Bachelor's thesis

Information and Communications Technology

2023

Tommi Leiritie

Automated Hardening of Linux Infrastructure

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information and Communications Technology

2023 | 33 pages

Tommi Leiritie

Automated Hardening of Linux Infrastructure

Automated hardening of virtualized and bare-metal computer systems is essential in large high security environments. Unlike Group Policies in a Microsoft Windows Domain, a built-in mechanism to automate the configuration of multiple hosts in a Linux environment does not exist, so an alternative solution is needed. Configuration management solutions provide the required consistency and determinism to ensure the technical security of Linux operating systems.

In this thesis, different hardening guidelines were studied, and the hardening of Linux operating system is presented according to the guidelines developed by the Center for Internet Security community. Also, HashiCorp Vagrant and Red Hat Ansible configuration management solutions are introduced.

The objective of the thesis was to develop an Ansible playbook which can be used to harden multiple Linux hosts and distributions simultaneously according to the Center for Internet Security Benchmarks and generate a report of the hardening actions.

The Ansible playbook was developed using HashiCorp Vagrant which was used to create and dispose of multiple Linux virtual machines. The final testing of the Ansible playbook was executed in a reference environment which consisted of virtualized and bare-metal Linux hosts.

According to the test results, the Ansible playbook meets the objectives and is a maintainable product. The produced Ansible playbook substantially reduces the need for manual hardening.

Keywords:

Hardening, Configuration management, Linux, Ansible, CIS

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja Viestintätekniikka

2023 | 33 sivua

Tommi Leiritie

Automatisoitu Linux-infrastruktuurin koventaminen

Automatisoitu työasemien, palvelimien ja sovellusten koventaminen on välttämätöntä suurissa korkean turvallisuustason ympäristöissä. Microsoft Windows -toimialueessa automaatio voidaan toteuttaa ryhmäkäytänteillä. Linux-ympäristöissä vastaavaa sisäänrakennetua mekanismia ei ole, joten vaihtoehtoinen ratkaisu on tarpeen. Konfiguraationhallintaratkaisut tuottavat vaadittavan johdonmukaisuuden ja determinismin Linux-käyttöjärjestelmien teknisen turvallisuuden takaamiseksi.

Työssä analysoidaan erilaisia kovennusoppaita ja esitellään Center for Internet Security -yhteisön kovennusoppaan mukaiset Linux-käyttöjärjestelmäkovennukset, sekä HashiCorp Vagrant ja Red Hat Ansible -konfiguraationhallintaratkaisut.

Työn tavoitteena oli tuottaa Ansible-pelikirja, jolla voidaan suorittaa Center for Internet Security -yhteisön tuottamia kovennuskomentosarjoja samanaikaisesti usealle Linux-jakelulle ja isäntäkonelle, sekä tuottaa raportti kovennustoimenpiteistä.

Ansible-pelikirja kehitettiin käyttämällä HashiCorpin Vagrant -työkalua, jolla luotiin ja hävitettiin useita Linux virtuaalikoneita. Ansible-pelikirjan lopputestaus toteutettiin referenssiympäristössä, joka koostui virtualisoiduista ja fyysisistä Linux-isäntäkoneista.

Testauksen perusteella Ansible-pelikirja täyttää asetetut tavoitteet ja on ylläpidettävä kokonaisuus. Ansible-pelikirja vähentää merkittävästi manuaalisen koventamisen tarvetta.

Asiasanat:

Koventaminen, Konfiguraationhallinta, Linux, Ansible, CIS

Contents

Figures

Code

List of abbreviations

AAA	Authentication, Authorization, and Accounting
API	Application Programming Interface
AWS	Amazon Web Services
BIOS	Basic Input Output System
CAT	Compliance Assessment Tool
CIS	Center of Internet Security
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DISA	Digital Information Security Agency
DNS	Domain Name System
DSC	Desired State Configuration
GCP	Google Cloud Platform
GPO	Group Policy Object
MAC	Mandatory Access Control
MAC	Media Access Control
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NTP	Network Time Protocol
HIPAA	Health Insurance Portability and Accountability Act
IaC	Infrastructure as Code

IP	Internet Protocol
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
OVAL	Open Vulnerability and Assessment Language
PCI DSS	Payment Card Industry Data Security Standard
QEMU	Quick Emulator
RADIUS	Remote Authentication Dial-In User Service
RHEL	Red Hat Enterprise Linux
RPC	Remote Procedure Call
SCAP	Security Content Automation Protocol
SSH	Secure Shell
STIG	Security Technical Implementation Guide
UEFI	Unified Extensible Firmware Interface
USGCB	United States Government Configuration Baseline
WMI	Windows Management Instrumentation
WSL	Windows Subsystem for Linux
XCCDF	The Extensible Configuration Checklist Description Format
YAML	Yet Another Markup Language

1 Introduction

As technical vulnerabilities are discovered on a day-to-day basis, operating systems and applications need to be protected against exploitation of both disclosed and undisclosed vulnerabilities. In 2021 the most common infection vector was exploitation of vulnerabilities which covered 37% of identified incidents (Mandiant 2022). A Security Information and Event Management (SIEM) solution can be tuned with high quality detection rules and threat intelligence feeds to detect known security threats. The SIEM can then be configured to command an Intrusion Prevention System (IPS) to prevent exploitation, but the challenge lies in defending against the unknown. Thorough hardening of operating systems and applications supplementing and enforcing a well-defined security policy is an effective strategy defending against undisclosed vulnerabilities or 0-day vulnerabilities. This is achieved by reducing both the technical and operational attack surface.

Large IT-environments require automation to reduce the workload of maintaining the environment. In Microsoft Windows environments where Active Directory Domain Services are present, Group Policy management enables centralized, automated, and a hierarchical way to manage the configuration of the operating system, application settings and user rights (Microsoft 2022a). Typically, Active Directory Domain Services are set up in an enterprise setting enabling centralized user account management for the administrators. Often PowerShell scripting is also used to supplement the Group Policy Objects (GPO) and further automate tasks. PowerShell scripts using the PowerShell remote management capabilities such as Windows Management Instrumentation (WMI) or Remote Procedure Call (RPC) can be run on the Domain Controller or the domain administrator’s workstation and the results of the remote management actions can be reported back to the manager machine (Microsoft 2022b).

In Linux environments Remote Authentication Dial-In User Service (RADIUS) can be used to provide Authentication, Authorization, and Accounting (AAA)

services. Active Directory services can be utilized for authentication and authorization using Lightweight Directory Access Protocol (LDAP) or Kerberos authentication (Red Hat 2022). None of these solutions provide a capable method of managing the configuration of Linux operating systems or application settings. As Linux environments lack a built-in solution to manage configuration, user accounts and user rights in large environments, 3rd party tools are required.

Configuration management solutions such as Ansible by Red Hat and Vagrant and Terraform by HashiCorp provide centralized, consistent, and deterministic tools for administrators to provision and maintain large Linux environments. In the context of provisioning and managing IT infrastructure through a configuration management solution, the term commonly used to describe this process is Infrastructure as Code (IaC). This means describing the desired state of the infrastructure in structured text files. When used in conjunction with version control software such as Git, modifications to the configuration can be tracked and managed.

The objectives of this thesis are to research different hardening guidelines and develop an automated solution to harden large high security Linux environments. The requirements for the automation are to enable granular control over hardening, compile a report of the implemented hardening and to be simple and maintainable. The requirements for the hardening are to be sufficiently comprehensive to meet certain security standards.

During the thesis work, HashiCorp Vagrant was used to repeatedly provision and dispose of a small Linux environment that was used to develop an Ansible playbook. The developed playbook enables a user to harden multiple virtualized and bare-metal Linux hosts simultaneously according to hardening Benchmarks developed by the Center of Internet Security (CIS) community. To use the playbooks the user is required to have access to the CIS Build Kits as the playbook itself does not contain any hardening actions to respect the end user agreement of the CIS Build Kits. The CIS Build Kits generate comprehensive logs that document the specific actions taken for configuration

recommendations. These logs include details on remediated recommendations, skipped recommendations deemed not applicable, failed recommendations due to errors, and recommendations that were already configured according to the Benchmark. Additionally, the logs identify configuration recommendations that need manual remediation. The developed playbook collects the logs to the Ansible controller for detailed reviewing of the hardening actions on each individual host and a Python script can be used to compile a summary report.

To ensure that the Ansible playbook achieves the objectives and meets the given requirements, the playbook was tested in a reference environment consisting of virtualized and bare-metal Linux hosts. The testing started by ensuring the configuration changes made by the hardening script do not prevent the usage of the Linux hosts as intended. Then the improvement of CIS Benchmark compliance was analyzed using the collected logs and finally, the compliance was verified and reported using the CIS Compliance Assessment Tool. After thorough testing, the developed hardening automation was introduced into production environments.

The thesis is structured as follows. Chapter 1 introduces the researched subjects of the thesis and the methodology used in the development and testing of the automation. Chapter 2 discusses the theory of hardening and introduces different hardening methods and guidelines. Chapter 3 reviews the process of configuration management and introduces the configuration management solutions used during this thesis work. Chapter 4 presents the development environment and tools used in the thesis work and discusses the structure of the developed Ansible playbook. Chapter 5 presents the testing environment and methodology and discusses the test results. Chapter 6 assesses the fulfillment of the objectives and presents recommendations for future improvements.

2 System Hardening

The process of hardening computer systems is a component of a comprehensive information security strategy. Hardening reduces the potential attack surface by ensuring the computer systems are up to date, nonessential services are turned off and in general, the systems are configured correctly and according to the best security practices such as least privilege access principle etc. In a comprehensive information security strategy procedures and policies are in place and the technical hardening measures supplement or enforce the procedures and policies.

Hardening a computer system thoroughly involves several steps to form layers of protection (NIST SP 800-152 2015). This involves selecting hardware from a trusted manufacturer that supports the required security features and hardening the UEFI/BIOS-settings. If the supply chain or firmware settings are not considered, the effectiveness of operating system and application hardening are diminished.

Many hardening guidelines, checklists and benchmarks have been developed by various communities, agencies, and enterprises. Also, standardized protocols, file formats and specification languages have been developed including the Security Content Automation Protocol (SCAP), Open Vulnerability and Assessment Language (OVAL) definitions and the Extensible Configuration Checklist Description Format (XCCDF) benchmarks. (Henttunen 2018)

2.1 Microsoft Windows Hardening

In a Microsoft Windows environment, the operating system and application hardening can be achieved using Group Policies. In an enterprise setting, where Active Directory Domain Services are present, Group Policies enable centralized, automated, and a hierarchical way to manage the configuration of the operating system, application settings and user rights (Microsoft 2022a). Different hardening guidelines, checklists and benchmarks have been

developed for Microsoft Windows; they are usually released as GPOs which can be deployed in a Windows Domain via the Domain Controller. Well established hardening guidelines available as GPOs include the Microsoft Security Compliance Toolkit Security Baselines (Microsoft 2023), the Digital Information Security Agency (DISA) Security Technical Implementation Guides (STIG) (DISA 2023), The National Institute of Standards and Technology (NIST) United States Government Configuration Baseline (USGCB) (NIST 2020) and the CIS Benchmark Build Kits.

On some Linux distributions such as Red Hat Enterprise Linux (RHEL) operating system hardening can be performed during installation. For common Linux distributions that do not provide such functionality, hardening methods have been developed by various organizations.

2.2 Hardening Guidelines

The CIS Benchmarks are developed for a wide range of organizations. In contrast, the NIST USGCB and DISA STIGs are designed specifically for the US Department of Defense and government agencies. CIS Benchmarks use roles and levels to distinguish between different hardening scenarios while USGCB and DISA STIG use control IDs to identify specific security configurations. The USGCB and DISA STIG are mapped to NIST frameworks such as NIST Cybersecurity Framework and NIST Special Publication 800-53 Security and Privacy Controls for Information Systems and Organizations (NIST SP 800-70r4 2018). The CIS Benchmarks are mapped to various industry standards such as NIST Cybersecurity Framework, Payment Card Industry Data Security Standard (PCI DSS) and Health Insurance Portability and Accountability Act (HIPAA) (Center of Internet Security n.d.)

CIS releases the Linux Benchmark Build Kits as shell scripts to the CIS community members while the DISA STIG and NIST USGCB are released in the XCCDF format. A hardening script can be generated from a XCCDF format hardening guideline using the OpenSCAP Workbench application, but these

scripts lack the granularity of the CIS community developed scripts which implement four levels of hardening and provide a mechanism for excluding even single hardening actions. The OpenSCAP tools provide a way to automate the hardening of Linux hosts according to the security guidelines but lack a reasonable way to manage the general settings of the hosts. This was one of the reasons in this thesis work to choose the combination of a configuration management platform and the CIS Build Kits to implement the hardening automation.

2.3 CIS Benchmarks

The CIS Benchmarks are recommendations of configuration to achieve technical hardening against cyber-attacks. The CIS Benchmarks are developed by the CIS community which consists of more than 12,000 IT security professionals and volunteers. The strength of the CIS Benchmarks is the development process and the community which consists of subject matter experts, technology vendors, public and private community members and academics across from different industries to debate and agree on the recommended configurations. Also, the Benchmarks can be a good source to learn as the recommendations are broken into description which summarizes the recommendation, rationale which explain why this recommendation is important, impact which describes the benefits of implementing the recommendation, audit which describes how to check compliance to the recommendation and remediation which provides the technical steps how to implement the recommendation. The Benchmarks are available for more than 25 vendor product families such as different cloud providers, mobile and network devices, operating systems, and desktop and server software. For Linux operating systems there are distribution specific Benchmarks for widely used Linux distributions such as CentOS, Debian, Fedora, Oracle Linux, RHEL, SUSE and Ubuntu. (CIS 2023a)

CIS also offers hardened virtual machine images on Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure and Oracle Cloud

marketplaces. The CIS Hardened Images are virtual machine images that have been pre-configured according to the CIS Benchmarks. (CIS 2023b)

2.4 CIS Distribution Independent Linux Benchmark

The CIS Distribution Independent Linux Benchmark is a distribution agnostic hardening guideline which can be applied to many different Linux distributions (CIS 2019). As the goal of this thesis is to produce automation to harden different Linux distributions using a single playbook, the Distribution Independent Benchmark is reviewed in this section to give an overview of steps taken in the process of hardening a generic Linux host.

The CIS Distribution Independent Linux Benchmark divides the process of hardening a Linux host into the following sections:

- Initial setup
- Service configuration
- Network configuration
- Logging and auditing
- Access, authentication, and authorization
- System maintenance

The initial setup consists of hardening steps that may be difficult to perform on a host that has been already installed. The steps are filesystem configuration, software update configuration, filesystem integrity checking, secure boot settings, additional process hardening, mandatory access control (MAC) and configuring warning banners.

The service configuration section ensures that unneeded services and service clients are not present in the system. The required services and clients that are present on the system according to the host’s role are configured according to the best practices and the integrity of their configuration is protected.

The network configuration section checks that the network parameters are configured in a way that unexpected network behavior is prevented such as a workstation routing network traffic. Also, uncommon, and unneeded network protocols and functionalities are disabled, and the host firewall is configured.

The logging and auditing section configures the system to log and audit the system events according to the best practices and to prevent tampering of the logs. Integrity, correctly configured retention, and logging of meaningful events is vital for efficient system monitoring both in performance and security perspective.

The access, authentication and authorization section ensure that both local and remote access is secured with strong cryptographic algorithms and authentication policies. Also measures to prevent the escalation of privileges by a non-privileged user are taken.

The system maintenance section checks that system file permissions and user, and group settings are configured correctly. This includes checking ownership and read, write and execution permissions of various directories and files both in system files but also user home directories.

3 Configuration Management

Configuration management is a process for maintaining computer systems, servers, and software in a desired, consistent, and reproducible state. It is a way to make sure that a system performs as it is expected even when configuration changes are made over time. (Red Hat 2019). The use of a configuration management solution eliminates manual configuration of single hosts. The desired state of the infrastructure is usually described in structured text documents. In contrast to traditional scripting where tasks are described in scripting languages such as Bash or PowerShell the configuration management software is responsible for performing the tasks according to the description. This method of describing the infrastructure as code also provides the ability to keep track of the current and past configuration when used in conjunction with version control software such as Git. It could be argued that the actual configuration files of the hosts could be kept under version control, but this would require keeping track of many more files than when using configuration management software.

As virtualization, containerization and cloud computing have become the leading solution in managing infrastructure, tools specialized in provisioning such resources have been developed. Such tools include HashiCorp Vagrant and Terraform, both focused on provisioning virtual machines and cloud computing resources. Vagrant is typically used in conjunction with Type-2 hypervisor such as Oracle VirtualBox to consistently provision virtual machines for testing and development purposes. In contrast, Terraform is typically used to provision virtual machine and container infrastructure on a Type-1 hypervisor such as VMWare ESXi or on a cloud computing platform such as Microsoft Azure or AWS. The main differences between HashiCorp Vagrant and Terraform is that Vagrant pulls the virtual machine image or a “Vagrant Box” from Vagrant Cloud and is focused on managing development environments while Terraform uses virtual machine templates on the virtualization platform and is a tool for building production infrastructure.

3.1 Vagrant

Vagrant is a command line utility for managing the lifecycle of virtual machines. Vagrant uses a structured text file called Vagrantfile to describe the virtual machines the user wishes to set up. Vagrantfiles use Ruby syntax, but prior Ruby knowledge is not needed to successfully create and modify Vagrantfiles. (HashiCorp 2022). Vagrant improves productivity as setting up a development environment can be as simple as acquiring a Vagrantfile and invoking a single command “vagrant up”. For development teams this enables consistency across the developer’s environments when the development environments are created with the same Vagrantfile. Also disposing of the environment created with Vagrant is as simple as invoking a single command “vagrant destroy”. Usually even for the simplest local tests utilizing a single generic virtual machine on a laptop, it is faster to create and dispose of using Vagrant than installing it by hand using an installation medium.

Code snippet 1 Simple Vagrantfile shows the simplest Vagrantfile possible. In the example, configuration version is defined as version 2 and Vagrant is directed to provision a single virtual machine with Debian 10 operating system, default central processing unit (CPU), memory and hard disk resources and connected to the VirtualBox default network. Vagrant uses Oracle VirtualBox as the default virtualization provider.

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/buster64"
end
```

Code 1. Simple Vagrantfile.

In addition to the default virtualization provider Vagrant officially supports multiple different providers, VMWare Fusion and Workstation, Docker, and Microsoft Hyper-V. The Vagrant community also maintain various custom providers such as QEMU (HashiCorp 2022)

To create a virtual machine, Vagrant pulls a “Vagrant Box” from the Vagrant Cloud. Vagrant Cloud is a service provided by HashiCorp which hosts the

Vagrant boxes created by HashiCorp and community contributors. The public Vagrant box catalogue contains packages of most major operating systems and many specialized boxes with commonly used configurations such as popular web application stacks. Vagrant Cloud also has paid subscription plans which enable the user to create and host private Vagrant Boxes on the platform. (HashiCorp 2020)

Vagrant enables the user to provision the virtual machines during the setup process. The user has access to multiple provisioning options from using shell scripts to using a configuration management solution such as Ansible. However, in this thesis the goal was to develop automation to harden infrastructure which does not use Vagrant so in the development environment minimal provisioning was used. Vagrant also enables the user to specify the virtual hardware of the virtual machines. This includes specifying the virtual network adapter types and virtual network connections, specifying the amount of CPU cores and maximum usage of the given cores and the size of the virtual machine's virtual memory and hard disks. (HashiCorp 2022)

3.2 Ansible

Ansible is an agentless configuration management platform developed by Red Hat. Ansible uses Secure Shell (SSH) connection by default to interact with Linux hosts and the tasks performed by Ansible are defined in YAML-files called Playbooks. Ansible expects a certain directory and file structure for the playbook to work.

The main components of an Ansible Playbook are the top-level playbook file, sub-playbooks, the inventory file, the roles directory containing directories and files for the roles and the variable directories. The top-level playbook is used to import sub-playbooks which enables to run the whole automation against all the hosts defined in the inventory file. (Red Hat 2021)

The sub-playbooks are structured by host role, i.e., web servers and database servers. This enables the user to run the automation against a subset of hosts

defined in the inventory file providing the first level of granularity. The sub-playbooks contain definitions of which hosts or host groups from the inventory file are addressed, and which roles will be applied to the given hosts. It is good practice to name the sub-playbooks with corresponding names as the host groups in the inventory file. (Red Hat 2021)

The inventory file contains the information of the hosts in the infrastructure and how the hosts are being grouped. The inventory file can be written in initialization (INI) file like format or in YAML-format. In general, the INI-format is more suitable for smaller environments as the syntax is simpler than the YAML-format. In the other hand the YAML-format is suited for more complex use-cases as the YAML-format inventory files can be generated dynamically and has more features altogether. (Red Hat 2023b)

The roles directory contains a directory for each role and their files and sub-directories. A role directory must contain at least one of the following sub-directories: tasks, handlers, defaults, vars, files, templates or meta. The tasks folder contains the main list of the tasks for the role, and it is common practice to have the individual tasks in task specific YAML-files. Handlers are special kinds of tasks which are run only when a change is made on a host. For example, restarting a service only when the configuration has been changed by Ansible. Defaults and vars directories both contain files defining role specific variables. Defining variables in the roles enables creating modular roles i.e., by defining Linux distribution specific variables for a role. Files and templates directories contain files that are deployed to the host. The difference between files and templates is that files are deployed unchanged, and templates contain variables that are filled in during the deployment. The meta directory contains metadata about the role such as dependencies to other roles, author, version, and license information. (Red Hat 2021)

The variable directories within the top-level of a playbook are group_vars and host_vars directories which contain host and group specific variables. These exist to help the playbook developer to organize the variables instead of writing the host and group variables into the inventory files. Defining variables in the

inventory file or the variable directories separates the variables from the playbook's functionality which enables the sharing of the playbook functionality without sharing details of the inventory. Ansible Vault is a tool to further protect the confidentiality of sensitive data such as credentials in the inventory or variables. Ansible Vault enables the user to encrypt variables and files using a password. (Red Hat 2023a)

4 Developing the automation

The development of the playbook was executed on a Windows 10 Enterprise workstation. To develop the playbook, Windows Subsystem for Linux (WSL), Microsoft Visual Studio Code, HashiCorp Vagrant, and Microsoft Hyper-V were used. The development environment was designed to require minimal adjustments and provide a straightforward workflow during the development process. By using time to plan and adjust the development environment and tools before the development process, the productivity and cognitive ergonomics of the actual development work can be improved considerably. By using linters and static code analysis tools, interruptions to the development work caused by the need to find semantical and syntactical errors can be reduced.

As the work's objectives and technologies to be used were clear, the development process was straightforward. First, the steps to achieve the objective were mapped: The Build Kit archive needs to be transferred securely to the hosts, the archive needs to be extracted, the main script needs to be executed and the host hardening logs need to be fetched to the Ansible controller. Then, techniques to implement the steps with Ansible were researched and Ansible modules to be used were selected. The steps were then implemented into Ansible tasks using the modules and techniques researched. After the core functionalities of the playbook were developed, sections that need to be able to change during playbook execution identified and converted into variables in the task definitions. These sections were mainly file paths as the different Build Kits have slightly different file and directory names. Finally, the structure of the inventory was designed to provide a logical way to control the variables and implemented.

4.1 Development environment

The Debian 11 Bullseye WSL image was used as the Ansible controller node. Ansible and its dependencies such as various Python 3 libraries were installed via the Debian package manager. Also, the Visual Studio Code Server was installed to enable connecting the Visual Studio Code editor to the WSL container. By default, WSL connects the container to a “Special” WSL Internal network Hyper-V virtual switch which enables communication to the host’s network using network address translation (NAT) as illustrated in Figure 1. Normally Hyper-V Internal network virtual switches only allow communication between virtual machines and the host but not to the host’s network. The WSL container was operated using the Visual Studio Code terminal as this provides convenience by enabling working within a single application.

The Visual Studio Code editor was used to write the Ansible playbook’s various YAML-files such as the task descriptions, variable files, and the inventory. Visual Studio Code can be extended to resemble an integrated development environment (IDE). In this development environment a WSL extension developed by Microsoft was used to enable connecting the Visual Studio Code editor to the Visual Studio Code Server running in the WSL container. Also, Ansible and YAML language support plugins developed by Red Hat were used to constantly run static code analysis for ensuring syntactical and structural correctness of the project files.

Hyper-V was used as the virtualization platform to host virtual machines as targets for the hardening automation. As the project required constant creation and deletion of virtual machines Hyper-V was selected because it is well supported by Vagrant and the network setup to connect to the WSL container was straightforward. A Hyper-V External network virtual switch was created for the virtual machines to share the physical network interface of the host for direct access to the host’s network as illustrated in Figure 1. This type of virtual network setup is often called a network bridge. The network bridge setup

allowed the WSL container to access the virtual machines through the NAT and for the virtual machines to access different package repositories on the internet.

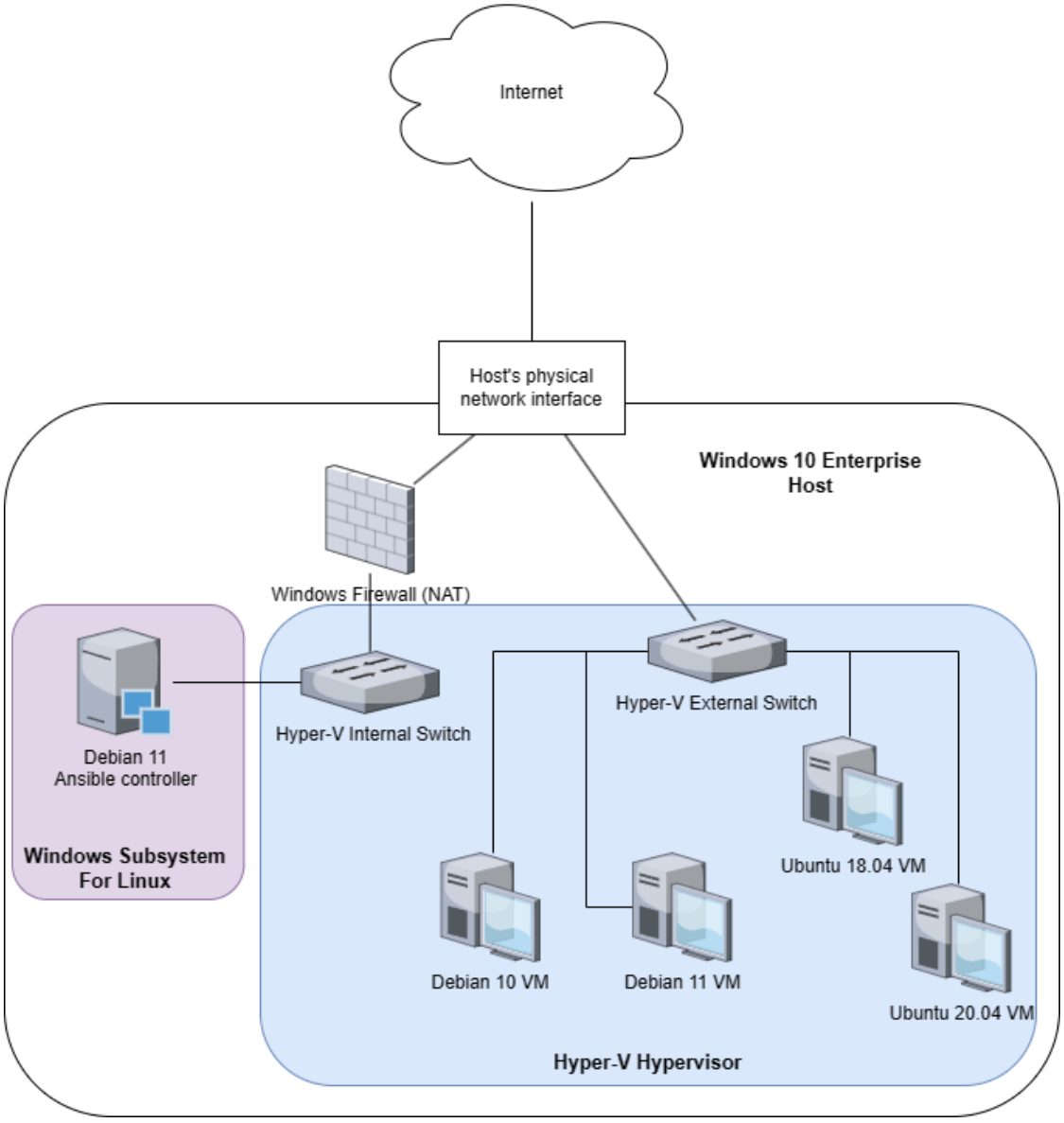

Figure 1. Diagram of the development environment.

Vagrant was an essential tool to create and dispose of the multi-machine virtual environment consistently and quickly for developing the playbook. The virtual environment consisted of four virtual machines using Debian 10, Debian 11, Ubuntu 18.04 LTS and Ubuntu 20.04 Linux operating systems. Vagrant was configured to allocate 4GB of virtual memory and 2 CPU cores for the virtual

machines to ensure enough computing resources are available to the virtualization host. The Media Access Control (MAC) addresses of virtual machines were configured within the Vagrantfile to ensure the virtual machines would always be assigned the same Internet Protocol (IP) addresses by the Dynamic Host Configuration Protocol (DHCP) server within the development network. Vagrant was also configured to not insert its own SSH public key to the virtual machines. Instead, a script was used to insert the public part of a pre-generated keypair to ensure access to the virtual machines via SSH for Ansible on the WSL Ansible controller.

Code snippet 2 Development Vagrantfile shows an example of the configuration for a single machine used in the development environment. For the three other machines the configuration was identical from the *'config.vm.define "debian10" do |v|'* to that block's closing '*end*' statement. The operating system versions, machine names and the MAC-address were unique for each virtual machine configuration.

```
Vagrant.configure("2") do |config|
  config.ssh.insert_key = false
  config.vm.define "debian10" do |v|
    v.vm.box = "generic/debian10"
    v.vm.network "public_network", bridge: "Bridge"
    v.vm.provider "hyperv" do |h|
      h.mac = "00:11:22:33:44:55"
      h.vmname = "debian10"
      h.maxmemory = 4096
      h.cpus = 2
    end
  end
  # retracted for redundancy
end
```

Code 2. Development Vagrantfile.

4.2 Structure of the playbook

The hardening functionality was achieved by developing a common role that has tasks to check if the target hosts already contain the CIS Build Kit archive file and if not, transfer the archive to the hosts. Then a task to check if the

archive has already been extracted into a directory and again, if not then extracts the archive. After the role has ensured that the Build Kit is available on the target hosts, a task populates the exclusion list in the Build Kit directory from variables set in the inventory. Before execution of the main script a task changes the permissions of the main script file of the Build Kit, so the script is executable, and another task executes the script. The main script executes the hardening actions and generates logs of the results in the Build Kit directory. Lastly a task searches for the most recent log files and passes this information to a task that fetches the most recent log file to the Ansible controller node. On the Ansible controller node a local Python script is executed which assembles the individual host results into a summary report. Tags were assigned to the tasks to enable the user to only do the checks, extraction, transfer, execution, or log fetching and summarizing for finer granularity of the playbook.

The inventory file's structure was designed to control which level of hardening and which Build Kit will be applied to which hosts. Top-level groups Level 1 Server, Level 2 Server, Level 1 Workstation, and Level 2 Workstation were defined. Distribution specific groups were defined as children to the top-level groups. The individual host information such as hostnames and IP addresses were defined in the distribution specific groups. The host's assignment into a group controls the group variables assigned to the host during playbook execution. The top-level group assigns a hardening profile variable to the host which is used during the playbook execution to answer to the Build Kit main script's prompt which asks for consent and level of hardening to be applied. The distribution specific groups assign variables for the name of the Build Kit archive, the Build Kit directory path, the name of the Build Kit main script and the path to the Build Kit log files. The variables for excluding hardening actions can be defined to either the distribution specific groups variables or individual host variables. This functionality improves the usability of the playbook both in homo- and heterogenous environments. For Linux distributions that do not have a distribution specific Build Kit developed by the CIS community, a generic Linux group was defined. Assigning a host to this group assigns the CIS Distribution Independent Benchmark's Build Kit to be run on the host.

5 Testing the automation

The developed automation was tested thoroughly as hardening has a potential to impair or even prevent legitimate usage of hosts and services. The testing consisted of deploying the automation in a reference environment, assessing the usability of the environment after the hardening, and finally assessing and verifying the compliance according to the CIS Benchmarks.

Even though the direct numerical improvement of compliance was substantial, it is important to identify that quantity does not compensate for quality. Different hardening actions have a different impact on the actual effectiveness of the hardening. Also, in a comprehensive security strategy, many different security controls are in place along hardening including both technical and operational security controls.

5.1 Testing environment

The test environment was a reference environment of a production environment consisting of several Linux servers and workstations and networking equipment such as switches and firewalls as shown in Figure 2. The Linux hosts were both bare-metal and virtualized. The networking equipment was used to provide the networking infrastructure for the communications between the Linux-hosts and the Ansible-controller. The Linux servers were providing numerous services such as off-line Linux repository mirrors, Domain Name System (DNS) and Network Time Protocol (NTP) services along with the customer’s proprietary services. The Linux workstations were generic workstations, and SSH-service was enabled on them so that the Ansible controller could connect to the workstations.

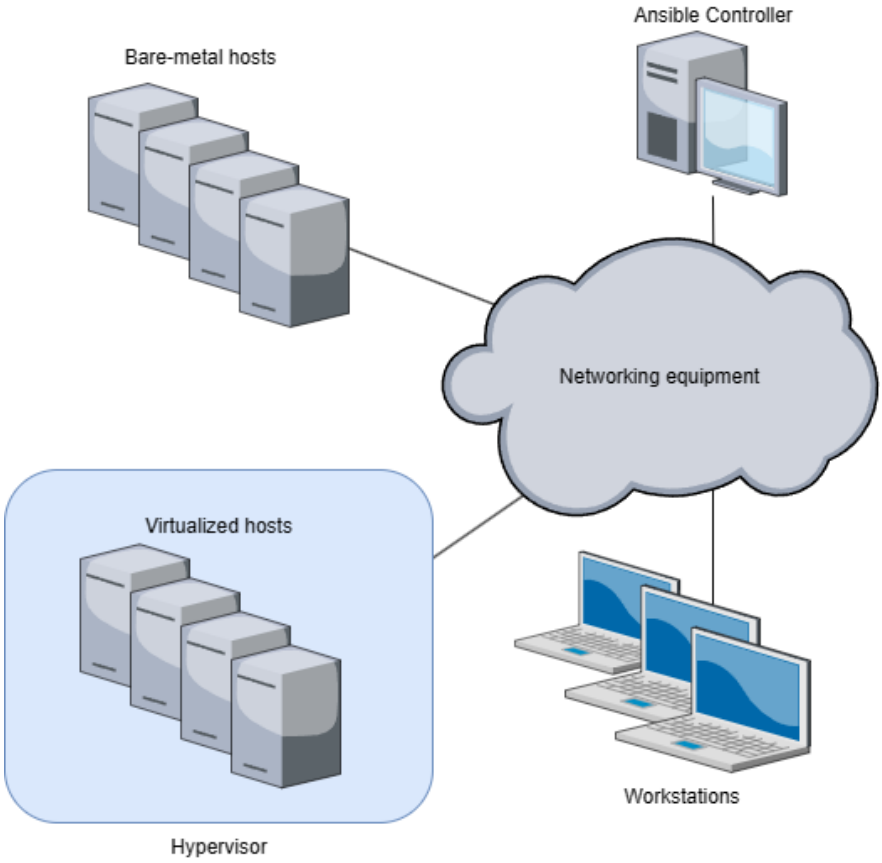

Figure 2. Testing environment principle

5.2 Post-hardening usability

The effects of the hardening recommendations were evaluated before running the Ansible playbook on the test environment. Hardening recommendations that would uninstall, disable, or impair a service provided by a server or an application used on a workstation were excluded. The planning and evaluation of applicable hardening actions is an essential step in reducing the possibility of disrupting the functionality of a host.

After running the Playbook in the test environment, the desired functionality of the services and applications were tested by hand, checking monitoring systems, and interviewing users. The most essential core services and functionalities were tested by hand immediately after the playbook was run to notice any catastrophic failures caused by the hardening. Only a few problems

with a host firewall occurred but these were located and fixed quickly. The monitoring systems were inspected for an extended period to notice any new errors and warnings, and none were present. After the technical functionality was ensured and the users had had time to use and test the hardened systems, the users were interviewed. In general, the systems functioned as intended. The most common complaint was regarding the inability to use USB storage on Level 2 workstations as recommendation number 1.1.10 disables USB storage. This complaint is more of a policy matter than a fault in the automation or the CIS Build Kits.

5.3 Compliance according to logs

The logs were analyzed after the hardening to evaluate the improvement of compliance to the CIS Benchmarks. The percentage of improvement was calculated by dividing the number of successfully remediated recommendations by the number of applicable recommendations. This gives an estimate of how much the potential attack surface has been reduced.

Across all the hosts hardened, the compliance to the CIS Benchmarks was improved by 35 percentage points. On serves where Level 1 hardening was applicable, the compliance was improved on average by 49 percentage points and on servers requiring Level 2 hardening by 36 percentage points. On Level 1 workstations the compliance was improved by 25 percentage points and on Level 2 by 30 percentage points. Some recommendations were not remediated by the CIS Build Kit scripts as the automatic remediation was not possible or the recommendation was not applicable because of the host's hardware or software configuration. The recommendations requiring manual remediation were mainly from the CIS Benchmark's Initial Setup section but also included recommendations that have potential lock legitimate users out of the host during hardening. The reason for this is that for example, recommendations regarding partitions and partition options cannot be remediated automatically as if the host's hard drive is not partitioned in a particular way and the automation should not set or change passwords without the user's consent.

5.4 CIS Compliance Assessment Tool

The CIS community develops and maintains a Compliance Assessment Tool (CAT) to assess and report compliance of hosts according to the CIS Benchmarks. CIS CAT functions in a similar way to a vulnerability scanner such as Nessus. The user specifies information of the hosts for CIS CAT to connect to and selects which Benchmarks are used in the assessment. After the configuration is done, the assessment is run and a report in a format requested by the user is generated.

A CIS CAT report consists of a summary, profiles used, assessment results and assessment details. The summary is a scored list of hardening sections and their subsections showing the number of passed and failed recommendations. The profiles section shows which Benchmark profiles were used on which host in the assessment as several Benchmarks can be applied to a single host, for example operating system and application hardening. The assessment results show a more list of individual hardening actions passed or failed. Finally, the assessment details show the full Benchmark details for each hardening action. As the report can be configured to show only failed/manual recommendations, it is a great tool for generating a “to-do” list for achieving ever higher compliance as the assessment details include instructions to implement the recommendation.

CIS CAT Pro was used in the reference environment to verify the information given by the collected host logs and the results between the logs and the CIS CAT report were identical. The results of the report and usability testing were assessed, and a decision was made to introduce the developed automation into production environments according to a separate plan.

6 Conclusions and recommendations

In this work, a functioning automation to harden multiple Linux hosts was developed. The resulting Ansible playbook fulfilled the objectives set by the client which were enabling granular control over hardening, compiling a report of the implemented hardening and being simple and maintainable. The granularity was achieved by implementing tags in the Ansible playbook for controlling the automation and utilizing the granular control the CIS Build Kits provided. The reporting functionality was implemented successfully, and the quality of the reports was verified with the CIS Compliance Assessment Tool. The simplicity and maintainability of the Ansible playbook were achieved by decoupling the hardening from the automation. By implementing the hardening automation this way, when the CIS Build Kits are updated, the automation does not need modifications. Also, by strictly following Red Hat's documentation and guidelines in the Playbook structure, the maintainability and modularity of the Playbook were ensured.

The importance of a reference environment for testing hardening before deploying into production was emphasized, as high levels of hardening have the potential to prevent the intended usage of a host or a service. Even though comprehensive hardening is essential when reducing technical attack surface, the hardening actions cannot impair the usability of a host or a service. By comprehensively testing the intended hardening actions in a sufficiently realistic reference environment, the number of problems when deploying to production is substantially reduced.

As there are hardening recommendations that cannot be executed automatically, a conscious decision needs to be made on how the remaining hardening recommendations are considered. The environment's security requirements need to be taken into consideration to evaluate whether the remaining hardening recommendations need to be implemented manually, mitigated by other security controls or the risk can be accepted as residual risk. The compliance achieved by automation can be improved by preparing the

environment to support the more demanding hardening recommendations such as the recommendations in the CIS initial setup section. This would consequently require more work and, therefore, is both a risk management and a business decision.

To improve the effectiveness and usability of the developed Playbook, several ideas were noted. By implementing tasks to configure firewall rules according to variables set in the inventory, the amount of manual configuration would be reduced. Also, a well-configured firewall reduces the network attack surface and provides visibility into network events. As CIS Build Kits for services and applications are developed, it would be relatively straightforward to implement variable and task structures to also utilize these Build Kits. This would further improve the overall hardening coverage. To automatically remediate the manual recommendations that require setting secrets such as various system passwords, Ansible Vault could be utilized. This would require developing tasks for the individual recommendations and would increase the complexity of the Playbook. To improve the user experience of the automation, a mechanism for displaying the progress during the hardening could be developed. When hardening hosts to CIS Level 2, the execution of the hardening script takes several minutes. This could lead to a misconception that the automation has stopped working.

References

CIS Distribution Independent Linux 2019. Distribution Independent Linux Benchmark. Referenced on 7. April 2023. https://www.cisecurity.org/cis-benchmarks

Center for Internet Security. 2023a. CIS Benchmarks. Referenced on 20. April 2023 https://www.cisecurity.org/cis-benchmarks-overview_new

Center for Internet Security. 2023b. CIS Hardened Images FAQ. Referenced on 9. May 2023. https://www.cisecurity.org/cis-hardened-images/cis-hardened-images-faq

Center of Internet Security. (No date available). Mapping and Compliance. Referenced on 16. April 2023. https://www.cisecurity.org/cybersecurity-tools/mapping-compliance

Defense Information Systems Agency. 2023. Group Policy Objects. Referenced on 20. April 2023. https://public.cyber.mil/stigs/gpo/

Henttunen, K. 2018. Automated hardening and testing CentOS Linux 7: security profiling with the USGCB baseline. Bachelor's Thesis. Information and Communications Technology. Turku University of Applied Sciences. Referenced on 7. April 2023. https://urn.fi/URN:NBN:fi:amk-2018060512649

HashiCorp. 2020. Vagrant boxes. Referenced on 23. April 2023. https://developer.hashicorp.com/vagrant/docs/boxes

HashiCorp. 2022. Vagrant Documentation. Referenced on 10. April 2023. https://developer.hashicorp.com/vagrant/docs

Mandiant. 2022. M-Trends 2022 Report. Referenced on 9. May 2023. https://mandiant.widen.net/s/bjhnhps2mt/m-trends-2022-report

Microsoft Corporation. 2022a. Active Directory Domain Services Overview. Referenced on 12. April 2023. https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview#understanding-active-directory

Microsoft Corporation. 2022b. Running Remote Commands. Referenced on 5. May 2023. https://learn.microsoft.com/en-

us/powershell/scripting/learn/remoting/running-remote-commands?view=powershell-7.3

Microsoft Corporation. 2023. What is the Security Compliance Toolkit (SCT)? Referenced on 20. April 2023. https://learn.microsoft.com/en-us/windows/security/threat-protection/windows-security-configuration-framework/security-compliance-toolkit-10

National Institute of Standards and Technology, Information Technology Laboratory. 2020. United States Government Configuration Baseline. Referenced on 20. April 2023. https://csrc.nist.gov/Projects/United-States-Government-Configuration-Baseline/USGCB-Content/Microsoft-Content

NIST Special Publication 800-53 Revision 5 National Security and Privacy Controls for Information Systems and Organizations. National Institute of Standards and Technology.

NIST Special Publication 800-70 Revision 4 National Checklist Program for IT Products – Guidelines for Checklist Users and Developers. National Institute of Standards and Technology.

NIST Special Publication 800-152 A Profile for U. S. Federal Cryptographic Key Management Systems. National Institute of Standards and Technology.

Red Hat. 2019. What is configuration management? Referenced on 7. April 2023. https://www.redhat.com/en/topics/automation/what-is-configuration-management

Red Hat. 2021. Best Practices. Referenced on 25. April 2023. https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html

Red Hat. 2022. Integrating RHEL systems directly with Windows Active Directory. Referenced on 18. April 2023. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/integrating_rhel_systems_directly_with_windows_active_directory/index

Red Hat. 2023a. Ansible vault. Referenced on 25. April 2023. https://docs.ansible.com/ansible/latest/vault_guide/index.html

Red Hat. 2023b. How to build your inventory. Referenced on 25. April 2023. https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html