the following three clusters. Names for the clusters were chosen by the author based on repeating themes in the questions. For cluster 3, there was no obvious connecting factor despite statistical similarity.

Cluster 1 - Personnel

- i Save costs in new employee orientation to systems.
- k Reduce resistance to DevOps change by allowing programmers to use familiar languages and patterns

Cluster 2 - Risk and cost

- a Reduce cost of protecting configuration management system
- c Reduce cost of scaling by leveraging file serving capabilities of commodity web servers
- e Improve resiliency and reduce costs with inter-operator scaling
- f Reduce cost to both sending and receiving organization when deploying new machines and IoT devices

- g Reduce cost and risk when preparing to deploy unknown number of machines or IoT devices
- l Save cost of errors when debugging dependencies
- n Reduce risk from supply chain dependency

Cluster 3

- b Improve security by better protecting master secret keys
- d Improve resiliency against network problems with efficient geographic scaling
- h Reduce relative cost and risk of running configuration management system in small networks
- j Faster time to market by leveraging existing skills

6 Conclusion

Tempted by illegal financial gains, criminals can operate large malware networks in hostile conditions. Despite the efforts of incident responders, security researchers, system operators and their tools, illegal profits from cybercrime offer incentives for criminals. When caught, criminals could face serious challenges. On one hand, the progress of malware and its command and control channels have improved in the face of adversity. On the other hand, the incentive of financial criminal profits has motivated criminals to continuously improve their methods. This research looked at taking and implementing some of their methods for the benign and legitimate control of companies' own computers.

This work contributed multiple improvements to configuration management systems. These contributions were based on the work done developing the stage model for comparing malware and configuration management systems. The improvements were designed and developed into two functional prototypes that were validated using multiple methods. Laboratory testing was used to evaluate their technical qualities and compare them to chosen leading tools in the industry. Case studies validated the applicability of these techniques in a realistic field environment.

Potential business benefits were identified and evaluated using individual semistructured expert interviews. Respondents agreed that the models and the Hidden Master architecture could reduce costs and risks, improve developer productivity and allow faster time-to-market. Protection of master secret keys and the reduced need for incident response were seen as key drivers for improved security. Low-cost geographic scaling and leveraging file serving capabilities of commodity servers were seen to improve scaling and resiliency. Respondents identified jurisdictional legal limitations to encryption and requirements for cloud operator auditing as factors potentially limiting full use of some concepts.

6.1 Hidden Master Architecture

The Hidden Master architecture is a key contribution of this work. Contrary to how leading configuration management tools were implemented and what was implied in configuration management research, it is possible to design a topology where master and agent (slave) never form a direct connection. Leading configuration management tools prefer a continuous or constantly repeating two-way connection. In the Hidden Master architecture, both catalogs (downstream from master to agents) and reports (upstream from agents to master) are transfered as encrypted and signed files through untrusted intermediaries called couriers and drops.

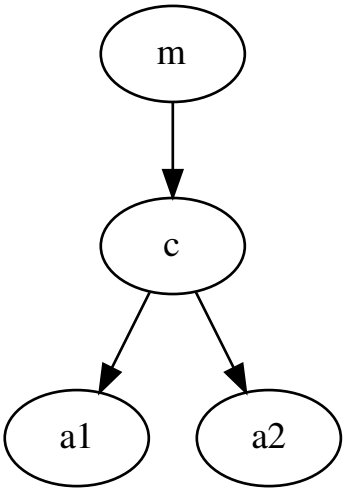

Figure 21: Downstream data flow in a simple Hidden Master architecture network

Downstream data flow in the Hidden Master architecture is shown in fig. 21. Master m only connects the network to upload encrypted and signed catalogs to an untrusted intermediary host. This data is periodically pulled by agents (slaves) a1 and a2.

The master private key is the most valuable piece of information among all computers controlled by it. In a mature environment applying configuration management and DevOps principles might include all computers and IoT devices of an enterprise. The Hidden Master architecture protects the master secret key better than industry leading solutions. Attack tree analysis shows that this eliminates whole categories of vulnerabilities, namely those applying common attack methods performed against servers.

The Hidden Master architecture protects the master's secret keys by allowing the system to operate while the master is not connected to the network. By the use of campaign keys, it is also possible to provision new agent nodes to the system without connecting the master to the network. Also the backchannel (from agent to master) is asynchronous, allowing the master to collect status information from agents for later analyses offline.

The Hidden Master architecture improves scalability even with single hosts by utilizing the capabilities of exising, purpose built servers for distributing static files. In a laboratory test, a limited RAM host with Conftero (advanced prototype) could deliver instructions to 20 times more agents than Salt, while Puppet failed to start in these conditions. Puppet and Salt are among the leading tools in the industry. For the advanced prototype, the agents pull configuration from a regular web server Apache, which is very good for serving static files. The load in other parts, master and slave, is not affected by the

size of the network.

In the Hidden Master architecture, the intermediate hosts (couriers and drops) do not have any encryption keys, so they do not need to be trusted. This allows for cheap geographic and inter-operator scaling. Even though the use of a web server makes single Conftero drop and courier highly scalable, they can scale further by adding these intermediate nodes and allowing agents to connect to them randomly.

The Hidden Master architecture was used in two case studies. In both cases, the the architecture showed promise in simplifying the network architecture and making it safer.

In the first smaller study involving a computer exercise evaluation, 23 students worked on their own VMs running on their own computer systems, running various versions of Ubuntu and Debian Linux. This resulted in multiple NAT situations, where none of the master or agent (slave) nodes on the network had public IP addresses, and most of the VMs were behind two NATs. The ability to deploy the intermediate host (courier/drop) into an untrusted cloud made it much simpler to protect the keys. The exercise was delivered through Conftero, and the information to evaluate the work was collected through Conftero too. Due to the asynchronous nature of Conftero, it was possible to evaluate the work even after the machines themselves were deleted.

The larger main case study involved a small Finnish company providing smart building technology. The company provided their own IoT platform with AI prediction. At the time of the study, Company X was well established in Finland and at the time of finishing this work, it had already expanded to foreign markets. The case study was performed by configuring and deploying a complex AV system to Client Y, a large university and a client of Company X. Deployment was performed by a junior operator of Company X who was observed and advised by the researcher.

It was found that the Hidden Master architecture could make it much faster to initially deploy devices to clients, as network and firewall configuration would not be needed. However, as Conftero was still a prototype, this was seen as too risky. It could also be expected that gaining the benefit of fast deployment might only be possible once the client security policy allows these type of deployments.

6.2 Improvements for Idempotent IaC

Key features of modern configuration management are idempotency and infrastructure as code. Idempotency means that only the target end state of the

system is defined. The configuration management tool only performs changes when they are needed. Eventually, the system should end up in a harmonious state where no changes are made. At this point, applying the same configuration again results in no changes. Infrastructure as code (IaC) simply means that the target state is written as plain text.

Leading tools in the industry use languages only used in each specific tool, invented only for this particular purpose. These are domain specific languages (DSL). In particular, they invent their own unique methods of dependencies and flow control. Sometimes industry leading solutions redefine common programming words (e.g. "class" in Puppet) or use highly unorthodox methods for flow control, such as Salt using Jinja templates to generate code for "for" loops. These tools also provide a large number (from one hundred to over 500) of predefined functions.

The improvements to agent configuration are:

- The use of *imperative general purpose programming language for defining idempotent infrastructure-as-code configuration*
- *A base resource model* to simplify building infrastructure as code definitions and languages
- A *simplified hasChanges model for resource revalidation* when applying configuration on agent.

Normal programming languages, such as Python or C, are imperative. They perform commands in top-down order, unless flow is diverted with flow control structures such as "if-else" or "for". This type of flow control is similar between different programming languages. These languages are general purpose languages (GPL), as they can be used for a wide variety of tasks and in ways not yet invented by the developers and inventors of these languages.

The large amount of code in and the number of functions provided by the industry leading systems might make it seem that developing a language for configuration management is an expensive endeavor. The requirement of idempotency might make it seem like a regular imperative GPL could not be idempotent.

This work contributes *a base resource model*. Based on analyzing real life configuration, it was seen that a small number of functions and control structures cover most of the code. For both Mozilla Engineering and the United States Government Configuration Baseline, the top five cover nearly 60% of all commands and flow control structures. Further analysis showed that rarely used commands or resources could be defined from a limited number of base resources.

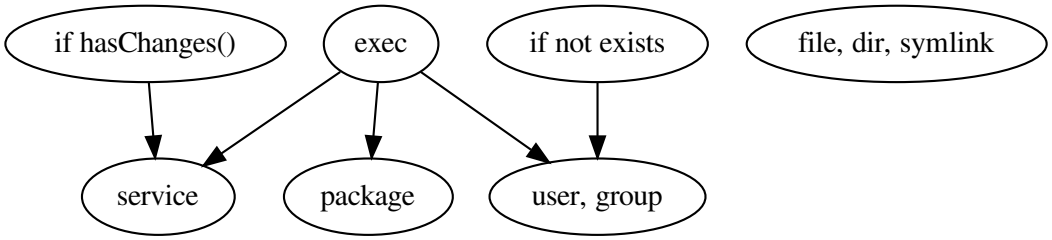

Figure 22: Dependencies in configuration management functions

This led to the base resource model (fig. 22). Idempotent configuration requires defining the base resources in a way that is idempotent. The base resource model shows how an idempotent IaC can be created by defining four lower level base resources and two control structures. Four more resources are then created based on these lower level resources. The higher layer of resources can be defined in a higher level language, as was later shown in the advanced prototype language design and implementation.

In the base resource model, the configuration management system provides idempotent definitions for file system related resources (file, directory, symbolic link, exists()) and resource revalidation with hasChanges(). In the prototype, these were implemented in a lower level Go language, and compiled statically into Conftero binaries. Higher level concepts of service, package and user management were then implemented using these lower level resources. In the prototype, these higher level concepts were implemented with the same Python-dialect high level language used by the operator of this system.

When a part of a configuration changes, other resources often need to be revalidated. For example, a very simple configuration for a web server might need a package (the software installed), a configuration file and to keep the daemon (the server application) running. When the package is updated or a configuration file changes, the daemon needs to be restarted for these changes to take effect properly. Industry leading tools provide multiple, unique and sometimes complex methods for defining inter-resource dependencies for different situations.

This work contributes a *simplified hasChanges() resource revalidation model*. If any earlier part of the configuration in a module has changes, function hasChanges returns True. This information is then used to revalidate resources. The use of hasChanges() makes IaC code shorter compared to the industry leading tools. By solving the specific dependency challenge of IaC, hasChanges() then allows familiar flow control from regular programming languages to be used in other situations.

The ideas identified and proven in malware led to the Hidden Master architecture

and multiple improvements in designing configuration.

The contributions relating to improvements in the idempotent IaC configuration emerged as part of answering multiple research questions and completing multiple objectives. Similar to the approach for the Hidden Master architecture, a stage model (RQ1, OBJ2) was used for identifying adaptable techniques (RQ2, OBJ1) and developed into key concepts for the configuration management tool (OBJ3) and into two functional prototypes (RQ4, OBJ4). Some of the improvements to IaC could be seen as emerging from the design. In addition to inspiration from malware, the improvements to idempotent agent configuration (RQ3) have partly emerged in the design and literature review phases. The prototypes and the concepts implemented in them were validated using laboratory testing (OBJ5), attack tree analysis (RQ5), case studies (RQ6, OBJ6) and semi-structured expert interviews to identify potential business benefits (RQ7, OBJ6). The research questions and objectives are listed in table 1.

6.3 Design and Prototype

Following a constructive approach - also known as design science - two prototypes were designed and implemented. These prototypes allowed the validation of the models and the new Hidden Master architecture. The validation included laboratory testing in simulated and emulated environments, two case studies and six individual semistructured expert interviews.

The design and implementation of a functional prototype is a contribution of this work. Both the process and the products (the prototypes) are part of the design and can help build knowledge. Two prototypes were built: a trivial prototype to test the feasibility of the Hidden Master architecture, and an advanced prototype that was used as the basis of the case studies and demonstrations for the expert interviews. The advanced prototype, Conftero, is a five thousand line program written in Go. Conftero implements the key contributions described earlier: the Hidden Master architecture, the imperative GPL for idempotent IaC, the hasChanges() resource revalidation model, and the base resource dependency model.

Design goals of the advanced prototype Conftero were, in addition to implementing the mentioned contributions:

- Protecting the master secret key
- Improving the network and geographical dispersion
- Decoupling subsystems.

The use of the Hidden Master architecture gave up on the requirement of

a direct two-way master-agent connection. By treating both upstream and downstream communication as individual encrypted files, it was possible form a clear and simple layer architecture as shown in table 64. Each layer can only communicate to the layer directly above or below it. Only the transfer layer can communicate between computers. This layer architecture made it possible to use an existing cryptosystem and to swap the transfer layer, both of which were done during this work.

The intermediate node plays an important role in the Hidden Master architecture. Other nodes are expected to be down for extended periods of time. For high security, the master could be shut down or disconnected from the Internet when not uploading new instructions or collecting agent reports. This makes it impossible to obtain master secret keys using traditional network based attacks at the tempting moment when the master is not being used and incident response might be unavailable or seriously delayed. It is expected that one of the courier/drop nodes is available most of the time in the network. Compared to industry leading solutions, renting cloud capacity for the intermediate nodes (instead of master nodes) is cheap and safe, as these nodes do not participate in the encryption layer and have no keys required to decrypt or modify the encrypted catalogs.

Table 64: Matrix of subsystems and components

Layers/Subsystems	Master	Courier/Drop	Slave
Configuration	Define configuration	n/a	Apply configuration
Encryption	Encrypt & sign	n/a	Decrypt & verify
Transfer	Upload to mule	Serve enc. catalog	Download

Even though the use of layer architectures is common in design, the actual decoupling of layers is not always easy to achieve. In laboratory testing, it was shown that the transport layer of Conftero could be easily changed. The SSH and HTTP based transport was changed to peer-to-peer (P2P) transport, making it possible to operate in conditions without upstream Internet connectivity and no preplanned network topology. As the transfer layer does not contain any encryption keys and is completely untrusted, it was possible to use the existing P2P application Syncthing without modifications.

Even though many programmers know not to “roll their own crypto”, implementation mistakes still make many cryptosystems vulnerable, as was shown with examples from malware CC. To combat these challenges, Conftero used a whole existing crypto system in addition to proven encryption algorithms.

OpenPGP standard offered an efficient public key crypto system with widely used processes for key management and verification. It was chosen as the basis of Conftero encryption layer. Expert interviews indicated that in addition to security, this could provide functional benefits. For example, PGP might make it possible and cryptographically secure to require signatures from two persons to allow changes to high value systems.

Backchannel from agents to master is part of the Hidden Master architecture. It was included in the implementation due to requirements and feedback from the case studies performed. Backchannel allows agents to send reports of configuration success and their state to the master, using the same security mechanisms that Conftero uses for downstream transfer.

Campaign keys allow secure configuration of an unspecified number of future agents. A company might need to configure a lot of similar computers continuously, but at unpredictable times and in unpredictable numbers. For example, new IoT devices are provisioned and deployed when a customer makes a request; or when automation provisions new virtual hosts as the load increases. Agent software can be installed during provisioning (operating system install or imaging), but it might require a human to validate the correctness of the keys - a tedious and error-prone operation. In Conftero, pre-generated agent binaries contain the campaign public key and initial configuration. To later address individual agents, an agent key pair is generated on the first launch and communicated to the master using the encrypted backchannel. This transfer is encrypted using the public key of the master.

The advanced prototype, Conftero, was developed in Go. The application itself can be delivered as a single, statically linked (linked against glib) binary. A single 'cct' binary can generate all other parts without any dependencies. The core code is approximately 4000 lines of pure Go, excluding third-party libraries. The imperative general purpose language for idempotent configuration is Starlark, a Python dialect. It is built into Conftero and does not depend on any external software. Following the base dependency model, the base resources are implemented at a lower level in Go. The higher level resources are implemented in the Python dialect. The simplified model for defining inter-resource dependencies, hasChanges, is also implemented in the higher level Python dialect. The transport layer is implemented using built in support for HTTP and SSH. Encryption uses OpenPGP, with built in support from standard libraries.

The advanced prototype implemented the novel concepts contributed by this work: the Hidden Master architecture, the hasChanges revalidation model and the base dependency model. Some additional features implemented in Conftero

include:

- Cross compilation for Linux, Windows and OSX using Go builtin cross compilation for pure Go programs. Only Linux support is currently implemented in the base resources.
- Static compilation which made it very easy to work in varying Linux distributions and environments. For maximum compatibility, the latest versions are linked against the four libraries that provide maximum compatibility.
- Live update of agent binary.

The prototype is the artifact, the construct created in the constructive research approach. In constructive research and design science, all of the design process, the construct and the validation can contribute to knowledge. The design of the prototype showed that the novel concepts work together and can be put into practice. Development of the advanced prototype included validation in a laboratory testing, in the field in two case studies and in its demonstration to expert interviewees.

6.4 Laboratory Testing

The Hidden Master architecture was validated using attack tree analysis, laboratory testing, two case studies in the field and semistructured expert interviews.

Laboratory testing used simulated and emulated environments to test the configuration management tools in various network topologies, load levels and adverse conditions. Most tests used fully virtualized computers in their own virtual IP network. Tested applications were the two prototypes, chosen industry leading tools and in some cases standard tools for the same protocols. These tools were tested both individually and in comparison.

A simple prototype, using a short script and an existing configuration management tool, was tested in optimal conditions. It showed that the approach is feasible in practice. A ready-made configuration was encrypted using PGP, then moved downstream to the agent, decrypted and applied.

The advanced prototype, Conftero, was tested in optimal conditions. As Conftero is a stand-alone tool implementing all features necessary for IaC configuration management, the test involved setting up the Hidden Master infrastructure, generating and handling the keys, synchronizing catalogs, automatically applying changes and reporting back. This test showed that the advanced prototype works in regular conditions. The prototype itself showed that it is possible to implement the novel contributions of this work, and to

indeed build such an application with the limited resources available.

Load testing indicated that the Hidden Master approach is highly scalable. The use of commodity web servers as the standard transport made even a single node highly scalable. In load testing, the Conftero based system was able to serve 10 000 agents with small configuration-only catalogs, and more than 1000 agents when delivering large binaries through the Conftero transport layer (both tests with a host limited to 1 GB RAM). As the Hidden Master architecture removes the requirement for two-way interactive and direct master-agent connection, it allows one to easily swap the transport layer. The default transport uses commodity web servers, which are highly optimized for serving static files, so the high scalability of a single courier/drop could be expected.

Scaling with a single node is only one way to scale in the Hidden Master architecture. By using multiple intermediate nodes (couriers/drops), and allowing agents to choose randomly between them, the problem becomes embarrassingly parallel. As the intermediate nodes are not trusted in the Hidden Master architecture, there is no need to vet, audit or make those nodes in-prem. Instead, cheap virtual services can be rented from multiple cloud operators, networks and different geographical areas, further reducing the risk related to each of these.

In low memory conditions (512 MB RAM), the Conftero based system fared much better than the alternatives. Compared to industry leading tools, Conftero could serve downstream 20 times more clients than Salt. Despite trying different versions and configurations, starting Puppet was not successful in low memory conditions. Similar to load testing, the significant benefits seem to come from delegating the bottleneck to a tool optimized for this purpose, a commodity web server. This is made possible by not delegating trust (or keys) to the courier or drop, and the asynchronous nature of the Hidden Master architecture. As the intermediate nodes (couriers/drops) are the only nodes that contact multiple nodes, the load is not expected to affect the agent or master nodes.

Adverse network conditions can disrupt communications, and some network problems can cause interference larger than the original problem. In this work, adverse network conditions were simulated using Linux kernel module netem, controlled by custom programs written by the researcher. The adverse network conditions tested were packet drop, corruption, duplication and latency.

Confero was the most resilient of the tested configuration management tools against packet loss and packet corruption. Latency, the time it takes for the first bit to arrive, had a measurable but insignificant effect on the tested configuration management tools in terms of their intended purpose. Packet

duplication had a negligible effect on the tools. Common networking tools not designed for configuration management were highly resilient to all adverse network conditions, and could outperform all configuration management tools tested. This indicates there are probably well tested and easily available methods for improving resiliency against adverse network conditions.

Peer to peer (P2P) operation allows each node to opportunistically exchange data with any other node on the network. Use of P2P architectures could allow operation when more common centralized, top-down network topology is not available. Such a scenario might rise from the disruption of a local ISP or cell phone network, or simply a blackout. Even though these events are often outside the control of organization requiring configuration management, their own computers might still be running using UPS, laptop batteries or the backup power of a local building.

Industry leading configuration management tools require direct two-way master-agent connection. Thus, they cannot work in P2P fashion. The Hidden Master architecture in Conftero removes this limitation and allows for better decoupling between layers. Thus, it was simple to swap the transport layer to use an external P2P tool.

Implementing a P2P application is not a trivial matter. In the Hidden Master architecture, the transport layer is not trusted. The transport layer does not handle any keys, and thus cannot manipulate or read manifests. Replay attack is prevented by versioning the catalogs with a monotonously growing counter inside the encrypted catalog. The transport layer can only see the metadata of traffic, such as the size and host identifiers. The P2P transport was created by outputting encrypted files downstream from Conftero, then transferring them using Syncthing P2P application. Only downstream transfer from the master to the agent was implemented and tested in this experiment.

Unlike industry leading configuration management tools, Conftero could operate in a shattered network where centralized and permanent topology was not available. In the simple scenario, agent nodes could transfer encrypted catalogs to each other even when the master was not available.

Air gapped networks are not connected to the Internet at all. However, some way for transferring software to those networks was required so that they could operate at all. As an implementation of the Hidden Master architecture, Conftero can communicate by asynchronously transferring files. In this scenario, files were successfully transferred using a simulated USB storage. The results indicated that the transfer could also be operated by a partially trusted person without access to the user interface of neither the master nor the agent.

6.5 Case Studies

Two case studies validated the benefits of the Hidden Master architecture, dependency model, resource re-evaluation model, idempotent imperative general purpose language and the related prototype in a realistic field environment. Case studies answered research question six: "What utility do the models and the research prototype provide when run in a field environment with business requirements?"

The first smaller case study used the advanced prototype Conftero for a computer exercise evaluation. All computers were behind NAT (network address translation), and used an untrusted cloud host to communicate. Some hosts were NAT'ted multiple times. The evaluation had to be performed after the computers (agents) were deleted with their contents. All tested computers were run by different operators, and they ran various Linux distributions.

Conftero could bypass multiple NAT using the the Hidden Master architecture. It operated in various distributions chosen by the candidates, except on 32-bit computer as such binaries had not been compiled. Due to its idempotent, imperative general purpose language, it was easy to adapt it to the requirement of monitoring and collecting data from agents, including opportunistic and automatic use of tools some candidates had installed. Installation of the agent appeared to be easy with tree point instructions. The first case study indicated that the prototype and the related features are easily adaptable to changing conditions and require minimal infrastructure to run without exposing master private keys to cloud operators.

The larger case study tested the prototype and related contributions in a complex environment. It involved two external organizations with their own requirements and policies. As the case involved working with established organizations, many existing tools, solutions and security policies created interesting demands and requirements for the deployment of new configuration management tool. In this study, an advanced multiroom AV solution was deployed in the premises of a large university, a client of Company X. The complex AV solution involved time limited mobile authentication, monitoring and automation. The implementation used interdependent containerized images.

The idempotent, imperative general purpose language showed promise for configuration management. Despite the very limited number of base resources as dictated by the *base resource dependency model*, a contribution of this work, the language proved powerful and adaptable. In the case study, a junior system operator with limited programming experience could quickly learn and use the language. It was possible to solve challenges that were completely unexpected

when the language and prototype were designed, namely that of controlling interdependent containers using a tool never seen by the author before. The use of regular flow control and structuring features of a regular programming language, such as functions, showed initial promise of simplifying abstraction in configuration management.

The Hidden Master architecture could solve multiple challenges faced by the case organization. An interesting area would be the automatic provisioning of IoT system images so that error prone and tedious manual key verification would not be needed. This could be an area of future research. When deploying new systems to external organizations, the Hidden Master architecture would allow remote access even without configuring networks and VPNs in both sending and receiving organizations. As the research prototype was not yet mature or security audited, additional security measures were used in production networks and such deployment was not tested in the larger case study.

In the spirit of constructive research / a design science approach, the case studies showed that a prototype implementing the contributions can indeed be created. In addition to providing end user benefits of their own, some contributions greatly reduced the burden of creating the prototype. The use of idempotent, imperative general purpose language could simplify and speed up learning and operating the system. But it also allowed the prototype to implement most of the language using an existing dialect of Python, and embedded Starlark interpreter. The base dependency model makes the basic configuration management system simpler to understand. But it also meant that only approximately ten base resources were needed. The hasChanges resource re-evaluation model meant that, unlike with industry leading configuration management systems, there was just a single type of dependency to implement. The use of standard programming language concepts (functions, if-else, for) greatly simplified both the implementation and explanation of the system. This is also an area that has shown itself as challenging to implement to DSLs (domain specific languages) as an afterthought, as shown by the use of Jinja templated code generation to implement loops in the industry leading tool Salt.

Some technical decisions in the prototype proved useful. Distributing the whole system as a single static (libc-linked) binary and allowing the master to work in any directory made multiple security realms and version control easy. Static linking made it simple to adapt to unexpected and changing execution environments. The bigger case organization turned out to have an environment where both policies and technical solutions made it complicated to install additional packages. A minor downside of static linking was the use of Python dialect instead of full actual Python, and the use of built-in SSH for initial

deployment.

The two case studies answered RQ6 and completed OBJ6. The case studies validated the concepts from RQ1-RQ4 (OBJ1-OBJ4). They helped guide the questions in the semi-structured expert interviews (RQ7, OBJ7). The research questions and objectives are listed in table 1.

6.6 Expert Interviews

The effect of organizational, contractual and legal factors was an interesting finding in the interviews. These non-technical matters limit the benefits provided by technology, both for existing solutions and the Hidden Master architecture. Some of these challenges might only be visible once the problems of manual work have been solved with configuration management systems.

Three themes raised from expert interviews:

- Administrative (non-technical)
- External factors (law, contracts, inter-organizational)
- Technology

Thematic analysis showed that non-technological aspects are interlinked with technology, and affect the benefit available from technology. In this regard, these themes bear some similarity to thematic analysis on DevSecOps publications by Rajapakse et al. (2022). Their four categories (people, practices, tools and infrastructure) also link technology with people and organizations.

Many administrative and external factors limit the benefit of configuration management systems. The systems are getting more complex. Advanced tooling requires more tooling. Integration challenges are created not only trough legacy systems but also due to limited scope of some systems. Combined with the temptation to scale systems to full potential of these tools and the resulting need for higher levels of abstraction, there is a risk of skill gap.

You might need less employees, but they need to be more skillful. Configuration management tools could help to retain these skilful workes, as the reduction of repetitive and tedious tasks makes the work more interesting.

The challenges faced by organizations embracing configuration management seem to only emerge after the previous level of challenges have been solved. Experts pointing out the adminstrative and external challenges are generally happy with the speed and productivity provided by CM, but seem to be looking for the next level of benefit from these tools.

The Hidden Master architecture and the prototype have to operate in this environment. Contracts, certificates and policies might limit or slow down

extracting the benefit provided by technology. For example, a policy requiring throughout, expensive audit or a new cloud provider could limit the use of inter-operator scaling provided by the Hidden Master. The advanced key management and encryption of HM could be limited by law in some markets, such as Russia or China.

Configuration management systems were found to speed up operations, in good and bad. Mistakes in repeating manual work were reduced, but any mistake happening with configuration management would be quickly deployed to large number of machines. The Hidden Master architecture was found to be very secure, but the model itself cannot remove all risk. Human factors would still provide an avenue of attack. The use of PGP keys would make it possible to reduce physical and human risk by using hardware security modules and by requiring two signatures.

All respondents agreed that the presented models and architecture could be expected to reduce costs and risks, improve developer productivity and allow faster time-to-market.

Interviewees found that the Hidden Master architecture and related contributions could improve security, scalability and resiliency. As potential target markets, most respondents identified embedded, rugged and IoT systems. Some also mentioned offensive work, developing markets and small businesses. Key management was found to be a crucial factor in the security of this architecture.

Technical immaturity of the prototype compared to existing, mature solutions in the market and inertia in organzations were found to be the main challenges for adoption of the software prototype.

Multiple respondents requested to be notified when the prototype and its source code would be publicly released. They also requested it to be released under a Free (FSF definition) license. Some expected that the HM architecture will be copied by other CM tools once public. Respondents offered multiple ideas to mature the work, such as the adding of integration points, documentation and productization.

Interviewees agreed on the multiple potential business benefits of the HM architecture. All respondents found that it could reduce costs and risks in protecting the system, better protecting the master secret key, scaling by leveraging the commodity web server file serving capabilities when running in small networks, and deploying IoT devices across organizations, scaling geographically.

All respondents found that the improvements to IaC could allow faster time-to-market and increase developer productivity, thus providing direct business

benefits. All respondents found the language and resource models very simple, easy to learn and understand which was one of the design goals of these systems.

6.7 Future Research

The Hidden Master architecture makes new transports both possible and cheap to implement. More transports could be researched and made the subject of experiment. Currently, many European countries are expecting electricity blackouts. This emphasizes the need for resiliency and security for configuration management systems, and could create demand for more research on peer-to-peer transports.

Configuration management aims for stability and a steady pace. As the typical use of configuration management tools does not require the speed of real time two-way connection between a master and agent, it could make sense for mature configuration management tools to implement the Hidden Master architecture. Based on some initial experiments, this might require large changes in some cases.

Configuration management should be simplified further. Complexity and the need for highly skillful workers was mentioned in multiple interviews. Complexity affects both the use and development of configuration management tools. As seen with the base resource model, the hasChanges revalidation model and the idempotent imperative general purpose language presented in this work, simplification can achieve multiple seemingly conflicting goals of simplifying development, improving ease of use and likely time-to-market - and even make the tool more adaptable. It is possible that further simplification - both in terms of concept and implementation - is possible. For example, maybe the single resource revalidation with hasChanges could become fully automatic in typical and uncomplicated cases.

The proliferation of Go-based malware was seen towards the end of this thesis. The builtin cross platform compilation could make it easy to support different platforms. The use of newer programming languages in malware can provide new inspiration for configuration management.

The secret key of the master is the single most valuable file in the whole network controlled by the configuration management. The use of PGP for the cryptosystem would allow bringing advanced features of key management to configuration management. Keys can be saved on hardware tokens, and multiple signatures from different persons could be required for changes on key infrastructure. Old catalogs are not needed, and in the advanced prototype they are automatically discarded. Maybe forward security could be used to

deny their use from the adversary in case of any major advancement in breaking cryptography.

Case studies and expert interviews in this study gave indications that imperative, general purpose language for idempotent configuration is easy to learn and debug. This could be studied further with multiple groups of configuration management students, such as those the researcher is teaching.

As the prototype matures, it could be security audited and deployed to production networks with less precautions, thus providing further validation and feedback from the field. Maturing to real life use would shed more light on the business benefits, which could then be more directly observed.

The approach taken by this thesis could be used for finding new concepts from malware, even those that are developed in the future. Suitable concepts could be identified using the stage model proposed in this work. These concepts could be developed into a coherent tool using a design approach inspired by the one described in this thesis. The design could be developed into a prototype and validated using laboratory testing, case studies and expert interviews. Based on the history of malware evolution, we can expect criminals to improve their malware in the future and provide new ideas to apply to benign configuration management.

Appendices

Appendix: Hidden Master Architecture Encryption Demonstration

This is the simple proof of the concept prototype. The advanced prototype is over 4000 source code lines of Go, and does not fit as an appendix.

README

```
Demonstrate *hidden master architecture* encryption operations
Copyright 2016 Tero Karvinen http://TeroKarvinen.com

'make' to run full demo.

stage/
    master/ # the computer with trusted keys,
            # only temporarily on the internet
    public/ # untrusted public web server
    slave/  # slave to be configured

The end result is the configuration management system
creating /tmp/helloTero.txt.
```

Agent Catalog

```
#!/usr/bin/env pup
# hello-slave.pp - These are the secret instructions to slave

file { "/tmp/helloTero.txt":
    ensure => "present",
    content=> "See you at TeroKarvinen.com!\n",
}
```

Makefile

```
# Demonstrate *hidden master architecture* encryption operations
# Copyright 2016 Tero Karvinen http://TeroKarvinen.com

all: clean build

clean:
```

```
	@echo "## Cleaning up... ##"
	# new
	rm -rf stage/ /tmp/helloTero.txt

build:
	@echo "## Generating keypairs... ##"
	./genkey.sh slave
	./genkey.sh master

	@echo "## Key exchange... ##"
	gpg2 --homedir stage/slave/ --export --armor \
	    |gpg2 --homedir stage/master/ --import
	gpg2 --homedir stage/master/ --export --armor \
	    |gpg2 --homedir stage/slave/ --import

	@echo "## Publish encrypted catalog to untrusted server... ##"
	mkdir -p stage/public/
	cp -v hello-slave.pp stage/master/
	gpg2 --homedir stage/master/ \
	    --sign --encrypt \
	    --recipient=slave@cct \
	    --trust-model always \
	    --armor \
	    --batch \
	    --output=stage/public/encrypted.asc \
	    stage/master/hello-slave.pp

	@echo "## Download and decrypt catalog to slave... ##"
	mkdir -p stage/slave/
	gpg2 --homedir stage/slave/ \
	    --trust-model always \
	    --output stage/slave/hello-slave.pp \
	    --decrypt stage/public/encrypted.asc
	puppet apply stage/slave/hello-slave.pp

	@echo "## This file is the result of configuration management on slave
	cat /tmp/helloTero.txt  # make will fail if file does not exist
```

genkey.sh

```
#!/bin/bash
# Generate GPG keypair
# Copyright 2016 Tero Karvinen http://TeroKarvinen.com

# sudo apt-get -y install rng-tools

if [ -z $1 ]; then
    echo "usage: genkey.sh NAME"
    exit
fi

set -o verbose

NAME=$1
GPGHOME="stage/$NAME/"

mkdir -p $GPGHOME
chmod og-rwx $GPGHOME

gpg2 --homedir $GPGHOME --batch --gen-key << ENDINST
    %echo Generating key for slave...
    Key-Type: default
    Key-Length: default
    # Subkey is required, or sign+encrypt fails with
    # "sign+encrypt failed: Unusable public key"
    Subkey-Type: default
    Subkey-Length: default
    Name-Real: $NAME INSECURE test key
    Name-Email: $NAME@cct
    Expire-Date: 0
    %no-protection # no passphrase
    #%pubring slave.pub
    #%secring slave.sec
    %commit
    %echo Done key generation.
ENDINST
gpg2 --homedir $GPGHOME --list-keys

echo "See you at TeroKarvinen.com"\
```

```
    |gpg2 --homedir $GPGHOME --sign --encrypt \
        --recipient $NAME@cct --armor
```

Appendix: Estimating the Size of Some Domain Specific Languages

Testing environment

```
$ salt --version; grep DESC /etc/lsb-release; uname -m
salt 2017.7.4 (Nitrogen)
DISTRIB_DESCRIPTION="Ubuntu 18.04.5 LTS"
x86_64
```

Classes controlling the agent state in Salt DSL

```
$ sudo salt-call --local sys.state_doc|grep -P '^    \w+:'|head -3
    acl:
    alias:
    alternatives:
$ sudo salt-call --local sys.state_doc|grep -P '^    \w+:'|wc -l
153
```

Functions controlling agent state the in Salt DSL

```
$ sudo salt-call --local sys.state_doc|grep -P '^    \S+:'|grep '\.'|head -3
    acl.absent:
    acl.present:
    alias.absent:
$ sudo salt-call --local sys.state_doc|grep -P '^    \S+:'|grep '\.'|wc -l
510
```

Length of Salt documentation (without control structures such as loop and if-else)

```
$ sudo salt-call --local sys.state_doc|wc -l
21962
```

Estimating the Size of Puppet DSL

```
$ puppet --version
5.4.0
$ dpkg --listfiles puppet|grep functions/|head -3
/usr/lib/ruby/vendor_ruby/puppet/functions/alert.rb
/usr/lib/ruby/vendor_ruby/puppet/functions/all.rb
/usr/lib/ruby/vendor_ruby/puppet/functions/annotate.rb
$ dpkg --listfiles puppet|grep functions/|wc -l
```

Appendix: Questionnaire for Semi-Structured Interview

This is the questionnaire for the semistructured expert interviews, conducted to complete objective 7 “Identify and validate potential business benefits in expert interviews”, and to provide an answer to RQ7 “What potential business benefits do experts see for the models and the research prototype?”.

The questions here worked as prompt for open-ended commentary. Participants were encouraged to give their free form feedback based on (or inspired by) any of the prompts and questions here. All participants commented and discussed the areas mentioned here.

Your answers to these questions will be included as part of the doctoral thesis and other research by Tero Karvinen. These answers are pseudonymous, and your name or your company name will not be published without your permission. If you want, you will be mentioned in the thank you list (Acknowledgments). The current prototype has had limited testing in a production environment, but it is not suitable for large scale distribution or environments with high security requirements. We can skip the technical details of your current company systems that are sensitive, such as details concerning the EDR (Endpoint detection and response) configuration.

- What is your current position?
- How long have you worked with the company?
- How long have you worked in management?
- What is your background with configuration management systems (centrally controlling a lot of computers)?
- What configuration management or DevOps tools does your company use?
- What do you consider the weak and strong points of these tools?
- What do you consider the main weaknesses of all tools in modern DevOps and configuration management systems?

Demonstration

A brief demonstration of the main research prototype is performed by the researcher. The main contributions implemented in the prototype are listed (Hidden master architecture, idempotency with imperative general purpose language, simplified IaC resource dependency handling, simplified model for resource functions). Network structure is shown. The main case study is described. If time permits, requested features are demonstrated.

Please evaluate how well the system provides these business benefits. The scale is from 1 (not at all) to 5 (extremely well). Feel free to ask for more details on any question.

Hidden Master Architecture - The hidden master keeps the master private keys out of Internet visible servers

- Reduces the cost of protecting the configuration management system
- Improves security by better protecting master secret keys
- Reduces the cost of scaling by leveraging the file serving capabilities of commodity web servers
- Improves resiliency against network problems with efficient geographic scaling (by adding untrusted intermediate servers in the cloud in different countries)
- Improves resiliency and reduces costs with inter-operator scaling (by renting untrusted intermediate servers from multiple different cloud operators)
- Reduces the cost to both the sending and receiving organization when deploying new machines and IoT devices (by bypassing NAT without exposing master, as only the intermediate servers need to be in public known addresses)
- Reduces the cost and risk when preparing to deploy an unknown number of machines or IoT devices (by leveraging campaign keys, back channeling individual agent key negotiations and predeploying campaign keys on provisioning)
- Reduces the relative cost and risk of running a configuration management system in small networks (where 24 hour incident response is not available and monitoring is limited).

Idempotent use of imperative general purpose language and improved resource models

- Saves costs in new employee orientations to systems (uses familiar language such as a Python dialect; and uses familiar control structures such as if-else for functions (<20 functions vs 500).
- Quickens the time to market by leveraging existing skills
- Reduces resistance to DevOps changes by allowing programmers to use familiar languages and patterns
- Saves cost on errors when debugging dependencies (hasChanges() versus complicated IaC resource models)
- Increases developer productivity by leveraging existing tool support, such as syntax highlighting

- Reduces the risk from supply chain dependency (Conftero's simplified resources model uses just ~5 base resources).

Summary

What additional benefits or downsides do you see here?

Do you have additional comments?

What future advice would you give for developing this system?

Misc

Do you wish to be mentioned in Aknowledgements?

Appendix: Correlation Matrix for Likert Scale Interview Questions

Spearman correlation of questions using Likert scale. Duplicate pairs are excluded. Upper line shows Spearman correlation of the pair, lower line shows p value. Statistically significant correlations ($p < 0.05$) are marked with an asterisk "*". Correlation of item with itself is 1.00 at p=0.00, and it's excluded.

-	b	c	d	e	f	g	h	i	j	k	l	m	n
a	0.71	1.00*	0.69	1.00*	1.00*	0.95*	0.71	0.58	0.71	0.74	0.95*	0.45	0.95*
	0.12	0.00	0.13	0.00	0.00	0.00	0.12	0.23	0.12	0.09	0.00	0.37	0.00
b	-	0.71	0.98*	0.71	0.71	0.78	1.00*	0.61	1.00*	0.67	0.78	0.63	0.78
	-	0.12	0.00	0.12	0.12	0.07	0.00	0.20	0.00	0.14	0.07	0.18	0.07
c	.	-	0.69	1.00*	1.00*	0.95*	0.71	0.58	0.71	0.74	0.95*	0.45	0.95*
	.	-	0.13	0.00	0.00	0.00	0.12	0.23	0.12	0.09	0.00	0.37	0.00
d	.	.	-	0.69	0.69	0.82*	0.98*	0.70	0.98*	0.73	0.82*	0.77	0.82*
	.	.	-	0.13	0.13	0.04	0.00	0.12	0.00	0.10	0.04	0.07	0.04
e	.	.	.	-	1.00*	0.95*	0.71	0.58	0.71	0.74	0.95*	0.45	0.95*
	.	.	.	-	0.00	0.00	0.12	0.23	0.12	0.09	0.00	0.37	0.00
f	.	.	.	.	-	0.95*	0.71	0.58	0.71	0.74	0.95*	0.45	0.95*
	.	.	.	.	-	0.00	0.12	0.23	0.12	0.09	0.00	0.37	0.00
g	.	.	.	.	.	-	0.78	0.73	0.78	0.83*	1.00*	0.71	1.00*
	.	.	.	.	.	-	0.07	0.10	0.07	0.04	0.00	0.12	0.00
h	.	.	.	.	.	.	-	0.61	1.00*	0.67	0.78	0.63	0.78
	.	.	.	.	.	.	-	0.20	0.00	0.14	0.07	0.18	0.07
i	.	.	.	.	.	.	.	-	0.61	0.82*	0.73	0.77	0.73
	.	.	.	.	.	.	.	-	0.20	0.04	0.10	0.07	0.10
j	.	.	.	.	.	.	.	.	-	0.67	0.78	0.63	0.78

```
 |.   |.    |.    |.    |.    |.    |.    |.   |-    |0.14 |0.07 |0.18|0.07
k|.   |.    |.    |.    |.    |.    |.    |.   |.    |-    |0.83*|0.71|0.83*
 |.   |.    |.    |.    |.    |.    |.    |.   |.    |-    |0.04 |0.12|0.04
l|.   |.    |.    |.    |.    |.    |.    |.   |.    |.    |-    |0.71|1.00*
 |.   |.    |.    |.    |.    |.    |.    |.   |.    |.    |-    |0.12|0.00
m|.   |.    |.    |.    |.    |.    |.    |.   |.    |.    |.    |-   |0.71
 |.   |.    |.    |.    |.    |.    |.    |.   |.    |.    |.    |-   |0.12
```

- a Reduce cost of protecting configuration management system
- b Improve security by better protecting master secret keys
- c Reduce cost of scaling by leveraging file serving capabilities of commodity web servers
- d Improve resiliency against network problems with efficient geographic scaling
- e Improve resiliency and reduce costs with inter-operator scaling
- f Reduce cost to both sending and receiving organization when deploying new machines and IoT devices
- g Reduce cost and risk when preparing to deploy unknown number of machines or IoT devices
- h Reduce relative cost and risk of running configuration management system in small networks
- i Save costs in new employee orientation to systems.
- j Faster time to market by leveraging existing skills
- k Reduce resistance to DevOps change by allowing programmers to use familiar languages and patterns
- l Save cost of errors when debugging dependencies
- m Increase developer productivity by leveraging existing tool support, such as syntax highlighting
- n Reduce risk from supply chain dependency

References

Adesemowo, A.K. and Thompson, K.-L. (2013). Service desk link into IT asset disposal: A case of a discarded IT asset. *2013 international conference on adaptive science and technology*. ICAST. November 2013. IEEE, 1–4. Available from https://doi.org/10.1109/ICASTech.2013.6707517.

Alkhateeb, S. (2016). Cyber crimes. *International Journal of Scientific & Engineering Research*, 7 (4), 918.

Al-Shaer, R., Ahmed, M. and Al-Shaer, E. (2019). Statistical learning of APT TTP chains from MITRE ATT&CK. 2019.

Anderson, P. and Cheney, J. (2012). Toward Provenance-based Security for Configuration Languages. *Proceedings of the 4th USENIX Conference on Theory and Practice of Provenance*. TaPP'12. 2012. Berkeley, CA, USA: USENIX Association, 2–2. Available from http://dl.acm.org/citation.cfm?id=2342875.2342877.

Anderson, R. et al. (2021). Silicon den: Cybercrime is entrepreneurship. *Workshop on the economics of information security*. 2021. Available from https://weis2021.econinfosec.org/wp-content/uploads/sites/9/2021/06/weis21-anderson.pdf.

Andrade, P. et al. (2012). Review of CERN Data Centre Infrastructure. *Journal of Physics: Conference Series*, 396 (4), 042002. Available from https://doi.org/10.1088/1742-6596/396/4/042002 [Accessed 3 June 2016].

Arfman, J.M. and Roden, P. (1992). Project athena: Supporting distributed computing at MIT. *IBM Systems Journal*, 31 (3), 550–563.

Balis, B. et al. (2018). Holistic approach to management of IT infrastructure for environmental monitoring and decision support systems with urgent computing capabilities. *Future Generation Computer Systems*, 79, 128–143. Available from https://doi.org/10.1016/j.future.2016.08.007.

Barford, P. and Yegneswaran, V. (2007). An inside look at botnets. *Malware detection*. Springer, 171–191.

Bhat, M. et al. (2022). *Innovation insight for continuous infrastructure automation*. Gartner.

Binsalleeh, H. et al. (2010). On the analysis of the Zeus botnet crimeware toolkit. *2010 Eighth Annual International Conference on Privacy Security and Trust (PST)*. August 2010. 31–38. Available from https://doi.org/10.1109/PST.2010.5593240.

Braun, V. and Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3 (2), 77–101. Available from https://doi.org/10.1191/1478088706qp063oa.

Brooks, B., Davidson, A. and Gregor, I. (2014). The evolving relationship between simulation and emulation: Faster than real-time controls testing.

Proceedings of the 2014 winter simulation conference, edited by tolk a, et al. 2014.

Brunnert, A. et al. (2015). Performance-oriented DevOps: A research agenda. *arXiv preprint arXiv:1508.04752*.

Burgess, M. (1998). Computer Immunology. *LISA*. 1998. 283–298.

Caballer, M. et al. (2023). Infrastructure manager: A TOSCA-based orchestrator for the computing continuum. *Journal of Grid Computing*, 21 (3), 51. Available from https://doi.org/10.1007/s10723-023-09686-7.

CentOS project. (2016). Index of /centos/7/os/X86_64/Packages. Available from http://mirror.centos.org/centos/7/os/x86_64/Packages/ [Accessed 22 November 2016].

CFEngine. (2016). LinkedIn infrastructure and operations. Available from https://cfengine.com/wp-content/uploads/2014/11/LinkedIn_CFEngine_Case_Study.pdf.

Champine, G.A., Geer, D.E. and Ruh, W.N. (1990). Project athena as a distributed computer system. *Computer*, 23 (9), 40–51.

Chen, T.M. and Robert, J.-M. (2004). The evolution of viruses and worms. *Statistical methods in computer security*, 1.

Cheshire, S., Aboba, B. and Guttman, E. (2005). *RFC 3927 - dynamic configuration of IPv4 link-local addresses*.

Colarik, A., Thomborson, C. and Janczewski, L. (2004). Update/patch management systems: A protocol taxonomy with security implications. In: Deswarte, Y. Cuppens, F. Jajodia, S. et al. (eds.). *Information security management, education and privacy*. IFIP international federation for information processing. 2004. Boston, MA: Springer US, 67–80. Available from https://doi.org/10.1007/1-4020-8145-6_5.

Coulter et al. (2017). Desired state configuration overview for decision makers. Available from https://docs.microsoft.com/en-us/powershell/dsc/decisionmaker.

Crnkovic, G.D. (2010). Constructive research and info-computational knowledge generation. *Model-Based Reasoning in Science and Technology*. Springer, 359–380. Available from http://link.springer.com/chapter/10.1007/978-3-642-15223-8_20 [Accessed 4 March 2017].

CVE-2020-11652. (2020). Available from http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-11652 [Accessed 11 December 2022].

Delaet, T., Joosen, W. and Van Brabant, B. (2010). A Survey of System Configuration Tools. *LISA*. 2010. 1–8. Available from https://www.usenix.org/event/lisa10/tech/full_papers/Delaet.pdf [Accessed 22 November 2016].

Dewri, R. et al. (2012). Optimal security hardening on attack tree models of

networks: A cost-benefit analysis. *International Journal of Information Security*, 11 (3), 167–188. Available from https://doi.org/10.1007/s10207-012-0160-y [Accessed 30 September 2016].

Dittrich, D. and Dietrich, S. (2008). P2P as botnet command and control: A deeper insight. *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*. 2008. IEEE, 41–48. Available from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4690856 [Accessed 3 June 2016].

Duplyakin, D., Haney, M. and Tufo, H. (2015). Highly Available Cloud-Based Cluster Management. *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. May 2015. 1201–1204. Available from https://doi.org/10.1109/CCGrid.2015.125.

Ellison, R.J. et al. (1999). Survivability: Protecting your critical systems. *IEEE Internet Computing*, 3 (6), 55–63. Available from https://doi.org/10.1109/4236.807008.

Endsley, M.R. (1988). Design and evaluation for situation awareness enhancement. *Proceedings of the human factors society annual meeting*. 1988. SAGE Publications Sage CA: Los Angeles, CA, 97–101.

Farooq, H.M. and Otaibi, N.M. (2018). Optimal machine learning algorithms for cyber threat detection. *2018 UKSim-AMSS 20th international conference on computer modelling and simulation (UKSim)*. 2018. IEEE, 32–37.

Fox, G.C. et al. (2015). Hpc-abds high performance computing enhanced apache big data stack. *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. 2015. IEEE, 1057–1066.

Franke, U. and Brynielsson, J. (2014). Cyber situational awareness–a systematic review of the literature. *Computers & security*, 46, 18–31.

Free Software Foundation. (2015). The Free Software Definition. Available from https://www.gnu.org/philosophy/free-sw.html [Accessed 22 November 2016].

F-Secure. (2020). SaltStack authorization bypass.

Fu, W. et al. (2017). *31st european conference on object-oriented programming (ECOOP 2017)*. 2017.

Fung, C. et al. (2005). Survivability analysis of distributed systems using attack tree methodology. *MILCOM 2005 - 2005 IEEE Military Communications Conference*. October 2005. 583–589. Available from https://doi.org/10.1109/MILCOM.2005.1605745.

Gu, G., Zhang, J. and Lee, W. (2008). BotSniffer: Detecting botnet command and control channels in network traffic.

Gyarmati, L. et al. (2013). Free-scaling your data center. *Computer Networks*, 57 (8), 1758–1773.

Hagemark, B. (1990). A Language and System for Configuring Many Computers as One Computing Site. *Proceedings of the Workshop on Large Installation System Administration III (USENIX Association)*. Available from https://cs.brown.edu/research/pubs/theses/masters/1990/hagemark.pdf [Accessed 10 February 2017].

Hariri, S. et al. (2003). Impact Analysis of Faults and Attacks in Large-Scale Networks. *IEEE Security & Privacy*, 1 (5), 49–54.

Harrison, H.E., Schaefer, S.P. and Yoo, T.S. (1988). Rtools: Tools for software management in a distributed computing environment. *Proceedings of the summer USENIX conference*. 1988. 85–93.

Hastings and Kazanciyan. (2016). Black Hat Asia 2016 17 DSCOMPROMISED A Windows DSC Attack Framework. Available from https://www.youtube.com/watch?v=WWJnMxv8P0g [Accessed 10 October 2016].

Hemminger, S. (2005). Network emulation with NetEm. *Linux conf au*. 2005. Citeseer, 2005.

Hevner, A. and Chatterjee, S. (2010). Design science research in information systems. *Design research in information systems*. Springer, 9–22.

Hevner, A.R. et al. (2004). Design science in information systems research. *MIS quarterly*, 75–105.

Hintsch, J., Görling, C. and Turowski, K. (2016). A Review of the Literature on Configuration Management Tools. Available from http://aisel.aisnet.org/cgi/viewcontent.cgi?article=1005/&context=confirm2016 [Accessed 22 November 2016].

Huang, C.D., Hu, Q. and Behara, R.S. (2008). An economic analysis of the optimal information security investment in the case of a risk-averse firm. *International journal of production economics*, 114 (2), 793–804. Available from https://doi.org/10.1016/j.ijpe.2008.04.002.

Huang, K., Siegel, M. and Madnick, S. (2018). Systematically understanding the cyber attack business: A survey. *ACM Computing Surveys (CSUR)*, 51 (4), 1–36.

Hummer, W. et al. (2013). Testing idempotence for infrastructure as code. *ACM/IFIP/USENIX international conference on distributed systems platforms and open distributed processing*. 2013. Springer, 368–388.

Hutchins, E.M., Cloppert, M.J. and Amin, R.M. (2011). Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1 (1), 80.

Jacquemart, Q. and Urvoy-Keller, G. (2018). *D3. 3–inter-site network virtualization orchestrator*. Available from https://prestocloud-project.eu/documents/deliverables/PrEstoCloud-D3.3.pdf.

Kartaltepe, E.J. et al. (2010). Social network-based botnet command-and-control: Emerging threats and countermeasures. *Applied Cryptography and Network Security*. 2010. Springer, 511–528. Available from http://link.springer.com/chapter/10.1007/978-3-642-13708-2_30 [Accessed 3 June 2016].

Karvinen, T. and Li, S. (2017). Investigating survivability of configuration management tools in unreliable and hostile networks. *2017 3rd international conference on information management (ICIM)*. April 2017. Chengdu, China: IEEE, 327–331. Available from https://doi.org/10.1109/INFOMAN.2017.7950402.

Keti, F. and Askar, S. (2015). Emulation of software defined networks using mininet in different simulation environments. *2015 6th international conference on intelligent systems, modelling and simulation*. 2015. IEEE, 205–210.

Kinkelin, H. et al. (2018). Trustworthy configuration management for networked devices using distributed ledgers. *NOMS 2018-2018 IEEE/IFIP network operations and management symposium*. 2018. IEEE, 1–5. Available from https://ieeexplore.ieee.org/abstract/document/8406324/.

Kinkelin, H. et al. (2019). Multi-party authorization and conflict mediation for decentralized configuration management processes.

Kitchenham, B. et al. (2012). Trends in the quality of human-centric software engineering experiments–A quasi-experiment. *IEEE Transactions on Software Engineering*, 39 (7), 1002–1017. Available from https://ieeexplore.ieee.org/abstract/document/6374196/.

Kos, J., Milutinović, M. and Čehovin, L. (2015). Nodewatcher: A substrate for growing your own community network. *Computer Networks*, 93, 279–296. Available from https://doi.org/10.1016/j.comnet.2015.09.021.

Kosar, T., Bohra, S. and Mernik, M. (2016). Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71, 77–91.

Kostromin, R. (2020). Survey of software configuration management tools of nodes in heterogeneous distributed computing environment. *ICCS-DE*. 2020. 156–165.

Kuhrmann, M., Fernández, D.M. and Daneva, M. (2017). On the pragmatic design of literature studies in software engineering: An experience-based guideline. *Empirical Software Engineering*, 22 (6), 2852–2891. Available from https://doi.org/10.1007/s10664-016-9492-y.

Kumara, I. et al. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology*, 137, 106593.

Ledakis, G. et al. (2018). *D5. 7 PrEstoCloud security enforcement mechanism-*

iteration. Available from https://prestocloud-project.eu/documents/deliverables/PrEstoCloud-D5.7.pdf.

Lemay, A. et al. (2018). Survey of publicly available reports on advanced persistent threat actors. *Computers & Security*, 72, 26–59.

Lenders, V., Tanner, A. and Blarer, A. (2015). Gaining an edge in cyberspace with advanced situational awareness. *IEEE Security & Privacy*, 13 (2), 65–74.

Lipton, P. and Lauwers, C. (2019). TOSCA simple profile in YAML version 1.3. Available from https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/csprd01/TOSCA-Simple-Profile-YAML-v1.3-csprd01.html.

Mansouri, Y., Prokhorenko, V. and Babar, M.A. (2020). An automated implementation of hybrid cloud for performance evaluation of distributed databases. *Journal of Network and Computer Applications*, 167, 102740. Available from https://doi.org/10.1016/j.jnca.2020.102740.

Marelli, M. (2022). The SolarWinds hack: Lessons for international humanitarian organizations. *International Review of the Red Cross*, 104 (919), 1267–1284.

Marsa-Maestre, I. et al. (2019). REACT: Reactive resilience for critical infrastructures using graph-coloring techniques. *Journal of Network and Computer Applications*, 145, 102402. Available from https://doi.org/10.1016/j.jnca.2019.07.003.

Mauw, S. and Oostdijk, M. (2005). Foundations of attack trees. *International Conference on Information Security and Cryptology*. 2005. Springer, 186–198. Available from http://link.springer.com/chapter/10.1007/11734727_17 [Accessed 28 September 2016].

Maymi, F. et al. (2017). Towards a definition of cyberspace tactics, techniques and procedures. *2017 IEEE international conference on big data (big data)*. 2017. IEEE, 4674–4679.

McMullin, C. (2023). Transcription and qualitative methods: Implications for third sector research. *VOLUNTAS: International Journal of Voluntary and Nonprofit Organizations*, 34 (1), 140–153. Available from https://doi.org/10.1007/s11266-021-00400-3.

Mead, N.R. et al. (2000). *Survivable network analysis method*. DTIC Document. Available from http://oai.dtic.mil/oai/oai?verb=getRecord/&metadataPrefix=html/&identifier=ADA383771 [Accessed 10 October 2016].

Mernik, M., Heering, J. and Sloane, A.M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37 (4), 316–344.

Mitchell, D. et al. (2020). Mozilla-releng / build-puppet. Available from https://github.com/mozilla-releng/build-puppet.

Mitre. (2019). ATT&CK matrix for enterprise. Available from https://attack.mitre.org/.

Moore, A.P., Ellison, R.J. and Linger, R.C. (2001). *Attack modeling for information security and survivability*. DTIC Document.

Moubarak, J., Chamoun, M. and Filiol, E. (2017). Comparative study of recent MEA malware phylogeny. *Computer and communication systems (ICCCS), 2017 2nd international conference on*. 2017. IEEE, 16–20.

NIST. (2016). The United States Government Configuration Baseline (USGCB) - NIST. Available from https://usgcb.nist.gov/index.html [Accessed 3 June 2016].

Noureddine, M.A. et al. (2016). A game-theoretic approach to respond to attacker lateral movement. *International conference on decision and game theory for security*. 2016. Springer, 294–313.

Parameswaran, U.D., Ozawa-Kirk, J.L. and Latendresse, G. (2020). To live (code) or to not: A new method for coding in qualitative research. *Qualitative Social Work*, 19 (4), 630–644. Available from https://doi.org/10.1177/1473325019840394.

Pasquale, L. et al. (2009). Distributed cross-domain configuration management. In: Krämer, B.J. Lin, K.-J. and Narasimhan, P. (eds.). *Service-oriented computing – ICSOC 2007*. Lecture notes in computer science. 2009. Berlin, Heidelberg: Springer Berlin Heidelberg, 622–636. Available from https://doi.org/10.1007/978-3-642-10383-4_45.

Peffers, K. et al. (2007). A design science research methodology for information systems research. *Journal of management information systems*, 24 (3), 45–77.

Perera, N. (2016). Automatic Configuration Management - Autodiscovery of Configuration Items and Automatic Configuration Verification. May 2016. American Institute of Aeronautics and Astronautics. Available from https://doi.org/10.2514/6.2016-2610 [Accessed 2 June 2016].

Phil, P.R. et al. (2014). OWASP Top 10: The Top 10 Most Critical Web Application Security Threats Enhanced with Text Analytics and Content by PageKicker Robot Phil 73. Available from http://dl.acm.org/citation.cfm?id=2788303 [Accessed 10 October 2016].

Piirainen, K.A. and Gonzalez, R.A. (2013). Seeking constructive synergy: Design science and the constructive research approach. *International conference on design science research in information systems*. 2013. Springer, 59–72.

Poat, M.D., Lauret, J. and Betts, W. (2015). Configuration Management and Infrastructure Monitoring Using CFEngine and Icinga for Real-time Heterogeneous Data Taking Environment. *Journal of Physics: Conference*

Series, 664 (5), 052020. Available from https://doi.org/10.1088/1742-6596/664/5/052020 [Accessed 2 June 2016].

Pols, P. (2017). The unified kill chain: Designing a unified kill chain for analyzing, comparing and defending against cyber attacks [Master's thesis]. Cyber Security Academy (CSA).

Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108, 65–77. Available from https://doi.org/10.1016/j.infsof.2018.12.004.

Rajapakse, R.N. et al. (2022). Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, 141, 106700. Available from https://doi.org/10.1016/j.infsof.2021.106700.

Rong, C. et al. (2022). OpenIaC: Open infrastructure as code - the network is my computer. *Journal of Cloud Computing*, 11 (1), 12. Available from https://doi.org/10.1186/s13677-022-00285-7.

SaltStack Inc. (2022). Hardening salt.

Schneier, B. (1999). Attack trees. *Dr. Dobb's journal*, 24 (12), 21–29.

Scott, J.A. and Nisse, D. (2001). Software configuration management. *SWEBOK*, 103.

Shambaugh, R., Weiss, A. and Guha, A. (2016). Rehearsal: A configuration verification tool for puppet. *Proceedings of the 37th ACM SIGPLAN conference on programming language design and implementation*. 2016. 416–430.

Sharma, T., Fragkoulis, M. and Spinellis, D. (2016). Does your configuration code smell? 2016. ACM Press, 189–200. Available from https://doi.org/10.1145/2901739.2901761 [Accessed 22 November 2016].

Sherman, A. et al. (2005). ACMS: The akamai configuration management system.

Silva, J. de C. et al. (2019). Management platforms and protocols for internet of things: A survey. *Sensors*, 19 (3), 676. Available from https://doi.org/10.3390/s19030676.

Silva, S.S.C. et al. (2013). Botnets: A survey. *Computer Networks*, 57 (2), 378–403. Available from https://doi.org/10.1016/j.comnet.2012.07.021 [Accessed 26 January 2017].

Siqueira, M.A. et al. (2006). An architecture for autonomic management of ambient networks. In: Gaïti, D. Pujolle, G. Al-Shaer, E. et al. (eds.). *Autonomic networking*. Lecture notes in computer science. 2006. Berlin, Heidelberg: Springer Berlin Heidelberg, 255–267. Available from https://doi.org/10.1007/11880905_21.

Steiner, J.G. and Geer Jr, D.E. (1988). Network services in the Athena environ-

ment. *Project Athena, Massachusetts Institute of Technology, Cambridge, MA*, 2139. Available from https://doi.org/10.1.1.31.8727 [Accessed 10 February 2017].

Sterbenz, J.P.G. et al. (2010). Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54 (8), 1245–1265. Available from https://doi.org/10.1016/j.comnet.2010.03.005 [Accessed 10 October 2016].

Stocker, A. et al. (2022). An ICT architecture for enabling ancillary services in distributed renewable energy sources based on the SGAM framework. *Energy Informatics*, 5 (1), 5. Available from https://doi.org/10.1186/s42162-022-00189-5.

Strom, B.E. et al. (2017). *Finding cyber threats with ATT&CK-based analytics*. Mitre.

Strom, B.E. et al. (2018). *MITRE ATT&CK: Design and philosophy*. Mitre.

Święcicki, B. (2016). A Novel Approach to Automating Operating System Configuration Management. In: Grzech, A. Borzemski, L. Świątek, J. et al. (eds.). *Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology – ISAT 2015 – Part II*. Advances in intelligent systems and computing. Springer International Publishing, 131–142. Available from https://doi.org/10.1007/978-3-319-28561-0_10 [Accessed 22 November 2016].

Tang, C. et al. (2015). Holistic configuration management at Facebook. 2015. ACM Press, 328–343. Available from https://doi.org/10.1145/2815400.2815401 [Accessed 30 September 2016].

Toffetti, G. et al. (2017). Self-managing cloud-native applications: Design, implementation, and experience. *Future Generation Computer Systems*, 72, 165–179. Available from https://doi.org/10.1016/j.future.2016.09.002.

Tomarchio, O., Calcaterra, D. and Di Modica, G. (2020). Cloud resource orchestration in the multi-cloud landscape: A systematic review of existing frameworks. *Journal of Cloud Computing*, 9 (1), 1–24.

Treese, G.W. (1988). Berkeley UNIX on 1000 workstations: Athena changes to 4.3 BSD. *USENIX winter*. 1988. 175–182.

Tripp et al. (1998). Supplement to IEEEi Guide for Synthetic Fault Testing of AC High-Voltage Circuit Breakers Rated on a Symmetrical Current Basis 8-32 Recovery Voltage for Terminal Faults. Available from http://ieeexplore.ieee.org/iel4/5748/15360/x0117410.pdf [Accessed 3 June 2016].

Ullah, A. et al. (2023). Orchestration in the cloud-to-things compute continuum: Taxonomy, survey and future directions. *Journal of Cloud Computing*, 12 (1), 135. Available from https://doi.org/10.1186/s13677-023-00516-5.

Välimäki, M. (2005). The rise of open source licensing : a challenge to the use of intellectual property in the software industry [Doctoral thesis]. Helsinki University of Technology.

Vanbrabant, B. (2014). A Framework for Integrated Configuration Management of Distributed Systems. Available from https://lirias.kuleuven.be/handle/123456789/453199 [Accessed 2 June 2016].

Von Neumann, J. and Burks, A.W.(Arthur.W. (1966). *Theory of self-reproducing automata*. Urbana, University of Illinois Press. Available from http://archive.org/details/theoryofselfrepr00vonn_0.

Wettinger, J. et al. (2013). Integrating Configuration Management with Model-driven Cloud Management based on TOSCA. *CLOSER*. 2013. 437–446.

Wurster, M. et al. (2020). The essential deployment metamodel: A systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*, 35 (1), 63–75.

Xu, J. and Russello, G. (2022). Automated security-focused network configuration management: State of the art, challenges, and future directions. *2022 9th international conference on dependable systems and their applications (DSA)*. 2022. IEEE, 409–420. Available from https://doi.org/10.1109/DSA56465.2022.00061.

Zhao, Z. and Guo, H. (2018). Method for enforcing access control policies on NCMS. *2018 IEEE international conference on service operations and logistics, and informatics (SOLI)*. July 2018. Singapore: IEEE, 226–231. Available from https://doi.org/10.1109/SOLI.2018.8476718.