Secure Configuration and Management of Linux Systems using a Network Service Orchestrator

Vijay Satti

A Thesis

in

The Department

of

Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science at

Concordia University

Montréal, Québec, Canada

May 2019

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Vijay Satti**

Entitled: **Secure Configuration and Management of Linux Systems using a Network Service Orchestrator**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Walter Lucia

_____ Examiner
Dr. Anjali Agarwal

_____ Examiner
Dr. Amr Youssef

_____ Supervisor
Dr. J. William Atwood

Approved by _____
Dr. Chadi Assi, Graduate Program Director
Department of Concordia Institute for Information Systems Engineering

_____ 2019

_____
Dr. Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Secure Configuration and Management of Linux Systems using a Network Service Orchestrator

Vijay Satti

Manual management of the configuration of network devices and computing devices (hosts) is an error-prone task. Centralized automation of these tasks can lower the costs of management, but can also introduce unknown or unanticipated security risks. Misconfiguration (deliberate (by outsiders) or inadvertent (by insiders)) can expose a system to significant risks.

Centralized network management has seen significant progress in recent years, resulting in model-driven approaches that are clearly superior to previous "craft" methods. Host management has seen less development. The tools available have developed in separate task-specific ways.

This thesis explores two aspects of the configuration management problem for hosts:

(1) implementing host management using the model-driven (network) management tools;

(2) establishing the relative security of traditional methods and the above proposal for model-driven host management.

It is shown that the model-driven approach is feasible, and the security of the model-driven approach is significantly higher than that of existing approaches.

Acknowledgments

Many have supported me in the completion of this work. I am deeply in their debt as their significant support helped me synthesize this thesis into a coherent form. I want to use this page to express my most profound thanks to all of them.

First and foremost, I find no words to express my sincere thanks to my faculty guide and supervisor Dr. J. William Atwood. Without his mentorship, continuous support and assistance, this project wouldn't have been successful. His guidance throughout this process introduced me to the underlying concepts that establish the foundation of this work. I feel proud to express his rich knowledge of literature in the context of research.

A huge thanks to my parents for their supreme confidence in me. This isn't just my work. Though they live far, their continuous moral support and belief in me make this a joint project. It would have been impracticable to accomplish this work without their motivation and financial support to facilitate my higher studies. I am likewise grateful to my dear sister for always being there when I need humor into my life.

I also want to extend the gratitude to my family, friends, and roommates who made this journey a delightful experience.

Finally, an utmost thanks to that supreme force which infused me with self-motivation to complete this work.

Contents

List of Figures

List of Tables

List of Abbreviations and Acronyms

AAA	Authentication, Authorization and Accounting
ACL	Access Control List
API	Application Programming Interface
CDB	Configuration Database
CIA	Confidentiality, Integrity and Availability
CLI	Command Line Interface
CPU	Central Processing Unit
DDoS	Distributed Denial of Service
DFD	Data Flow Diagram
DSL	Domain Specific Language
GUI	Graphical User Interface
JSON	JavaScript Object Notation
IAB	Internet Architecture Board
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
ISO	International Organization for Standardization
IP	Internet Protocol
IT	Information Technology
KVM	Kernel-based Virtual Machine
LAN	Local Area Network
MIB	Management Information Base
MitM	Man in the Middle

NACM	NETCONF Access Control Model
NCS	Network Control System
NED	Network Element Driver
NFV	Network Functions Virtualization
NETCONF	Network Configuration Protocol
NIST	National Institute of Standards and Technology
NSO	Network Services Orchestrator
OS	Operating System
OSI	Open Systems Interconnection
RAM	Random Access Memory
REST	Representational State Transfer
RFC	Request For Comments
ROM	Read-Only Memory
RPC	Remote Procedure Call
SDN	Software Defined Network
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VLAN	Virtual Local Area Network
VM	Virtual Machine
VPN	Virtual Private Network
XML	eXtensible Markup Language
YAML	Yet Another Markup Language
YANG	Yet Another Next Generation

Chapter 1

Introduction

Computer networks are the backbone of an enterprise's Information Technology (IT) infrastructure. The size of these networks varies from a small number of devices residing at a room in the head-office to hundreds and thousands of devices spread across the entire globe. These networks are dynamic and have different operational needs and network management requirements. Network management is simple when the number of devices is small. However, it becomes increasingly complex as the device count goes up.

A survey conducted by Sherry, Hasan, Scott, Krishnamurthy, Ratnasamy, and Sekar (2012) on 57 enterprise networks of small to very large sizes and ranging from approximately a few hundred to 100,000 devices found that these networks comprised of a broad spectrum of a heterogeneous set of devices called as middleboxes. It was also found that even small networks incur substantial management and hardware expenses with the management team personnel requirement going up to hundreds in larger enterprises. Furthermore, a large number of enterprise network administrators reported misconfiguration as the primary cause of network failures (Sherry et al., 2012).

The survey shows the trends that due to the broad category of network devices and the dynamic nature of networks, network complexity increases substantially. Thus, an efficient and systematic configuration and management of the network continues to be a challenging task for network administrators.

Configuration and management of today's IT infrastructure can be predominantly classified into two categories; network management and host management. Although there has been much study and effort invested in both fields individually, however so far only little study has been done that focuses on the management of host devices using the network configuration protocol NETCONF and its data modeling language YANG (Yet Another Next Generation).

This study contributes to the literature focusing on future work regarding network management based on the requirements put forward by the network engineers in a workshop held by the Internet Architecture Board (IAB) in 2002. The requirements are stated in the Internet Engineering Task Force's (IETF) informational document RFC3535 and are still very relevant today. A key requirement put forward was that transactions are an essential mechanism and it could significantly simplify the process of configuration and management across a large number of devices (Schoenwaelder, 2003).

Furthermore, while investigating the problems and challenges faced by network engineers of large enterprises, it was found that a transactional-oriented network management protocol such as NETCONF could potentially address and solve these problems. On the contrary, other host configuration management tools lack transactions and employ scripts with the configurations laid out in sequential order. Hence, this study is motivated by the apparent lack of research and discussion on the real-world use of NETCONF and YANG for the management of host systems and attempts to examine the feasibility and security that would result from the use.

The security of a system is mainly affected by two components; internal factors and external factors. For an enterprise, internal security is a major concern. The insiders or the employees have access to sensitive data and can intentionally or unintentionally leak the information. Inadvertently, a security loophole could also be exposed as a result of a misconfiguration by an employee. On the contrary, an outsider intentionally tries to break the outer defenses first before they can gain access to the internal network. It is, therefore, imperative to do a comparative in-depth threat analysis to identify the potential threats that could arise by using NETCONF and YANG for host management.

In brief, the thesis explores the NETCONF-YANG based configuration and management of Linux-based host systems using a Network Service Orchestrator (NSO) by employing the Open Systems Interconnection's (OSI) network management model. The International Organization for Standardization (ISO) has standardized the network management model called FCAPS, an acronym for Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management (ISO, 1989). The focus of this study is primarily on configuration management and security management aspects.

Structure and organization of the thesis:

- **Chapter 2, Background Study,** covers the history, the background, and the literature of the several concepts that establish the foundation for a reader to understand the terminologies used in this thesis. The respective subsections cover only those protocols, tools, and concepts that are relevant for this study.
- **Chapter 3, Problem Statement,** defines and describes the network configuration and management problems addressed in this thesis.
- **Chapter 4, Proposed Solution,** describes the proposed solution that addresses the problem identified in the problem statement chapter.
- **Chapter 5, Threat Models,** illustrates the threat models and covers the security analysis related to the two different configuration management approaches for host management discussed in this thesis.
- **Chapter 6, Conclusion and Future Work,** summarizes the thesis and introduces possible opportunities for future research.
- **Appendix,** comprises the lab arrangement and includes the device configurations and the steps required to set up the tools.

Chapter 2

Background Study

Networks vary in size, ranging from small home networks to enterprise Intranet (LAN) to wide area networks (WAN). Smaller networks are easily manageable and can often be configured manually. However, as the network grows in size, it is self-evident that a network management solution is required. Network management is a process that can be modeled using the OSI/ISO FCAPS framework. The framework can be split into five discrete areas, viz. Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management (ISO, 1989). This chapter emphasizes on the configuration management and security management aspects of network management.

Configuration management is the management of change in the state of a network-element using a tool that enforces and ensures the change in the state. A network-element can be any network device or host in a network management framework that performs a specific functionality.

Security management is making sure who can make changes and view configuration information of the network-elements. There are various dimensions to characterize the security of a system, the most important being Confidentiality, Integrity, Availability (also knows as CIA triad (Perrin, 2008)), Authentication, Authorization, and Non-repudiation.

2.1 Network Management

In the late 1980s, the Internet was developing fast. With the growing number of network devices, such as routers, switches, and servers, it became increasingly challenging to manage all these devices and make sure they were performing optimally. The number of vendors was growing larger, and each was having its own set of management mechanisms. A network administrator was expected to learn all the proprietary management systems. This lead to severe management problems because of a lack of an accepted standard for the management of the network devices. It was at this point that it became apparent to the Internet community that a standard management protocol had to be developed.

In October 1987, an experimental management approach appeared called high-level entity management system (HEMS) and high-level entity management protocol (HEMP), stated in RFC1021 and RFC1022 respectively (Partridge & Trewitt, 1987b) (Partridge & Trewitt, 1987a). After a few weeks, another protocol was proposed called simple gateway monitoring protocol (SGMP), stated in RFC1028 (Davin, Case, Fedor, & Schoffstall, 1987). Work was also done on a third proposal called common management information services and protocol over TCP/IP (CMOT), which appeared in April 1989 and is stated in RFC1095 (Warrier & Besaw, 1989). As reported in RFC1052, the IAB expressed its gratitude for the efforts of the three working groups and directed IETF to coordinate on the on-going developments. It was later decided to drop the development of HEMS/HEMP on favor of the CMOT and SGMP group. It was also concluded that the Internet community should start to adopt simple network management protocol (SNMP, successor of SGMP) as the common protocol for network management (Cerf, 1989).

Furthermore, the OSI/ISO splits network management into five distinct areas. The configuration management part of the model for network-elements can be performed using several methods such as configuration files, command line interface (CLI), web-based graphical user interface (Web UI), simple network management protocol (SNMP), and network configuration protocol (NETCONF). Configuration files usually employ a domain specific language (DSL), which consists of configuration data in various formats, such as

Ruby, JavaScript object notation (JSON), INI files, or custom syntaxes. The subsequent sections explore the characteristics of CLI, SNMP, and NETCONF protocol.

2.1.1 CLI for Network Devices

Most network devices use a CLI as the default network management interface. It is a text-based interface that is used for reading and writing the configurations to and from the device. The interface is usually implemented using a command line shell that converts input commands into device specific functions. User entered input commands are transferred to a configuration file and then reloaded to the device if required. Configurations are written using the device's internal logic or in-built application programming interface (API).

A CLI can be accessed using a console, Telnet, or a secure shell (SSH) connection. A console is a physical port available on a network device that is used to connect directly to the device as essentially there is no display on a network device. Whereas, Telnet and SSH are the protocols that are used connect remotely to a device. A CLI connection is established using a terminal emulator application such as PuTTy and SecureCRT. Figure 2.1 shows a CLI session using a terminal emulator on a Cisco router.

```
CSR1000v#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet1       192.168.122.41  YES DHCP   up                    up
GigabitEthernet2       unassigned      YES unset  administratively down down
GigabitEthernet3       unassigned      YES unset  administratively down down
```

Figure 2.1: CLI session on a Cisco router

2.1.2 SNMP

Simple network management protocol (SNMP) was introduced in 1988 and was accepted as a full Internet standard in May 1990. The protocol allows a network administrator to configure SNMP-based devices and enables the monitoring of the device state. For example, it can be used to restart a router or to check the speed of an interface on a switch. It not only supports routers and switches, but can also be used to manage and monitor any

device that support SNMP operations, e.g., Unix-based systems, Windows-based systems, printers, firewalls, and more. All messages are transported via user datagram protocol (UDP) over commonly used port 161 and 162. The protocol is simple, easy to implement, and widely adopted. Despite that, it has many shortcomings because of which consistent efforts have been made for the continuous development and support of the protocol.

SNMPv1 was the initial release of the protocol, defined and published by the IETF in RFC1157 (Case, Fedor, Schoffstall, & Davin, 1990). The security of the protocol is based on community strings, which is merely a plain-text string that allows any SNMP based applications to gain access to the device's management information by just entering the string. The security of the messages sent through the protocol thus depends on the security of the internal network of the enterprise. Due to the shortcomings such as weak security and relatively poor performance when a large amount of management data needs to be retrieved, two work groups were established, one to focus on security, and the other to focus on the performance (this group was called SMP). These two groups later merged and formed SNMPv2.

SNMPv2, also known as SNMPv2c, is defined in RFC 1901, 1905, 1906 (Case, McCloghrie, Rose, & Waldbusser, 1995) (Case, McCloghrie, Rose, & Waldbusser, 1996) (Case, McCloghrie, & Rose, 1996). This version has better management features but borrows the security from SNMPv1 in which community strings are in plain-text and pre-shared with the communicating parties. An eavesdropper can easily capture community strings because all the data is unencrypted. The failure of the working group to deliver the security as per the agreement slowed down the overall acceptance of SNMPv2. SNMPv3 later replaced this protocol.

SNMPv3 is the latest version of the SNMP protocol, defined in RFC 2571 to 2575. The main reason for the development of this version was to add cryptographic security and enhanced support for remote configurations (Stallings, 1998). It added implementation support for following three security levels:

- **NoAuthNoPriv:** Communication without authentication and without privacy.

- **AuthNoPriv:** Communication with authentication but without privacy. Authentication is based on either HMAC_MD5 or HMAC_SHA algorithm.

- **AuthPriv:** Communication with authentication and with privacy. Authentication is based on either HMAC_MD5 or HMAC_SHA algorithm. Privacy is based on CBC_DES algorithm.

SNMP works in a client-server mode. A client, also know as SNMP manager, has a network management station (NMS) application installed and running and is used to manage the SNMP managed devices. A server, also known as SNMP agent, is a managed device having a management agent application installed and running. SNMPv3 re-named the SNMP agents as client entities.

When an SNMP server encounters an issue, it sends a message to the client known as SNMP trap. Trap messages are special code with a unique ID called object identifier data (OID). The management capabilities of SNMP are represented in a file called management information base (MIB). MIBs are used to expose an agent's management data to the manager. The translation of an OID is actually stored in MIBs. SNMP manager understands the alerts and sends the configuration data by translating the MIBs exposed by the agent. However, SNMP is mostly used for alerts and not for configuration management because of a lack of standardized MIBs (Schoenwaelder, 2003).

2.1.3 NETCONF

In June 2002, the Internet Architecture Board (IAB) held a workshop on network management. The notes from the workshop are documented in RFC3535 (Schoenwaelder, 2003). This RFC states that:

> The goal of the workshop was to continue the important dialog started between network operators and protocol developers, and to guide the IETFs focus on future work regarding network management.

Based on the requirements put forward by network operators, it was concluded that

simple network management protocol (SNMP) is not adequate for configuration management and a newer protocol was required that would address the issues raised in the workshop. Complying to the stated requirements, the network configuration protocol (NETCONF) was published in 2006 in RFC4741 (Enns, 2006). The protocol has gained acceptance since its launch. The original RFC4741 was updated and replaced with RFC6241 in 2011 (Enns, Bjorklund, Schoenwaelder, & Bierman, 2011).

NETCONF, as it stands for, network configuration protocol, is an IETF standardized network configuration and management protocol. It provides a method to add, modify, and delete the configuration of network devices. It is a client-server based protocol. A client is typically a NETCONF manager, and a server is a device that supports NETCONF operations.

It is an Extensible Markup Language (XML) Remote Procedure Call (RPC) style protocol. XML is used to transport data while RPC is a request from one computer program to another program. All the content transported through RPC requests and responses is sent through XML, encoded in UTF-8. The communicating parties use rpc and rpc-reply elements for requests and responses. Each rpc and rpc-reply element has a message ID number and a namespace (e.g., urn:ietf:params:xml:ns:netconf:base:1.0).

A NETCONF connection provides confidentiality, integrity, and authentication. Messages sent through NETCONF are connection oriented. Connections are encrypted, and messages are transported over the SSH protocol. The default transmission control protocol (TCP) port assigned by the Internet Assigned Numbers Authority (IANA) for NETCONF over SSH is 830. Some RFCs define NETCONF over transport layer security (TLS), simple object access protocol (SOAP), and blocks extensible exchange protocol (BEEP), but they are infrequently used.

It is a transactional protocol. When a configuration change requires updates across multiple devices, changes happen all-or-nothing and all-at-once. That means, changes all succeed or all fail. Transactions are essential to avoid unknown, inconsistent, in-determinant configuration state of the devices. For example, a virtual private network (VPN) update may require a change in many devices. A change successful only on a few devices may

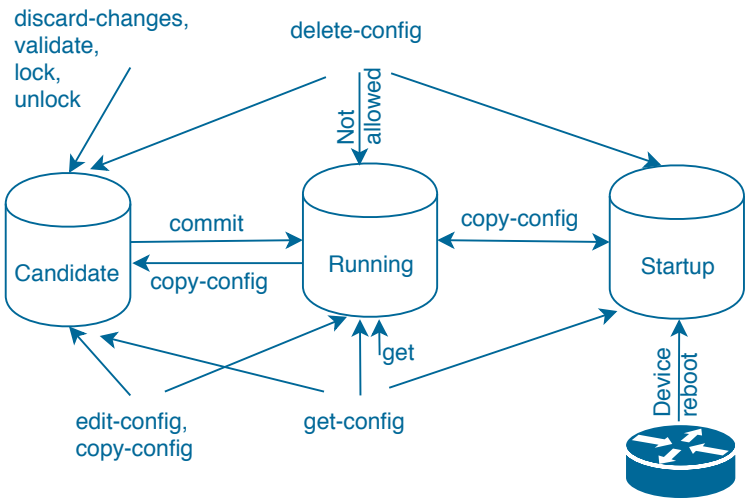

Figure 2.2: NETCONF datastores

lead to unexpected behavior in the network. A NETCONF operation applies configuration locks on all target devices. Locks are released once all the configuration changes are successfully implemented or discarded.

A typical NETCONF system includes one-or-more type of datastores. Data stored in the database can be conceptually categorized into configuration data and state data. Configuration data is the writeable device configuration, and state data is the device state information. Figure 2.2 represents these datastores with the base operations that can be performed on them. Candidate datastore holds the copy of data that has not yet been uploaded to the device. All the configurations are first pushed into candidate datastore, where they are validated and then activated by performing a commit onto running datastore. Running datastore holds the current active configuration on a device. Startup datastore represents the configuration used by the device at startup. Depending on the capabilities, candidate and startup may or may not be supported by the NETCONF manager, whereas, running is a mandatory datastore.

When a NETCONF manager connects to a NETCONF device, they exchange "hello" messages as shown in Figure 2.3. The "hello" message indicates the capabilities supported by each side. These capabilities include information such as the NETCONF version that

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <capabilities>
      <capability>urn:ietf:params:netconf:base:1.0</capability>
   </capabilities>
</hello>
]]>]]>
```

Figure 2.3: Example of a NETCONF "hello" operation

the client and server supports and the YANG data models advertised by the device.

NETCONF exists in version 1.0 and version 1.1. These two versions use different framing mechanisms. In version 1.0, a special character sequence]]>]]> is added at the end of each XML message to indicate a break between XML packets. Whereas, version 1.1 has a run-length encoding. Also, there are a number of tags and attributes that can be used within a specific operation. A list of all these tags and attributes can be found in RFC6241 (Enns et al., 2011).

NETCONF Base Operations:

- **get:** This operation is used to retrieve both configuration and state data of the device. If the operation succeeds, NETCONF server sends an rpc-reply message. If it fails, an rpc-error message is included in the rpc-reply.
- **get-config:** This operation is used to retrieve configuration data only. Additional tags such as "filter" can also be used within get-config for sub-tree filtering to retrieve information based on a specific data model of that device.
- **edit-config:** This operation is used to make changes to the configuration of a device. To perform this operation, the target datastore needs to be specified in the rpc request.
- **copy-config:** This operation is used to copy or replace configuration data between different datastores, and to specified URLS. It uses "source" and "target" tags to specify the source and target datastore.
- **delete-config:** This operation is used to remove configurations from a specified

datastore.

- **lock:** This operation is used to lock the datastores. It is used to ensure the consistency of the device configuration so that another client would not be able to make any changes while a client session is already active.
- **unlock:** This operation is used to release an active "lock". Requirement for performing this operation is that the requested "lock" should be currently active, and should be issued by the same session as "unlock".
- **close-session:** This operation is used to gracefully close a session. All the acquired "locks" and resources are released before the session is terminated by the server.
- **kill-session:** This operation is used to forcefully close a session using its session ID.

A couple of new operations are added and described in RFC6241, e.g., "commit" operation to copy the device configuration from candidate datastore to running datastore, "discard-changes" operation to roll-back candidate datastore configuration to the active running configuration, "cancel-commit" operation to cancel an active confirm commit operation, "validate" operation to validate the configurations stored in candidate datastore. These operations can only be used when candidate datastore is enabled (Enns et al., 2011).

2.1.4 YANG

Yet Another Next Generation (YANG) is a data modeling language designed for use with the network configuration protocol (NETCONF). In simple terms, it is the grammar and vocabulary of each device that is managed using the NETCONF protocol. It was proposed in 2010 in RFC6020 (Bjorklund, 2010) and updated in 2016 to RFC7950 (Bjorklund, 2016).

When NETCONF was launched in 2006, there was no standardized data modeling language available to define the data carried over the protocol. Standardization of YANG was established on the basis of the work done by an IETF group called NETMOD. The IETF released version 1.0 of YANG in 2010, and subsequently released version 1.1 in 2016.

YANG is a data modeling language used to define the syntax and semantics of the data sent over the NETCONF protocol. In other terms, it is used to model the configuration and state data in NETCONF as the data in YANG maps one-to-one to XML. To understand how it is used in a network management system, consider a scenario where a management tool has to connect to different vendor equipments. Each vendor has its own proprietary command-line interface (CLI) and application programming interface (APIs). Making a change, like adding a device or adding a new service would require a change at multiples levels as the new device might talk to other devices, or the corresponding new service may incorporate multiple devices. A YANG data modeling was developed to make this approach more efficient as it permits vendors to use a common language to define the data models for their devices. So, next time a new device is added, it might have different data models as compared to devices from other vendors, but YANG; the data models governing language is the same.

YANG data is stored in files called modules. A module for a device can be published by the IETF, the OpenConfig, or the device Vendors. An example YANG module (Figure 2.4) (Bjorklund, 2016) and the breakdown of important module statements with their meaning is listed below:

- **module:** Each module starts with a module statement and the module name. A module name should be same as the name of the file.
- **namespace:** It is a unique identifier used to distinguish a module from other YANG modules. A typical NETCONF based system can have multiple modules from various devices and different data model publishers.
- **prefix:** This statement is used to assign a prefix, which is used to refer to a module namespace.
- **import:** This statement is used to refer to data from other modules, and an include statement refers to data from submodules into a main module.

```
module example {
        namespace "...";
        prefix "...";

        import ... { ... }
        include "...";

        organization "...";
        contact "...";
        description
            "...";

        revision ... {
            description "...";
        }

        typedef ... {
        ...
        }

        container ... {
            leaf ... {
                ...
            }

            leaf-list ... {
                ...
            }

            container ... {
                leaf ... {
                    ...
                }

                list ... {
                    key ...
                    leaf ... {
                        ...
                    }
                    leaf ... {
                        ...
                    }
                    leaf ... {
                        ...
                    }
                }
            }
        }
    }
```

Figure 2.4: Representation of an example YANG module

- **organization, contact, description:** These statements describe the information specific to the party that owns the module.
- **revision:** This statement describes the history related to the changes of the module.
- **typedef:** A data type may be a base type or derived type. All the base types are listed in the RFC6020. A derived data type is a user specified data type structure. typedef statement is used to define a derived type named my-base-str-type. It has a base type as string, and can take a string of length 1 to 255. A leaf named class uses this derived data type.
- **container:** This statement is used for organizing a tree. It groups leaf, leaf-list, and list. It does not hold any data and does not have a data type. Containers can be

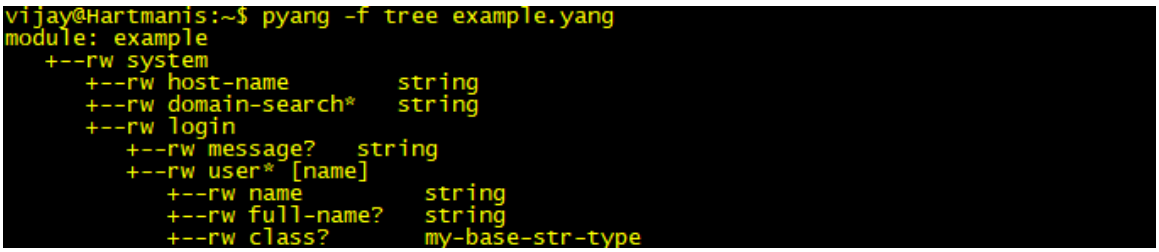

Figure 2.5: pyang tree format translation of a YANG module

nested. In Figure 2.5, a login container is nested inside a system container.

- **leaf:** A leaf is like a variable that holds a single data value of a specified data type. In Figure 2.5, leaf host-name has a data type assigned as string. Additional attributes can be defined within a leaf, e.g., "mandatory true;" defines that the configuration must be present, "config true;" states that it is the configuration data whereas, "config false;" means that it is the state data.

- **leaf-list:** A leaf-list is like an array that can hold values of a specified data type. Unlike leaf, it can hold multiple data values.

- **list:** A list is like a table of variables. It has a "key" statement, which defines the entries that are key values.

A network programmer uses the YANG modules published by either the vendor, the IETF, or the OpenConfig based models to describe the configuration hierarchy for a device or service in the network. These YANG models are then encoded in XML and transferred to the server using NETCONF or any other API (such as RESTCONF, or gRPC). Figure 2.5 shows a tree format translation for a YANG module. A YANG module can also be translated into an XML-based syntax called YIN using the yin filter in the "pyang" python-based YANG validator tool. YIN format output simplifies the task of a developer to validate, apply data filters and perform other operations on the YANG modules.

2.2 Host Management

Host management, interchangeably known as configuration management, is the management of a change in the state of a system throughout its lifecycle. Depending on the operating system (OS), a system can be classified as Linux-based or Windows-based. Configuration management started as a process within the US military in the late 1960s to manage the lifecycle of the various changes in the system. The process has evolved since then and has been adopted for building a system or software quickly and efficiently.

Until recently, configuring IT infrastructure was a manual task. As IT infrastructure has multiplied, technologies to manage these infrastructures have evolved as well. The growing understanding of IT and infrastructure as a code has influenced the work on configuration management processes. The best practices for configuration management can be found under the "Information Technology Infrastructure Library (ITIL) Asset and Configuration Management" Service Transition processes (BMC, n.d.).

Configuration management has many applications, e.g., infrastructure migration, mass deployment, configuration change failure rollbacks, and more. Consider a university IT infrastructure, which typically has hundreds and thousands of desktops running on Linux and Windows-based systems. A system administrator needs to deploy a security patch on these systems. If done manually, he or she has to log in into each machine and then install the patch. This process is time-consuming and prone to human errors. A configuration management tool can do the mass deployment by simply entering a command on a master system that pushes the patch installation request to all the connected client machines. This automated process not only saves time but is also less susceptible to human errors.

2.2.1 CLI for Host Systems

Despite advances in host management technologies, there are still many network-elements in a network that offer CLI as the default configuration management interface. Modern configuration management protocols provide connectivity through a secure remote connection protocol such as SSH over TCP or through other transport mechanisms. On the

```
vijay@Hartmanis:~$ ifconfig eno1
eno1      Link encap:Ethernet  HWaddr **:**:**:**:**:**
          inet addr:132.205.9.62  Bcast:132.205.9.63  Mask:255.255.255.252
          inet6 addr: fe80::2e27:d7ff:fe46:c8b9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1715924 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1150153 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1454882551 (1.4 GB)  TX bytes:126440310 (126.4 MB)
          Interrupt:20 Memory:fb000000-fb020000
```

Figure 2.6: Network adapter information on a Linux-based system

```
C:\Users\Vijay Satti>netsh interface ip show addresses "Wi-fi"
Configuration for interface "Wi-Fi"
    DHCP enabled:                         Yes
    IP Address:                           172.30.114.8
    Subnet Prefix:                        172.30.0.0/17 (mask 255.255.128.0)
    Default Gateway:                      172.30.0.1
    Gateway Metric:                       0
    InterfaceMetric:                      35
```

Figure 2.7: Network adapter information on a Windows-based system

other hand, many vendors still offer CLI as the only way to connect to a network-element using an unsecured unreliable connection.

Figure 2.6 and Figure 2.7 shows the output of command to retrieve the IP address of a network interface controller (NIC) on a Linux-based and a Windows-based system respectively. Even though the purpose is same, the syntax of the command and the output format is different for different systems.

Configuring a heterogeneous network through different CLIs could be an inefficient task, but CLI is still handy in scenarios when simple network management protocol (SNMP) or network configuration protocol (NETCONF) connectivity is unavailable over the network, or when the network speed is low. For example, a denial-of-service (DoS) attack may hinder the remote connectivity to a device. In situations like this, direct access to device CLI through the serial link could admittedly be helpful (Jones, 2004).

2.2.2 Configuration Management Tools

There are many open-source and commercial configuration management tools available in the market. Choosing the correct tool for an enterprise involves many factors (Delaet, Joosen, & Van Brabant, 2010). Below is a brief overview of the most popular configuration management tools:

Puppet is an open-source tool, initially released in 2005, and has a client-server architecture. Puppet server runs on one or more servers, and puppet clients runs as an agent on each client machine that an administrator wants to manage. The tool can be used to manage a variety of Linux, Unix, and Windows-based systems. The configurations are defined in a file called Puppet manifests. Enterprise version of Puppet provides GUI, CLI, and API support for client management. The tool is written in C++ and Clojure. All the system related data are written in a custom declarative language or Ruby domain specific language (DSL).

Chef is an open-source tool, initially released in January 2009, and has a client-server architecture. Chef server runs on the master machine, and chef clients run as agents on each client machine. The tool can be used to manage Linux, Unix. and Windows-based systems. It uses SSH to connect and certificates to authenticate communication with the clients. All the configurations are stored in a machine called workstation, usually a master server, where it is tested and then sent to the clients. These configurations are defined in a file called recipes (grouped as cookbooks). Once the cookbooks are uploaded to the server, an administrator can access Web UI, CLI, or API to send the configurations to the clients. The tool itself is written in Ruby and Erlang and uses Ruby DSL for writing the recipes.

Salt (also known as Saltstack) is an open-source tool, initially released in March 2011, and has a client-server architecture. The server machine is called as master, and clients are called minions. Master runs remote execution commands to send the configuration data to minions. It is a CLI based tool written in Python language and can be used to manage Linux/Unix and Windows-based systems. Saltstack is fast and reliable as it uses Zeromq library for message transport. Client-related configurations are executed through Salt files, which are defined in Yet Another Markup Language (YAML).

Ansible is an open-source configuration management and orchestration tool, initially released in February 2012, and later acquired by Red Hat in 2015. Unlike Puppet, Chef and Salt, it has an agent-less architecture, which means client machines need not have an agent or a daemon installed to connect to the server. The tool can be used to manage Linux/Unix

and Windows-based systems. Ansible server uses a secure SSH or remote PowerShell connection to push configurations to the client machines. All the configurations are defined in files called Playbooks. Besides, it also comes with built-in modules to perform various operations. Modules can be directly executed on client machines or can be used in playbooks. The tool is primarily accessed using CLI. The enterprise version comes with Web UI solution called Ansible Tower. The tool is built on Python language and uses YAML for writing the playbooks. Custom modules can be written in a scripting language such as Python, Ruby, Perl, and Bash (Ansible, n.d.).

2.3 Network Service Orchestration

Network service orchestration (NSO) in computer networking is defined as the provisioning of network services by automating the network-elements for end-to-end service lifecycle management. An orchestrator tool keeps a high level track of the service workflows, from allocating resources, to scaling resource, to deploying resources, thus simplifying the process of end-to-end network service creation. The network device that is automated can typically be a router, switch, gateway, firewall, and more, and service typically includes configuring virtual private network (VPN), virtual LAN (VLANs), subnets, routing, trunks, quality of service (QoS), etc. A network device can be physical or virtual and can span across LAN, WAN or a remote data center.

While configuration management tools (such as Puppet, Chef, Saltstack, Ansible, discussed in Subsection 2.2.2) are ideally suited for host system configuration such as Linux, Unix, and Windows-based systems, network service orchestration tools are primarily used for network configuration and end-to-end service lifecycle management of network devices. Some configuration management tools also support network configuration management, however comparatively it is not as efficient. Configuration management tools require building a network from ground zero which includes building manifests, cookbooks, or playbooks whereas, NSO uses the concept of abstraction for representing the device configurations.

Consider a scenario in which a network engineer wants to create a VLAN in a network. Using the traditional approach would require to log into each device and to enter the commands using its command line interface (CLI) or application programming interface (API) to configure the VLAN. NSO, on the other hand, can automatically discover the current configurations on the devices across the entire network and present the network engineer the network topology and all dependencies for creating the VLAN. He or she can then instruct NSO to apply the configurations on multiple devices without resorting to manual login into each device or without modifying scripts that call the device APIs.

In a real sense, a network service orchestration tool is a vendor-neutral solution for device configuration management. It uses device abstractions to map a vendor-specific CLI or API through its YANG data models, thus providing a model-driven network-wide interface for all network devices. It lets a network engineer create new and modify existing services using standardized models. The services are delivered faster and more efficiently without any disruption, ensuring on-time deliveries and contractual service level agreements (SLAs). The implementation of NSO varies from industry-to-industry on a case-per-case basis. Many vendor solutions (e.g., Cisco NSO, evolution of the Tail-f Network Control System (NCS), Juniper's Contrail Service Orchestration and Anuta Networks NCX) provide NSO integration with the existing networks.

2.4 Security Context

2.4.1 Justification

Security depends on both internal and external factors. To secure the network both internally and externally, proper secure design methodologies include multiple layers of defenses. Each layer implements various security policies and controls designed in a way such that authorized users can access the network, while a user trying to perform any malicious activity is blocked. For example, in the TCP/IP protocol suite, security can be applied at the Internet or IP layer using IPSec, or security can be applied at the Transport or TCP/UDP layer using SSL/TLS, or security can be applied at the Application layer

using SSH. Since NETCONF messages are transported through SSH, the protocol is briefly discussed in the next subsection.

2.4.2 SSH Protocol

Secure shell (SSH) is a protocol designed for secure connection to a remote computer and secure remote command execution. It was initially released in 1995 by a Finnish researcher Tatu Ylönen. The protocol works on a client-server architecture and is implemented at the application layer of the TCP/IP protocol stack. A client is typically an application that wants to connect to the SSH server. Default TCP and UDP port assigned by the IANA for SSH operations is 22. The protocol establishes a secure cryptographic tunnel between the communicating parties, authenticating each side to other using a public-private keypair, allowing them to encrypt-decrypt the requests and replies back and forth. It does so by breaking the data into a series of packets. The packet includes the packet length, padding amount, payload and the padding. The packet also has a message authentication code (MAC) for integrity checks. The whole packet excluding the packet length is encrypted and sent forward to the other end. The receiving party receives the packet, decrypts it and extracts the data (Ylonen & Lonvick, 2006) (Ylonen & Lonvick, 2005).

2.4.3 Security Analysis Techniques

While the protocols above ensure the security of data in transit, there are several threat modeling techniques developed for secure design analysis and testing of a system. A few of them are:

- Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege (STRIDE) by Microsoft (Kohnfelder & Garg, 1999).
- Damage, Reliability, Exploitability, Affected Users, and Discoverability (DREAD) by Microsoft (Shostack, 2008).
- Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) by Carnegie Mellon Software Engineering Institute (Caralli, Stevens, Young, & Wilson, 2007).

- Common Attack Pattern Enumeration and Classification (CAPEC) by MITRE (Barnum, 2008)
- Threat Agent Risk Assessment (TARA) (Wynn et al., 2011) and Threat Agent Library (TAL) by Intel (Casey, 2007).
- Guide to Data-Centric System Threat Modeling (Draft Special Publication 800-154) by National Institute of Standards and Technology (NIST) (Murugiah & Scarfone, 2016).

Furthermore, there are industry-wide best practices and frameworks for cyber risk management and assessment. A few of them are:

- NIST Framework for Improving Critical Infrastructure Cybersecurity (Barrett, 2018).
- Control Objectives for Information and Related Technologies (COBIT) (COBIT, 2018)
- Center for Internet Security (CIS) Controls and Benchmarks (Controls, n.d.) (CIS, n.d.).
- MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) Framework (Strom et al., 2017).

The white paper presented by Shevchenko, Chick, O'Riordan, Scanlon, and Woody (2018) lists the summary of some of these methods from a variety of sources (Shevchenko, Chick, O'Riordan, Scanlon, & Woody, 2018). Despite having various such techniques, STRIDE is chosen as the preferred threat modeling methodology for this study.

The term STRIDE is an acronym for six different security threats namely, Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. It was developed in 1999 and adopted in 2002 by Microsoft as an internal drive to produce secure software applications. Although STRIDE's application is not limited to just software, it was subsequently improved and used to model enterprise architectures, networks, systems, or any particular asset with an intent to be protected against security threats (Bodeau, McCollum, & Fox, 2018).

2.4.4 Security Analysis of Network and Host Management

The white paper by Shevchenko et al. (2018) (Shevchenko et al., 2018) shows that a significant amount of work has been done in developing techniques that allow administrators to evaluate the security of a system. Additionally, the study conducted by Bellovin and Bush (2009) (Bellovin & Bush, 2009) explores the security aspect of configuration management of five different scenarios and outlines the challenges related to each of these. Their research concludes that configuration management is vital for organizational security. However, it is stated that there are many security-related challenges and further research work needs to be done. Despite having the availability of various techniques, the use of the existing techniques to do the security analysis of the configuration management tools used in this study is missing from the existing literature.

Chapter 3

Problem Statement

This chapter discusses the limitations related to the existing configuration and management solutions that lead to the motivation presented in Chapter 1. The problem statement itself has been classified into various segments and attempts to shed some light on many issues with the current configuration and management schemes.

3.1 CLI and API Limitations

Today's network infrastructure layer spans various domains consisting of network-elements from multiple vendors with vendor-specific legacy CLIs and APIs. CLIs and APIs are designed to access the native functionality of the device; however, there is no standardization on writing and implementing the same. Each vendor has its own proprietary CLI and set of APIs, which leads to the manual configurations using the custom build scripts thus making it difficult to automate the network. No automation means spreadsheet-based implementation, which subsequently leads to human errors and misconfiguration.

CLI also has the limitation of lack of transaction management. A configuration change may involve changes to multiple devices or multiple changes on the same device. These configuration changes cannot be implemented partially as this may leave the network to an undefined state. Therefore, in case of any failure, there should be a mechanism to roll back the changes and the network to its initial state. CLI fails to address this issue (Rizos,

2016).

CLIs are designed to work with humans, but not the APIs for programmatic access. Besides, a change in a device from a specific vendor in a network leads to change in API calls across the entire network. Thus, every time a vendor makes a change in network CLI or API, a network administrator has to re-write the scripts, which makes the network costly to maintain.

Using a CLI has its pros and cons. CLIs are made to be user-friendly and more accessible to operate. However, in a multi-vendor network, different types of network devices can have different proprietary CLIs, and each operates through a unique set of commands. A simple task requires a different set of commands across different CLIs, giving the same output, but in a different format. Network engineers and operators spend many hours in learning the syntax of the commands, thus spending a large chunk of time and money in the learning process.

A CLI is fundamentally a tool to promote a vendor's technology. In certain means, it is a barrier to the adoption of another vendor's equipment. By getting network engineers highly invested in their CLI with its certification programs, equipment providers and vendors try to entice the market into their ecosystem.

3.2 SNMP Limitations

SNMP was developed for both configuration management and monitoring of the devices. However, for several reasons, it is used more frequently for monitoring and alerts and not for configuring the changes — the main reason being the security and the stateless nature of the protocol. SNMPv1 is the most widely deployed version, but it is not secure. SNMPv2 development promised security but failed to achieve it. SNMPv3 was developed with better security; however, it was not adopted as much by the industry because of its complexity.

Security is not the only issue with SNMP. Because of its complexity, it also lacks standardized MIB modules, which makes it difficult for equipment vendors to design and

implement the MIBs in a timely manner.

Another issue with SNMP is that it provides an unreliable transport mechanism as it uses UDP for establishing the connection between the manager and the agents. UDP is a session-less protocol, which means that messages can be easily lost during the transport.

SNMP uses a transactional model, but because of its session-less nature, SNMP manager (or, NMS) may not know whether or not the devices have succeeded or failed. Since messages are sent over UDP, the NMS and the agents need an added measure to determine the sequence of interactions. As another solution, the IETF has defined RFC3430 for using SNMP over TCP. Although, the transactional model makes it further complicated to implement the MIBs (Schoenwaelder, 2003) (Schoenwaelder, 2002).

Furthermore, SNMP is not efficient in large systems. It provides reasonable performance when retrieving a small amount of data from a large set of devices. However, it becomes slow when retrieving a large amount of data even from a small set of devices (Schoenwaelder, 2003).

3.3 Host Management Limitations

Host management or configuration management tools are used for the automated management of large systems. Tools such as Puppet, Chef, Saltstack, and Ansible offer CLI and GUI as the prime interface to push configurations to the managed devices. These configurations are usually sent through a file called DSL. A DSL contains the device specifications in a syntax very similar to the device implementation language. A multi-domain multi-vendor network would require the configuration files for each type of device in their own syntaxes. In such scenarios, management of configuration files in a heterogeneous network becomes a burdensome task for a network administrator.

Additionally, the architecture of a host configuration management tool can be push or pull based. In pull-based architecture, managed device agents fetch the configurations from the manager. In push-based architecture, managers push the configuration to the managed devices. In either case, authentication needs to take place between the manager

and the agent. In push-based architecture, authentication information is written in plain-text in configuration files (as demonstrated in A.5). Sending login credentials to managed devices in plain-text through configuration files exposes a security risk (Delaet et al., 2010). Another such scenario is a configuration change that requires a push operation to change the username and password or keys on managed devices.

Configuration management tools also lack transactional support. Transactions are significant when a device configuration fails, and the state of all the devices need to be rolled back to the initial configuration state thus avoiding any misconfiguration in the network. Because configurations are sent through configuration files, there is no automated mechanism to roll back the state of the devices into its initial state.

3.4 Security Limitations

SNMP has security drawbacks and using host management tools itself does not provide any concrete security. While reviewing the host management tools for configuration management, it was found that the existing literature lacks the security analysis of configuration management tools. Therefore, a security analysis needs to be done and it needs to be determined how introducing NETCONF and YANG for the configuration management of the host systems affects the overall security posture of the network.

3.5 Demerits of Host Management using Traditional Methods

This section lists the demerits of the traditional host management methods. Subsequently, Chapter 4 will introduce the merits of host management using an advanced NETCONF and YANG based method.

- SNMP, CLI, and most of the configuration management tools lack transaction management. There is no in-built mechanism to roll back the network to its initial state in case a configuration failure occurs.

- CLIs and APIs are vendor-specific proprietary solutions. There is no standardization on writing and implementing the same.
- CLIs are designed to work with humans and are hard to automate.
- Initial versions of SNMP were not secure and were less advanced. Despite various improvements, SNMP is used more frequently for monitoring and alerts, and not for configuring the changes.
- SNMP is an IETF standardized protocol, but it lacks standardized MIB modules and has an unreliable transport mechanism.
- Using configuration management tools, configurations are usually sent to devices through DSL files, which are a collection of files and scripts. In a heterogeneous network, configuration management using DSL files becomes a burdensome task for a network administrator.

Chapter 4

Proposed Solution

This chapter outlines how introducing NETCONF and YANG for the configuration and management of host systems could address the problems discussed in Chapter 3. The proposed solution employs a model-driven approach for host management and offers a single point of management through a programmatic interface, thus reducing the complexity of the network, and providing network-wide transactions, reliability, and security.

4.1 NETCONF and YANG based Host Management

To address the challenges and limitations with the network configuration and management solutions, IETF has defined two key standards:

(1) NETCONF protocol API

(2) YANG data modeling language

The proposed solution includes a combination of a network configuration management solution called NSO (NSO, n.d.) and a data model driven management plane framework called ConfD (ConfD, n.d.). Combining NSO and ConfD provides a NETCONF and YANG based management framework for the host device configurations in the network. It also extends support for management interfaces such as CLI, SNMP, and legacy interfaces that are still widely used in configuring the network devices as well as host systems.

NSO at a Glance

NSO is a network configuration and management application that provides a central configuration and management point for multi-vendor device automation and orchestration of the entire network. It has support for various northbound and southbound network management protocols such as NETCONF, SNMP, CLI, and more (as shown in Figure 4.1). For a NETCONF operation, NSO can be loaded with the YANG data models of a device. These data models are made available by the IETF, the OpenfConfig, or the device vendor itself. Once NSO has the data models, it can be instructed to manipulate the device configuration using the NETCONF protocol.

ConfD at a Glance

ConfD is a data model driven management framework that is meant to be integrated with network devices that want to implement NETCONF protocol. It executes as a Unix daemon on the target network device and acts as a NETCONF agent for the NETCONF protocol. ConfD's LinuxCfg package interacts with the ConfD daemon and provides a set of managed network interfaces for the Linux OS. The network properties and functionalities that can be managed using LinuxCfg package depend on the data models described in the YANG modules specifications structured within the individual components of the package. ConfD also has support for additional northbound and southbound management protocols such as SNMP and CLI (as shown in Figure 4.1).

The Glue - NSO Network Element Driver (NED)

Network-wide transactions are initialized within NSO by a network engineer or a network management application through one of the northbound management protocols or interfaces. An initiated transaction spans all the way to a device connected to a southbound interface using a NED. To interact with a Linux managed object enabled with ConfD, a NED is built using the Pioneer package in NSO. A NED can be built using the YANG data models for a NETCONF device or using the MIBs for an SNMP device.

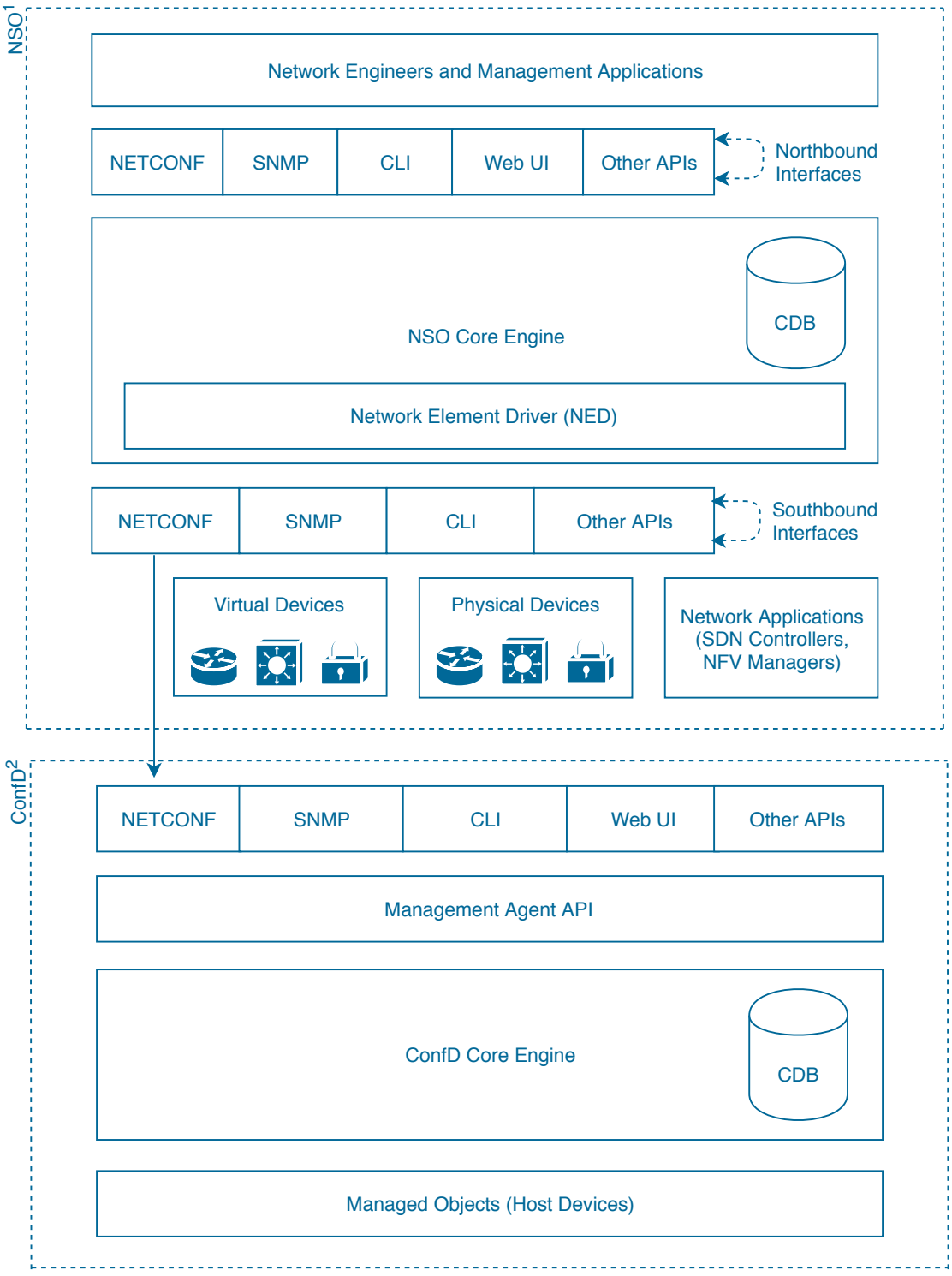

Figure 4.1: NETCONF and YANG based management of host systems

NSO[1] (NSO, n.d.), ConfD[2] (ConfD, n.d.)

4.2 Solution Capabilities

NETCONF and YANG based management of host systems provides many improvements as compared to other alternatives discussed in the problem statement chapter. The following subsections summarizes the superiorities of the proposed solution:

Transactions and Roll-backs

Host management tools (or, configuration management tools) do not have any primitives for transactions. NETCONF protocol has in-built primitives that support transaction-oriented operations. Transactions are very important when a configuration change needs to be implemented on multiple devices at once. Transaction-oriented property of NETCONF ensures that configuration change operation is successful on all devices, and if it fails for any reason, then the protocol instructs the management application to reverse the network to its initial state. Therefore, when a configuration change is pushed, all network devices are either transformed into a new state, or they are rolled-back to its initial state. This assures there are no half-done, broken or stray configurations in the network (Enns et al., 2011).

Transactions are performed by acquiring configuration locks on all target devices in the network. The lock is kept until configurations are applied on all devices and the validation is performed for the applied configurations. Validate operation checks for any syntax or semantics errors in the configurations before it is loaded into the device. Locks are released on all target device once the configuration change is completely implemented or completely discarded. All these operations are performed simultaneously in a transactional order on all target devices in the network (Enns et al., 2011).

In addition, if the device does not support roll-backs, NSO can calculate the difference between the current and the previous state of the device and can accordingly apply the configuration changes. When a device is on-boarded to NSO, NSO knows the capabilities supported by that device. Depending on the device capabilities, NSO can issue the

appropriate commands to the device using the configuration copy that is saved in its configuration database CDB at all times. CDB is shown in Figure 4.1.

Reliability

NETCONF-based management provides a persistent connection between the client and the server. All protocol messages are sent over TCP. TCP is a reliable protocol that checks if everything sent by the sender was received by the receiving end without any packet loss. That means acknowledgment of each and every configuration sent to the client is being reported to the server. In case of any errors, all the connection failures are also reported to the server. If a connection drops, the server can then automatically release all the locks acquired by the client. The locks are kept until it is explicitly released by the client or until the connection times out. This ensures efficient and reliable resource consumption (Enns et al., 2011).

Multi-vendor Interoperability and Extensibility

The YANG data modeling language provides a model-driven approach to network programmability and management. Using the YANG models in NSO, it becomes easy to inter-connect devices supported by different vendors in an enterprise. These devices need not necessarily support NETCONF as the management protocol in-built into them. It is because a network administrator can simultaneously manage a CLI, SNMP, and a NETCONF-based device using a single programmatic interface.

Besides, YANG provides a modular approach to add or remove any network function and controls using its data models. If at any time, a new network feature needs to be added to a device type, the network administrator need not re-write the whole CLI or API for the device. For example, if a network administrator needs to make particular changes in a host system to add functionalities such as IP address, DNS, NTP, etc., then a programmer can build the YANG module for these specific configurations without re-writing the data models for the entire host system. Thus, YANG modules are extensible and make it easy to add or remove functionalities needed for the device management in an ad-hoc fashion.

Performance and Scalability

Hedstrom et al. did a head-to-head quantitative comparison of NETCONF and SNMP. Their research found that NETCONF is also better than SNMP in terms of performance and scalability. SNMP is faster for a lower number of devices, whereas NETCONF provides better performance as the device count goes up. This clearly shows that NETCONF transactional property provides better performance in large networks. They also found that NETCONF has higher bandwidth utilization as compared to SNMP. However, the higher bandwidth requirement is because of the protocol overheads due to the added security measures, connection-oriented nature, and capability exchange features (Hedstrom, Watwe, & Sakthidharan, 2011).

4.3 Security Efficiencies

NETCONF-based management of host systems introduces a secure management scheme. The protocol possesses various security improvements as compared to legacy configuration management solutions such as SNMP and CLI. It enhances security in two ways. One, it provides a secure transport mechanism to send and receive sensitive information. Second, it provides an access control mechanism to restrict the use of sensitive information. Regardless, security also depends on a third factor, which is the security of application that stores or releases such sensitive information (Enns et al., 2011). The following subsections discuss each of these security considerations.

4.3.1 Secure Transport

Secure transport mechanism of NETCONF provides confidentiality, integrity, authentication, replay protection, and sequential data delivery for the sensitive information in transit between the NETCONF server (a manager) and the client (a device). Sensitive information can be the configuration data, state data or the data related to protocol messages.

Configuration data often contains usernames, passwords, keys, and network topology related information. Any malicious activity in the network can open doors for a man-in-the-middle (MitM) attack allowing an adversary to modify or inject data in the configuration files. NETCONF provides secure transport of the configuration data and secures the transmission against an adversarial attack.

NETCONF transport layer security can be implemented in many ways. NETCONF over SSH is the most widely used and mandatory implementation of the protocol. The protocol specifications for NETCONF over SSH are defined in IETF RFC6242 (Wasserman, 2011). A NETCONF connection over SSH is initiated by a client with the server over TCP on IANA assigned port 830. SSH transport layer protocol (Ylonen & Lonvick, 2005) mandates the authentication of both the client and the server before any configuration, or state data can be exchanged between both the parties, thus avoiding the exchange of messages with an incorrect identity. Once the connection is established, SSH then provides strong encryption of messages for data privacy and also supports integrity checks. This is harder to achieve using SNMP.

Another implementation of NETCONF is the use of the TLS protocol to secure the NETCONF message exchanges. Specifications for NETCONF over TLS are defined in IETF RFC7589 (Badra, Luchuk, & Schoenwaelder, 2015). A NETCONF connection over TLS is initiated by a client with the server over TCP on IANA assigned port 6513. The client starts the TLS handshake process with the server, which is used to authenticate each other's identity using certificates (Dierks & Rescorla, 2008) (Rescorla, 2018). Once the TLS handshake process is finished, the client can start sending and receiving encrypted XML documents with the server in the rpc and rpc-reply messages respectively.

Additionally, NETCONF protocol design further allows and extends the implementation of NETCONF over SOAP, defined in RFC4743 (Goddard, 2006), and NETCONF over BEEP, defined in RFC4744 (Lear & Crozier, 2006), thus opening the scope for further security enhancements in environments that support these protocol implementations.

4.3.2 Access Control Model

Since configuration and state data can be confidential and sensitive, therefore it should be accessible to the authorized users and systems only. Without a standard access control model, every NETCONF protocol user has the permissions of a user equivalent to the root user. NETCONF access control model (NACM) provides security by limiting the resources and data that a user or an adversary can reach or attack. A user can be part of a group, and a group is assigned access to the defined protocol operations. Once a user has authenticated itself to the server, the NETCONF transport layer transfers the username and the group associated with the client to the server. The server then enforces the access control on the client based on the username and the group (Bierman & Bjorklund, 2012) (Bierman & Bjorklund, 2018)

NACM limits NETCONF user's access using control rules such as permit and deny, and operations such as create, read, update, delete (CRUD), execute, and none (Bierman & Bjorklund, 2012). On the contrary, SNMPv1 and SNMPv2 have the concept of defining communities. These communities are defined in SNMP communities page which associates communities with access rights. Whereas, CLIs employ an extra login sequence to give access to the configuration mode. Some proprietary CLIs allow users to be added to groups that have limited access privileges. However, NACM offers possibilities that are not available in SNMP and CLI as they have no precise, well-defined, standardized method to limit authenticated user's rights and permissions.

4.3.3 Security of Management Application

There are many different management applications, and their implementation varies from one enterprise to another. These management applications usually do not actively impose any security measures. However, their application and implementation are designed to improve the overall security posture of the entire network. For example, NSO improves the security as it provides configuration rollbacks, which is useful to avoid any stray configurations in the network that could arise while implementing changes such as

VPN or ACLs. Besides, it enforces consistent security policies across all devices in the network. It also avoids human-related configuration errors and saves audit logs, which helps keep track and in minimizing any security risk caused due to misconfiguration.

4.4 Testing NETCONF and YANG based Host Management

The testing method involves the description of the process involving the compilation of standard YANG data models to access and modify the configuration and management of some common properties within a host device that exposes a NETCONF interface.

The NETCONF interface on a host device is exposed by installing ConfD management interface tool on the device. In ConfD, confdc compiler is the main development tool that is used to build the ConfD's LinuxCfg application package. ConfD's LinuxCfg package implements some widely used YANG specifications such as system management, interface management, IP management, etc. Since, YANG is a modular data modeling language, a network engineer with programming experience can add or remove modules on the basis of the requirements of the customer.

During the first step of building the LinuxCfg package, confdc compiler in ConfD uses the YANG data model and compiles it into to a .fxs file. A .fxs file is the compiled binary version of the YANG data model. When ConfD starts, all compiled .fxs files form the schema of the database. Since LinuxCfg uses C programming, confdc further takes the .fxs schema file as an input and generates a header file with all the namespaces and constants that contains mapping to interact with the ConfD core API. The next step in the build process involves the generation of OS object file and linking the .o file to executables which are linked to ConfD as well as system library. At this point, the build process is finished and ConfD could be started to interact with the Linux host.

The compilation process can be tested using the confdc command to compile a YANG data module ietf_system.yang.

```
vijay@host:~/confd-6.6/examples.confd/linuxcfg/ietf_system$ confdc -c ietf-system.yang
```

Confdc compiler generates a .fxs file, which can be further compiled to generate a

header file ietf_system.h. The .h and C programming code in the .c file in LinuxCfg then interact with the Linux kernel for configuring the properties supported by the YANG modules.

The command below shows the information about the generated .fxs file. It is important to note that the generated .fxs file has dependencies on three standard namespaces.

```
vijay@host:~/confd-6.6/examples.confd/linuxcfg/ietf_system$ confdc --get-info ietf-system.fxs
fxs file
Confdc version:          "6.6_1" (current Confdc version = "6.6_1")
uri:                     urn:ietf:params:xml:ns:yang:ietf-system
id:                      urn:ietf:params:xml:ns:yang:ietf-system
prefix:                  "sys"
flags:                   118
type:                    cs
mountpoint:              undefined
exported agents:         all
dependencies:            ['urn:ietf:params:xml:ns:yang:iana-crypt-hash',
                          'urn:ietf:params:xml:ns:yang:ietf-inet-types',
                          'urn:ietf:params:xml:ns:yang:ietf-yang-types']
embedded YANG source:    ietf-system@2014-08-06
source:                  "ietf-system.yang"
checksum:                4760dac8fdc2ffcd66f115aa2c3fd67e
cdb checksum:            4597898ec785e962e88c18cf9207022a
```

All these dependencies listed above should be supported by the NETCONF server on the host machine. This could be tested and verified by connecting to the northbound NETCONF interface of the host machine using the southbound NETCONF interface of the NSO. If the device returns the list of YANG modules, it means it is reachable through NSO.

For example, below is the test for a host device added by the name "test" using pioneer package in NSO. Pioneer fetches the list of YANG modules supported by the host device. Before performing this test, the device is first added to an authgroup and then added into NSO using the device credentials as described under Section A.3.8 of Appendix A.

```
admin@ncs(config)# devices device test pioneer yang fetch-list
Retrieving module list from device
Device does not report support for netconf-monitoring, trying anyway

Found out the names for a total of 19 modules
hello message: 19
netconf-monitoring subtree: 0
netconf-monitoring xpath: 0
Marked module ietf-netconf for download
Marked module ietf-ip for download
```

```
Marked module IF-MIB for download
Marked module ietf-interfaces for download
Marked module ietf-inet-types for download
Marked module IANAifType-MIB for download
Marked module IP-MIB for download
Marked module tailf-acm for download
Marked module iana-crypt-hash for download
Marked module ietf-netconf-acm for download
Marked module ietf-system for download
Marked module UDP-MIB for download
Marked module INET-ADDRESS-MIB for download
Marked module iana-if-type for download
Marked module TCP-MIB for download
Marked module ietf-netconf-with-defaults for download
Marked module ietf-yang-types for download
Marked module SNMPv2-TC for download
Marked module tailf-aaa for download
message Marked 19 modules for download, skipped 0
yang-directory /tmp/download/test
```

A successful test run of the above command indicates that the NETCONF connection from NSO to the host device is feasible and that the host device supports the YANG modules returned by the device in the pioneer fetch list. The final step is to build the NED package and then reload it to NSO as described under Section A.7 of the Appendix A. Once NED is reloaded, a network engineer can log into NSO CLI and can perform the configuration supported by the device.

Below is a test run of the ietf_system YANG component to change the hostname of the device:

```
host(config)# system
Possible completions:
  clock  contact  dns-resolver  hostname  location  ntp
host(config)# system hostname ServerEV8Concordia
host(config)# commit
Commit complete.
ServerEV8Concordia(config)#
```

The structure and syntax of the CLI commands can be well understood by looking over the Pyang tree format output of the ietf_system YANG module. From Figure 4.2, system is the container and hostname is one of the leaves in that container. Hostname has a derived type inet:domain-name defined in the urn:ietf:params:xml:ns:yang:inet-types namespace under imported ietf-inet-types module.

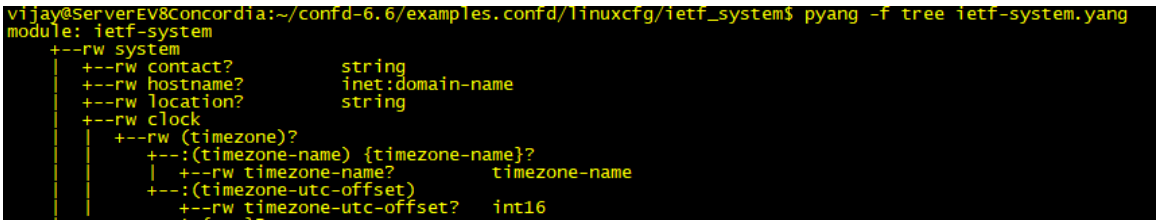

Figure 4.2: pyang tree format translation of ietf_system YANG module

4.5 Merits of Host Management using NETCONF and YANG

It is obvious to have the thought that the same job of changing a hostname or any other configuration can be achieved using the configuration management tools such as Ansible or simply using the device CLI. However, an enterprise-level implementation of NETCONF and YANG based host management using Cisco NSO has a number of benefits as compared to other approaches, such as:

- Network-wide transactions with support for roll back in case of configuration failure while provisioning services on a device in the network.
- Multi-vendor interoperability through a single network-wide CLI or Web UI for all the devices in the network.
- Easier network discovery and topology scan of the devices in the network making it simple for a network engineer to understand and provision network services.
- Extensibility by adding standardized YANG models as per the business requirements of the customer.
- Automation of network tasks using a rich set of northbound APIs such as Java and Python.
- Integration with SDN controllers and NFV managers.

Chapter 5

Threat Models

This chapter examines and evaluates the security of the two different approaches for host management. One is configuration and management using a configuration management tool such as Ansible and the other being NETCONF and YANG based management solution using NSO for the configuration and management of host systems.

5.1 Threat Modeling

A threat is a security loophole that could have a possible impact on the integrity of a network, a system, or software application. Threat modeling is a risk assessment technique useful to brainstorm and classify these potential security threats.

There are several threat modeling techniques available from a variety of sources. The application of each and every technique varies from industry to industry (Shevchenko et al., 2018). Some of these techniques are STRIDE (Kohnfelder & Garg, 1999), DREAD (Shostack, 2008), OCTAVE Allegro (Caralli et al., 2007), CAPEC (Barnum, 2008), TARA/-TAL (Wynn et al., 2011) (Casey, 2007), NIST SP 800-154 (Murugiah & Scarfone, 2016), and Attack Trees (Schneier, 1999). For this study, STRIDE is chosen as the preferred method to identify the security threats because the technique focuses on the security analysis of the complete system from an end-user's perspective.

The idea behind using STRIDE is that any application falls under a predictable set of

threats. For the identification of these potential threats, a data flow diagram (DFD) of the application is drawn in Microsoft Threat Modeling Tool (Microsoft, n.d.). A DFD is drawn using the components by selecting processes, external interactors, data stores, data flows, trust line boundaries, and trust border boundaries. Once the visualized sketch is ready, STRIDE modeling is applied at each data entry and exit point and the processes to identify the threats that put application and assets at risk. Any data interaction that crosses trust boundaries exposes further opportunities for adversaries. Finally, on the basis of the data flow diagram, all the potential threats are listed as well as categorized by their severity. Based on the findings, appropriate mitigations are reported.

Prior to doing STRIDE, a network operator would have to "guess" what security mechanisms are needed to find out the potential threats in a system.

Applying STRIDE gives a full list of the threats and the STRIDE done in this thesis gives suggested mitigations for each of these. This would give confidence to the network operators in building systems that are secure and less vulnerable to security breaches.

The list below describes the meaning of each of these threats with examples, the corresponding security property that is violated and example mitigation that could be applied to improve the security.

- **Spoofing:** It is pretending to be something or someone else by illegitimately acquiring the identity. For example, spoofing someone's IP or MAC address. This is a threat against authentication and can be mitigated using passwords and digital signatures.
- **Tampering:** It is the modification of legitimate information for a malicious purpose. For example, modifying username, password, or something saved on disk. This is a threat against data integrity and can be mitigated using ACLs and digital signatures.
- **Repudiation:** It is claiming, denying or disowning to have performed an action. For example, removing logs from the system and claiming not to have acted. This is a threat against non-repudiation and can be mitigated using secure logging and auditing, and digital signatures.

Component Type	Component Symbol	Description
Process	Circle	It represents processes, programs, or services run by the computer, e.g., a kernel process.
Interactor	Square or rectangle	It represents the endpoints connected to the system, e.g., users, clients, servers, etc.
Data Store	Two parallel horizontal lines	It represents databases, files, registry keys, etc.
Data Flow	One way arrow	It represents data in motion over networks, e.g., data over TCP, UDP, or RPC channels.
Trust Line Boundary	Dotted line	It represents a trust boundary between trusted and untrusting components.
Trust Border Boundary	Dotted square or rectangle	It represents a trust boundary between trusted and untrusting components.

Table 5.1: Threat modeling DFD components

- **Information Disclosure:** It is a breach of information to someone not authorized to access it. For example, tapping data and data leak in transit. This is a threat against confidentiality and can be mitigated using encryption and ACLs.
- **Denial of Service:** It is the disruption of service for legitimate users. For example, not able to access a network device or host system because of exhausted resources. This is a threat against availability and can be mitigated using packet filtering, load balancers and ACLs.
- **Elevation of Privilege:** It is getting a higher privilege to access a network device or a host system that a user is not authorized to log into. For example, getting root login access to a network device without proper authorization. This is a threat against authorization and can be mitigated using input validation and ACLs.

The very first step to analyze a system using STRIDE is to brainstorm and build the DFD model of the system. A DFD contains the essential components of the system and its interaction with the other internal and external components. Table 5.1 lists the components available in the Microsoft Threat Modeling Tool to build a DFD and describes their meaning. Each of these components is susceptible to a set of threats listed in Table 5.2. Table 5.2 does not include trust boundaries because all trust boundaries are subjective in nature.

Component Type	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Elevation of Privilege
Process	X	X	X	X	X	X
Interactor	X		X			
Data Store		X		X	X	
Data Flow		X		X	X	

Table 5.2: Threat matrix of components

5.2 Threat Analysis

The security model of two different approaches has been constructed under Subsection 5.2.1 and 5.2.2. As the first step, DFD is derived using the architecture of the application under inspection. For the security evaluation, the STRIDE threat modeling is applied to the respective component types listed in Table 5.1. Next, appropriate mitigation is suggested for all the listed threats. Finally, a threat matrix is generated as per Table 5.2.

It is important to note that security of any application needs to be considered from day one commencing with the requirements analysis phase. The following security analyses are not intended to analyze the vulnerabilities of the applications through penetration testing and code analysis techniques. Instead, it is intended to analyze the security of a system from an architectural point of view.

5.2.1 Security Analysis of Configuration Management using Ansible

There is a wide variety of configuration management tools available in the market, and their implementation varies from industry to industry (Delaet et al., 2010). For this study, Ansible is chosen as the preferred tool as it is not only used for configuration management and automation, but it also supports the orchestration of the network devices and host systems. Besides, it also has modules that provide support for unified configuration management with Cisco NSO using Ansible NSO modules.

DFD Overview

Ansible has a straightforward architecture. Figure 5.1 demonstrates the high-level DFD of Ansible in a typical IT environment. Any host system that has Ansible installed is known as the Control Node. All modules, plugins, tasks, playbooks are stored in the ansible control node. A control node is a host machine that sends configuration commands to the Managed Nodes. A managed node is a host machine or a network device that can be configured using Ansible. It is not required to have a specific application or a package to be installed on managed nodes, except SSH for enabling the connection from control nodes. Ansible ships with many connection plugins; however, SSH is the default protocol used for system administration.

A network engineer connects to the Ansible control node using a bi-directional remote connection application having support for Telnet or SSH to send the configuration commands to the managed nodes. In a standard IT environment, a control node and managed node are located at the data center within the trusted private network of the corporation. Trusted network of an enterprise is referred by the term Corpnet in the Microsoft threat modeling tool. The term Corpnet is used very often by the enterprises to define their trust boundary. A network engineer can connect to the control node through Corpnet LAN or from outside the trusted network using Telnet or SSH. Enterprises also enable public network connections to their trusted private network using IPsec or SSL VPN.

Processes

From the DFD in Figure 5.1, there are two processes viz., control node and managed node. Since only the control node process is under scrutiny, any security relationship with the managed node will be explained separately. From Table 5.2, processes are susceptible to all six STRIDE threats.

- **Spoofing:** An attacker may spoof the identity of the control node. For example, spoofing of identities such as IP address and MAC address may lead to information

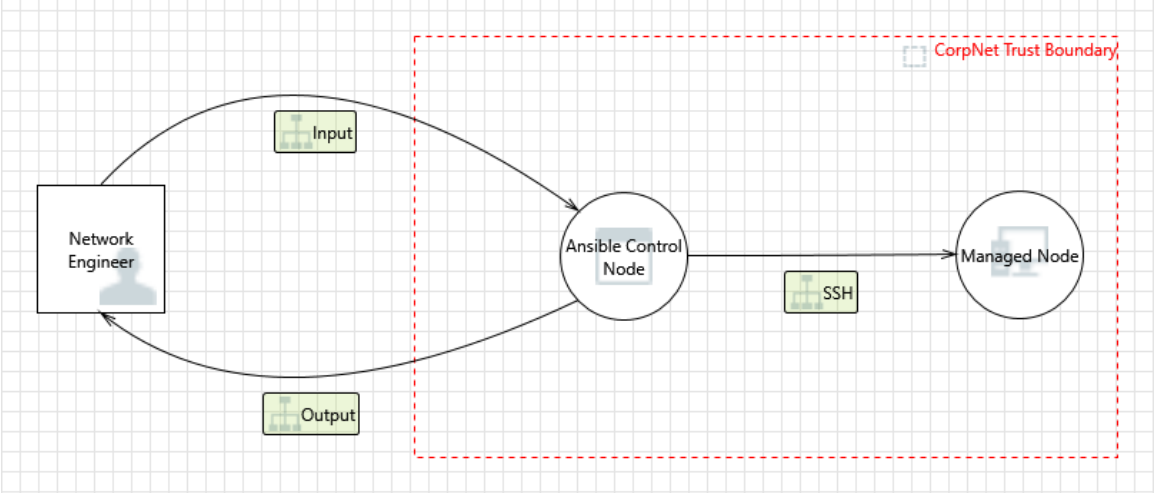

Figure 5.1: High-level DFD of configuration management using Ansible

disclosure such as username and password if sent in plain-text by a network engineer. If an attacker impersonates the IP address of a control node, she can modify the packet headers so that the packet appears to come from a trusted system. She may subsequently send incorrect configurations or may completely shut down all the managed nodes. To mitigate spoofing attacks, a mutual authentication mechanism must be used to identify the destination process.

- **Tampering:** An attacker may tamper with the input data in transit. Failure to verify the input data entering the firewall northbound interface can cause a large number of exploitable issues. For example, this may lead to attacks such as information disclosure, denial of service or an elevation of privilege attack against Ansible control node. Therefore, all input data must be verified for correctness using a standard input validation technique. An attacker can also modify or tamper with the process binaries. However, the Linux kernel is not under scrutiny in this threat model. To mitigate tampering attacks, a message authentication mechanism, filtering using ACLS, or digital signatures must be used.

- **Repudiation:** An attacker or even a legitimate user logged into the control node may make configuration changes on the managed node and may later deny having

done so. To mitigate repudiation attacks, logging or auditing is used. Mitigation for this attack on the control node exists within the Ansible application itself. By default, Ansible logging is off unless a path such as #log_path = /var/log/ansible.log is specified in the "ansible.cfg" config file. Therefore, to mitigate this attack, a network manager can define the #log_path variable in the Ansible config file.

- **Information Disclosure:** An attacker may sniff the data sent by the network engineer to the control node. This data can lead to information disclosure attack revealing the network configurations and can later be misused to attack other parts of the system. Therefore, all input and output connections coming in and going out from the control node through a trust boundary must be encrypted. To mitigate this attack, use of secure connection protocol such as SSH or a privacy-enhanced protocol such as TLS is recommended.

  Information disclosure may also be caused due to a vulnerability in the Ansible native application. For example, a recent high-risk vulnerability in the Ansible "user" module allowed an insider adversary to access sensitive information on the targeted system (CVE-2018-16837, 2018). To mitigate such issues, it is advised to update the application and apply the appropriate patches.

- **Denial of Service:** An attacker may flood the Ansible control node with illegitimate requests causing the services to crash, halt or run slowly. This violates the availability matrix as the control node might not be able to serve any further requests by the network engineer. To mitigate this attack, techniques such as source address filtering, hardening of the server, implementation of load balancers and high availability using the management of primary and secondary control node is recommended (Montgomery, Sriram, Carson, & Santay, 2016).

- **Elevation Of Privilege:** An attacker may get privileged access to the Ansible control node and may change the entire topology of the system. For example, in an environment without proper authentication, an attacker with a privileged set of read-only permissions may elevate the permission set to read-and-write. In order to get write

access, an attacker may create or modify the user profile having the privileged set of access. To mitigate this attack, processes should be run with the least privilege possible. Countermeasure techniques such as ACLs, authentication, and data validation are recommended.

Interactors

From the DFD in Figure 5.1, the network engineer is the interactor interacting with the Ansible control node. From Table 5.2, interactors are prone to spoofing and repudiation attack.

- **Spoofing:** Spoofing attack corresponds to the impersonation of a valid user. If an attacker impersonates the identity of a valid network engineer, she may be able to perform several attacks on the system and may instruct the control node to implement incorrect network configurations. An impersonated network engineer may also lead to data being sent to the attacker's target instead of network engineer. A spoofing attack is mitigated by the use of a two-way authentication mechanism in which both the parties, network engineer and the control node, authenticate each other at the same time. An example of a two-way authentication technique is Kerberos.

- **Repudiation:** An attacker or even an insider network engineer may send a illegitimate request to the Ansible control node and may later deny doing so. This can be avoided by using logging or auditing to record the source, time and summary of the received data on the control node. All event logs in the network can also be forwarded and saved in the Syslog server.

Data Flows

From the DFD in Figure 5.1, there are three data flows corresponding to the Ansible control node viz., the inbound flow of traffic from network engineer to control node, the outbound flow of output data from the control node to network engineer and the outbound

flow of configuration data from the control node to managed node. From Table 5.2, data flows are vulnerable to tampering, information disclosure and denial of service attack.

- **Tampering:** Data in transit between the network engineer and the control node can be tampered with by injecting malicious information. For example, an attacker can tamper with the sensitive input configuration changes sent by the network engineer to the control node. She can also sniff the operational data sent back by the control node to the network engineer. The sniffed data can be later manipulated to perform a high severity attack. As a countermeasure, the data flow between the network engineer and control node must be protected using digital signatures or a message authentication code technique. Data flow between the control node and the managed node is protected using SSH by default.

- **Information Disclosure:** As mentioned under the previous bullet point, an attacker can sniff the data flow at various interfaces to perform a planned attack on other managed devices or servers. The data flow from an inbound and outbound interface on the northbound side of the control node can be protected by encrypting the data flow using a secure connection protocol such as SSH. The mitigation for protection of data flow from the southbound interface of the control node to the northbound interface of the managed node exists in Ansible architecture using SSH.

- **Denial of Service:** As a result of DoS attack, an attacker or an external unauthorized interactor may interrupt data flow in either direction. If an attacker floods the data flow from network engineer to control node, then the control node would crash, and no further communication could be established by the network engineer with the control node. If an attacker floods the data flow from the control node to network engineer, then the network engineer system would crash; however, the likeliness of this occurrence is negligible. If an attacker floods the data flow from the control node to the managed node, then the managed node would crash and would not be able to provide the intended services. As a countermeasure, packet filtering firewalls, IP restriction, network monitoring, and bandwidth control methods can be implemented.

Component Type	Interaction	S	T	R	I	D	E
Processes	Control Node	X	X	*	X	X	X
	Managed Node	X	X	X	X	X	X
Interactors	Network Engineer	X		X			
Data Flows	Network Engineer to Control Node		X		X	X	
	Control Node to Network Engineer		X		X	-	
	Control Node to Managed Node		*		*	X	

[1] X Indicates mitigation exists by the suggested method.
[2] * Indicates mitigation mechanism exists in the application architecture.
[3] - Indicates the likeliness of this event does not affect the main process.
[4] Blank indicates that the threat does not affect the component.

Table 5.3: Threat matrix of configuration management using Ansible

Threat matrix generated by doing the security analysis of the components involved is outlined in Table 5.3.

Note: In some enterprises, Ansible architecture may also consist of cloud-based managed nodes hosted on the Internet as well as a repository of configuration items (CIs) however, their analysis is out-of-scope for this study.

5.2.2 Security Analysis of NETCONF and YANG based Host Management using NSO

For this study, Cisco NSO is chosen as the preferred orchestration tool as it is free to use for non-production use. It is enabled by NETCONF and YANG and bridges technologies such as SDN and NFV for automating services across traditional and modern networks.

DFD Overview

As demonstrated in Figure 5.2, Cisco NSO has comparatively a complex architecture. The NSO application package is installed on a Linux or macOS based host platform. This machine serves as the NETCONF client and manages host machines that act as the NETCONF server. All YANG related data such as device models and service models are stored in an external datastore entity called CDB or YANG datastore. All connections are TCP/IP based and default connection type used for network administration is SSH.

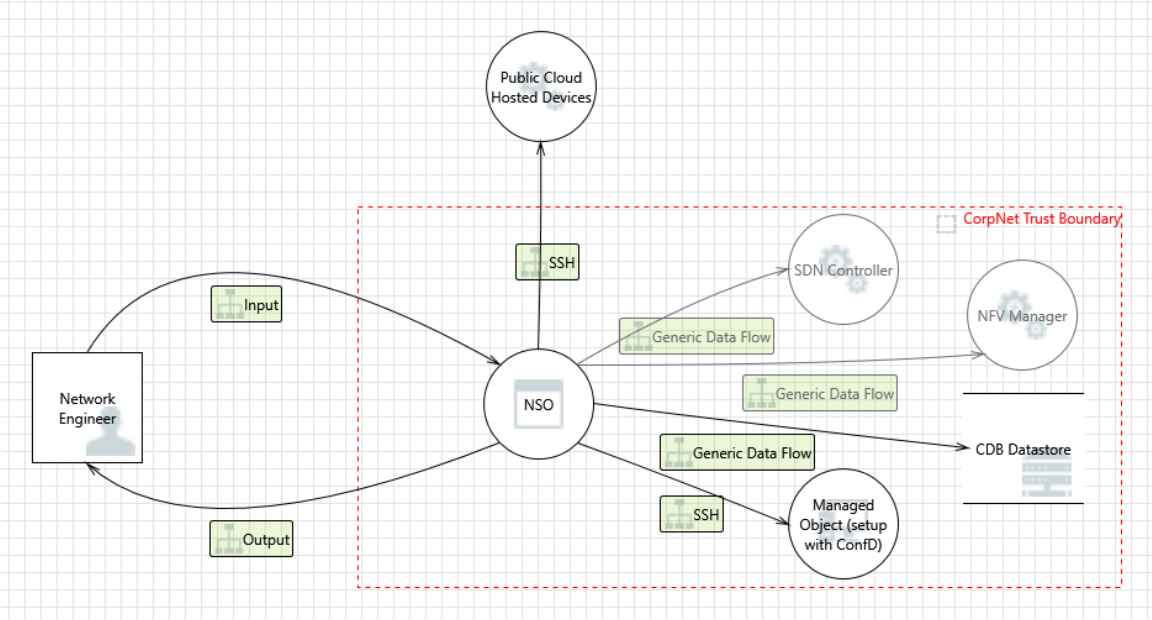

Figure 5.2: High-level DFD of NETCONF and YANG based host management using NSO

A network engineer connects to the NSO host machine to send the configuration commands to the managed objects. In a standard IT environment, the NSO, the YANG datastore, and the managed objects are all located at the data center within the Corpnet trusted network. A network engineer can connect to NSO through Corpnet LAN or from outside the trusted network using VPN. All cloud hosted devices are present at remote locations and their connectivity to NSO is enabled using SSH.

As a complete configuration management solution, NSO sits on top of network applications such as software-defined networking (SDN) controllers and network functions virtualization (NFVs) managers, as shown in Figure 4.1. However, SDN and NFV have different applications and therefore they have not been included in this analysis.

Processes

DFD in Figure 5.2 consists of five processes viz., NSO (the main process under scrutiny), the managed object, public cloud hosted devices, SDN controller and NFV manager. Any security relationship of NSO with the connected components will be described individually. From Table 5.2, processes are susceptible to all six STRIDE threats.

- **Spoofing:** An attacker may impersonate the identity of the NSO process, which may lead to unauthorized access to other connected processes. A process should be able to verify the identity of NSO exactly as needed using a proper authentication mechanism. Mitigation for this threat exists in the NSO application architecture itself. Each device's credential in NSO is set up using an "authgroup", which is a PGM group (username, password, enable password). Once a device is added to authgroup, NSO then pulls the SSH keys from the device to verify the identity to each other. Similarly, NSO has in-built bi-directional authentication mechanism for the network engineer, the source processes and the other supported northbound interfaces. A user can be authenticated using local authentication, pluggable authentication module (PAM), and external authentication. PAM can be configured to work with remote authentication dial-in user service (RADIUS) and lightweight directory access protocol (LDAP).

- **Tampering:** Data flowing across northbound input interface may be tampered with by an attacker. Failure to verify that input is a root cause of a considerable number of exploitable issues. This may lead to several attacks against NSO such as information disclosure, denial of service or an elevation of privilege attack. Kernel binaries of the native OS could also be modified or tampered by an attacker. However, the Linux or macOS kernel is not under scrutiny in this threat model. To mitigate this attack, consider all input paths and the way they handle data. Verify that all input is verified for correctness using an approved list input validation approach. Use of ACLs to specify the access rights can also be used as a countermeasure technique.

- **Repudiation:** NSO can claim that it did not receive configuration change instructions from a source outside the trust boundary. Similarly, an attacker or even a legitimate user may log into NSO and make configuration changes and may later deny having done so. To mitigate this attack, logging or auditing is used to record the source, time, and summary of the events. NSO has inbuilt mechanism for capturing logs. Logging can be enabled via "ncs.conf" configuration file in NSO. Logs are stored under "audit.log" and "ncs.log" file and are located at /var/log/ncs folder for system

installs or under local installation folder for local install.

- **Information Disclosure:** An attacker may sniff data flowing across the NSO northbound input interface sent by the network engineer. This data may include sensitive network configurations thereby gaining access to underlying network routing and topology. Furthermore, depending on what type of data an attacker can read, it may be used to attack other parts of the system or can be a disclosure of information leading to compliance violations. To mitigate this attack, consider encrypting the data flow using SSH or TLS. SSH provides a secure connection mechanism whereas TLS ensures confidentiality by using symmetric data encryption and authentication using TLS handshake.

- **Denial of Service:** DoS or distributed DoS is a tactic used to flood a system with illegitimate requests from multiple compromised computers. NSO under DoS attack may crash, halt or run slowly. This may cause a significant business impact as the NSO process would not be able to serve any immediate configuration changes. To mitigate this attack, techniques such as source address filtering, hardening of the server, implementation of load balancers and high availability using the management of the primary and secondary process is recommended. NSO has connection time out feature to release the resources after a specified period of time. However, in case of a DoS attack that does not make much sense as the attacker would keep on consuming the resources as soon as they are released. Another feature that supports high availability in NSO is the NSO clustering feature that can serve local and remote devices both at the same time.

- **Elevation Of Privilege:** NSO may be subjected to an elevation of privilege using remote code execution by an external malicious entity. A recent vulnerability in the NSO application allowed an unauthenticated malicious user to gain unauthorized access of affected NSO system. The vulnerability could be exploited because of an incomplete input validation for authentication performed by NSO plug and play component package. Once exploited, an attacker could gain unauthorized access to

the configurations of the managed objects connected to NSO (CVE-2018-0463, 2018). Another similar vulnerability in the NSO CLI parser allowed the attacker to execute arbitrary commands with root privileges on the targeted systems (CVE-2018-0274, 2018). To mitigate such attacks, techniques such as input validation, data execution prevention, running applications with least privilege and patching with updated versions are recommended.

Interactors

From the DFD in Figure 5.2, network engineer is the interactor interacting with NSO. From Table 5.2, interactors are prone to spoofing and repudiation attack.

- **Spoofing:** An attacker may impersonate a network engineer's identity by spoofing their IP address. The firewall or NSO's northbound interface may have ACL rule for incoming connections; however, if an attacker is able to bypass the access rule, she can perform several attacks on the system. To mitigate this attack, the use of a bi-directional authentication system is recommended. In NSO, user logs in through CLI, SNMP, Web UI or via NETCONF protocol. In either case, the user needs to log in using the login credentials or a public key in order to gain access. Once a user is authenticated using AAA infrastructure, group membership is established for that user and then all operations that can be performed by that user are authorized.

- **Repudiation:** An attacker or a trusted user may perform unauthorized requests and may later deny doing so. To mitigate such attacks, the use of logging or auditing is recommended. NSO comes with extensive logging functionality. All event logs in the network can also be forwarded and saved in the Syslog server. Syslog logs using NSO can be redirected to a local or a remote server.

Data Stores

NSO has a tree-structured database called as the configuration database (CDB). This database stores a copy of all device-related configurations and is controlled by a YANG

schema. From Table 5.3, a datastore is vulnerable to tampering, information disclosure and denial of service attacks.

- **Tampering:** An attacker can modify or delete the data stored in CDB. By default, CDB is stored under the path "$NCS_DIR/var/ncs/cdb". Location of CDB can be separated from the NSO server to a server that denies all traffic by default and is open only to specific locations or users that are allowed to make changes to it. To mitigate this attack, firewall rules for the database server, ACLs and hardening of the database server are recommended.

- **Information Disclosure:** CDB contains sensitive information related to the network such as routing and network topology. This data is stored in XML format. Unauthorized access by an attacker can lead to a leak of sensitive information about the network and can later be used for other planned attacks. To mitigate this risk, data encryption techniques are recommended.

- **Denial of Service:** CDB datastore can be made unavailable using a denial of service attack by an external malicious entity. Although NSO in a system install is implemented in a high availability cluster, it is recommended to store the CDB data backup on a separate location. CDB keeps its data in two files; A.cdb and C.cdb. If a datastore is not available, a copy of these files could be used, and the copy is then used as the full backup of CDB. To mitigate DoS attacks, additional measures such as packet filtering, load balancers, and ACLs are recommended.

Data Flows

From the DFD in Figure 5.2, there are multiple data flows corresponding to the main NSO process viz., the inbound flow of traffic from network engineer to NSO, the outbound flow of output data from NSO to network engineer and the outbound flow of configuration data from NSO to managed object, the CDB database and the public cloud hosted devices. From Table 5.2, data flows are vulnerable to tampering, information disclosure and denial of service attacks.

Component Type	Interaction	S	T	R	I	D	E
Processes	NSO	*	X	*	X	X	X
	Managed Object	X	X	X	X	X	X
	Public Cloud Hosted Devices	X	X	X	X	X	X
Interactors	Network Engineer	*		*			
Data Stores	CDB Data Store		X		X	X	
Data Flows	Network Engineer to NSO		*		*	X	
	NSO to Network Engineer		*		*	-	
	NSO to Public Cloud Hosted Devices		*		*	X	
	NSO to CDB Data Store		*		*	X	
	NSO to Managed Object		*		*	X	

[1] X Indicates mitigation exists by the suggested method.
[2] * Indicates mitigation mechanism exists in the application architecture.
[3] - Indicates the likeliness of this event does not affect the main process.
[4] Blank indicates that the threat does not affect the component.

Table 5.4: Threat matrix of NETCONF and YANG based host management using NSO

- **Tampering:** Data flow across various interfaces may be tampered with by an attacker. Data in transit between the network engineer and NSO can be tampered with by injecting malicious information. However, NSO has an in-built SSH server. Therefore, all NETCONF related data is sent over SSH. Data across southbound NETCONF interface is protected using SSH as well.

- **Information Disclosure:** Data flow across various interfaces can be sniffed by an attacker. As a countermeasure, the use of encryption is recommended. By default, all XML related data in NETCONF are within an SSH session as an SSH subsystem.

- **Denial of Service:** Using a DoS attack, an attacker may interrupt data flow in either direction, and the devices under attack would therefore not be able to provide the intended services. The only exception being, if an attacker floods the data flow from the NSO to the network engineer, then the network engineer system would crash; however, from an attacker's perspective, the likeliness of this event does not make much sense. As a countermeasure, packet filtering firewalls, IP restriction, network monitoring, clustering, and bandwidth control methods can be implemented.

Threat matrix Table 5.4 represents the summary of the security analysis.

5.3 Security Insights

Even though Subsections 5.2.1 and 5.2.2 cover the technical details of the security analysis, this section focuses on the insights derived from the analysis.

Threat modeling insight

This chapter comprised the threat modeling of two approaches for host management using the STRIDE threat modeling technique. The analysis helped in finding out the potential threats and based on component exposed to that threat, appropriate mitigations are suggested using industry-wide recognized mitigation techniques.

The actual threats and the mitigation techniques may vary from enterprise to enterprise because the security and the threats rely on the design of the network. Besides, the configuration management (CM) tools discussed in this thesis expose management API of a network therefore care must be taken to ensure the security of the system. Any security exploits in the application or an unauthorized access to the management API can break down the whole network. Being a large technology company, Cisco and Red Hat have already taken care of many security-related issues within the application architecture and would continue to do so to provide on-going support to their customers. The security that comes from additional security measures in the NETCONF-based management is prevalent because of secure transport and access control mechanism NACM mentioned under Section 4.3.

Threat matrix threat count insight

As mentioned before, it is obvious that the total number of threats may vary a lot depending on the components that are added to the DFD of the application. Keeping the focus on the main process and the interactions related to it, the threat count is calculated by taking the sum of the number of threats within each item in the legend of the threat matrix table. In the threat matrix table legend, X represents the threats for which mitigation is suggested and * represents the threats for which mitigation exists in the application

Approach	Mitigation Suggested (X)	Mitigation Exists Within Application (*)	N/A (-)
CM using Ansible	13	3	1
CM using NSO	11	14	1

Table 5.5: Threat matrix summary

under scrutiny. Based on the threat count of X and * for the main process and its related components, threat matrix summary Table 5.5 is generated.

CM using Ansible has 13 threats and CM using NSO has 11 threats for which mitigations are suggested using appropriate measures. However, the actual number of threats may vary from enterprise to enterprise due to its dependency on the design of the network. Therefore, this information does not provide any tangible insight about the security of the approach under inspection.

There is one threat in each approach, marked as N/A, which does not affect the main process under scrutiny because of the fewer odds of the event.

Furthermore, CM using Ansible has 3 threats and CM using NSO has 14 threats for which mitigation exists within the application itself. This information is indeed more important. A higher number of mitigation measures within the application indicates better security. These threat mitigations are enforced by the in-built security measures in the application. Whereas, mitigations for some threats in the CM using NSO approach also exist because of the additional security measures available in the NETCONF protocol.

Attack surface insight

In the context of this research, an attack surface is the count of different threats, or various attack vectors (paths), in a system that can be exploited by an attacker. The larger the attack surface, the higher is the security risk. Therefore, to construct a secure system, the goal is to minimize the attack surface as much as possible.

Host CM using legacy CM protocol SNMP has comparatively a larger attacker surface as compared to the NETCONF and YANG based host CM using NSO. CM using SNMP constitute for a larger attack surface because of its demerits such as unreliable transport

service UDP and unencrypted transmission of configuration data in the initial versions of the protocol. On the contrary, NETCONF and YANG based host CM using NSO and ConfD constitute for a smaller attack surface because configuration related data is transmitted over connection-oriented service TCP and communication paths are encrypted using SSH 5.3.

Figure 5.3: Attack surface

Simpler IT infrastructure security management insight

In most IT environments, network and security teams work in silos, which restricts the horizontal communication between both the teams. NSO makes it possible for network and security teams to work together to deliver the end-to-end network services. This could be well understood by taking the example of implementing VPN services in an enterprise. Once all the network and security devices are added to NSO, NSO provides a single abstraction layer for all the devices in the network using well-defined device YANG models. A network engineer can use these YANG models to design and automatically change the VPN service provisioning. Also, sensitive configurations such as VPN are automated using the transactional model discussed in Section 4.2. Any change to a VPN device is either entirely implemented or entirely discarded. Therefore, there is a smaller chance of security loopholes related to stray configurations in the network. Similarly, a YANG based approach can also simplify the firewall rules and ACLs management of network and host devices. This is harder to achieve using CLI based configuration management tools because of the lack of device abstraction.

Finally, while consulting the security analysis from this thesis, a system designer, a solution architect or an application designer may have different opinions about the design of the DFDs and the populated threat matrices. However, different opinions, discussions, and suggestions would only help in further strengthening the security.

Chapter 6

Conclusion and Future Work

CLI, API, SNMP, and various other configuration management technologies have dominated the configuration management of host systems for a very long time. This thesis addresses the problems with these configuration management techniques and proposes a NETCONF and YANG based host management solution that can overcome these issues. The thesis also covers the security analysis of the two different approaches for host management. One is configuration and management using a configuration management tool such as Ansible and the other being the presented NETCONF and YANG based management solution using NSO and ConfD. The material in Section 4.4 shows the actual result of using NSO/ConfD to set a single (non-networking) parameter on a Linux host. This confirms that the approach is viable, and thus allowed us to proceed with the primary result of this thesis, which is the demonstration of the superior security of the NSO/ConfD approach. The results of the security analysis, the listed solution capabilities, the security efficiencies and with the advancements in automation and orchestration technologies, it can be concluded that a NETCONF and YANG based host management solution is superior to the current legacy management solutions in many ways.

The proposed solution can be further extended and examined by doing a real-world scalability and performance test and by adding support for additional YANG modules and interfaces to encourage the use of NETCONF-YANG based host management in large

enterprises. Since the tools used in this thesis are proprietary, therefore not much information is available in the existing literature about their usefulness and application. The good news is NETCONF is free to use in the basic version of ConfD, and that is why a NETCONF connection from NSO to Linux boxes was made feasible using ConfD as the middle component. The thesis could be used as a starting point by someone who wants to use NETCONF and YANG for host management using NSO and can develop the package in ConfD as per their desired management requirements. The information in this thesis can also be continuation work for a student interested in developing the package by utilizing open-source YANG interfaces. Finally, all threat analysis results also need to be verified using a thorough penetration testing as well as static and dynamic code analysis of the native applications in use.

That being said, the transition of technology from a current environment to a new environment in an enterprise is laborious and distasteful. However, the transition can eventually reap significant performance, cost, productivity, and security benefits.

Appendix A

Appendix

A.1 Lab Machines Specifications

NSO and Ansible are installed on a dedicated Linux machine named Hartmanis running Ubuntu 16.04.5 LTS version codename Xenial with 64-bit OS type on an Intel Core i5-2400 CPU at 3.10GHz clock speed with 4GB of RAM and 242GB of HDD. The host machine model is HP Z210. It uses a dedicated PCI ethernet card for its connectivity to other machines in the network.

Cisco CSR 1000V NETCONF router is installed on a dedicated Linux machine named Gingko running Ubuntu 16.04.5 LTS version codename Xenial with 64-bit OS type on an Intel Core i5-2400 CPU at 3.10GHz clock speed with 6GB of RAM and 238 GB of HDD. The host machine model is HP Z210. It has three dedicated PCI ethernet cards for its connectivity to other machines in the network.

Cisco CSR 1000V SNMP router is installed on a dedicated Linux machine named Tarjan running Ubuntu 16.04.5 LTS version codename Xenial with 64-bit OS type on an Intel Core i5-2400 CPU at 3.10GHz clock speed with 6GB of RAM and 242GB of HDD. The host machine model is HP Z210. It has three dedicated PCI ethernet cards for its connectivity to other machines in the network.

Rest of the machines have relatively similar OS configurations as the machines above. Backus, Iversion, Ritchie, Minsky, and Dijkstra has an Intel Core 2-6300 CPU at 1.86GHz

clock speed with 2GB of RAM and 155GB of HDD. The host machine model on these machines is Dell 390. Cocke is a Dell T3400 machine with an Intel Core 2 Duo-E6750 CPU at 2.66GHz clock speed with 2GB of RAM and 155GB of HDD. McCarthy is a Dell T3400 machine with an Intel Core 2 Duo-E7400 CPU at 2.80GHz clock speed with 2GB of RAM and 244GB of HDD.

Cisco CSR 1000V routers are installed on Ubuntu machines using the .iso file on a KVM hypervisor. Installation process requires manually creating the VM on the hypervisor. ConfD is installed and setup on Iversion.

A.2 Installing KVM

To check if the processor supports hardware virtualization:

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

Output value 0 means that the CPU does not support hardware virtualization.

Output value 1 or more means the processor does support virtualization, but it still needs to be ensured that the virtualization is enabled in the BIOS.

Alternatively, we can execute:

```
$ kvm-ok
```

This would return the following output:

```
INFO: /dev/kvm exists
KVM acceleration can be used
```

A.2.1 Dependencies

The command below will make sure that all the dependencies are installed that are required for the functioning of KVM. Different packages will serve different purposes such as administering QEMU and KVM instances, building virtual machines, and creating a bridge from the network to the VMs.

```
$ sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder
   ↪ bridge-utils
```

A.2.2 Adding user to libvirtd group

```
$ sudo adduser vijay libvirtd
Adding user 'vijay' to group 'libvirtd' ...
```

The user needs to login again for them to become a member of *libvirtd* group. The member of this group can on run virtual machines.

A.2.3 Verifying installation

```
$ virsh list --all
 Id Name                 State
----------------------------------
$
```

The output would be empty since at this point no VM has been added.

A.2.4 Installing GUI

```
$ sudo apt-get install virt-manager
```

virt-manager is the GUI tool that will be used subsequently to add and manage the virtual machines.

A.2.5 Autostarting VM on startup

```
$ virsh autostart CSR1000v-Netconf
```

Once the VM has been created (explained later), it can be configured so that it starts automatically with the system reboot.

Next, the following steps are performed on the KVM server:

A.2.6 Creating the Cisco CSR 1000V VM

Installing Cisco CSR 1000V routers .iso files using *virt-manager* is a straightforward process. Installation requirements and a step-by-step guide on the installation process is given in the "Cisco CSR 1000V Series Cloud Services Router Software Configuration Guide" that ships with the package file.

A.2.7 Booting the Cisco CSR 1000V VM

By default, the Cisco CSR 1000V router is accessed using the virtual VGA console. Alternatively, the VMs can also be configured to use the Serial console port. Configuring a serial console port would allow access to the router using a Telnet or SSH connection from any other machine in the lab. Following command can be used to Telnet to the VM:

```
$ telnet host-ip portnumber
```

For the console access to work through the serial port, the router must be configured using *platform console serial* command:

```
R1>en
R1#conf t
R1(config)#platform console serial
R1(config)#end
R1#wr
Building configuration...
[OK]
R1#reload
```

A.2.8 Assigning hostname to the VM

```
R1>en
R1#conf t
R1(config)#hostname CSR1000v-NETCONF
R1(config)#end
R1#wr
Building configuration...
[OK]
```

Similarly, the other router is setup and configured with the hostname CSR1000v-SNMP.

A.3 Installing NSO

Cisco's NSO (NSO, n.d.) supports Linux and Mac OS X platforms. The installer file is distributed by the OS name Linux and Darwin respectively. Installation steps are performed on the Ubuntu platform; a Linux distribution based on Debian. The package can be installed in the following two ways:

- Local installation – used in lab environment for evaluation, proof of concept and development of the product.
- System installation – used for deployment of the product in production environment.

NSO installation file archive has the following naming pattern:
nso-VERSION.OS.ARCH.installer.bin
The variables in the file name define the version, OS and the CPU architecture.

A.3.1 Dependencies

Additional packages are required that will serve different purposes such as building NSO examples, using NETCONF console, etc.

```
$ sudo apt-get install default-jdk
$ sudo apt-get install ant
$ sudo apt-get install python-paramiko
```

Verifying the installation of dependencies:

```
$ java -version
$ ant -version
```

Web UI can be accessed through Firefox browser installed on Ubuntu by visiting the following URL: http://127.0.0.1:8080/login.html

A.3.2 NSO Local Install

Installation file used is *nso-4.1.linux.x86_64.installer.bin*. Add-ons are tested on a newer version *nso-4.7.linux.x86_64.installer.bin*.

(a) NSO is installed in a single directory (i.e. *$HOME* in this case), under folder name *ncs-4.1*, using attribute value *–local-install*:

```
$ sh nso-4.1.linux.x86_64.installer.bin $HOME/ncs-4.1 --local
   ↪ -install
```

Installation notes generated from the above command are as following:

```
INFO  Using temporary directory /tmp/ncs_installer.15038 to
   ↪ stage NCS installation bundle
INFO  Unpacked ncs-4.1 in /home/vijay/ncs-4.1
INFO  Found and unpacked corresponding DOCUMENTATION_PACKAGE
INFO  Found and unpacked corresponding EXAMPLE_PACKAGE
INFO  Generating default SSH hostkey (this may take some time
   ↪ )
INFO  SSH hostkey generated
```

```
INFO  Environment set-up generated in /home/vijay/ncs-4.1/
   ↪ ncsrc
INFO  NCS installation script finished
INFO  Found and unpacked corresponding NETSIM_PACKAGE
INFO  NCS installation complete
```

(b) Installation program creates a shell script file named ncsrc (under *$HOME/ncs-4.1/*). This file need to be executed using source command to set the environment variables into the current shell script:

```
$ source $HOME/ncs-4.1/ncsrc
```

To check the environment variable populated after running the above command:

```
$ echo $NCS_DIR
```

This would return */home/vijay/ncs-4.1*; the installation directory of NSO.

(c) Verifying the NSO status:

```
$ ncs --status | grep status
$ ncs --version
```

(d) Creating a run-time directory where NSO would store its database and log files:

```
$ ncs-setup --dest $HOME/ncs-run
```

(e) Starting NSO daemon under the run-time directory:

```
$ cd $HOME/ncs-run
$ ncs
```

NSO daemon need not be run at this point as there are no real or simulated devices that have been added or created so far. It will be followed in the subsequent installation steps.

A.3.3 Setting up a simulated network of devices using Netsim

NSO comes pre-loaded with a network simulator called Netsim. Netsim provides the capability within NSO to create dummy devices that run-in localhost using NEDs. It also creates a management plane for the devices without exposing any control plane or data plane. Netsim devices give a fair indication of how the configurations would behave on the real devices.

In the lab, example packages that come with NSO installation bundle are used to demonstrate how a set of simulated network of Cisco IOS, Juniper Junos and a Netconf device (using its YANG file) can be created and added using Netsim. Subsequently, real devices are added using a system build NED and the NEDs in *$NCS_DIR/packages/neds/* directory.

Creating Netsim devices and running the simulator:

Very first step before running any example is to make sure to source the *ncsrc* file in *$NCS_DIR*:

```
$ source $HOME/ncs-4.1/ncsrc
```

Then go to the following directory:

$NCS_DIR/examples.ncs/getting-started/using-ncs/1-simulated-cisco-ios/

(a) Generate a network simulator with 3 devices c0, c1, c2. They all speak Cisco CLI:

```
$ ncs-netsim create-network $NCS_DIR/packages/neds/cisco-ios
   ↪ 3 c
DEVICE c0 CREATED
DEVICE c1 CREATED
DEVICE c2 CREATED
```

(b) Start the simulated devices:

```
$ ncs-netsim start
DEVICE c0 OK STARTED
```

```
DEVICE c1 OK STARTED
DEVICE c2 OK STARTED
```

(c) Run the CLI towards one of the simulated devices:

```
$ ncs-netsim cli-i c0
```

(Above command opens a Cisco IOS style CLI for device c0.)

```
c0> enable
c0# show running-config
```

(Above command shows that the device has some initial config.)

pwd will print the working directory for this simulated device:

```
c0# pwd
/home/vijay/ncs-4.1/examples.ncs/getting-started/using-ncs/1-
   ↪ simulated-cisco-ios/netsim/c/c0
```

Structural device configurations are stored in the following file:

$NCS_DIR/examples.ncs/getting-started-using-ncs/1-simulated-cisco-ios/netsim/c/c0/cdb/ios.xml.

A.3.4 Setting up and starting NSO

Go to directory *$NCS_DIR/examples.ncs/getting-started/using-ncs/1-simulated-cisco-ios*

Create NSO environment ready to run towards the simulated devices using *ncs-setup* command:

```
$ ncs-setup --netsim-dir ./netsim --dest .
```

Above command creates the run-time directories needed by NSO, links the NED package and populates the NSO database with meta-data of the simulated devices.

Start NSO:

```
$ ncs
```

Start the NSO CLI as user admin with a Cisco style CLI:

```
$ ncs_cli -C -u admin
admin@ncs#
```

NSO also supports a Juniper style CLI, that is started by using a -J modification to the command as follows:

```
$ ncs_cli -J -u admin
admin@ncs>
```

To switch between Cisco IOS and Juniper style CLI:

```
admin@ncs> switch cli
```

A.3.5 Adding Netsim devices to NSO

Go to the directory where Netsim devices were created and execute the following commands:

```
$ ncs-netsim list
$ ncs_cli -C -u admin
admin@ncs# config
admin@ncs(config)# devices device c0 address 127.0.0.1 port 10022
   ↪  device-type cli
admin@ncs(config-device-c0)# authgroup default
admin@ncs(config-device-c0)# state admin-state unlocked
admin@ncs(config-device-c0)# commit
admin@ncs(config-device-c0)# ssh fetch-host-keys
admin@ncs(config-device-c0)# sync-from
admin@ncs(config-device-c0)# end
admin@ncs#
```

A.3.6 Setting up run-time directory and loading NED packages for adding real devices to NSO

Create a new directory (let's say under *$HOME*) and execute the following commands:

```
$ mkdir nso-data
$ ncs-setup --dest nso-data
$ cd nso-data
$ ncs
```

Under *~/nso-data/packages*, create a symlink for packages from ncs-4.1 directory:

```
$ ln -s ~/ncs-4.1/packages/neds/cisco-ios cisco-ios
```

Similarly, create symlinks for all the packages that will be used in the lab setup. Reload the packages once symlink has been created:

```
~/nso-data/packages$ ncs_cli -C -u admin
admin@ncs# packages reload
```

A.3.7 Initial configuration of real NETCONF device

Virtual router CSR1000v-NETCONF is configured using the following initial configuration:

Enabling SSH connectivity on the router:

```
~/nso-data/packages$ ncs_cli -C -u admin
CSR1000v-NETCONF#conf t
CSR1000v-NETCONF(config)#ip domain-name concordia.ca
CSR1000v-NETCONF(config)#crypto key generate rsa
2048
CSR1000v-NETCONF(config)#username admin priv 15 password 15
CSR1000v-NETCONF(config)#enable password admin
CSR1000v-NETCONF(config)line vty  0 4
```

```
CSR1000v-NETCONF(config)#transport input all
CSR1000v-NETCONF(config)#end
CSR1000v-NETCONF#wr
Building configuration...
[OK]
```

Enabling NETCONF on the router:

```
CSR1000v-NETCONF>en
Password:
CSR1000v-NETCONF#conf t
CSR1000v-NETCONF(config)#netconf-yang
CSR1000v-NETCONF(config)#exit
CSR1000v-NETCONF#wr
Building configuration...
[OK]
```

To check the status of the software processes required to support NETCONF-YANG:

```
CSR1000v-NETCONF#show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
vtyserverutild   : Running
opdatamgrd       : Running
nginx            : Running
ndbmand          : Running
```

Check for NETCONF-SSH connectivity to the router:

```
vijay@Gingko:~$ ssh -s admin@192.168.122.182 -p 830 netconf
```

A successful connection to the device will return the list of capabilities supported by the device.

Note – if connection fails, then go to device and remove the *aaa new-model* config or, apply the following config required for NETCONF-SSH connectivity and edit-config operations:

```
CSR1000v-NETCONF>en
Password:
CSR1000v-NETCONF#conf t
CSR1000v-NETCONF(config)# aaa authorization exec default local
CSR1000v-NETCONF(config)#exit
CSR1000v-NETCONF#wr
Building configuration...
[OK]
```

Sending hello message to the device to inform the device that you are a NETCONF client:

NSO comes with *netconf-console* tool that can be used for testing NETCONF operations. In the example below, an explicit device username, password, and the NETCONF port is specified with the hello command to connect to the device and print its capabilities:

```
$ netconf-console --host=192.168.122.182 -u admin -p admin --port
   ↪  830 --hello
```

or, using an XML file RPC request:

```
<?xml version="1.0" encoding="UTF-8"?>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <capabilities>
      <capability>urn:ietf:params:netconf:base:1.0</capability>
   </capabilities>
```

```
</hello>
]]>]]>
```

NETCONF get-config operation:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="1"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
         <running />
      </source>
   </get-config>
</rpc>
]]>]]>
```

To get a constructed view, the output can be passed through an XML formatter tool.

NETCONF edit-config operation:

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc message-id="1"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" message-id="101">
   <edit-config>
      <target>
         <running />
      </target>
      <config>
         <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-
            ↪ native">
            <hostname>CSR1000v-NETCONF-New</hostname>
         </native>
```

```
      </config>
   </edit-config>
</rpc>
]]>]]>
```

Similarly, other NETCONF operations can be performed.

A.3.8 Adding real NETCONF device using Pioneer

Pioneer is an add-on that can be used to perform NETCONF operations on a NETCONF supported device. It also allows to build a custom NETCONF NED for the device using its YANG data models.

Install following additional dependencies:

```
$ sudo apt install python-pip
$ pip install paramiko
$ sudo apt-get install xsltproc
```

To check if all the dependencies are fulfilled:

```
$ which ncs && which python && python -c "import␣paramiko" \
&& which xsltproc && which xmllint && which bash && which pyang \
&& echo "All␣Fine"
```

Downloading, installing and building the NSO Pioneer package:

```
$ cd packages
$ git clone https://github.com/NSO-developer/pioneer
$ cd pioneer/src; make all; cd -
PYTHONPATH=../python:$PYTHONPATH python ../python/pioneer/action.
   ↪ py
/home/vijay/ncs-run/packages
```

Reloading packages:

```
$ ncs_cli -C -u admin

admin connected from 132.205.2.217 using ssh on Hartmanis
admin@ncs# packages reload

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has
   ↪ completed.
>>> System upgrade has completed successfully.
reload-result {
    package pioneer
    result true
}
```

Creating an authgroup:

```
admin@ncs# conf
admin@ncs(config)# devices authgroups group thesis-project
   ↪ default-map remote-name admin remote-password admin remote-
   ↪ secondary-password admin
admin@ncs(config-group-thesis-project)# commit
exit
```

Adding an IOS device using built-in Cisco IOS NED:

```
admin@ncs(config)# devices device CSR1000v address 132.205.9.49
   ↪ authgroup thesis-project device-type cli ned-id cisco-ios
admin@ncs(config-device-CSR1000v)# state admin-state unlocked
admin@ncs(config-device-CSR1000v)# ssh fetch-host-keys
admin@ncs(config-device-CSR1000v)# commit
```

```
admin@ncs(config-device-CSR1000v)# top
admin@ncs(config)# exit
admin@ncs# devices sync-from
admin@ncs# show devices device CSR1000v
```

The device has now been added to NSO using the built-in Cisco IOS NED that ships with the NSO installation package. This NED doesn't support NETCONF operations. Pioneer is used to build the NETCONF NED using the device YANG models.

Creating a device list entry for the NETCONF device, committing the changes and connecting to the device:

```
admin@ncs(config)# devices device CSR1000v-NETCONF address
   ↪ 132.205.9.49 port 830 authgroup thesis-project device-type
   ↪ netconf trace raw
admin@ncs(config-device-CSR1000v-NETCONF)# state admin-state
   ↪ unlocked
admin@ncs(config-device-CSR1000v-NETCONF)# ssh fetch-host-keys
admin@ncs(config-device-CSR1000v-NETCONF)# commit
admin@ncs(config-device-CSR1000v-NETCONF)# top
admin@ncs(config)# exit
admin@ncs(config)# connect
```

At this point, any attempt to *sync-from* will fail since no NETCONF NED has been build for the device yet.

Testing pioneer NETCONF hello:

```
admin@ncs(config)# devices device CSR1000v-NETCONF pioneer
   ↪ netconf hello
```

This will return a *hello* response.

Fetching YANG models list from the device:

```
admin@ncs(config)# devices device CSR1000v-NETCONF pioneer yang
   ↪ fetch-list
admin@ncs(config)# devices device CSR1000v-NETCONF pioneer yang
   ↪ download
```

This will return a list of YANG files supported by the device. If the *fetch-list* doesn't work, YANG models need to be retrieved manually from the device owner.

Building NETCONF NED:

```
admin@ncs(config)# devices device CSR1000v-NETCONF pioneer yang
   ↪ build-netconf-ned
```

Installing NETCONF NED:

```
admin@ncs(config)# devices device CSR1000v-NETCONF pioneer yang
   ↪ install-netconf-ned
```

Reloading packages to use the package in NSO:

```
admin@ncs# packages reload
```

Checking sync-from:

```
admin@ncs(config)# devices device CSR1000v-NETCONF sync-from
sync-result {
 device CSR1000v-NETCONF
 result true
}
```

While building a NETCONF NED, it is possible that some YANG files might already be in NSO, or some may return an error. If so, NSO will return a circular dependency error. It can be fixed by disabling the broken YANG types and then building the NED again.

Currently, there is no built-in method in NSO to build the NED for a device. Future versions of NSO, 4.8 onwards, will supposedly have a built-in NETCONF NED builder

through which it will be possible to generate, build and load YANG-based NETCONF NEDs automatically. It will significantly reduce the number of manual steps to onboard the devices to NSO.

A.3.9 Adding real SNMP device to NSO

To manage an SNMP device, its data models should be imported into NSO. These data models are defined in MIBs. NSO uses these data models to generate configuration commands. These data models are converted to YANG data models and the YANG data models are then compiled to build the NED.

Also, to onboard the device to KVM, it is obvious to apply the initial configuration similar to NETCONF device, with some additional configurations specific to SNMP devices.

Configuring SNMP v3 on a Cisco CSR 1000V router:

```
CSR1000v-SNMP#conf t
CSR1000v-SNMP(config)#snmp-server group G1 v3 priv
CSR1000v-SNMP(config)#snmp-server user U1 G1 v3 auth sha snmpapwd
   ↪  priv aes 128 snmpepwd
CSR1000v-SNMP(config)#end
CSR1000v-SNMP#show snmp user
User name: U1
Engine ID: 800000090300525400DD61A2
storage-type: nonvolatile        active
Authentication Protocol: SHA
Privacy Protocol: AES128
Group-name: G1
CSR1000v-SNMP#wr
Building configuration...
[OK]
```

Using *ncs-make-package* command to create an SNMP NED package:

ncs-make-package command can be used to create a NED package for a device using its MIB files. MIB for a device can be downloaded from Cisco's online MIB Locator utility. A list of MIBs translated to its equivalent YANG data models can also be downloaded from the online YangModels Git repository.

Once MIBs are retrieved, following commands can then be used to create a package for the device:

```
$ ncs-make-package --snmp-ned ~/Downloads/snmpmib/ snmpned
$ cd snmpned/src; make
```

The above command takes attribute value *–snmp-ned* for building an SNMP package (or, attribute value *–netconf-ned* for building a NETCONF package) and creates a package named snmpned assuming its MIB files are under the folder snmpmib.

The generated package tells NSO about the MIBs the device implements and can also be used to simulate Netsim devices. Once the package has been built, it is loaded into NSO, and the SNMP device is then added using the same procedure just like any other device. We can also use the example NED package that ships with NSO with built-in SNMP NED component. The package is available under the directory *$NCS_DIR/examples.ncs/snmp-ned/*.

A.4 Installing Ansible

Ansible (Ansible, n.d.) is a standalone application installed on a server and does not require an Ansible agent running on client machines. The only requirement is Python and SSH installed and enabled on the client machines that need to be configured via Ansible.

A.4.1 Dependencies

Install the following package and add the ppa repository on the Ansible server machine:

```
$ sudo apt-get install software-properties-common
```

```
$ sudo apt-get update
$ sudo apt-add-repository ppa:ansible/ansible
```

A.4.2 Installing Ansible package

To install Ansible:

```
$ sudo apt-get install ansible
```

To check the version installed:

```
$ ansible --version
ansible 2.6.3
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/vijay/.ansible/plugins
     ↪ /modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-
     ↪ packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.12 (default, Dec  4 2017, 14:50:18) [GCC
     ↪ 5.4.0 20160609]
```

A.4.3 Onboarding client machines

Although the client that needs to be configured using Ansible does not require to have any specific package installed, there needs to be a couple of things that need to be checked. e.g.:

- **SSH keypairs:** required to connect to the client machines without the need to login manually everytime a configuration is pushed. Ansible server has the private key installed on it, whereas the client machines have the server's public key.
- **sudo:** to configure the root file systems (escalate privilege).

- **Static addressing:** Ansible server can have a DHCP assigned address as the configuration commands will initiate from this machine and nothing connects to it. Client machines could have DNS hostname or DHCP assigned address as well, but static addressing is the most reliable way to configure client machines using Ansible.

All the machines in the lab environment have a static IP address. The IP address of these machines are added to *hosts* file on Ansible server to connect using hostnames:

```
$ sudo nano /etc/hosts
```

Add the following lines to ping client machines by name:

```
132.205.9.61    Backus
132.205.9.57    Iversion
132.205.9.53    Ritchie
132.205.9.34    Cocke
132.205.9.33    Minsky
132.205.9.45    Tarjan
132.205.9.49    Gingko
132.205.9.10    Dijkstra
132.205.9.9     McCarthy
```

A.4.4 Generating SSH keys

SSH keypair is generated using ssh-keygen command:

```
$ ssh-keygen
```

Under .ssh folder, two keypairs would be generated, i.e. ida_rsa (private key) and id_rsa.pub (public key):

```
$ ls .ssh
id_rsa  id_rsa.pub
```

Assuming both the server and the client machines have the same username (in this case vijay), transfer the public key to the client machines. It is important to note that a private key is never sent to any other computer or user.

```
$ ssh-copy-id -i .ssh/id_rsa.pub Backus
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ".
   ↪ ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key
   ↪ (s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if
   ↪  you are prompted now it is to install the new keys
vijay@backus's password:
Number of key(s) added: 1
Now try logging into the machine, with: "ssh 'Backus'"
and check to make sure that only the key(s) you wanted were added
   ↪ .
```

Similarly, keys are copied to the other machines that need to be managed and configured using Ansible server.

While copying the public key, explicitly define the username if the client's machine has a different username than the Ansible server:

```
$ ssh-copy-id -i .ssh/id_rsa.pub dev@Backus
```

To bypass the sudo login everytime sudo is used (privileged access), go to the client machine's visudo to edit the */etc/sudoers* file:

```
$ sudo visudo
```

Add the following line to the bottom of sudoers file and save:

```
vijay ALL=(ALL) NOPASSWD: ALL
```

A.4.5 Adding device groups

Next, add device groups so that the configurations can be fetched and sent to multiple devices using group name instead of sending it to individual devices:

```
$ sudo nano /etc/ansible/hosts
```

Add the following lines to the bottom of this file and save:

```
[nso_machines]
Tarjan
Gingko
[lab]
Backus
Iversion
Ritchie
Cocke
Minsky
Tarjan
Gingko
Dijkstra
McCarthy
```

A.4.6 Checking connectivity from Ansible server to client machines

Client machines can now be pinged from Ansible server. If the ping response is SUCCESS, it means client machines are ready to be configured using Ansible server:

```
$ ansible -m ping all
Gingko | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

```
Tarjan | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
Ritchie | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
Iversion | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
Backus | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
Minsky | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
Cocke | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
Dijkstra | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

```
McCarthy | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

A.5 Integrating Ansible and NSO

The built-in northbound REST API in NSO can be used to fetch the device related data by using the *curl* command. The command can be modified to retrieve data in the JSON format. The data retrieved can then be converted into YAML code. This YAML code is used in creating Ansible playbooks to perform various actions on the devices using Ansible NSO modules.

The curl command below fetches the information of the device named c0 from the NSO configuration database (CDB). The output of this command can be copied and parsed into YAML format using an online *json2yaml* utility.

```
$ curl -s -u admin:admin -H "Accept:␣application/vnd.yang.data+
   ↪ json" http://localhost:8080/api/config/devices/device/c0/
   ↪ config?deep
```

Playbook outlining Ansible NSO modules:

```
---
- hosts: 127.0.0.1
  connection: local
  gather_facts: yes

  tasks:
  - name: Sync from devices
    nso_action:
      url: http://localhost:8080/jsonrpc
```

```
    username: admin
    password: admin
    path: /ncs:devices/sync-from

- name: Query device name and device type
  nso_query:
    url: http://localhost:8080/jsonrpc
    username: admin
    password: admin
    xpath: /ncs:devices/device
    fields:
    - name
    - device-type

- name: Apply device configuration
  nso_config:
    url: http://localhost:8080/jsonrpc
    username: admin
    password: admin
    data:
      tailf-ncs:devices:
        device:
        - name: c0
          tailf-ncs:config:
            tailf-ned-cisco-ios:interface:
              FastEthernet:
              - name: 1/0
                ip:
```

```
                    address:
                      primary:
                        address: 10.0.0.10
                        mask: 255.255.255.0
```

Playbook test run showing the tasks in action and the device change state:

```
$ ansible-playbook -v nsoansible.yaml
Using /etc/ansible/ansible.cfg as config file

PLAY [Ansible NSO Modules]
   ********************************************************

TASK [Gathering Facts]
   ********************************************************
ok: [127.0.0.1]

TASK [Sync from devices]
   ********************************************************
changed: [127.0.0.1] => {"changed": true, "output": {"sync-result
   ": [{"device": "c0", "result": "true"}, {"device": "c1", "
   result": "true"}, {"device": "c2", "result": "true"}]}}

TASK [Query device name and device type]
   ********************************************************
ok: [127.0.0.1] => {"changed": false, "output": [["c0", "ios-id:
   cisco-iosssh"], ["c1", "ios-id:cisco-iosssh"], ["c2", "ios-id:
   cisco-iosssh"]]}
```

```
TASK [Apply device configuration]
   ****************************************************
changed: [127.0.0.1] => {"changed": true, "changes": [{"from":
   null, "path": "/ncs:devices/device{c0}/config/ios:interface/
   FastEthernet{1/0}/ip/address/primary/address", "to": "
   10.0.0.10", "type": "set"}, {"from": null, "path": "/ncs:
   devices/device{c0}/config/ios:interface/FastEthernet{1/0}/ip/
   address/primary/mask", "to": "255.255.255.0", "type": "set"}],
    "diffs": []}

PLAY RECAP
   ****************************************************
127.0.0.1                  : ok=4    changed=2    unreachable=0
      failed=0
```

A.6 Installing ConfD

Just like NSO, ConfD (ConfD, n.d.) installation file is also a self-extract archive which is OS/CPU specific. The distribution file used in lab environment is *confd-basic-6.6.linux.x86_64*. To install and unpack the archive file, execute the following command and specify the installation directory as an argument in this command:

```
$ sh confd-basic-6.6.linux.x86_64.installer.bin ~/confd-6.6
```

Installation notes generated by executing the above command are as follows:

```
INFO  Unpacked confd-basic-6.6 in /home/vijay/confd-6.6
INFO  Found and unpacked corresponding DOCUMENTATION_PACKAGE
INFO  Found and unpacked corresponding EXAMPLE_PACKAGE
INFO  Generating default SSH hostkey (this may take some time)
INFO  SSH hostkey generated
```

```
INFO  Environment set-up generated in /home/vijay/confd-6.6/
   ↪ confdrc
INFO  ConfD installation script finished
```

Installation program creates a shell script file named *confdrc* (under *$HOME/confd-6.6/*). The file needs to be executed using the *source* command to set the environment variables into the current shell script:

```
$ source $HOME/confd-6.6/confdrc
```

To check the environment variable populated after running the above command:

```
$ echo $CONFD_DIR
```

This should return */home/vijay/confd-6.6;* the installation directory of ConfD.

Next step is to install LinuxCfg. LinuxCfg is a tool that is included with ConfD and can be used to manage a set of objects for a Linux OS using their YANG data models. It has an inbuilt component that handles the interaction with ConfD and a set of individual components that handle the management of the objects using their data model defined in YANG.

A.6.1 Requirements for LinuxCfg

Building the package requires:

- GNU make C compiler
- ConfD version 6.3.1 or higher (YANG 1.1 is required by some components)

A.6.2 Building and Installing LinuxCfg

Go to *$CONFD_DIR/examples.confd/linuxcfg* directory. On the top level of this directory, there is a makefile with instructions to build and install the components. This file can be modified to add or remove any components that need to be installed.

To install using makefile:

```
$ make install
```

A.6.3 Running LinuxCfg

To start the ConfD and LinuxCfg daemon, following commands can be executed in the given order:

```
$ confd --start-phase0
$ confd --start-phase1
$ ./linuxcfg
$ confd --start-phase2
```

Note: root privileges might be required to run LinuxCfg.

Or, simply start using the make file:

```
$ make start
```

To start ConfD CLI:

```
$ make cli
```

To enter *config* mode in the ConfD prompt:

```
admin connected from 132.205.2.209 using ssh on Iversion
Iversion# config
Entering configuration mode terminal
Iversion(config)#
```

To test components connectivity, execute *system-restart* command from ConfD terminal to restart the system using LinuxCfg *ietf_system* YANG module component:

```
Iversion(config)# system
Possible completions:
  system  system-restart  system-shutdown
```

Similarly, *ietf_interface* and *ietf_routing* component can be tested to perform various operations such as modify network settings, configure IP routes, etc.

```
Iversion(config)# interfaces interface
Possible completions:
  enp4s0  enp5s4  enp5s5  lo  range  virbr0  virbr0-nic
```

To stop the LinuxCfg daemon:

```
$ make stop
```

To stop the ConfD daemon:

```
$ confd --stop
```

A.7 Integrating ConfD and NSO

Once NSO and ConfD have been setup correctly, it is relatively a trivial task to integrate both the tools. Both NSO and ConfD support NETCONF on the northbound and the southbound. This capability can be leveraged to access ConfD's northbound NETCONF interface using NSO's southbound NETCONF interface. In order to achieve this, a NED needs to be build using NSO Pioneer package. This NED enables a NETCONF interface between NSO and the NETCONF enabled ConfD managed object.

To access the northbound interface on ConfD device, make sure that NETCONF is enabled in the *confd.conf* file under the directory: *$CONFD_DIR/etc/confd/*

Also, check for following recommended settings:

```
---
<datastores>
  ---
  <running>
    <access>writable-through-candidate</access>
  </running>
```

```
    ---
  </datastores>
  ---
  <capabilities>
     ---
     <writable-running>
        <enabled>false</enabled>
      </writable-running>
      ---
  </capabilities>
---
```

Creating an authgroup for the ConfD based Linux device:

```
admin@ncs# config
admin@ncs(config)# devices authgroups group confd default-map
   ↪ remote-name vijay remote-password <password>
admin@ncs(config-group-confd)# commit
```

Adding the Linux device to NSO:

```
admin@ncs(config)# devices device LinuxDevice address
   ↪ 132.205.9.54 authgroup confd port 2022 device-type netconf
admin@ncs(config-device-LinuxDevice)# state admin-state unlocked
admin@ncs(config-device-LinuxDevice)# ssh fetch-host-keys
admin@ncs(config-device-LinuxDevice)# commit
admin@ncs(config-device-LinuxDevice)# top
admin@ncs(config)#
```

Fetching YANG models list from the device:

```
admin@ncs(config)# devices device LinuxDevice pioneer yang fetch-
   ↪ list
```

```
admin@ncs(config)# devices device LinuxDevice pioneer yang
   ↪ download
```

Building the NED using retrieved YANG models:

```
admin@ncs(config)# devices device LinuxDevice pioneer yang build-
   ↪ netconf-ned
```

Installing the NETCONF NED:

```
admin@ncs(config)# devices device LinuxDevice pioneer yang
   ↪ install-netconf-ned
```

Reloading package to use the NED in NSO:

```
admin@ncs# packages reload
reload-result {
    package pioneer
    result true
}
reload-result {
    package LinuxDevice
    result true
}
```

References

Ansible, R. (n.d.). *Ansible is simple it automation*. https://www.ansible.com/.

Badra, M., Luchuk, A., & Schoenwaelder, J. (2015). *Using the netconf protocol over tls with mutual x. 509 authentication*. https://tools.ietf.org/html/rfc7589.

Barnum, S. (2008). Common attack pattern enumeration and classification (capec) schema description. , *3*. https://capec.mitre.org/documents/documentation/CAPEC_Schema_Description_v1.3.pdf.

Barrett, M. P. (2018). *Framework for improving critical infrastructure cybersecurity, version 1.1* (Tech. Rep.). https://www.nist.gov/publications/framework-improving-critical-infrastructure-cybersecurity-version-11.

Bellovin, S. M., & Bush, R. (2009). Configuration management and security. *IEEE Journal on Selected Areas in Communications*, *27*(3), 268–274. https://www.cs.columbia.edu/~smb/papers/config-jsac.pdf.

Bierman, A., & Bjorklund, M. (2012). *Network configuration protocol (netconf) access control model* (Tech. Rep.). https://tools.ietf.org/html/rfc6536.

Bierman, A., & Bjorklund, M. (2018). *Network configuration access control model* (Tech. Rep.). https://tools.ietf.org/html/rfc8341.

Bjorklund, M. (2010). *Yang - a data modeling language for the network configuration protocol (netconf)* (Tech. Rep.). https://tools.ietf.org/html/rfc6020.

Bjorklund, M. (2016). *The yang 1.1 data modeling language* (Tech. Rep.). https://tools.ietf.org/html/rfc7950.

BMC. (n.d.). *Itil asset and configuration management*. http://www.bmcsoftware.ca/

guides/itil-asset-configuration-management.html.

Bodeau, D., McCollum, C., & Fox, D. (2018). Cyber threat modeling: Survey, assessment, and representative framework. *HSSEDI, The Mitre Corporation*. https://www.mitre.org/publications/technical-papers/cyber-threat-modeling-survey-assessment-and-representative-framework.

Caralli, R. A., Stevens, J. F., Young, L. R., & Wilson, W. R. (2007). *Introducing octave allegro: Improving the information security risk assessment process* (Tech. Rep.). Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8419.

Case, J., Fedor, M., Schoffstall, M., & Davin, J. (1990). Rfc 1157: Simple network management protocol (snmp). https://www.ietf.org/rfc/rfc1157.txt.

Case, J., McCloghrie, K., & Rose, M. (1996). *S. waldbusser, transport mappings for version 2 of the simple network management protocol (snmpv2)* (Tech. Rep.). RFC 1906, January. https://tools.ietf.org/html/rfc1906.

Case, J., McCloghrie, K., Rose, M., & Waldbusser, S. (1995). *Introduction to community-based snmpv2* (Tech. Rep.). https://tools.ietf.org/html/rfc1901.

Case, J., McCloghrie, K., Rose, M., & Waldbusser, S. (1996). Rfc 1905: Protocol operations for version 2 of the simple network management protocol (snmpv2). *Network Working Group*. https://tools.ietf.org/html/rfc1905.

Casey, T. (2007). Threat agent library helps identify information security risks. https://www.researchgate.net/publication/324091298_Threat_Agent_Library_Helps_Identify_Information_Security_Risks.

Cerf, V. (1989). *Rfc 1052: Iab recommendations for the development of internet network management standards*. https://tools.ietf.org/html/rfc1052. April.

CIS. (n.d.). *Cis benchmarks*. https://www.cisecurity.org/cis-benchmarks/.

COBIT. (2018). *Cobit 2019*. http://www.isaca.org/cobit/pages/default.aspx.

ConfD. (n.d.). *Confd basic | tail-f systems*. https://www.tail-f.com/confd-basic/.

Controls, C. (n.d.). *Cis controls*. https://www.cisecurity.org/controls/.

CVE-2018-0274, C. (2018). *Cisco network services orchestrator arbitrary command execution vulnerability.* https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180606-nso.

CVE-2018-0463, C. (2018). *Cisco network services orchestrator network plug and play information disclosure vulnerability.* https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20180905-nso-infodis.

CVE-2018-16837, R. (2018). *Cve-2018-16837 - red hat customer portal.* https://access.redhat.com/security/cve/cve-2018-16837.

Davin, J., Case, J., Fedor, M., & Schoffstall, M. (1987). Rfc 1028-simple gateway monitoring protocol. *IETF*. https://tools.ietf.org/html/rfc1028.

Delaet, T., Joosen, W., & Van Brabant, B. (2010). A survey of system configuration tools. In *Lisa* (Vol. 10, pp. 1–8). https://www.usenix.org/legacy/event/lisa10/tech/full_papers/Delaet.pdf.

Dierks, T., & Rescorla, E. (2008). Rfc 5246-the transport layer security (tls) protocol version 1.2. *Internet Engineering Task Force*. https://tools.ietf.org/html/rfc5246.

Enns, R. (2006). Netconf configuration protocol-rfc 4741. https://tools.ietf.org/html/rfc4741.

Enns, R., Bjorklund, M., Schoenwaelder, J., & Bierman, A. (2011). Rfc 6241, network configuration protocol (netconf). *IETF*. https://tools.ietf.org/html/rfc6241.

Goddard, T. (2006). *Using netconf over the simple object access protocol (soap)* (Tech. Rep.). https://tools.ietf.org/html/rfc4743.

Hedstrom, B., Watwe, A., & Sakthidharan, S. (2011). Protocol efficiencies of netconf versus snmp for configuration management functions. *University of Colorado, Master Thesis*. https://pdfs.semanticscholar.org/5664/44aa2023ac8cf9910cc33ead8582ace4c9c4.pdf.

ISO, . (1989). *Iso/iec 7498-4:1989 - information processing systems – open systems interconnection – basic reference model.* https://www.iso.org/standard/14258.html.

Jones, G. (2004). *Operational security requirements for large internet service provider (isp) ip network infrastructure* (Tech. Rep.). https://tools.ietf.org/html/rfc3871.

Kohnfelder, L., & Garg, P. (1999). *Microsoft sdl threat modelling*. https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling.

Lear, E., & Crozier, K. (2006). *Using the netconf protocol over the blocks extensible exchange protocol (beep)* (Tech. Rep.). https://tools.ietf.org/html/rfc4744.

Microsoft, T. M. (n.d.). *Getting started - microsoft threat modeling tool - azure | microsoft docs*. https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-getting-started.

Montgomery, D., Sriram, K., Carson, M., & Santay, D. J. (2016). *Advanced ddos mitigation techniques | nist*. https://www.nist.gov/programs-projects/advanced-ddos-mitigation-techniques.

Murugiah, S., & Scarfone, K. (2016). *Sp 800-154, guide to data-centric system threat modeling*. https://csrc.nist.gov/publications/detail/sp/800-154/draft.

NSO. (n.d.). *Cisco network services orchestrator (nso) - cisco*. https://www.cisco.com/c/en/us/solutions/service-provider/solutions-cloud-providers/network-services-orchestrator-solutions.html.

Partridge, C., & Trewitt, G. (1987a). *High-level entity management protocol (hemp)* (Tech. Rep.). https://tools.ietf.org/html/rfc1022.

Partridge, C., & Trewitt, G. (1987b). *High-level entity management system (hems)* (Tech. Rep.). https://tools.ietf.org/html/rfc1021.

Perrin, C. (2008). The cia triad. https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA.

Rescorla, E. (2018). *The transport layer security (tls) protocol version 1.3* (Tech. Rep.). https://tools.ietf.org/html/rfc8446.

Rizos, C. (2016). *Why use netconf/yang when you can use snmp and cli?* https://snmpcenter.com/why-use-netconf/.

Schneier, B. (1999). *Academic: Attack trees - schneier on security*. https://www.schneier.com/academic/archives/1999/12/attack_trees.html.

Schoenwaelder, J. (2002). Simple network management protocol over transmission control protocol transport mapping. https://tools.ietf.org/html/rfc3430.

Schoenwaelder, J. (2003). *Overview of the 2002 iab network management workshop* (Tech. Rep.). https://tools.ietf.org/html/rfc3535.

Sherry, J., Hasan, S., Scott, C., Krishnamurthy, A., Ratnasamy, S., & Sekar, V. (2012). Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, *42*(4), 13–24. https://people.eecs.berkeley.edu/~sylvia/papers/fp150-sherry.pdf.

Shevchenko, N., Chick, T. A., O'Riordan, P., Scanlon, T. P., & Woody, C. (2018). Threat modeling: A summary of available methods. https://resources.sei.cmu.edu/asset_files/WhitePaper/2018_019_001_524597.pdf.

Shostack, A. (2008). Experiences threat modeling at microsoft. In *Modsec@ models*. https://adam.shostack.org/modsec08/Shostack-ModSec08-Experiences-Threat-Modeling-At-Microsoft.pdf.

Stallings, W. (1998). *Snmp, snmpv2, snmpv3, and rmon 1 and 2*. Addison-Wesley Longman Publishing Co., Inc. https://dl.acm.org/citation.cfm?id=521036.

Strom, B. E., Battaglia, J. A., Kemmerer, M. S., Kupersanin, W., Miller, D. P., Wampler, C., ... Wolf, R. D. (2017). *Finding cyber threats with att&ck^{TM}-based analytics* (Tech. Rep.).

Warrier, L. B., & Besaw, L. (1989). Rfc 1095.". *Common Management Information Services and Protocol over TCP/IP (CMOT),"*. https://tools.ietf.org/html/rfc1095.

Wasserman, M. (2011). *Using the netconf protocol over secure shell (ssh)* (Tech. Rep.). https://tools.ietf.org/html/rfc6242.

Wynn, J., Whitmore, J., Upton, G., Spriggs, L., McKinnon, D., McInnes, R., ... Clausen, L. (2011). *Threat assessment & remediation analysis (tara): Methodology description version 1.0* (Tech. Rep.).

Ylonen, T., & Lonvick, C. (2005). *The secure shell (ssh) transport layer protocol* (Tech. Rep.). https://tools.ietf.org/html/rfc4253.

Ylonen, T., & Lonvick, C. (2006). Rfc 4254 the secure shell (ssh) connection protocol. https://tools.ietf.org/html/rfc4254.

All URLs are valid as of the publication date of this document.