Politecnico di Torino

POLITECNICO DI TORINO

Master Degree Course in Computer Engineering

Master Degree Thesis

Security assessment and threat response through SCAP

Supervisors
prof. Antonio Lioy
prof. Andrea Atzeni

Massimiliano Torchio

Academic year 2021-2022

Summary

Managing the security of IT systems is becoming increasingly complex. As the number and variety of devices and their functionalities grows, so does the opportunities for attackers to successfully exploit vulnerabilities and infiltrate a system. Thus the need for the introduction of automatic processes and tools that can help security administrators in evaluating and maintaining the desired level of security.

One option that can be used as guidance in this integration of automation and security is the Security Content Automation Protocol (SCAP). SCAP is a framework that provides a standard for the format and nomenclature of security-related information, and its components and reference data can be used to ensure consistency and interoperability between security automation tools.

As first subject of this thesis, the current state of SCAP is analysed and discussed from a theoretical viewpoint. Each specification included in SCAP is described by referencing the official documentation, illustrating their relationships and their use in an automation context.

Next, the real world implementation of an open source security scanner, called OpenSCAP, is presented. This tool allows for the automatic evaluation of the security of a system, using the techniques and standards provided by SCAP. Starting from this base scanner, several tools and utilities have been developed to add new functionalities, such as scanning remote machines and evaluating container images.

Further analysis is dedicated to virtualized environments. In particular, it's discussed how OpenSCAP can be used to evaluate the components of a network functions virtualization infrastructure (NFVi), which is typically comprised of virtual machines or containers.

The current gaps and limitations are then discussed, pointing out which aspects of SCAP can be improved and where the biggest implementation flaws can be found. Of these, three areas in particular were selected for improvement, proposing and developing new alternative solutions.

The first proposed solution concerns the integration of the concepts of software weakness and common attack pattern in SCAP. These new components, which are represented by Common Weakness Enumeration (CWE) and Common Attack Pattern Enumeration and Classification (CAPEC) respectively, are not currently part of SCAP. Their inclusion can be useful in the process of vulnerability assessment, as they can provide additional information on how the specific vulnerabilities that are present on a system can be exploited, which in turn provides guidance on how to apply effective countermeasures.

In order to achieve this, a tool has been developed that can receive as input the result of a vulnerability assessment performed with OpenSCAP and then relate each found vulnerability to the corresponding weaknesses and attack patterns. This information can be easily accessed through a report that is generated by the tool that also contains the links to the reference pages of those weaknesses and attack patterns.

The second solution consists of a tool that allows the automatic remediation of container images, which is not part of the OpenSCAP features. The term "remediation" is used to refer to the modification of the image so that it results in a state of compliance to a desired security policy. It has been developed a software that is compatible with multiple container images and utilizes Podman and OpenSCAP in conjunction to create secured versions of those images.

Lastly, the third solution is a continuous monitoring tool. Continuous monitoring is the process of constantly assessing the security of a system by automated means. It has been developed a tool that can detect potentially dangerous changes in a system and automatically re-evaluate it and apply remediations. This solution can also be employed in a more complex use case, where a single main host acts as orchestrator and monitors multiple remote systems at the same time.

Each one of these solutions has been thoroughly tested, and solves at least partially the shortcomings that were targeted for improvement.

Contents

Chapter 1

Introduction

1.1 Background

The use of IT technologies has seen an unprecedented growth in the last decades. Virtually any aspect related to human activity has some sort of connection to the digital world. The new capabilities provided by computing devices and software applications are of extreme value, especially when they are deployed in a business context.

This trend of digital transformation has been further accelerated during the years of pandemic: a study from July 2020 conducted on almost a thousand company executives reported that the digitalization of customer interactions accelerated by three to four years, and the share of digitalized products or services was seven years ahead of the pre-pandemic rate of adoption [1].

The advent of these IT tools has brought with it new potential threats and security risks. The amount of valuable or confidential information that is digitally processed or stored is creating new targets for malicious agents. As institutions and companies proceed further in the digitalization process, more and more aggressors emerge that can spend more time and resources in trying to attack their IT systems.

As a consequence, new vulnerabilities and software flaws are discovered every day. New powerful exploits are also created to infiltrate systems and cause damage or steal data. The available attack surface increases and systems can be targeted by a considerable amount of diverse types of attacks.

Therefore, managing the security of IT systems is becoming an increasingly complex task. Each system requires specific configurations and security measures that can vary depending on its environment. In order to raise the confidence in the security of a system, and help maintaining that level of security throughout time, it is necessary to include some degree of automation in the security process.

1.2 Security automation

The use of automatic tools and techniques in the security process is becoming crucial and their prevalence is rising. The human activity is still the foundation and key factor in the context of security, but more and more aspects can be treated with automatic processes that allow a faster and more reliable analysis.

A survey conducted among several qualified IT security stakeholders indicates that the vast majority (over 90%) of companies is using some degree of automation in their security processes, and the ones with a higher level of automation can address more easily a vast number of security alerts and promptly act on them [2].

There are many areas that are related to security that are well suited for automation. Primarily, anything related to the collection of data regarding a system such as characteristics or

log files. Automatic tools can gather information in a much shorter period of time, allowing the security administrators to dedicate their time to more valuable subjects (e.g. the analysis of the collected data).

One of the first steps of the security process is to define the desired level of security, and from that specify the rules and characteristics that the system must adhere to. The following step is to verify the actual state of the system, i.e. perform a security assessment, that can be carried out with the help of automatic tools.

Automation can also be used to maintain the level of security of a system over time. This can be achieved both with tools that can detect ongoing attacks, behaving like Intrusion Detection Systems (IDS), or possibly even respond to such events, in this case as Intrusion Prevention Systems (IPS).

These techniques can be used to quickly respond to threats, allowing for a much safer environment, by taking advantage of the functionalities provided by such automatic tools.

The main objective of SCAP is to act as standard reference for these contexts of security automation. The SCAP framework provides a set of specifications and rules that, if properly followed, can ensure the interoperability among tools and the consistency of security-related data retrieved from different sources.

Chapter 2

The SCAP framework

2.1 Overview

The Security Content Automation Protocol (SCAP) is defined by NIST as "A suite of specifications that standardize the format and nomenclature by which software flaw and security configuration information is communicated, both to machines and humans." Its objective is to provide a standard for companies and individuals managing various aspects related to cybersecurity, specifying how to express, store, and organize security-related information. SCAP is comprised of many standards and can be applied in numerous use-cases such as verifying the absence of known vulnerabilities, checking system security configurations, performing inventory scanning, and automatically manage patches [3].

SCAP was created to assist companies in maintaining the security of their systems. Assessing and monitoring system security is a challenging task for multiple reasons:

- The sheer number and variety of systems to be secured poses great difficulty. First, a company may have hundreds if not thousands of different devices (desktop computers, servers, routers and switches, etc.) running on different operating systems. Then, each device has its own set of applications, and the same software may require different security configurations on different hosts.
- New vulnerabilities are discovered every day, and maintaining the security of a system requires constant threat mitigation via patches or software reconfiguration. Companies need to be quick in reacting to risk in order to minimize the window of exposure, preventing their system from being vulnerable for an extended period of time.
- A wide variety of security tools are available on the market, and many of them use proprietary content and terminology that are incompatible to one another. This lack of interoperability can slow the process of security assessment and even generate inconsistencies, to which the remediation can consume even more time and resources.

2.2 SCAP terminology

SCAP standardizes a set of specifications, described as *SCAP component specifications*. A product that utilizes SCAP employs a combination of these specifications for specific functions, called the *SCAP use cases*. SCAP content takes the form of an XML content named *SCAP data stream*. SCAP data streams come in two forms as *SCAP source data stream* for input content and *SCAP result data stream* for output content. The most relevant elements of a data stream are called *stream components*.

If a product or a source content wants to claim conformance to a SCAP use case, it must follow the relative specifications of NIST SP 800-126, and it's then referred to as *SCAP conformant*.

SCAP conformant products can be of two types: *content producers*, able to generate appropriate SCAP source data stream content, and *content consumers*, that can process existing SCAP source data stream content and output SCAP result data streams.

2.3 SCAP use cases

The following is a list of common uses of SCAP and their relative recommendations. This list is to be intended as a general, non-comprehensive guide, with the objective of providing a high-level description of some of the SCAP capabilities. Any organization actually employing SCAP should check if specific additional requirements are needed based on their system.

2.3.1 Security Checklist Verification

A company that wants to assess the security of its system should follow a security guidance. Many organizations produce security guidance for various devices and platforms, that should be both human and machine-readable to allow security verification in an automatic environment. SCAP provides this functionality by using SCAP-expressed security checklists that can be processed by SCAP-validated content consumers. These checklists are a collection of specific configurations to which a system should conform, that can be compared to the actual status of the system to confirm compliance or point out differences from the requirements of the guidance. Checklists can contain multiple profiles to support the same products in different environments and can be modified by companies with unique security requirements.

General SCAP checklists, that include many aspects of a system (for example, an operating system security checklist) can be very useful and are often paired with more specialized checklists. Specialized SCAP checklists can verify a particular characteristic of a system, such as the presence of a specific patch, to highlight potential security issues. Often vendors make SCAP conformant data publicly available to perform patch checking of their product. SCAP checklists are valuable also for testing, as a tool to check that new applications do not alter the security configuration required by the company.

Usually, a company should acquire security checklists from the National Checklist Program (NCP), maintained by NIST, and only trust digitally signed SCAP checklists. After selecting the appropriate checklists that refer to the products and needs of the company, the next step is to modify them according to the specific security requirements of the company, in a process called tailoring. For example, a specific check could be omitted because the company has other means to secure that particular aspect, or a rule could be restricted even more for a higher level of security.

2.3.2 Software and Hardware Identification

Being able to identify which software or hardware is present on a system is highly valuable for companies. Even when that software or hardware is uninstalled, it leaves behind some traces of its past existence: executables, code libraries, specific configurations, and drivers are some examples of what is called an *artifact*. SCAP checklists can retrieve those artifacts and link them to the actual software or hardware that produced them.

SCAP checklists enable companies to perform inventory operations, identifying which software is installed on a system. With the ability to link artifacts to its original generator, the company can also identify software that was installed and then removed. This process can be very useful to detect possible security threats, such as violations of the policies of the company (for example in the case of a blacklist or whitelist) or finding evidence of a particular malware attack. The real benefit of SCAP is that incident response team and software vendors can quickly make public SCAP-expressed checklists that address the security issue, and companies can use their SCAP-validated tools to verify the checklist. The use of SCAP allows for a reduction of the window of exposure, avoiding extra steps from the tool vendors in developing their version of the checklists, as the time consumed in testing and deploying the checklist can be significant.

2.3.3 Providing Evidence of Conformance

Many organizations need to demonstrate that they conform with legal mandates and regulations, depending on which country and in which field they operate in. Some examples are the Federal Information Security Management Act (FISMA) for the U.S. government IT systems, the Health Insurance Portability and Accountability Act (HIPAA) for healthcare, and the General Data Privacy Regulation (GDPR) for privacy in the EU.

It's usually very challenging for the companies to demonstrate that they implemented the security requirements needed. The linking of the high-level abstract requirements of the regulations to the low-level actual configurations of the system is often complicated, ambiguous, and error prone. To provide assistance, SCAP content can characterize these links. Some organizations already provide SCAP checklists that officially provide evidence that the regulations are respected. One example of this is NIST that provides SCAP-expressed checklists that provide compliance evidence for FISMA on many systems.

2.3.4 Continuous Monitoring and Remediation

SCAP can be used to automatically monitor changes in the system, if not truly continuously, at least periodically. With the integration of SCAP-expressed checklists and SCAP-validated scanners, any deviation from the requirements that could negatively affect security can be quickly identified and remediated. Almost all systems need some sort of continuous monitoring: either some new software or hardware are added to the system, or a new patch is available, or a user changes some configurations (with or without malicious intent). Some of these events require new security checklists and/or modifications to existing ones, others can be detected by simply performing again the security checklist verification.

Since SCAP poses great attention and detail in the standardization and interoperability of its contents, the results of an SCAP analysis can easily be used as input for other tools. The field of Data analytics studies how to find patterns of suspicious activity after retrieving a large amount of data, and SCAP results are immensely useful for the retrieving part. SCAP checklists results, and possibly their subsequent elaboration, both help immensely when implementing remediations. Although automatic remediation is not directly provided by SCAP, it certainly constitutes the bases for a clear communication between tools and between humans and machines, that enables fast and effective automatic remediation of security problems.

2.4 SCAP Specifications

There are twelve individual specifications adopted by SCAP, known as the SCAP component specifications, grouped in five categories:

- **Languages.** The SCAP languages provide standard vocabularies and conventions for expressing security policy, technical check mechanisms, and assessment results. These specifications are Extensible Configuration Checklist Description Format (XCCDF), Open Vulnerability and Assessment Language (OVAL), and Open Checklist Interactive Language (OCIL).
- **Reporting formats.** The SCAP reporting formats provide the necessary constructs to express collected information in standardized formats. The SCAP reporting format specifications are Asset Reporting Format (ARF) and Asset Identification.
- **Identification schemes.** The SCAP identification schemes provide a means to identify key concepts such as software products, vulnerabilities, and configuration items using standardized identifier formats. They also provide a means to associate individual identifiers with additional data pertaining to the subject of the identifier. The SCAP identification scheme specifications are Common Platform Enumeration (CPE), Software Identification (SWID) Tags, Common Configuration Enumeration (CCE), and Common Vulnerabilities and Exposures (CVE).

- **Measurement and scoring systems.** The SCAP measurement and scoring system specifications are Common Vulnerability Scoring System (CVSS) and Common Configuration Scoring System (CCSS). These specifications are used to measure specific characteristics of a security weakness and generate a severity score for that weakness.
- **Integrity.** An SCAP integrity specification helps to preserve the integrity of SCAP content and results. Trust Model for Security Automation Data (TMSAD) is the SCAP integrity specification.

Regarding software flaws and security configurations, SCAP uses as standard reference data those provided by the National Vulnerability Database (NVD), managed by NIST.

These component specifications are used together to create an SCAP-conformant security guidance. For example, a security guidance can describe the desired security configurations and installed patches. Such requirements would be expressed as a checklist using XCCDF, specifying which checks need to be performed automatically (with OVAL) or manually (with OCIL). The security configurations would be enumerated with CCE, the software flaws with CVE, and the target platforms with CPE and SWID tags. The resulting checklist would be an SCAP-expressed checklist that can be utilized by SCAP-validated tools.

The next sections describe the functionalities of the SCAP component specifications one by one.

2.4.1 XCCDF

Extensible Configuration Checklist Description Format is an XML based language capable of describing security checklists in a standardized way. It allows for an easier and more uniform creation of automated security checklists. In the context or SCAP, XCCDF can be described as the general descriptive language that contains the other specifications, organizing the various components and how they relate to each other.

Here are some of the most relevant capabilities provided by XCCDF:

- A more consistent and accurate exchange of security related information, between vendors and companies, and between security experts or auditors of the same company.
- A faster and uniform generation of security checklists that include rules, technical procedures, advisories and measures for remediations.
- Allows the combination and/or modification of different security checks from different sources.
- Ensures compliance to multiple policies and facilitates the scoring and reporting of compliance results.

The core components of a security checklist are called rules: a single rule describes a specific state or condition that the target system must be in. An example of rule can be how stringent the password requirements should be, or assessing the presence of a specific vulnerable configuration in the system.

The key functionality of XCCDF is tailoring: checklists users can merge and customize different checklists to better suit their organizational or situational needs. Tailoring is highly valuable in many different cases. A specific company may want to further restrict the password policies or set them differently for different parts of the system. Another example is if some particular configurations are in conflict with other applications, the security team of the company can disable some of the rules.

Tailoring provides three customization options:

- Selectability: Single or multiple rules together can be selected or de-selected to respectively include or exclude them from the check

- Value Modification: XCCDF can contain variables that are then referenced in the document itself. The values of these variables can be substituted with locally significant values, such as specific addresses of parts of the system or the required length of passwords.
- Property Modification: This option includes all the possible customizations that are not selectability or value modification. One example could be altering the weight value of some rules. Any modification must still conform to the specifications, in order to guarantee interoperability with SCAP-conformant tools.

XCCDF supports and facilitates tailoring. Checklists authors are encouraged to include descriptive text, associated with each rule, that should contain useful information that can guide users in tailoring choices. It's also possible to create tailoring documents which define different tailoring profiles. Some combination of rules could be in conflict or mutually exclusive, and profiles can help avoiding these inconsistencies.

Lastly, XCCDF defines a standardized reporting format for test results. The objective is storing rule checking results with a defined set of useful information, allowing different security tools to operate consistently on these results. The XCCDF results shall include data such as the start and end time of the test, the target system, the check results, and which tailoring choices were made [4].

2.4.2 OVAL

Open Vulnerability and Assessment Language is a community-developed language that standardize the assessment and report of a system security state. OVAL is maintained by the Center for Internet Security (CIS), and its repository is currently hosted by MITRE. OVAL is one of the two checking systems supported by SCAP (the other being OCIL). It allows automatic checks and can retrieve a wide variety of information from the system: it should be used whenever is possible, and the use of OCIL should be restricted only to checks that cannot be performed automatically.

The OVAL language is comprised of three parts that corresponds to three key steps of system assessment:

- OVAL Definitions. The first part of the language is designed to describe the specific state of a system.
- OVAL System Characteristics. This second part defines a standard for representing system configuration information.
- OVAL Results. The last part specifies how to format and store the evaluation results.

The OVAL Definitions are grouped in four classes, based on what the objective of the check is. Vulnerability Definitions describe the conditions that must be present on a system to assess the existence of a vulnerability. Patch Definitions state which patches are appropriate for a system. Inventory Definitions describe the conditions that determine whether a specific software is installed on a system. Compliance Definitions define the conditions of a system to be considered compliant to a specific policy.

The main feature of OVAL is its ability to perform checks: the definitions specify which tests need to be passed, and the tests are considered passed when specific objects are in the required states (objects and states are referred to with other components, such as CPE or CCE). This process allows the checks of an SCAP checklist to be performed automatically.

2.4.3 OCIL

The Open Checklist Interactive Language defines a framework for representing non-automatable questions. It aims at retrieving information either directly from a person or from previous data

collections. OVAL and OCIL are both part of SCAP because they complement each other. The use of OCIL questionnaires must be limited only to checks that cannot be performed by OVAL.

OCIL is used to express manual security checks. There are multiple instances in which these manual checks are needed:

- A software product might not be compatible with OVAL, for example by not having the correct APIs.
- Some kind of physical information could be of use, such as a serial number affixed to a device.
- The user's own information might be needed, for example asking whether he or she took part in a security training session in the last six months.

OCIL checks can be included in SCAP checklists but usually have some throwbacks: the automation component of SCAP becomes somewhat reduced, requiring data that cannot be automatically retrieved by security tools.

2.4.4 ARF

The Asset Reporting Format defines a standardized data model for the transport of information about assets, and the relation between assets and reports. It aims at facilitating the interoperability of security related information between organizations, vendors, and tools. ARF is specifically designed to merge and consolidate multiple result files, coming from different specifications (for example, XCCDF and OVAL results).

Different products may produce reports that relate to the same asset but that use different representations of it. By combining reports, report requests, and asset correlations (defined later in Asset Identification), different reports can be correlated and fused, enabling the spread of a common structure for ARF reports.

2.4.5 Asset Identification

Asset Identification is a framework that describes attributes and methods to uniquely identify an asset. The term "asset" is used here to indicate anything that has value for an organization, ranging from people and devices to networks and software. The Asset Identification specification allows companies to quickly correlate various information from different sources, identifying assets by merging these data.

In the context of SCAP, Asset Identification is used within an ARF report to identify the target asset that was under assessment. It can also contain additional identification information to further facilitate interoperability for future identifications.

2.4.6 CPE

Common Platform Enumeration is, as defined by NIST, a standardized method of describing and identifying classes of applications, operating systems, and hardware devices present among an enterprise's computing assets. Its objective is to identify classes of products with specific names, allowing tools to collect information from actual assets and matching them to CPE names [5].

CPE is a complex and evolving specification, currently composed by four individual specifications that form the CPE stack:

- **Naming.** Defines standardized methods for assigning names to IT product classes. It defines the *well-formed names* (WFN), logical constructs that include many parameters such as the name of the product, the version, and the language.

- **Name Matching.** Defines the procedures for comparing WFNs to each other, checking whether they refer to the same class of products. The name comparison results are expressed as set relations, determining if one name is more general and contains the other, or if they represent two completely different classes of products.
- **Dictionary.** Defines the concept of a CPE dictionary as a collection of CPE names associated with some kind of meta data. It specifies how to create and maintain a dictionary repository, as well as how to search for specific CPE names or a related class of products.
- **Applicability Language.** Defines a structure for forming complex logical expressions combining WFNs. These expressions are used to tag security checklists with specific information about the products that they refer to. One example is combining the CPE name of an Operating System and the one of a specific application, ensuring that both products are installed at the same time.

The official CPE dictionary is located at the National Vulnerability Database (NVD), managed by NIST, and should be referred to when using CPEs in SCAP content. Additional third-party dictionaries can be included, if they follow the official specifications. CPE is used in SCAP to uniquely identify the classes of products, including the target system where the security check is going to be performed.

2.4.7 SWID

Software Identification is a standard that provides the format and the structure for describing a software product. SWID tags are a collection of metadata that contains information about a specific software, that are added to the system at the moment of installation of the software and deleted during the uninstall process. When the SWID standard is correctly employed, the presence of a SWID tag guarantees the presence of the related software on the system.

SWID tags are complementary to CPE and can be mapped to specific CPE names. Their use can be incorporated to facilitate the assessment of the presence of a specific software product. This procedure is especially useful for inventory purposes.

From a security standpoint, maintaining accurate software inventories is crucial for multiple reasons:

- Ensures that all installed software adhere to the desired version, reducing the presence of variable software and thus reducing the surface of attack.
- Enables the verification that all software patches are up-to-date and no known vulnerabilities affect the system.

NIST is currently working on incorporating SWID tags data in the NVD, so that vulnerable software affected by a specific CVE can be more easily identified.

2.4.8 CCE

Common Configuration Enumeration assigns a unique identifier to a particular software configuration that presents a security issue. The official CCE list, maintained by NIST, is a repository of CCE identifiers[1]. Each entry has a unique CCE identifier number, together with a human-readable description of the configuration issue and the technical parameters that characterize the configuration. In this sense, CCE provides a connection between the natural language, the security configuration policies and the machine-readable technical implementations.

[1]The entire list can be found at: https://ncp.nist.gov/cce/index

CCE must be used in SCAP when assessing the security of the system software configuration. CCE does not include other kinds of possible configurations, such as hardware or physical, and operates only for software-based configurations. The use of the CCE list allows for a standardized exchange of information about configuration issues, enhancing the interoperability of security tools and the consistency across multiple sources.

2.4.9 CVE

Common Vulnerabilities and Exposures is, similarly to CCE, a reference dictionary of known software flaws and vulnerabilities. It provides a unique identifier for each vulnerability, accompanied by a human-readable description of the issue, the date of the discovery, and links to other related resources. The CVE standard is maintained by MITRE and the CVE repository of reference is the National Vulnerability Database (NVD) by NIST[2].

CVE is the standard for communicating software vulnerabilities information between organizations, vendors, and security tools. Unlike CCE, CVE is not restricted only to software vulnerabilities but can identify a broader range of vulnerabilities, including hardware-related ones.

CVE identifiers are assigned to each publicly known vulnerability and are considered as the base references for vulnerability assessment in the context of SCAP.

2.4.10 CVSS

The Common Vulnerability Scoring System is a metric to evaluate the severity of software vulnerabilities. A CVSS score can be represented as a string vector containing the values used to derive the final score. Higher scores correspond to higher severities, which means that the vulnerability can be easily exploited or could have devastating consequences. CVSS scores can help companies in prioritizing which vulnerabilities need to be remediated quickly. CVSS refers to CVEs to identify vulnerabilities.

There are two additional metrics that are part of CVSS: temporal and environmental scores. While the base score is calculated once and stored alongside CVEs in the NVD, the other two can be used to modify the base score for specific situations. Temporal scores take into consideration parameters that can change over time due to external events, environmental scores are customized scores that reflect how impactful a vulnerability can be on a specific company. SCAP allows the use of all the above metrics to better suit the needs of the different organizations.

2.4.11 CCSS

The Common Configuration Scoring System is a framework for measuring the severity of software security configuration issues, assigning a score to each particular issue. CCSS is derived from CVSS and operates in analogous ways, but they are not used exactly in the same way.

While CVSS scores provide a direct measure of the vulnerability severity and can be informative by themselves, CCSS scores can be useful to companies, but need to be considered together with the security policies of the company and in the broader context of the system.

2.4.12 TMSAD

The Trust Model for Security Automation Data defines a trust model that can be applied to SCAP to establish its security, including integrity of data and authentication. TMSAD focuses on securing XML files, as it's the common format for SCAP files. The model is composed of recommendations on how to effectively use digital signatures, cryptographic algorithms, and identity information.

[2]NVD vulnerability page: https://nvd.nist.gov/vuln

A company that acquires SCAP content should always verify the validity of the digital signature applied to it, and reject any content that either fails the validity check or doesn't include a signature at all. After any process of tailoring, a company should sign again the modified document with a new signature, to ensure that later modifications cannot be applied with malicious intent.

Chapter 3

The OpenSCAP tools

3.1 The OpenSCAP ecosystem

The OpenSCAP project, initially sponsored by Red Hat, started in late 2008 with the objective of creating useful open source tools to interact with the SCAP standard, and in particular the development of a security scanner that can deal with SCAP content [6]. This scanner is called OpenSCAP Base.

Their ecosystem is now comprised of several tools, built on top of the base scanner, that address a wide range of real world scenarios. The next sections describe the most relevant aspects of the tools.

The other fundamental aspect of the ecosystem is the security content, which is as important as the scanner itself. For this reason the SCAP Security Guide (SSG) project was created. It consists of an open source collection of security policies written in SCAP form.

SSG contains security checklists with well described rules that often include remediation scripts. It also implements the security requirements of many authorities, such as PCI DSS, STIG, and USGCB. Each document contains multiple profiles so that a user can select the appropriate one for its use case.

3.2 OpenSCAP Base

3.2.1 Introduction

OpenSCAP is, at its core, a set of open-source libraries that allow the manipulation of the various standards of SCAP. OpenSCAP also represents a command line tool, called *oscap*, that can evaluate SCAP components, scan target systems and generate documents with the proper format.

OpenSCAP has been validated by NIST as conformant to SCAP version 1.2 on February 2017 [7].

The following are the certified SCAP components that are supported by OpenSCAP with their relative version:

- AI (1.1);
- ARF (1.1);
- CCE (5);
- CCSS (1.0);
- CPE (2.3);

- CVE;
- CVSS (2.0);
- OVAL (5.10.1);
- TMSAD (1.0);
- XCCDF (1.2);

Since the date of the certification, OpenSCAP has been updated and now supports SCAP 1.3. In particular, OVAL was updated to support version 5.11.1.

3.2.2 Installation

OpenSCAP can be installed on many Linux distributions using their relative package manager command:

- Fedora: dnf install openscap-scanner
- Red Hat Enterprise Linux (RHEL) 5: yum install openscap-utils
- RHEL 6, RHEL 7, CentOS 6, CentOS 7: yum install openscap-scanner
- Debian, Ubuntu: apt-get install libopenscap8
- Scientific Linux 6, Scientific Linux 7: yum install openscap-scanner
- openSUSE Leap, openSUSE Tumbleweed: zypper install openscap-utils
- SUSE Linux Enterprise Server 12 and 15: zypper install openscap-utils

On Windows systems, the OpenSCAP installer can be downloaded from:

https://www.open-scap.org/tools/openscap-base/#download

Some Windows systems may need an additional installation of Visual C++ Runtime libraries:

https://aka.ms/vs/15/release/VC_redist.x86.exe

Alternatively, OpenSCAP can be directly built from the source code that is available on GitHub. The precise instructions are located at:

https://github.com/OpenSCAP/openscap/blob/maint-1.3/docs/developer/developer.adoc

The first step is to get the source code either from a release tarball or from the git repository. Then the build dependencies need to be installed; note that these dependencies are different between operating systems. Once the installation is done, the OpenSCAP library can be built. The library contains self-checks that can be used to test the correctness of the build. Finally, OpenSCAP can be installed.

3.2.3 Command modules

The OpenSCAP command line tool takes the following general form:

oscap [general-options] **module** operation [operation-options-and-arguments]

There are only two general options:

- **-V, –version**: prints the version of oscap and a list of supported specifications, shows inbuilt CPE names and supported OVAL objects;
- **-h, –help**: shows the help screen.

The capabilities that oscap provides are divided in modules that represent the different SCAP specifications:

info determines type and print information about a file;

xccdf evaluates XCCDF checklists;

oval evaluates OVAL Definitions;

ds manipulates SCAP data streams as a whole;

cpe validates and searches CPE names;

cvss calculates CVSS scores;

cve validates and searches for CVE Identifiers;

The following descriptions of operations are based on oscap 1.3.5. All the examples were executed on a Fedora 32 machine, using files that are part of the SCAP security guide (SSG) project. SSG can be installed with a single command (*dnf install scap-security-guide* on Fedora).

Info Module

This module takes as input an xml file and prints information about the SCAP content in it. It determines the file type and additional information such as date of creation and specification version. In the case of xccdf or source data stream files, the info module retrieves the IDs of incorporated profiles, components, and data streams.

Using as example the *ssg-fedora-xccdf.xml* file, the following is the result of the info inspection:

```
Document type: XCCDF Checklist
Checklist version: 1.1
Imported: 2021-03-19T15:18:22
Status: draft
Generated: 2021-03-19
Resolved: true
Profiles:
        Title: OSPP - Protection Profile for General Purpose Operating Systems
               Id: ospp
        Title: PCI-DSS v3.2.1 Control Baseline for Fedora
               Id: pci-dss
        Title: Standard System Security Profile for Fedora
               Id: standard
Referenced check files:
        ssg-fedora-oval.xml
               system: http://oval.mitre.org/XMLSchema/oval-definitions-5
        ssg-fedora-ocil.xml
               system: http://scap.nist.gov/schema/ocil/2
```

The *–profile* option can be specified, using the Id of a profile contained in the checklist, to print more information about that specific profile. For example, when using the *standard* Id, the output description is:

> This profile contains rules to ensure standard security baseline of a Fedora system. Regardless of your system workload all of these checks should pass.

Using *pci-dss*, the output instead is:

> Ensures PCI-DSS v3.2.1 related security configuration settings are applied.

Oscap info can also be used on result type files, such as ARF, in which case it reports useful information such as date and time of evaluation and source benchmark and profile.

XCCDF Module

The supported version of XCCDF is 1.2. There are six main operations concerning this module.

The first one is **eval**. It performs the evaluation of an xccdf file given as input. It prints the result of each rule contained in the checklist. The evaluation can be performed also if the input file is a source data stream file, in which case oscap evaluates the first xccdf checklists present in the data stream. If multiple xccdf checklists are present, the option *–xccdf-id* allows to select a specific component.

It can be specified which particular profile is to be used, and it can even be specified a particular rule to be evaluated. If a tailoring file or tailoring component is available, it can be included by specifying its Ref-Id.

Oscap has some inbuilt CPE names, but the *–cpe* option allows the use of CPE dictionaries or languages for applicability checks.

There are many options for generating different types of result files and reports, namely XCCDF, OVAL and ARF result files and HTML reports.

XCCDF checklists can contain remediation scripts for some of the checks. If a given rule is failed, the execution of these scripts should correct the system and bring it to a state of compliance with that rule. The *–remediate* option executes the remediation fixes for failed rules immediately after the scan. Since this option automatically executes some potentially dangerous scripts, it should be used only for trusted XCCDF content.

The second operation is **remediate**, that provides post-scan remediation. It takes as input an XCCDF Results file (that can be generated, for instance, with the *oscap xccdf eval* command), then for every failed rule it checks if a fix script is available and executes it. Again, this procedure can be dangerous from a security point of view, therefore it should be used only for trusted content. Reports and result files can be generated in a way similar to the *eval* operation.

Another operation is **resolve**, that takes as input an XCCDF file and resolves it as described in the NISTIR 7275 specification [4]. Available XCCDF checklists are typically already resolved, such as the content of SSG.

The **validate** operation validates an XCCDF file against an XML schema and shows every found error. Starting from XCCDF version 1.2 the validation is based on a Schematron, which renders the validation process much faster.

The fifth operation is **export-oval-variables**. Given an XCCDF file, this operation collects all the XCCDF values that would be used by OVAL during an evaluation, then it creates an OVAL external-variables document containing such values. Single profiles or checklists included in a data stream can be selected with the corresponding options *–profile* and *–xccdf-id*.This procedure is needed prior to the scan of OVAL definitions in some particular cases, described in the next section (OVAL Module).

Lastly, the **generate** operation, starting from an XCCDF file or a result file, creates another document such as a security guide or a report. There are many submodules that can generate different files: HTML security guides from a benchmark, HTML reports on evaluation results, fix scripts that can apply remediation to the system, or custom output files given an XSLT file.

OVAL Module

OpenSCAP supports oval 5.11.1. The tool can probe the system and evaluate all definitions contained in an OVAL Definitions file. To start an evaluation, the input file can either be an OVAL Definitions file or a SCAP source data stream.

In the case of SCAP content that is distributed along multiple XML files, such as an OVAL Definitions file and a separate XCCDF file, some of the OVAL definitions may depend on variables that are assigned during the XCCDF evaluation. This means that a separate evaluation of those OVAL definitions may produce inconclusive results.

The XCCDF *–export-oval-variables* operation creates a dedicated file that can be used in a subsequent OVAL scan with the *–variables* option referencing to that file. This allows for a correct evaluation of OVAL definitions.

The OpenSCAP tool can also perform data collection regarding OVAL not by scanning a system, but by analyzing a file that contains system characteristics. This input file may have been generated on a separate system. This **analyse** operation produces an OVAL Results file as output.

The **validate** and **generate** options are similar to the ones described in the XCCDF section.

Additionally, the **collect** operation allows oscap to gather system characteristics for all objects in an OVAL Definitions file, showing if those objects are present on the system or not.

Data Stream Module

The ds module can manipulate SCAP data streams. In particular, it can create a source data stream file starting from an XCCDF file: the tool can automatically detect dependencies present in XCCDF benchmarks, like OVAL files, and create a single data stream that contains those files.

Once the source data stream file is created, new components can be added. New components can either be OVAL, XCCDF or CPE Dictionary files.

The opposite operation can also be performed: starting from a bundled source data stream file, oscap can split it into multiple files (XCCDF, OVAL definitions, etc) allowing further inspection of the single components.

Additionally, this module manages result data streams. Given a source data stream and XCCDF result file, possibly followed by additional OVAL results, OpenSCAP can generate a result data stream in ARF format.

Both source and result data streams can be validated against an XML schema to check for errors.

CPE Module

CPE operations are quite simple: given a CPE name, oscap can check whether the name is in a correct CPE format, and if also given a CPE dictionary it can find an exact match of the name in that dictionary. It can also validate the dictionary against an XML schema.

CPE 2.3 is the supported version.

CVSS Module

The OpenSCAP tool can take as input a CVSS vector and calculate base, temporal and environmental scores. It can also describe the individual components of a CVSS vector in a human readable format and show the partial scores of the components.

OpenSCAP supports CVSS 2.3.

CVE Module

This module can validate a given CVE data feed. With the *find* operation, taking as input a CVE identifier and a data feed, the tool checks if the given CVE is present in the data feed and reports base score, vector string and vulnerable software list.

OpenSCAP supports CVE Version 2.0.

3.3 OpenSCAP Utilities

Several additional tools are built on top of the OpenSCAP library, with each of them providing additional capabilities. The core functioning is the same of the oscap base command line tool, but these utilities allow the user to perform advanced tasks with a single command.

All the following tools can be installed with the installation of a single package, *openscap-utils*.

3.3.1 Tailoring

The command line tool **autotailor** enables the creation of a tailoring file that can be used alongside an existing SCAP source data stream, adding a new customized profile that extends an existing one. There are three tailoring operations: adding a new rule with the *–select* option, removing a rule with the *–unselect* option, or modify the value of an XCCDF variable with *–var-value*. All these options can be used multiple times in the same command. Adding or removing a rule requires the user to specify the rule ID.

Autotailor is a basic tool that doesn't provide any validation of the tailoring. This means that it allows the extension of non-existing profiles, selection or de-selection of non-existing rules, and modification of non-existing variables. The resulting tailoring file would be meaningless and any evaluation using it will result in an error.

There are better solutions for creating tailoring files. For example, SCAP Workbench is an application with a user-friendly graphical interface that provides the same functionalities and much more control on the correctness of the changes.

3.3.2 Scanning remote machines

OpenSCAP can be used to perform evaluation of remote machines using an SSH connection with **oscap-ssh**. This tool copies the input SCAP content to the remote machine, runs the evaluation on the target system and downloads the results back to the local machine. After the evaluation, no content or temporary file remains on the remote system. Note that only the target system needs the oscap tool installed on its end, which is not needed on the local machine.

After the SSH connection, the usage of the tool is equivalent to the regular oscap. The evaluation of XCCDF content cannot be done by passing as input a standalone XCCDF file but requires a source data stream. Evaluation of OVAL content and collection of system characteristics can be performed using OVAL files as input.

Oscap-ssh checks a particular environmental variable on the local machine, named sshadditionaloptions, and copies its contents on the SSH command line in the proper position where those options are expected. These contents may be needed to establish and manage the SSH connection.

There are some security concerns regarding this tool. Usually, OpenSCAP needs to gain privileged access on the system in order to function properly.

One possibility is to enable the "PermitRootLogin" directive in etc/ssh/sshdconfig on the remote machine, but this operation is a security issue by itself and should be avoided.

A much better option is to enable a non-privileged user to run only the oscap commands as root, with the *sudo* option, by properly configuring the remote machine.

3.3.3 Scanning Linux containers and images

The **oscap-podman** tool allows the scanning of containers and container images by simply specifying the ID of the container or image. The tool functionalities are analogous to the base oscap tool, but it can only run under root privileges.

This tool utilizes a technique called *offline scanning*: the file system of the container is mounted to a directory of the host in read-only mode. This means that the container is not changed in any way by OpenSCAP, which acts from the host system. Hence, remediation of the container cannot be performed by oscap-podman in an autonomous way but requires manual configuration.

The following is an example of evaluation of a RHEL 8 container. First of all, the ID of a container or a container image present on the system can be retrieved by the command *podman images*:

```
$ podman images
REPOSITORY                          TAG     IMAGE ID      CREATED    SIZE
registry.access.redhat.com/ubi8  latest  3269c37eae33  1 week ago  208 MB
```

Once the desired image ID is chosen the evaluation can be performed, using the contents of SCAP Security Guide, with the following command:

```
$ oscap-podman 3269c37eae33 xccdf eval --report report.html --profile ospp \
    /usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

3.3.4 Scanning Docker containers and images

The **oscap-docker** tool can assess vulnerabilities and check compliance to security policies of running Docker containers or Docker images, using the offline scanning technique.

The tool can perform compliance scanning as the base oscap tool, specifying which container or image is the target of the command. As an example, scanning a RHEL 7 image with a given XCCDF checklist file and generating an HTML report can be done with the following command:

```
$ oscap-docker image rhel7 xccdf eval --report report.html xccdf.xml
```

In the case of a running container, it's sufficient to change the word "image" to "container".

Another interesting functionality is the ability to perform a vulnerability assessment of a container or image, testing them against a CVE stream. A CVE stream is a collection of vulnerabilities that are applicable to the target container or image, written in SCAP format, downloaded directly from the product vendor.

Using again as example a RHEL 7 image, the command takes the following form:

```
$ oscap-docker image-cve rhel7 --report report.html
```

The command will attach the desired docker image, determine the operating system variant/version, download the applicable CVE stream from Red Hat, and then evaluate the target image for vulnerabilities.

In this case the additional OpenSCAP options can only be for the creation of HTML reports or OVAL Result files.

For the evaluation of a container, the correct command needs "container-cve" instead of "image-cve".

3.3.5 Scanning virtual machines

The **oscap-vm** tool can perform SCAP evaluation of virtual machine domains or virtual machine images. Again, it uses the offline scanning technique: the evaluation happens from the host so that the virtual machine is not changed or damaged in any way, and doesn't require the installation of any additional software, including OpenSCAP.

The usage requires first to specify the type of target, either *domain* or *image*, and then identify it by the domain name or the image path. After that, the rest of the options are passed to the base oscap, that can perform evaluation of XCCDF or OVAL content, and collect OVAL System characteristics. The same restrictions of the previous offline scanning tools apply to this tool, notably that it cannot perform automatic remediation.

3.3.6 Scanning arbitrary file systems

The **oscap-chroot** tool can perform offline SCAP evaluations of file systems mounted at an arbitrary path. It can be used when evaluating custom objects that aren't covered by previous tools, for example containers of different formats. The modes of operation and options mimic the oscap ones.

3.4 OpenSCAP Daemon

The OpenSCAP Base tool and its utilities are very adequate to perform solicited single evaluations, but lack the possibility of planned scanning or continuous monitoring. Thus the OpenSCAP Daemon project was created to assist the users in scheduling SCAP evaluations.

The project is comprised of two main components: the daemon itself, running in sleep state in the background until a task has to be processed, and a command-line interface tool for managing tasks.

3.4.1 Installation

OpenSCAP Daemon can be installed on Fedora systems with the following command:

```
$ dnf install openscap-daemon
```

On other Linux distribution, first download the latest release from GitHub[1], then the tool can be installed either as a python2 or as a python3 application:

```
cd openscap-daemon
# as a python2 application
sudo python2 setup.py install
# as a python3 application
sudo python3 setup.py install
```

In order to allow further task operations and executions, the daemon needs to be started with the proper command by a privileged user:

```
$ oscapd
```

After that the command line tool is ready to be executed by opening a different terminal window.

3.4.2 Task management

The entire functioning of the tool is centred around tasks. The first operation is the creation of a task, using the command:

```
$ oscapd-cli task-create -i
```

After that, a simple interactive window is presented to the user that can set the desired options for the task. The target of the evaluation can be specified to be either local or remote, using an SSH connection. After that the user can select the proper SCAP content, optionally followed by a tailoring file, and a suitable profile.

Lastly, the schedule is created by specifying two values: "not before", indicating the starting time of the evaluation, and "repeat after", that can be set to a specific amount of hours or with the predefined intervals (daily, weekly, monthly).

The following is an example of the creation of a task that evaluates a local machine with a Ubuntu 20.04 data stream, using the standard profile, and is set to be repeated daily:

[1] https://github.com/OpenSCAP/openscap-daemon/releases/tag/0.1.10

```
$ oscapd-cli task-create -i
Creating new task in interactive mode
Title: Daily Example
Target (empty for localhost):
Found the following SCAP Security Guide content:
       1: /usr/share/xml/scap/ssg/content/ssg-centos7-ds.xml
       2: /usr/share/xml/scap/ssg/content/ssg-centos8-ds.xml
       ...
       26: /usr/share/xml/scap/ssg/content/ssg-ubuntu2004-ds.xml
       27: /usr/share/xml/scap/ssg/content/ssg-vsel-ds.xml
       28: /usr/share/xml/scap/ssg/content/ssg-wrlinux1019-ds.xml
       29: /usr/share/xml/scap/ssg/content/ssg-wrlinux8-ds.xml
Choose SSG content by number (empty for custom content): 26
Tailoring file (absolute path, empty for no tailoring):
Found the following possible profiles:
       1: Standard System Security Profile for Ubuntu 20.04
              (id='xccdf_org.ssgproject.content_profile_standard')
       2: (default)
              (id='')
Choose profile by number (empty for (default) profile): 1
Online remediation (1, y or Y for yes, else no):
Schedule:
 - not before (YYYY-MM-DD HH:MM in UTC, empty for NOW):
 - repeat after (hours or @daily, @weekly, @monthly, empty or 0 for no
     repeat): @daily
Task created with ID '1'. It is currently set as disabled. You can enable it
with oscapd-cli task 1 enable.
```

Tasks are disabled by default when created, and need to be enabled with the suggested command. After every evaluation, the results are saved with an ascending Id and can be viewed either as ARF files or HTML reports.

3.5 Practical examples

The following two sections describe two common use-cases of OpenSCAP, providing step by step the instructions for reproducing the tests.

The first section analyses the content of SCAP Security Guide and then checks the system's compliance to the chosen security policy.

The second section shows a vulnerability scanning using a CVE stream, which is a collection of OVAL definitions that check for the presence of specific vulnerabilities, directly retrieved by the system's vendor.

3.5.1 Security compliance on Fedora

Every test described below was executed on a freshly installed Fedora 32, which can be downloaded from:

https://dl.fedoraproject.org/pub/fedora/linux/releases/32/Workstation/x86_64/iso/

Two components are needed in order to test the system: the OpenSCAP Base scanner and the security content of SCAP Security Guide. The former, which gives access to the oscap command line tool, can be installed with:

```
$ dnf install openscap-scanner
```

The latter is installed with:

```
$ dnf install scap-security-guide
```

The SCAP Security Guide contains many security policies written in SCAP form that are installed at /usr/share/xml/scap/ssg/content/. Inside this folder there are seven files that target a fedora machine, and all their names start with *ssg-fedora*.

After the installation of oscap it's possible to check the installed version, along with some other useful information, by using the command "oscap -V". Since the output is quite long, some superfluous parts were omitted and replaced by three dots:

```
$ oscap -V
OpenSCAP command line tool (oscap) 1.3.5
Copyright 2009--2021 Red Hat Inc., Durham, North Carolina.

==== Supported specifications ====
SCAP Version: 1.3
XCCDF Version: 1.2
OVAL Version: 5.11.1
CPE Version: 2.3
CVSS Version: 2.0
CVE Version: 2.0
Asset Identification Version: 1.1
Asset Reporting Format Version: 1.1
CVRF Version: 1.1

==== Capabilities added by auto-loaded plugins ====
No plugins have been auto-loaded...

==== Paths ====
Schema files: /usr/share/openscap/schemas
Default CPE files: /usr/share/openscap/cpe

==== Inbuilt CPE names ====
Red Hat Enterprise Linux - cpe:/o:redhat:enterprise_linux:-
Red Hat Enterprise Linux 5 - cpe:/o:redhat:enterprise_linux:5
...
Fedora 32 - cpe:/o:fedoraproject:fedora:32
Fedora 33 - cpe:/o:fedoraproject:fedora:33
Fedora 34 - cpe:/o:fedoraproject:fedora:34
Fedora 35 - cpe:/o:fedoraproject:fedora:35

==== Supported OVAL objects and associated OpenSCAP probes ====
...
```

SCAP Content info

One useful operation that is allowed by oscap is the retrieval of information using the info module. Given an input file, this command generates a description of the file and of its contents. The amount of information displayed can vary greatly depending on the input file type. The command syntax is:

```
$ oscap info filepath
```

In this context, the files located in /usr/share/xml/scap/ssg/content/ that have "ssg-fedora" in their names will be the subject of the test:

- **ssg-fedora-cpe-dictionary.xml**:

```
Document type: CPE Dictionary
Imported: 2021-03-19T15:17:53
```

- **ssg-fedora-cpe-oval.xml**:

```
Document type: OVAL Definitions
OVAL version: 5.11
Generated: 2021-03-19T00:00:00
Imported: 2021-03-19T15:17:53
```

- **ssg-fedora-ocil.xml**:

```
Document type: OCIL Definitions file
```

- **ssg-fedora-oval.xml**:

```
Document type: OVAL Definitions
OVAL version: 5.11
Generated: 2021-03-19T00:00:00
Imported: 2021-03-19T15:18:22
```

In these previous cases, only the document type and little additional information is displayed.

- **ssg-fedora-ds.xml**:

```
Document type: Source Data Stream
Imported: 2021-03-19T15:19:01

Stream: scap_org.open-scap_datastream_from_xccdf_ssg-fedora-xccdf-1.2.xml
Generated: (null)
Version: 1.3
Checklists:
       Ref-Id: scap_org.open-scap_cref_ssg-fedora-xccdf-1.2.xml
              Status: draft
              Generated: 2021-03-19
              Resolved: true
              Profiles:
              Title: OSPP - Protection Profile for General Purpose
                  Operating Systems
                     Id: xccdf_org.ssgproject.content_profile_ospp
              Title: PCI-DSS v3.2.1 Control Baseline for Fedora
                     Id: xccdf_org.ssgproject.content_profile_pci-dss
              Title: Standard System Security Profile for Fedora
                     Id: xccdf_org.ssgproject.content_profile_standard
              Referenced check files:
                     ssg-fedora-oval.xml
                     system:
                         http://oval.mitre.org/XMLSchema/oval-definitions-5
                     ssg-fedora-ocil.xml
                     system: http://scap.nist.gov/schema/ocil/2
Checks:
       Ref-Id: scap_org.open-scap_cref_ssg-fedora-oval.xml
       Ref-Id: scap_org.open-scap_cref_ssg-fedora-ocil.xml
       Ref-Id: scap_org.open-scap_cref_ssg-fedora-cpe-oval.xml
Dictionaries:
       Ref-Id: scap_org.open-scap_cref_ssg-fedora-cpe-dictionary.xml
```

First note that the output for the file *ssg-fedora-ds-1.2.xml* would be the same with the only exception being "Version: 1.2" instead of "Version: 1.3".

The document is composed of one stream that contains one XCCDF checklist. This checklist has three different profiles, and references two files that were previously analysed: *ssg-fedora-oval.xml* and *ssg-fedora-ocil.xml*. Lastly it shows the reference-Id for checks and dictionaries.

- **ssg-fedora-xccdf.xml**:

```
Document type: XCCDF Checklist
Checklist version: 1.1
Imported: 2021-03-19T15:18:22
Status: draft
Generated: 2021-03-19
Resolved: true
Profiles:
        Title: OSPP - Protection Profile for General Purpose Operating
            Systems
                Id: ospp
        Title: PCI-DSS v3.2.1 Control Baseline for Fedora
                Id: pci-dss
        Title: Standard System Security Profile for Fedora
                Id: standard
Referenced check files:
        ssg-fedora-oval.xml
                system: http://oval.mitre.org/XMLSchema/oval-definitions-5
        ssg-fedora-ocil.xml
                system: http://scap.nist.gov/schema/ocil/2
```

This document is an XCCDF checklist that is the same checklist contained in the data stream. The only notable differences are the profile Ids, where only the final part of the names is written, omitting the domain and organization part. This abbreviated version of the profile Ids can be used in the OpenSCAP tools, avoiding the typing of long and complex identifiers.

Scanning the system

The scan of the system can be executed with the following command:

```
$ oscap xccdf eval --profile standard --results ssg-fedora-results.xml \
 /usr/share/xml/scap/ssg/content/ssg-fedora-ds.xml
```

In this case OpenSCAP utilizes the xccdf module to evaluate the checklist contained in the source data stream *ssg-fedora-ds.xml*.

The profile option allows the user to define which profile is going to be used, in this case the Standard System Security Profile for Fedora. The results of the scan are stored in the *ssg-fedora-results.xml* file, as specified by the results option.

This checklist can take a considerable amount of time (10 to 15 minutes), especially in assessing the first rule. When the scan is complete, the results can be easily converted to an HTML report, rendering the results much easier to be examined by a human, with the following command:

```
$ oscap xccdf generate report ssg-fedora-results.xml > ssg-fedora-report.html
```

Alternatively, the execution of the scan and the generation of the report can be issued with one single command, by adding the report option.

The report shows that the tested system satisfied 27 rules, failed 46, and 5 resulted in an error.

Tailoring

Before proceeding further, it can be useful to deselect some specific rules to greatly reduce the time consumed by the scan. The quickest way for tailoring an existing checklist is to create a tailoring file using the SCAP Workbench tool. The tool can be installed with:

```
$ dnf install scap-workbench
```

After opening SCAP Workbench the *ssg-fedora-ds.xml* file can be selected, specifying which profile is the target of the tailoring, and then the specific checks can be selected or deselected. In the context of this example, the *ssg-fedora-ds-tailoring1.xml* is a tailoring file created by deselecting six specific rules from the standard profile:

- Verify File Hashes with RPM;
- Verify and Correct File Permissions with RPM;
- Ensure auditd Collects Information on the Use of Privileged Commands;
- Verify that Shared Library Files Have Restrictive Permissions;
- Disable Kernel Support for USB via Bootloader Configuration;
- Set Default firewalld Zone for Incoming Packets.

The first four rules were deselected to reduce the time spent during the scan, while the last two were removed to avoid possible problems that could arise if remediated: one disables all devices connected via USB, the other could deny access to the system if not properly handled.

Remediation

It's now possible to quickly scan the system with the provided tailoring file:

```
$ oscap xccdf eval --profile standard_custom_1 \
--tailoring-file ssg-fedora-ds-tailoring1.xml \
--results tailoring-1-results.xml --report tailoring-1-report.html \
 /usr/share/xml/scap/ssg/content/ssg-fedora-ds.xml
```

The new report shows 25 passed rules, 42 failed and 5 errors.

OpenSCAP allows automatic remediation of checks if the relative XCCDF file contains instructions on how to fix the problem. The checklists provided by SCAP Security Guide have such fix elements for certain rules. The remediation procedure is performed by simply adding the remediate option (**note**: the remediation scripts require root privileges to be executed):

```
$ oscap xccdf eval --remediate --profile standard_custom_1 \
--tailoring-file ssg-fedora-ds-tailoring1.xml \
--results tailoring-1-remediation-results.xml \
--report tailoring-1-remediation-report.html \
 /usr/share/xml/scap/ssg/content/ssg-fedora-ds.xml
```

Oscap performs a normal evaluation of the system, searching for the fixes of the failed rules. Then it executes the remediation scripts and re-evaluate the rules, to check if the fixes were successfully applied. After this operation, the new results are that 64 rules passed, 5 failed and 3 reported an error.

After further inspection of the report, it is clear that the 5 failed rules had no suitable fix, as the XCCDF file did not contain a remediation shell script for them. On the other hand, the 3 rules that resulted in an error all have a remediation script, but that script can be applied only on certain systems that have a specific package installed. Since the test system does not present that package, the remediation script simply ended with "Remediation is not applicable, nothing was done".

3.5.2 Vulnerability scanning on Ubuntu

The target system of this example is a Ubuntu 20.04 (Focal) machine in which OpenSCAP Base was installed with the following command:

```
$ apt-get install libopenscap8
```

The available version for this distribution is 1.2.16

The Ubuntu foundation, and in particular the security team of Canonical Ltd, provides OVAL data for all the supported Ubuntu releases. These data are SCAP conformant and can be utilized by any third-party SCAP tools.

Ubuntu's OVAL data are a collection of OVAL definitions that can check whether the system is affected by known vulnerabilities and can determine if the appropriate security patches are available for the system.

The instructions can be found at:

https://ubuntu.com/security/oval

First, the following command is needed in order to download the correct OVAL data for the target system:

```
$ wget https://security-metadata.canonical.com/oval/com.ubuntu.$(lsb_release
    -cs).usn.oval.xml.bz2
```

In this example, the downloaded file becomes *com.ubuntu.focal.usn.oval.xml* after decompressing.

After that the CVE assessment can be performed with OpenSCAP:

```
$ oscap oval eval --results oval-vuln-results.xml --report \
    oval-vuln-report.html com.ubuntu.focal.usn.oval.xml
```

The report inspection shows that the total number of OVAL definitions is 520, of which 519 are patch definitions. The only non-patch definition is an inventory definition that checks if Ubuntu 20.04 is installed.

Of the 519 patch definitions 473 resulted as false, meaning that the system is not affected by those vulnerabilities, and 46 are true.

The report presents these 46 vulnerable states alongside useful links that instruct on how to remediate the system. Often a security patch is already released and can be installed to prevent any possible exploitation of those vulnerabilities.

The presence of a considerable amount of vulnerabilities was expected on the target system, as at the time of evaluation it hadn't been updated for approximately six months. After the installation of all the most recent patches available, a new assessment was made with the same OVAL data and this time no vulnerability was reported. By this example alone it clearly emerges how important patch management is in preventing malicious attacks, as by simply updating the system all of the 46 previously found vulnerabilities were eliminated.

Chapter 4

Security automation in virtualized environments

4.1 Network Functions Virtualization

Network Functions Virtualization (NFV) is a network architecture concept that consists in the virtualization of some common network services. These kind of services include routers, firewalls, intrusion detection systems (IDS), load balancers, and many other that traditionally require a dedicated hardware device.

NFV allows the creation of multiple virtual machines or containers on a single hardware device, each of them providing a dedicated network functionality via software. This eliminates the need of additional hardware components that are often proprietary, reducing the economic cost of a network and facilitating interoperability.

Another major benefit of NFV is a greatly improved scalability and agility, as new services can be added or updated without the need of new components, thus allowing service providers to change their offers based on the customer's need.

4.1.1 NFV structure

The reference for NFV architecture is developed by the European Telecommunications Standards Institute (ETSI), that defines standards that can reach the high level of reliability and performance required by network services, and promotes better stability and interoperability [8].

The NFV structure is comprised of three main components:

- Virtualized network functions (VNFs) are the actual software implementation of network functions;
- Network functions virtualization infrastructure (NFVi) includes all the hardware and software components that are needed to build and run the NFV. Local machines, the network itself, and the virtualized network functions are all part of the NFVi;
- Management, automation and network orchestration (MANO) provides the framework for managing NFV infrastructure, including the deployment of new virtualized network functions.

4.1.2 NFV security considerations

The advent of NFV and its growing use poses some security issues, as new software flaws can generate new vulnerabilities. The ability of NFV to automate the creation and management of

new network functions brings additional potential exploits in network configuration, orchestrator management, and malicious misconfiguration.

NFV also provides some security benefits, in particular it can help in mitigating distributed denial of service (DDoS) attacks: an NFV orchestrator can automatically instantiate new VNFs to handle the surge in incoming traffic, greatly reducing the incident response time [9].

The security flaws of NFV emerge when the security threats associated to physical networking intersect the security issues of virtualization (i.e. virtual machines or containers). It follows that checking the security of virtual machines and containers that deploy the network functions is of vital importance, and it's precisely the objective of the security group of Open Platform for NFV (OPNFV), discussed in the following section.

Numerous kinds of attacks can be performed against an NFV environment, and security best practices are already available in the form of security guidances. The effective implementation of these security guidelines can, at least in principle, be automatically checked by assessment tools such as OpenSCAP.

4.2 Open Platform for NFV

Open Platform for NFV (OPNFV) is a collaborative project supported by the Linux Foundation. It's an open-source effort that aims at facilitating the deployment and integration of NFV. The OPNFV community follows the standard references for NFV and promotes the use of open-source conformant tools. It is also actively engaged in developing, testing and deploying tools for an NFV infrastructure, with particular emphasis on conformance and performance [10].

One group in particular, the OPNFV Security Group, is dedicated to improve the security of many aspects related to OPNFV. It follows multiple projects such as OPNFV Security Vulnerability Management, provides secure coding guidelines, and automated security scanning. This last project makes use of OpenSCAP and it's discussed in the following section.

4.2.1 OPNFV Security Scanning

The Security Scanning project aims at providing compliance checking and vulnerability assessment of a NFV infrastructure. Being a child project of OPNFV, it follows a standardized framework and utilizes an open-source tool: in this case SCAP was chosen, alongside OpenSCAP.

The project is designed to be included in an automated process. Given an NFVi, it retrieves all the nodes and performs a security scan on each of them utilizing SCAP content. The security evaluation is performed by the OpenSCAP tool and can consume standard SCAP content.

Depending on the given input content, the security scan can guarantee compliance to the desired security policy and checking the presence of known vulnerabilities.

The project has three main areas of development, each of them with different scopes: the deployment of security scan in a CI/CD process, the creation and support of SCAP content, and a standalone application development of Security Scanning [11].

Integration in a CI/CD process

The OPNFV project has developed a Continuous Integration / Continuous Delivery process, and uses the open-source software Jenkins as automation server. The OPNFV security group incorporated the code for automatic scanning in the Jenkins build platform.

The following list sums up the steps needed to perform the security evaluation of a NFV infrastructure:

1. The OPNFV Jenkins build is initiated.

2. A dedicated security script is executed, and it's given a configuration file.

3. The IP address of each node of the NFVi is gathered.

4. The appropriate scanning profile is selected according to the node type.

5. OpenSCAP is remotely installed on each node.

6. The security scan is performed on each node.

7. Security reports are downloaded back to the local system.

The configuration file mentioned in step 2 contains the information needed to differentiate the scan between different kinds of NFVi nodes. It can easily be modified to include new host types or modify the evaluations.

This is an example of configuration file:

```
[controller]
user = heat-admin
scantype = xccdf
secpolicy = /usr/share/xml/scap/ssg/content/ssg-rhel7-xccdf.xml
cpe = /usr/share/xml/scap/ssg/content/ssg-rhel7-cpe-dictionary.xml
profile = stig-rhel7-server-upstream
report = report.hmtl
results = results.xml
reports_dir=/home/opnfv/functest/results/security_scan/
clean = True
```

Additional host types can be specified inside additional square brackets. As the *secpolicy* and *cpe* fields show, the contents of SCAP Security Guide are installed on the machines alongside OpenSCAP. This allows the selection of the appropriate benchmarks and profiles for a wide variety of use-cases.

The last value of the configuration file indicates whether the installed packages and the generated reports should be eliminated from the nodes or not. If the value is set to "True" all of the scan traces are removed, ensuring an unobtrusive evaluation.

Regarding the third step, the IP address gathering can be performed in a number of ways. Currently it utilizes the OpenStack project "nova", that contains the appropriate APIs for virtualized servers.

SCAP content authoring

The OPNFV security group is also involved in the creation of SCAP content that can reflect the wide variety of operating systems and applications that are part of a NFVi.

Due to the intrinsic nature of a NFV environment, which can include countless different software versions and various nodes, a considerable amount of diverse SCAP content is needed. The production of useful SCAP content is non-trivial and requires a good amount of technical knowledge.

The ultimate goal of OPNFV security group is to incorporate new SCAP content directly in the SCAP Security Guide project, so that all kinds of users can easily access the already developed SCAP content. There is already a wide variety of operating systems and applications that have dedicated benchmarks in SSG, but the production of new content is always needed in order to extend the support to an even wider selection of use-cases.

Standalone application

A standalone Security Scanning application has been proposed, which should provide the security functionalities outside the strict context of OPNFV projects. It should be possible to autonomously deploy the Security Scanning application with its own installation scripts.

Unfortunately this project is still not deployed, and the security scanning functionalities reside only as part of the OPNFV Functional Testing ("Functest") project.

OPNFV Functional Testing

The Functest project provides a framework for testing the OPNFV platform for NFVi. It includes testing methodologies, test suites and test cases that can verify the functionalities of a OPNFV-deployed NFVi.

The project page is located at:

https://wiki.opnfv.org/display/functest/Opnfv+Functional+Testing

Amongst the many functionalities provided by Functest there is the security scanning of NFVi nodes. This framework serves as the CI/CD context previously discussed.

Functest is available as a Docker image on the public dockerhub registry[1]. After the image is pulled and the container is properly deployed, all the functionalities can be exploited from within the container.

Functest contains many test suites; one of them, "Features", implements the Security Scanning steps previously mentioned. Although some simple scans can in fact be performed, the project has not been fully developed and expanded.

The most critical limitation is the support for package installers: different nodes of an NFVi may require different connection methods by which the installers can connect. The package installers are fundamental in the process as they are needed to install OpenSCAP and the relative content to each node. The only installer currently supported is the Oracle Apex environment.

4.3 Container remediation

A number of OpenSCAP utilities were presented in section 3.3. Two of them in particular, oscap-podman and oscap-docker, can scan images or running containers checking for compliance to a security policy and/or perform a vulnerability assessment.

Both these utilities mount the image's file system in read-only mode, adhering to the so-called offline scanning technique. This guarantees that the scanning operation doesn't harm or modify in any way the image, that can be a useful feature in some cases but also a great limitation in others.

Let's take in consideration a possible scenario in which applying container remediation could be useful. The OpenSCAP scanner, by itself, already provides multiple remediation possibilities. All of these possible actions cannot be performed by the two oscap utilities, as they cannot modify the original image or container.

Oscap-podman and oscap-docker can potentially generate fix scripts and save them to a file, using the review mode for remediation. The first step is to perform the desired scan specifying the result file:

```
$ oscap-podman IMAGE-ID xccdf eval --results results.xml \
/usr/share/xml/scap/ssg/content/ssg-rhel8-ds.xml
```

[1] https://hub.docker.com/r/opnfv/functest/

Then the specific results ID can be retrieved with the info command:

```
$ oscap-podman IMAGE-ID info results.xml
```

And finally with the obtained ID the appropriate fix document can be generated:

```
$ oscap-podman IMAGE-ID generate fix --fix-type bash --output fix-script.sh \
--result-id RESULT-ID results.xml
```

Now the remediation script can be downloaded back to the original system and can be manually reviewed by a security expert, which will then apply the desired remediations by executing the scripts on the running container.

The biggest issue with this approach is that it eliminates the automation in the process. The aforementioned procedure is time consuming and impractical, especially in highly virtualized environments where there can be hundreds of running containers that each require different remediations.

The next section presents one possible solution to this issue, a tool that allows for automatic remediation of images or containers.

4.3.1 The Atomic scan tool

Atomic scan is a scanning tool developed by Red Hat developers that aims in particular at detecting, and possibly remediating, container vulnerabilities [12].

The development process followed some strict requirements:

- The tool shouldn't be installed on the main host, and also shouldn't run on very high privileged containers. This approach eliminates by design multiple potential security issues.
- Multiple container frameworks should be included as possible targets, i.e. the scan capabilities should not to be limited to a single framework such as Docker.
- The tool should score high in performance, with times of execution short enough to be scalable to a high number of containers.

The Atomic tool is able to download and run containers that include scanning tools. In order to avoid security-related issues, the scanning tools themselves aren't privileged. Atomic can mount read-only root file systems from the host machine, then pass these file systems to the scanning container.

The Atomic scan tool is already installed in every atomic image but can still be installed on other systems, such as Fedora or CentOS, with the proper installer command:

```
$ yum install atomic
```

The structure of the tool is shown with the use of the help option, that prints the following output:

```
atomic scan [-h] [--scanner {available_scanners}]
 [--scan_type SCAN_TYPE] [--list]
 [--all | --images | --containers]
 [scan_targets [scan_targets ...]]
```

Atomic utilizes by default the OpenSCAP scanner, which is bundled together in the same package, but can also utilize different scanners if they are properly configured. The *scanner* option lets the user specify which scanner to use. The *scan_type* option is used to specify the scan operation that the scanner needs to perform.

The last options allow the user to specify if the target of the scan: it can either be all the images, all the containers, or both. Alternatively, the ID of single images or containers to be scanned can be listed.

Atomic scan is quite a flexible tool, in the sense that any scanner can be plugged-in to satisfy a wide variety of users and to support multiple use-cases. The scan types can be configured to perform compliance checking or vulnerability scanning for different images.

4.3.2 Scanning and remediating containers with Atomic

The following steps describe how to scan and remediate an image that is based on a Red Hat product, in this case Red Hat Enterprise Linux 7 (RHEL 7) [13]. This target was chosen because Red Hat-based products are the ones best supported by Atomic, OpenSCAP and SCAP Security Guide (SSG).

First, the OpenSCAP container image has to be downloaded from the Red Hat Container Catalogue (RHCC)[2], as it already contains the SCAP content of SSG. This enables the evaluations using all the profiles and options contained in SSG. The image can be downloaded with the command:

```
$ atomic install rhel7/openscap
```

A standard compliance scanning will be performed first. With the Atomic list command, the list of available scan types is shown. The *configuration_compliance* scan gives access to some common compliance checking benchmarks. It is now possible to scan a target image by running the following command:

```
$ atomic scan --scan_type configuration_compliance IMAGE_ID
```

An interesting feature of Atomic is that the specified target image can also be a Red Hat image that is not present on the host system, in which case Atomic will automatically pull the image directly from the registry. This process occurs when the image ID is replaced with the image name in the format *registry.access.redhat.com*, followed by the intended version (e.g. "rhel7:latest" or "rhel7/rhel-tools").

The Atomic tool generates a subdirectory for storing the scanning results. The directory is located at */var/lib/atomic/* and takes the name of the utilized scanner, which in this case is *openscap*, plus a time stamp of when the evaluation was performed. This new directory can be accessed to retrieve the ARF results file and optionally the scan reports in HTML format, if the *report* option is specified.

After this example of standard scanning, now the remediation process will be analysed. From the viewpoint of a user, there isn't much difference in the process. By simply adding the *remediate* option in the previous command, Atomic will apply the remediations:

```
$ atomic scan --remediate --scan_type configuration_compliance IMAGE_ID
```

It's important to note that the remediation process results in the creation of a new container image with an altered configuration, which is built on top of the original image as a new layer. The original image is not altered per se, but the new layer contains all the configuration improvements.

Due to the intrinsic structure of images, that are made of layers that are always added on top of each other, the result of the Atomic remediation effectively solves the "container remediation" problem.

One last additional consideration is needed: as the structure of the image changes, meaning that the resulting image is different from the original one, any digital signature that was originally posed on it will no longer be valid. There are certain scenarios in which this issue has to be addressed. A company should for example apply a new signature to the new image with the company's own signatures.

4.3.3 Container scanning integration in a CI/CD process

As the prevalence of container-based environments is growing, especially in large and complex systems, the need of having a reliable and well-functioning security process grows accordingly.

[2] https://catalog.redhat.com/software/containers/search?q=OpenSCAP

One of the ways in which this can be accomplished is by introducing automated scanning and remediation of containers in a CI/CD pipeline [14].

Especially in the context of companies, having a robust and secure catalogue of hardened containers is vital, as it ensures that the tools that interact in the CI/CD process can reliably pull images. Thus the need of constantly updating an image repository by pushing the security-remediated images.

The CI/CD integration also offers continuous enforcement and provides uniformity of security policies.

The first component of this process is an automation server such as Jenkins[3], that enables the creation of pipeline steps to scan, remediate and push back secured images to a repository.

The pipeline can be built in numerous ways, but in this example is comprised of three steps:

1. Pull an image from a repository;
2. run the Atomic tool to scan and remediate the image;
3. assign a new image tag and push back the secured image into the repository.

The repository of step one, from which to pull the image, depends on the single organization that manages it. It can be a public Docker repository, or more likely a private repository of the company, allowing a much stricter control over the source of images and their security.

Regarding the second step, the Jenkins dashboard allows parameters to be passed to the Atomic tool. These parameters can specify the name of the target image, and one possible configuration of these variables can be:

```
${REPOSITORY}
${IMAGE-NAME}
${VERSION}
```

That represent respectively the name of the repository, the image name and the image version. These three parameters allow for a simple identification of the scan target. The command needed for this step takes this form:

```
$ sudo atomic scan --remediate --scan_type configuration_compliance \
       ${REGISTRY}/${SRC_IMAGE}:${VERSION}
```

Additional options can be included to generate human-readable reports and possibly pushing them to a dedicated section in Jenkins, allowing further inspections by security experts. The Jenkins ecosystem is quite wide and many additional plugins are available that provide this feature. One example could be HTML Publisher[4] that can push the reports directly in the Jenkins dashboard.

[3] https://www.jenkins.io/

[4] https://plugins.jenkins.io/htmlpublisher/

Chapter 5

Limitations and areas of improvement

Despite being well supported and used by many stakeholders, the SCAP framework and its derived tools still present some limitations, especially when responding to the rapidly growing need of protection from vulnerabilities.

This chapter describes some of the limitations of SCAP at various levels of abstraction. This includes both the issues that are ascribed in particular to the OpenSCAP tools, as well as the limitations of the SCAP components themselves. SCAP as a whole is also considered and suggestions on the possible incorporation of new components are provided.

Some of the issues that are addressed in the following sections are considered critical and are currently being addressed by the SCAP team of NIST. The resolution of these issues is the main goal of the next significant update to SCAP version 2.

5.1 CPE and SWID tags

Regarding the use of SCAP for inventory purposes, the current approach relies heavily on CPE as a base for software and hardware identification. CPE was never intended to be a software inventory standard, but rather a software identifier, and has very poor scalability.

There has been a rapid growth of CVE assignments in recent years: in the years 2005 to 2016, the number of new CVE-assigned vulnerabilities ranged between 4 to 7 thousand. Then in 2017 the number jumped to 14,645 and never stopped growing, reaching its maximum in 2021 with just over 20 thousand new vulnerabilities discovered. Figure 5.1 shows the number of new CVE added in the NVD each year, with colours representing the CVSS severity distribution. Data for 2022 are updated as of March 16.

The NVD analyses all the new vulnerabilities and produces CPE names in order to identify the affected assets, which in turn means that for an effective vulnerability scanning the CPE mapping should be predictable and unique. This is definitely not the case, as it will be shown later below.

SWID tags appear to be the next replacement for CPE, as they solve some of the CPE's key problems.

First, SWID tags can be produced by the software vendors and are managed directly on an endpoint by software means, which is much more scalable and supports a larger portion of inventory use cases. Second, SWID tags allow for a much clearer characterization of software and identification of patches.

The SCAP community is actively encouraging the adoption of SWID tags as standard for software inventory, facilitating the integration of SWID tags in the software release process. One of the goals of SCAP version 2 is the transition to the use of SWID instead of CPE, which will be deprecated [15].

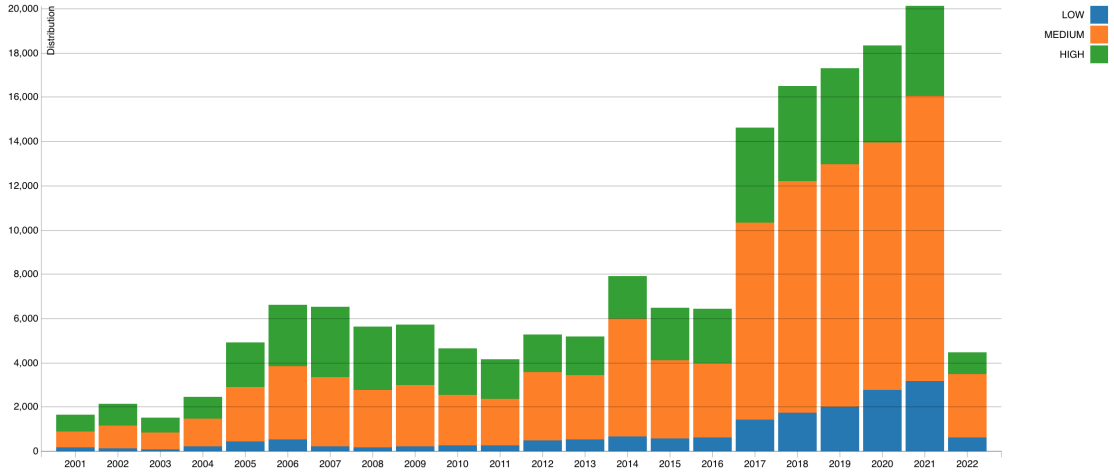


Figure 5.1. Number of new CVE by year. Colours represent CVSS severity. (source: NVD).

5.1.1 CPE limitations for open-source software

One critical area in which CPE shows its limitations is the identification of open-source software and libraries. Open-source libraries come from a wide variety of sources and lacks a centralized control on their naming. Additionally, the pace at which new libraries are created or updated hinges on the CPE scalability issue.

The CPE mapping procedure between open-source libraries and CPE entries is not well defined, leading to a relationship that isn't one-to-one, but rather multiple libraries can be mapped to a single CPE name.

Open-source libraries often come with very generic and inconclusive names and authors. Developers often use many of these libraries in their projects: both the developers and the final users of the projects would strongly appreciate a vulnerable-free software, at least with respect to publicly known vulnerabilities. The problem here arises because there is no definitive way of mapping the names and authors of the libraries to CPE names, thus it's impossible to reliably search for vulnerabilities that affect the specific libraries used in the software [16].

Let's take as example a very common Node.js library, **unzipper**[1], which has over five million monthly downloads. Starting from the package information it should be possible to create a meaningful CPE name that can then be used to identify which CVEs, if any, affect a specific version of the library.

The first attribute of the CPE name, *part*, surely cannot be the one for operating systems or hardware devices, so it will be "a" for application. The second attribute should be the person or organization that created the product. There is no clear information directly in the NPM page, so a closer inspection of the GitHub repository of the project is needed.

The repository is located at:

https://github.com/ZJONSSON/node-unzipper

Some useful information can be retrieved from the *package.json* file partially reported here:

```
{
  "name": "unzipper",
  "version": "0.10.12",
  "description": "Unzip cross-platform streaming API ",
  "author": "Evan Oxfeld <eoxfeld@gmail.com>",
```

[1] https://www.npmjs.com/package/unzipper

```
  "contributors": [
    {
      "name": "Ziggy Jonsson",
      "email": "ziggy.jonsson.nyc@gmail.com"
    },
    {
      "name": "Evan Oxfeld",
      "email": "eoxfeld@gmail.com"
    },
  ...
  "repository": {
    "type": "git",
    "url": "https://github.com/ZJONSSON/node-unzipper.git"
  },
  ...
}
```

Since no organization is ever mentioned, it is safe to assume that the vendor attribute should be some form of the author's name or, less likely, the name of one of the contributors.

Lastly the product name should simply be "unzipper". It is now possible to search for matching CPE names directly in the NVD, which then provides the list of CVEs that affect the products identified with that CPE name.

The NVD search engine for CPE names is located at:

https://nvd.nist.gov/products/cpe/search

The results are always empty for any combination of names. The problem lies in the vendor attribute, that cannot be found from the available information. Only if the vendor field is omitted, then the search gives some meaningful results.

In particular, there are 61 matching records if the input query is "cpe:2.3:a:*:unzipper". All these results are different versions of products named "unzipper" from two different vendors: "r-company" and "unzipper-project". The first one targets android devices, while the second targets Node.js and so it refers to the intended library.

This example shows the arbitrary assignment of CPE names to software products and the not so easy process of CPE mapping that should be far more precise and reliable: the "unzipper project" is never mentioned in any resource related to the unzipper library, and there is no clear reason that the vendor attribute in the CPE name should be as it is.

The complexity of this process could potentially lead to the unconscious use of a vulnerable product: this is the case of unzipper for versions before 0.8.13 that do have a vulnerability (CVE-2018-1002203[2]) that can easily be missed if using only CPE names as a source for vulnerability assessment.

5.2 Continuous monitoring

At the current state of development, most tools that implement SCAP use a periodic scanning approach for collecting system's data and evaluating security: this is the case for OpenSCAP Daemon that can, as previously stated, generate tasks for scheduled evaluations.

Periodic scanning isn't an ideal solution, mainly for two reasons:

- There is the need for a trade-off between the level of confidence that in any given moment the system actually adheres to the desired state, and the workload that's put on the scanning machine itself. The first aspect leads to a higher scanning frequency, while the second poses a ceiling to that frequency.

[2]https://nvd.nist.gov/vuln/detail/CVE-2018-1002203

- A fixed-interval pre-scheduled scanning can leave the system vulnerable for lengthy amounts of times, ranging from several hours to even days.

Another even worse limit of periodic scanning is that clever enough attackers can erase their presence or effects on the system between subsequent scanning, possibly preventing the detection of critical security events.

The proposed solution for these problems is event driven reporting: the SCAP version 2 architecture will include new standards that allow the automatic sending of notification when a security related event occurs.

This change of structure will allow future tools to respond in real-time to system modifications that can generate security issues. When the event is detected a new evaluation can be performed and, when possible, remediations can be applied automatically.

In case no automatic remediations are possible, the system will still signal the issue so that the security administrators can promptly intervene [15].

5.3 Container scanning and remediation

Checking for compliance and scanning for vulnerabilities in an image or a container can be done effectively by some of the OpenSCAP utilities, but they cannot remediate those images.

When it comes to image scanning and remediation, chapter 4 presented a solution using in particular the Atomic scan tool. Unfortunately, this tool has been recently discontinued, so no further development is in progress. There is no guarantee that the Atomic tool will be supported in the future. The functionalities and scope of the Atomic project has been replaced by Podman [17].

An exact replacement of the container remediation functionality of Atomic is still not available in Podman. An open issue in the Podman GitHub repository addresses this missing feature:

https://github.com/containers/podman/issues/8305

The implementation of a Podman "scan" command is technically in progress, although there has been very little activity on this issue in the last few months.

There is a workaround that can mimic the steps of the Atomic scan command, which is to create an image with an installed scanner, such as OpenSCAP, and then run the image to scan the other target images. It's still not a complete solution as it lacks the automation and the flexibility of Atomic scan, meaning that it requires a manual configuration that can be different case by case.

5.4 Incorporating CWE and CAPEC

The SCAP standard is continuously expanding to new components and refining the already included ones. The addition of Common Weakness Enumeration and Common Attack Pattern Enumerations and Classifications in the SCAP work flow and their integration with existing tools can enhance the capabilities of SCAP.

Common Weakness Enumeration (CWE) is a structured catalogue of software and hardware weaknesses that can have security repercussions. The term "weakness" is used in a general sense and indicates security flaws at different levels such as:

- Hardware architecture design;
- faults in software code;
- bugs in software or hardware implementation.

The community-developed CWE list is maintained by MITRE and it enumerates, classifies and describes the weaknesses in a hierarchical structure. It serves as a language for communicating weakness-related information in a consistent manner between vendors, consumers, and security experts [18].

CWE primarily aims at preventing the development of flawed products from the very early stages of design, providing a guide for developers and security practitioners. The use of CWE also provides these additional benefits:

- Allows an effective description and discussion of software and hardware weaknesses in a common language.
- Facilitates the check for weaknesses in existing software and hardware products.
- Evaluation of tools that target these weaknesses.
- Provides a common baseline standard for weakness identification, mitigation, and prevention efforts.
- Prevent software and hardware vulnerabilities prior to deployment.

5.4.1 CWE List

The CWE list is the dictionary of all the CWEs and can be found at:

https://cwe.mitre.org/data/index.html

First developed in 2006, it initially included software weaknesses only. Since 2020 the support for hardware weaknesses has been added due to the increasing concern about hardware related issues, as seen in recent years with the discovery of vulnerabilities like Meltdown, Spectre, or Rowhammer.

The list is continually refined and updated with each new release. It is also fully searchable and downloadable, but an interesting aspect is the possibility of navigating the CWE list on selected "views". The views organize the CWE structure in a specific manner so that it can best serve the specific needs of a user. These unique viewpoints are divided in standard views, external mappings, and additional helpful views.

Standard views

The first standard view is "Research Concepts", that focuses on weakness behaviour. This view is expected to include every CWE and can be used to identify theoretical gaps within CWE. It's mainly intended for researchers and analysts that work with a high level of abstraction and are interested in conceptual dependencies between weaknesses. This view follows a deep hierarchical structure with more levels than any other view.

The second is "Software Development". This view organizes weaknesses around concepts that are frequently used or encountered in software development, therefore it's primarily aimed at developers or educators. It provides a variety of categories that include all aspects of the software development life cycle including architecture, design, and implementation.

The third view is "Hardware Design", that organizes weaknesses around concepts that are frequently used or encountered in hardware design. This view can align closely with the perspectives of hardware designers, manufacturers and educators.

External Mappings

This family of views typically represent a subset of weaknesses that are related by some external factors.

Some views are a "Top N" list of weaknesses: the CWE Top 25 is an annual list of the 25 most dangerous software weaknesses that is created by analysing the number and severity of new vulnerabilities that appear in the NVD. The OWASP Top 10 instead provides the most important issues for web application developers.

Other external mappings view include:

- Most Important Hardware Weaknesses List;
- Seven Pernicious Kingdoms;
- SEI CERT Oracle Coding Standard for Java;
- SEI CERT C Coding Standard;
- CISQ Quality Measures.

Helpful Views

These are additional views that were created in order to match some commonly encountered use cases. Some of them focus on a particular moment of the software life cycle, other views are specifically targeting a programming language and its most common associated weaknesses:

- Introduced During Design;
- Introduced During Implementation;
- Quality Weaknesses with Indirect Security Impacts;
- Software Written in C;
- Software Written in C++;
- Software Written in Java;
- Software Written in PHP;
- Weaknesses in Mobile Applications.

Lastly, some views were created to show only selected CWEs such as CWE Composites, CWE Named chains, CWE Cross-section, and CWE Simplified Mapping.

5.4.2 CWE in relation to SCAP

Although CWE is not included amongst the SCAP components, it still plays a role in the scoring of vulnerabilities. The NVD utilizes a subset of the CWE list (a "CWE Slice") that allows to differentiate between vulnerability types and helps the NVD analysts in assigning a score to the CVEs [19].

The CWE structure is composed of several levels that become increasingly specific: from a single CWE that generally indicates a type of weakness can follow many different specific weaknesses, that can often be themselves the parents of even more specific weaknesses.

For example, from CWE-74 "Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')" can be further specified the CWE-79 "Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')". Different variants of cross-site scripting are classified with different CWE identifiers, for example CWE-83 "Improper Neutralization of Script in Attributes in a Web Page".

This hierarchical structure allows the NVD analysts to evaluate and score vulnerabilities at different levels of granularity, which is a needed feature as CVEs can vary greatly in specificity.

In turn, the NVD also influences some CWE views. As previously mentioned, the CWE Top 25 is calculated each year from the NVD data. The ranking is calculated by counting how many CVE entries map to a specific CWE with respect to the total number of CVEs (i.e. the frequency that the CWE is the root cause of a vulnerability), multiplied by the average CVSS score of those vulnerabilities.

This scoring procedure orders the weaknesses by how prevalent they are and how dangerous can an exploitation be on average.

5.4.3 Scoring weaknesses with CWSS

The Common Weakness Scoring System (CWSS) is a mechanism for prioritizing software weaknesses that was developed by MITRE alongside CWE. It's the result of a community-based effort to provide an open, consistent and robust scoring system [20].

The three main capabilities provided by CWSS are:

- A quantitative measure of the severity of unfixed weaknesses present in a software;
- a common framework that allows the coordination between different people and interoperability between tools;
- a customized prioritization of weaknesses according to the specific needs and environment of a company.

CWSS scoring is mainly used by security analysts, both directly and by using dedicated code analysis tools: typically an initial score is automatically calculated by a tool, then the analyst incorporates and refines the score metrics to produce a final score.

The other important use of CWSS lies in the making of ranking lists such as CWE Top 25 or OWASP Top 10, where CWSS scores are used in conjunction with multiple other metrics to prioritize weaknesses.

CVSS vs CWSS

CVSS is perhaps the most similar scoring system to CWSS from a conceptual viewpoint (both are comprised of several metrics that can be compared and in some way overlap), but the two differ in usage scenarios.

CVSS is a reactive measurement that is calculated after a vulnerability is discovered, analysed and verified. CWSS can be used much earlier in the process, identifying weaknesses before they are fully examined to determine if they will result in vulnerabilities. In this sense CWSS is a proactive scoring system.

CVSS doesn't allow for incomplete information: a score can be calculated only if every metric has a value assigned. The NVD approach in the case of vulnerabilities with incomplete information is to assign the maximum possible score in order to get a conservative score. This approach leads to inflating scores and can be viable only when the missing information is very limited.

In the case of weaknesses the scarcity of information is very common and CWSS addresses this scenario by explicitly supporting missing metrics, allowing CWSS to still provide a meaningful score for these weaknesses.

5.4.4 CAPEC

As CWE functions as a detailed catalogue of weaknesses, Common Attack Pattern Enumeration and Classification (CAPEC) categorizes attack patterns that are commonly used by adversaries in exploiting those weaknesses. The term "attack pattern" is used to describe the actual mechanism and techniques by which an attacker can successfully breach a system [21].

The structure of CAPEC is similar to CWE, but the two can be considered complementary to each other: CWE focuses on the intrinsic attributes and characteristics of weaknesses and their structural relations, while CAPEC approaches weaknesses from the viewpoint of an attacker, describing the actual implementations of techniques used to exploit weaknesses. In this case the relation between attack patterns follows the practical approach of attackers, highlighting possible chains of actions that can lead to a successful exploitation of weaknesses.

CAPEC can provide additional information regarding a weakness, and it's especially helpful in detecting possible attacks to a system: even if a company knows which weaknesses affect their system, information retrieved by CWE alone may not be sufficient to actually check whether those weaknesses are being exploited or not. On the other hand, CAPEC specifically describes the mechanism by which an exploitation is carried out, which allows the security analysts to look for specific patterns in their system and compare them to the typical attack patterns.

CAPEC integration with SCAP can be difficult as there isn't any SCAP component which is directly linked to it. Nevertheless, once CWE is incorporated the subsequent integration of CAPEC is much more feasible.

The utility of its integration can be found especially in the context of vulnerability assessment and in the monitoring of the system, enabling an easier and possibly faster threat detection by checking for the traces of the specific patterns used by attackers.

5.5 OpenSCAP limitations

The OpenSCAP organization maintains multiple repositories on their GitHub page:

https://github.com/OpenSCAP

The community of developers and contributors is still very much active in many projects related to OpenSCAP Base and other aspects of SCAP. Many aspects described below are currently tracked in one or more open GitHub issues.

5.5.1 Additional SCAP components

There are two component specifications that are part of SCAP that do not have a dedicated module in OpenSCAP, but can still be handled by the tool correctly. These components are SWID tags and Asset Identification.

The presence of SWID tags on a system can be assessed, using OpenSCAP, by using OVAL inventory class definitions. These definitions can be part of a SCAP source data stream or a standalone OVAL Definition file. To the user, there is no difference between the procedure of identifying SWID tags or a standard evaluation of SCAP content. Both the commands *xccdf eval* in the case of source data streams, and *oval eval* in the case of OVAL Definition files, perform SWID tags detection if the input files contain a reference to an OVAL inventory class definition that searches for the presence of a matching SWID tag.

The Asset Identification (AI) specification, in the context of the SCAP standard, is limited to the identification of the target asset, which is the system that is put under evaluation by a source data stream. OpenSCAP correctly handles AI, incorporating its use in ARF reports. In particular, XCCDF and OVAL results files contain the information of the target asset of evaluation, that is then represented in the dedicated AI part of the ARF report. Typically the AI fields that are used to identify the target asset are its connection addresses (MAC, IPv4 and IPv6), the hostname, and the fully qualified domain name.

5.5.2 Unsupported SCAP components

The SCAP Validation Program by NIST certified OpenSCAP as SCAP conformant, specifying two product capabilities: "Authenticated Configuration Scanner" and "Common Vulnerabilities and

Exposures (CVE)". In total, the SCAP Validation Program can verify three product capabilities: the one that is missing is "Open Checklist Interactive Language (OCIL)" [7].

OCIL functionalities are not implemented in OpenSCAP. Nevertheless, the tool can detect and identify OCIL checklists: the *info* module, when given an OCIL file as input, simply outputs "Document type: OCIL Definitions file".

If the input file is a source data stream containing OCIL checks, those checks are identified with their relative reference Id and listed as referenced check files. When unpacking bundled source data stream, using the *sds-split* command, if an OCIL check file happens to be among the extracted files it will be shown in the list of files. In any case, OCIL checks cannot be performed using OpenSCAP.

Lastly, the SCAP specification states that the scoring system for software vulnerabilities should be CVSS version 3, while OpenSCAP only uses the 2.3 version. The differences between the two versions are not critical, but several changes occurred.

For example, in the Base metric group two new metrics were added (User Interaction and Privileges Required), and a new value (Physical) was incorporated in the Attack Vector metric. Another change example is in the Environmental group, with the introduction of Modified Base Scores. Even the number of severities were increased from three (Low, Medium, and High) to five (adding None and Critical).

Chapter 6

Proposed solutions

Of the limitations presented in the previous chapter three areas in particular were selected to be addressed, with the objective of reducing some of those limitations and developing new solutions that can be applied in many use-cases, widening the range of meaningful utilization of SCAP in the process of IT security.

The three areas subject of improvement are:

- Incorporating CWE and CAPEC;
- container image scanning and remediation;
- continuous monitoring.

The first area aims at incorporating weaknesses and attack patterns in the vulnerability assessment process. As with OpenSCAP is possible to perform vulnerability scanning to highlight which particular CVEs affect a system, so the proposed solution links vulnerabilities to their correspondent weakness, identified by a CWE number, and finally to the CAPEC attack patterns that can exploit those weaknesses.

The second subject is to create an automatic process that can scan and remediate a given Podman image. This solution does not use the offline scanning technique, thus enabling the modification of the input image so that it's compliant to a security policy, and finally commits the new remediated image.

Lastly, the proposed solutions on the subject of continuous monitoring are both a standalone script that can be executed on a machine, waiting for specific events to occur and then automatically applying remediations, and a broader process that incorporates the script in a more complex structure, where an orchestrator can remotely monitor multiple machines.

The user and developer manuals can be found in Appendix A and Appendix B respectively.

6.1 Extending vulnerability assessment to CWE and CAPEC

This section presents and analyses the first-hand developed tool "CVEtoCWE", which can be found on the following public GitHub repository:

`https://github.com/maxwhy/CVEtoCWE`

The first section covers the objectives and the motivations that led to the creation of this tool and, additionally, it describes on a theoretical level its core functionalities.

The second section describes in particular the actual structure of the tool, from the chosen programming language to the organization of the code, and analyses the main functions contained within the tool.

6.1.1 Objectives

Common Weakness Enumeration (CWE) is not currently included amongst the SCAP components, although there are good reasons for its integration in the security automation process.

Section 5.4.2 presented the relation between CWE and SCAP at its current state: for every new vulnerability that receives a CVE identifier and appears in the NVD, the correspondent weakness that made possible the vulnerability in the first place is analysed. The type of weakness, among many other factors, determine the severity of the CVE and consequently its assigned score.

CWE is by far the most widely adopted standard reference for software weaknesses and it's used to convey weakness information between security professionals, vendors, and consumer companies. The NVD also utilizes CWE when determining the weakness type that stands at the base of a CVE.

This first step of the integration of CWE already indicates the close relation between CWE and CVE, i.e. weakness and vulnerability. The lack of support for CWE in SCAP and its tools leaves a significant gap in the security process, especially regarding vulnerability assessment.

In fact, the final objective of vulnerability assessment is to eliminate all the known vulnerabilities from a system, but it also serves the purpose of prioritizing the vulnerabilities. In an ideal world every vulnerability should be avoided and fixed immediately, but in the actual state of things there is a need for prioritizing which issues to fix first, both in terms of time spent and economic resources that can be allocated.

This prioritization is typically carried out with the use of CVSS scores, which are certainly useful but are not the be-all and end-all solution. Different users may have different needs that can change the actual mitigation priority for them and their particular use-cases.

In this scenario the usefulness of CWE can emerge as additional information for a correct and well-suited prioritization. A particular type of weakness can be of much more interest for a particular company, or have lower or greater impact on different systems within the company IT assets.

Having the knowledge of the actual weakness that causes a vulnerability can direct the remediation efforts towards one particular vulnerability, as that weakness can be more damaging for that particular system.

Let's take as example two particularly prevalent CWEs, which are respectively number one and three in the 2021 CWE Top 25 list:

- CWE-787: Out-of-bounds Write[1];
- CWE-125: Out-of-bounds Read[2].

These are both very dangerous weaknesses but their effects are different: CWE-787 typically leads to the corruption of data in the system, while CWE-125 typically can allow attackers to read sensitive information.

This difference in effects can shift the attention from one weakness to the other: a company, or a particular system within that company, that contains highly confidential or sensitive data should concentrate the efforts in fixing the Out-of-bounds Read weakness first. Another company or system that relies heavily on data consistency should concentrate on the Out-of-bounds Write weakness instead.

This prioritization cannot be performed effectively by the CVSS scores alone. Consider for example two vulnerabilities, CVE-2021-42008[3] and CVE-2021-38115[4], which have a CVSS score

[1] https://cwe.mitre.org/data/definitions/787.html

[2] https://cwe.mitre.org/data/definitions/125.html

[3] https://nvd.nist.gov/vuln/detail/CVE-2021-42008

[4] https://nvd.nist.gov/vuln/detail/CVE-2021-38115

of 7.8 (High) and 6.5 (Medium) respectively. By these scores alone, the natural prioritization is to first mitigate the first vulnerability, then the second one.

After further inspecting these vulnerabilities, it emerges that the first one has the CWE-787 (Out-of-bounds Write) assigned to it and the second has the CWE-125 (Out-of-bounds Read) instead. So, in the case of a system that treats highly sensitive data the actual order of remediation should be inverted, disregarding the severity scores of Medium and High.

For these reasons the inclusion of CWE in the process of security, and in particular vulnerability assessment, is desirable and meaningful. The CWE data that is linked to a CVE cannot be retrieved from any of the OpenSCAP family of tools, and it appears that no other tool exists with this scope.

The fulfilling of this gap is precisely the objective of the "CVEtoCWE" tool, which aims at providing a feature that would otherwise require a manual inspection that would be both time consuming and tedious.

Additionally, attack patterns can now also be included. Starting from the list of CWE it is possible to link the common attack patterns, identified with CAPEC, that are used to exploit those weaknesses. Utilizing CAPEC can provide insights on which aspects of the security of a system must be hardened and monitored to detect and possibly prevent threats.

The tool was developed with additional requirements in mind:

- It should cover the widest possible range of use-cases;
- It should work with standards and references that are supported by SCAP;
- It should be a command line tool with a simple interface, similarly to OpenSCAP.

6.1.2 Functionalities

The main concept of the CVEtoCWE tool is to take as input the results of previous vulnerability assessment evaluations, and then match every vulnerability that is present on the evaluated system to the corresponding weakness. Additionally, the tool can also retrieve the common attack patterns that are linked to those weaknesses.

This tool is supposed to work in the context of automatic vulnerability assessment, which can be performed with the OpenSCAP tools, and can be integrated in the process allowing its functionalities to extend the amount of information that is acquired during an evaluation.

Vulnerability assessment in the SCAP work flow is typically performed using OVAL Vulnerability definitions for the specific vulnerabilities that apply to the target system, bundled together in an OVAL Definitions file. The tool that performs the scan evaluates each definition and generates an OVAL Results file containing the collected results of each definition.

An interesting aspect to keep in mind is that as long as the scanning tool is SCAP conformant, it will generate a conformant OVAL Results file. This means that both OpenSCAP Base and its utilities, when used for vulnerability assessment, generate OVAL Results files that can be utilized by the CVEtoCWE tool regardless of the target of evaluation.

Thus there are many use cases that can be satisfied by the tool, as it can process OVAL Results file generated by:

- OpenSCAP Base scanning of the local system;
- oscap-ssh evaluation of remote machines;
- oscap-podman and oscap-docker scanning of images and containers;
- oscap-vm scanning virtual machines.

This consideration also applies to scanning tools that are not part of OpenSCAP, such as Atomic: if a container vulnerability assessment is made with the proper options to generate an OVAL Results file, that file too can be given to the CVEtoCWE tool.

After the OVAL Results file is given as input to the tool, it then searches for all the failed vulnerability definitions (i.e. the vulnerabilities that are actually present on the system). Then, for each found vulnerability it retrieves the corresponding assigned CWE. The mapping between CVE and CWE is for the vast majority of cases one-to-one, but there may be some marginal cases where multiple weaknesses are related to a single vulnerability: as an estimate, this portion accounts for less than 2% of the total number of CVEs. To avoid filling the tool results with too much information, only the first listed weakness is considered.

This retrieval step relies on the NVD as the source of data. The NVD was the natural choice because it's listed as the source of standard reference data for vulnerabilities in the SCAP standard. The tool utilizes the official APIs provided by the NVD itself. These APIs can provide, starting from a CVE Identifier, a variety of information related to that vulnerability, including the associated CWE.

After that, the correspondent CAPEC entries are presented and referenced. The correlation is performed using a custom list that matches weaknesses with the related attack patterns. This list has been generated from the official CAPEC list found on the MITRE site.[5] In this case, it's quite common that multiple attack patterns are mapped to a single weakness, and the tool can correctly handle such cases.

The two main functionalities of the CVEtoCWE tool consist in:

- Showing the mapping of CVE identifiers to the correspondent weaknesses;
- generating a report that contains the CVE to CWE mapping, the related attack patterns, and the references for those CWE and CAPEC entries.

In the first case, the command line tool simply outputs on the terminal the list of CVEs alongside the CWE that is attributed to them.

In the second case the tool also includes the CAPEC entries that are related to those weaknesses and saves this list on a HTML report. The report contains the reference link to the MITRE page of each weakness and attack pattern, as MITRE is the organization that maintains the CWE and CAPEC list. These convenient links serve the purpose of facilitating further analysis of weaknesses and attack patterns, as the official page of a CWE or a CAPEC contains useful additional information.

As already mentioned in section 5.4.2, the NVD actually uses a subset of all the available CWEs, i.e. a CWE slice, which is comprehensively described on this page:

https://nvd.nist.gov/vuln/categories

There are two "special" CWE entries in this list that do not identify a weakness, reported here with their description:

- NVD-CWE-Other ("Other"): NVD is only using a subset of CWE for mapping instead of the entire CWE, and the weakness type is not covered by that subset.
- NVD-CWE-noinfo ("Insufficient Information"): There is insufficient information about the issue to classify it; details are unknown or unspecified.

The first one is applied when no weakness of the NVD CWE slice matches the actual weakness of the vulnerability. Given that the NVD slice is considered quite complete, the occurrence of this particular situation should be rare and typically means that the underlying weakness is relatively

[5] https://capec.mitre.org/data/downloads.html

uncommon (for example, every CWE contained in the 2021 Top 25 is also included in the NVD slice).

The exact criteria for which CWEs are included or not in the NVD slice is unknown, but presumably enough weaknesses are part of the slice to allow the NVD analysts to effectively categorize vulnerabilities.

The second one is used when there is not enough information available to determine which weakness is causing the vulnerability. It's sometimes used in the initial stages of the CVE discovery, when the analysts haven't yet fully studied and inspected the vulnerability, and then it's usually substituted by a normal CWE entry.

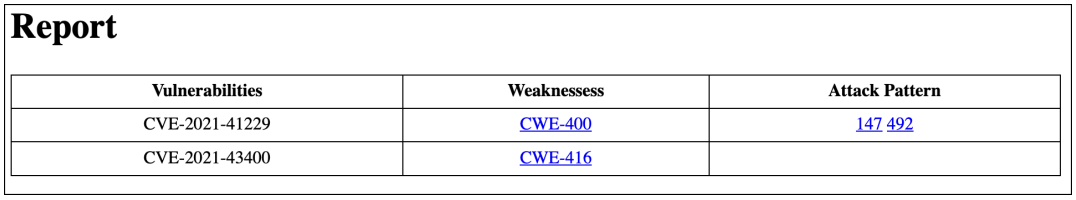
Whenever the tool encounters vulnerabilities that fall into the aforementioned categories, it reports the correspondent name but cannot provide the MITRE reference link, as such link doesn't exist.

There are some OVAL Vulnerability definitions that include references to CVEs that are not present on the NVD, typically because those vulnerabilities are marked as "RESERVED". This vulnerability status is applied when a CVE Numbering Authority (CNA) assigned the CVE but hasn't been able to recover enough information about the vulnerability to populate the CVE entry.

In this case the tool shows a warning that the vulnerability hasn't been found on the NVD, and then it does not include it in the report, as there is no meaningful information about that vulnerability.

The mapping between weaknesses and attack patterns is not one to one: some weaknesses don't have any CAPEC entry directly associated with them, while other more general and comprehensive CWEs present multiple attack patterns. The report can correctly handle both these situations, showing all of the related CAPEC numbers, or leaving an empty space if no attack patterns are related to a particular weakness.

Figure 6.1 shows a sample report generated by the CVEtoCWE tool that contains some of the aforementioned situations. Notice how the first vulnerability is related to two attack patterns while the second doesn't have any correlation.



Report

Vulnerabilities	Weaknessess	Attack Pattern
CVE-2021-41229	CWE-400	147 492
CVE-2021-43400	CWE-416	

Figure 6.1. Sample report generated by the CVEtoCWE tool.

6.1.3 Structure

The CVEtoCWE tool takes the form of a command-line interface (CLI) tool, as it's the most effective and simple way of interacting with OVAL files and setting the options for the different use-cases. Additionally, the target users of this tool are usually very familiar with CLI tools, as CVEtoCWE is meant to be included in the work flow of other OpenSCAP tools that are almost exclusively command-line based.

The tool also resembles the usage of OpenSCAP Base, as the various commands generally take the form of the name of the tool followed by the operation to be performed, then the additional options and finally the input file.

Code organization

The programming language chosen for the development of this tool is JavaScript. The primary reason for its use is that the NVD APIs, on which the functioning of the tool is based, are

constructed using REST services and JSON format for data.

Both these standards are very well supported by JavaScript: JSON stands for "JavaScript Object Notation" and is natively included in JavaScript, and REST APIs are widely used for web-based communication between applications, for which JavaScript is the most adopted programming language.

JavaScript is also a flexible and relatively secure language by itself, it's very well maintained and is used by many developers.

CVEtoCWE utilizes the Node.js[6] runtime environment for JavaScript. Node.js is open source, platform independent, and allows the execution of JavaScript code for CLI tools. Its architecture allows for asynchronous input/output operations and thus possesses very high scalability.

Node.js is widely used and there are many libraries already available that provide a high variety of functionalities. Node.js comes with a pre-installed package manager, called npm, that allows the developer to easily access the available libraries and packages via the npm registry.

The general structure of the code is as follows:

- **index.js**: the starting file that defines the functioning of the tool;
- **commands**: the folder that stores the files that contain the actual commands code;
- **save.js**: contains the code for the "save" command;
- **show.js**: contains the code for the "show" command.

The tool takes the following general form for its functioning:

cvetocwe command [options] *input-file*

Code functioning

Both the available commands of the tool require an OVAL Results file as input. The input file is opened and parsed as JSON, allowing the file contents to be easily accessed by JavaScript. CVEtoCWE then searches for any failed vulnerability definition.

For every failed definition the tool searches for the relative references that are found on the OVAL definition itself. These references can be of many types but in this process, staying in line with the scope of the tool, only the CVEs are retrieved. It's quite common that a single OVAL definition references multiple CVEs, and in that case all the referenced CVE are correctly retrieved.

Then for each found CVE the relative NVD API is called. The complete instructions for these APIs can be found on the NVD site[7]. For each CVE a particular URL needs to be constructed in the following way:

```
url = "https://services.nvd.nist.gov/rest/json/cve/1.0/" + CVE_ID
```

Then an asynchronous request is sent to that particular URL with an HTTP GET method. The use of asynchronous requests is crucial for scalability: the code can continue its execution without waiting for each single response, that depending on the network status can take a considerable amount of time to return.

Being able to send multiple requests in succession is especially needed for moderately vulnerable systems, as some failed OVAL definitions can result in dozens of different CVE requests. In those cases, waiting for a response that can take up to a few seconds to arrive would have a great impact on the tool performance in terms of speed.

[6] https://nodejs.org/en/

[7] https://nvd.nist.gov/developers/vulnerabilities

Once the response returns, it's again parsed as JSON and the CWE values are finally retrieved for each corresponding CVE. If the response contains any kind of error, such as "404: Not Found", the terminal will show a warning message stating that a particular CVE returned that error. The message is shown as a warning, and not an error, because these responses typically mean that the CVE is marked as reserved.

If the original issued command is *show*, then the tool writes on the terminal the entries composed as "CVE : CWE".

If instead the command was *save*, an HTML file is constructed so that it contains a table with one column containing the CVE identifiers, a second one containing the correspondent CWE, and a third column containing the related CAPEC entries. Any CVE that returned as an error is not included in the HTML file, but it's still shown as warning in the terminal. Each shown CWE and CAPEC in the report is also a link to the MITRE page of that weakness or attack pattern.

The location and name of this HTML report can be specified by adding the *reportname* option in the command, otherwise it's saved on the current directory as "report.html". One example of a report generated by the tool in a typical use case is shown in figure 6.2. Notice the reported weakness "NVD-CWE-noinfo" and the variable amount of related attack patterns.

Report

Vulnerabilities	Weaknessess	Attack Pattern
CVE-2021-43400	CWE-416	
CVE-2021-23192	NVD-CWE-noinfo	
CVE-2021-41160	CWE-787	
CVE-2021-38493	CWE-119	10 100 123 14 24 42 44 45 46 47 8 9
CVE-2021-30535	CWE-415	
CVE-2021-41159	CWE-787	
CVE-2020-25722	CWE-863	
CVE-2016-2124	CWE-287	114 115 151 194 22 57 593 633 650 94
CVE-2021-25633	CWE-295	459 475
CVE-2021-41229	CWE-400	147 492
CVE-2021-25634	CWE-295	459 475
CVE-2020-25718	CWE-732	1 122 127 17 180 206 234 60 61 62 642

Figure 6.2. Example of report generated by the CVEtoCWE tool.

6.2 Container image remediation

This section discusses the issue of container image remediation, and in particular analyses the proposed solution "ContainerRem", which can be publicly found at:

`https://github.com/maxwhy/ContainerRem`

The following paragraphs present the motivations and objectives that led to this solution, and then go more in detail regarding the actual code functionalities, organization, and structure.

6.2.1 Objectives

The OpenSCAP organization developed some utilities that can automatically perform security scanning of images. These utilities, **oscap-podman** and **oscap-docker**, can be of great help in determining whether an image is compliant to a security policy, checking the image system and generating reports. Both tools perform evaluation in the so-called offline scanning technique, which mounts the images in read-only mode so that the scan cannot damage or modify the images in any way.

The choice of utilizing this technique limits the possibilities of OpenSCAP, as at the present state it cannot remediate those images. The only current available work flow for remediating images is to first scan the images with the previously mentioned utilities, then manually inspect the reports to determine which rules aren't satisfied; after the appropriate fixes are selected, the interested security manager has to manually modify the images that can then be re-evaluated with OpenSCAP; finally, the newly generated reports can be inspected again to check if the remediations actually worked.

This process is clearly not scalable to a very high number of images, and requires a considerable amount of time and work. The solution to this problem is a tool that can automatically perform as much of the described steps as possible, leaving only marginal work on the actual human operator.

Another important aspect is that the tool should support a wide variety of image types, as images and containers are by their very nature suitable for highly diverse environment, where different types of systems are required. On this aspect the real limitation isn't posed by the images themselves, as the tool could theoretically function on almost all the common images, but by the SCAP content itself. The source data streams that are needed to perform SCAP evaluations must be designed at least for a specific operating system, if not for specific versions of it.

Additionally, there is also a performance consideration to be made. The image remediation work flow may not have hard requirements on time consumed, as it's a process that it's typically performed on fixed schedules and/or on regular intervals, but should still take the minimum amount of time possible. This is even more true in context with many images and containers that are running simultaneously. A similar consideration can be done regarding the size of images, that should not greatly increase as images are sometimes used in specialized environments with strict space requirements.

6.2.2 Functionalities

From the user perspective, the ContainerRem tool consists in a bash script that is executed giving one parameter as input. This input is the name of the image to be remediated. The tool identifies the image and behaves differently based on the image type. At the moment, three types of image are supported:

- Fedora;
- Ubuntu 18.04;
- Ubuntu 20.04.

Starting from the desired input image, the tool builds a new image by using a dedicated Dockerfile. Note that the name of this file could also have been "Containerfile" but given that the former name is adopted as standard by the vast majority of developers, the choice was to use it even if Podman is used instead of Docker.

ContainerRem copies the appropriate source data stream to the image. This content is taken from the SCAP Security Guide project, but in order to reduce time and occupied disk space not all the SSG contents are installed. Thus the image will only have the file that is necessary for the scanning and remediation.

Then the tool installs OpenSCAP onto the images by using the appropriate package manager. This is perhaps the longest part of all the process, as OpenSCAP isn't a very small package to install. Depending on the internet connection speed, it can take up to several minutes to install. This inconvenient occurs only on the first installation, as thanks to the layer caching mechanism of Podman, any subsequent use of the ContainerRem tool on the same image will no longer need the long installation process.

After the installation is completed, the tool can perform the desired evaluation of the image by running it as container, scanning and remediating the container with OpenSCAP and the source data stream. Once the remediation is completed, the generated report is copied back to the original host.

Finally a new image is committed based on the newly remediated container. In order to keep the work space clean, the container is then removed from the list of stopped containers.

The SSG content that refers to the operating system of the containers presents multiple rules that were made with a physical system as target, or at the very least a virtual machine. Containers present some differences to "real" systems, meaning that a non negligible amount of rules are not applicable to containers.

This aspect is reflected in the fix scripts that are contained in those rules. The remediation procedure of OpenSCAP actually searches for these fix scripts and tries to execute them. When rules are marked as "not applicable", the remediation script begins takes this general form:

```
# Remediation is applicable only in certain platforms
if [ ! -f /.dockerenv ] && [ ! -f /run/.containerenv ]; then
...
# Remediation script here
...
else
    >&2 echo 'Remediation is not applicable, nothing was done'
fi
```

In these cases, neither evaluation or remediation for those specific rules can be performed on a container image. Nevertheless, the rest of the rules can still be correctly evaluated and remediated.

6.2.3 Structure

The code structure is organized with the objective of keeping the files that refers to different images separated. The main script that manages the various images is the bash script "ContainerRem.sh". The rest of the files are stored in three different folders, one for each type of image that is supported (Fedora, Ubuntu 18.04 and Ubuntu 20.04)

Each folder contains the relative source data stream from the SSG content: *ssg-fedora-ds.xml*, *ssg-ubuntu1804-ds.xml*, and *ssg-ubuntu2004-ds.xml*. These files are already present amongst the tool code so that those files can be directly copied inside the images when they are built.

Each folder also includes the Dockerfile that is invoked during the image building. Each Dockerfile is similar but unique for the different images, as it is responsible for the installation of OpenSCAP and the copy of the source data streams.

The Podman **run** command is then used to run the image to be remediated as a container and executing an OpenSCAP evaluation, and subsequent remediation, on the running container. After the remediation is done, the report can be copied from the now remediated container to the host system with a Podman **cp** command. The reports are located inside the correspondent folder of the image.

A new remediated image is created by committing the remediated container. As the new image is created, the container is no longer needed and can be removed.

6.3 Continuous Monitoring

This section discusses the proposed solutions on the subject of continuous monitoring. There are two main aspects that are analysed below: the first is the development of a continuous monitoring tool that can be installed on a machine and can respond to potentially dangerous security-related events in real-time. The second is the implementation of the same principles applied in a more complex use-case, where an orchestrator manages multiple remote machines that are monitored, and each one of them sends their reports to an external resource.

These next sections describe the objectives, functionalities, code organization and structure of the "SCAPmonitor" tools, which can be found at the following GitHub repository:

https://github.com/maxwhy/SCAPmonitor

6.3.1 Objectives

The concept of continuous monitoring can be highly valuable in many scenarios, especially when security and policy compliance are priorities for a given system. Unfortunately, SCAP as a framework never contemplated continuous monitoring, at least until version 2 will be released. OpenSCAP tried to fill this gap by providing scheduling and planned scanning with OpenSCAP Daemon.

Nevertheless, the gap is still present as pre-scheduled scans do not and cannot provide the same level of security confidence that can be achieved with continuous monitoring, for a variety of reasons already presented in section 5.2. The main objective of SCAPmonitor is in fact to eliminate this gap and provide not only continuous monitoring, but also active remediation.

Due to the intrinsic differences of both operating systems and security policies, it is very unlikely that a single tool could be employed in many different scenarios. A more conservative approach of concentrating on more specific use-cases is needed to develop a functioning tool, of which core ideas and functionalities can then be extended to a growing number of additional cases.

The SCAPmonitor development process started with the objective of monitoring the specific events that can put a system in a non-compliant state with respect to a specific security policy. The work flow can be summarized in these steps:

1. A specific operating system is selected (e.g. Ubuntu 20.04).
2. A matching SCAP content is retrieved (e.g. the SSG source data stream for Ubuntu 20.04).
3. Each rule within the content is analysed and a list of events that could change the result of the rule evaluation is created.
4. It's implemented a new piece of code that can detect when those events occur in the system.
5. When a new event is detected, an OpenSCAP evaluation and remediation is issued.

This procedure leads to a tool that is sensible to specific events in the system that refers to that particular security policy. As different security policies check different aspects of a system, so the pool of events to be monitored are different.

It's important to note that while the previous statements are true, so is the fact that many security policies are similar to one another and that similar systems require similar event detections. This means that, for example, any version of Ubuntu 20.04 requires the same kind of monitoring regardless of it being 20.04.1, 20.04.2, or 20.04.3. Also newer or older versions of Ubuntu may need a monitoring tool with only small adjustments to properly work.

The SCAPmonitor tools were designed with the goal of providing a script that can be simply started by a user and then can be left running in the background. The user shouldn't note any considerable difference in the behaviour of the system while SCAPmonitor is running. In order to avoid system overhead, the act of monitoring and waiting for events should pose minimum effort on the machine.

Another objective of the tools is to behave as Intrusion Prevention Systems (IPS), not being limited to simply detect changes, but also automatically act on them and possibly restore the system to a compliant state. This can be achieved by running an OpenSCAP remediation in an autonomous way on every event detected.

6.3.2 Functionalities

The SCAPmonitor tool consists in a script that can be started from the command line with a single command, passing as argument the path where the correspondent source data stream is located. The execution starts and the tool begins waiting for the specific events that refer to the current system and policy.

The tool doesn't show any activity until an event is detected. When a detection occurs a message is shown on the terminal window indicating the type and location of the event, and then OpenSCAP is used to evaluate the system and apply remediation if it appears to be in a state of non-compliance to the security content provided.

SCAPmonitor doesn't end its execution until the user actually stops it, so that any succession of events can be detected automatically and without pause. The functioning of the tool is built in such a way that even in the presence of simultaneous events those events can all be detected. Different events that occur in various locations of the system or present different behaviours are all separately detected and a new remediation is issued for each one of them.

On the other hand, if the same kind of events happen in rapid succession or during a single OpenSCAP evaluation, the events that are immediately after the first one will not issue a new evaluation. This execution flow assures that no infinite evaluation loop can occur but it still covers all those events: the remediation of the system, which is performed as the final step of an OpenSCAP evaluation, forces the targets of those events back to a compliant state.

After every remediation a new report is created by OpenSCAP, showing which rules are passed or not and, additionally, describing what remediations were applied and if they have been effective.

Two similar scripts has been developed for two specific targets:

- **SCAPmonitor**: monitors a Ubuntu 20.04 system with respect to the SSG content *ssg-ubuntu2004-ds.xml*
- **SCAPmonitor18**: monitors a Ubuntu 18.04 system with respect to the SSG content *ssg-ubuntu1804-ds.xml*

There are only minor differences between the two but nonetheless the proper one must be applied in the appropriate context in order to correctly function.

6.3.3 Structure

The code structure is now analysed by referring to the SCAPmonitor for Ubuntu 20.04 script. After this analysis only the most noticeable differences with respect to the Ubuntu 18.04 version are presented.

SCAPmonitor is constructed with numerous sub-processes each monitoring different aspects of the system and listening for different events. In theory, each rule contained in the source content should have its own sub-process checking the validity of that particular rule. Practically implementing a tool with such granularity is both time consuming for the developer and, most importantly, requires too much effort for the system.

It is a much better solution to group similar rules that are compatible to each other, allowing a single sub-process to check and wait for multiple events. This reduces the overall amount of processing time and greatly enhances the performance of the tool.

From a security perspective it has been followed a conservative approach, meaning that in order to reduce the number and complexity of rule checking, the tool actually detects a broader range of events than the theoretical minimum amount that would be necessary.

This choice can result in a little decrease in performance, as some evaluations can be issued even if the detected event did not effectively pose a security threat to the system. This increase of false positive detections is typically not noticeable and doesn't dramatically affect the overall performance of the tool.

There are two main kinds of sub-processes that are utilized by SCAPmonitor:

- An **inotifywait** command wrapped in a cycle;
- A while cycle that repeats every fixed amount of seconds.

The first approach requires the installation of the "inotify-tools" package. Among those tools, inotifywait is a function that uses the Linux *inotify* APIs to efficiently wait for specific changes to files. inotify creates watches that can monitor individual files or entire directories, signaling when specific types of events occur that affect those files or directories. An event can be the creation or removal of files within a directory, the opening and closing of files (either with or without a modification), and the change of metadata such as permissions or extended attributes associated with a file or directory.

The use of inotifywait consists in specifying which files or directories need to be monitored and which type of events should be listened to. Once the script reaches an inotifywait command, it blocks the execution of the program and waits for the event occurrence. When that happens the compliance scanning and remediation with OpenSCAP is then executed.

Each instance of inotifywait is enclosed in an infinite cycle, so that after the event detection and subsequent remediation a new inotifywait command is executed and starts listening again for its events. All the different cycles are executed in parallel by SCAPmonitor, so that any desired file and directory can be concurrently monitored.

The second approach is used when the condition of certain rules cannot be monitored by changes in files or directories, as they refer to the presence or absence of specific packages or whether a service is running on the system.

In those cases the chosen mechanism for detecting changes is polling, i.e an infinite while cycle that waits a set amount of time between each execution. The exact amount of time between each execution can be set depending on the needs of each different use-case. A high frequency cycle can raise the confidence in the actual security of the system and can more rapidly respond to threats, at the cost of a higher execution load on the system.

In any case the execution cost of the entire cycle, both in terms of time consumed and computational power required, it's quite low and not that impactful. As a consequence the delay between each cycle can be relatively short, in the order of a dozen seconds, ensuring a good level of security for most use-cases.

Rule checking implementation

The following paragraphs generally describe how each rule contained within the *ssg-ubuntu2004-ds.xml* file is checked.

First, five rules were counted as exceptions as they aren't monitored by the tool. These rules all refer to specific directories being mounted on separated partitions on the disk. As stated in the SSG content guide itself, to modify and create separate logical volumes at run time is possible but non-trivial. Hence delegating these operations to an autonomous script that is continuously running in the background could easily damage the system if not handled with extreme care. Following this consideration the aforementioned rules are considered to be outside the scope of SCAPmonitor.

There are multiple rules that refer to a specific service being installed and running on the system. These rules are checked by verifying that the list of active processes contains the desired service, meaning that the service is both active and, by consequence, installed.

On the other hand, other rules check that a package is not installed on the system. The process for these rules is similar to the previous ones but it consists in checking from the list of packages the status of the undesired packages. Both this group of rules and the previous one are checked using the second approach of polling.

A large number of rules refer to ownership and permissions of files, and can be monitored with inotifywait by waiting for an "attrib" type of event, which is issued when metadata like ownerships and permissions change.

Inotifywait is also used to monitor modifications to files that can potentially be dangerous for the system. Multiple rules consist in verifying that a particular line of a file is present or not. Unfortunately inotifywait cannot distinguish between modifications on one line or another of the file, as any type of modification issues the same event.

This is perhaps where the most false positive detections arise, but it can also be considered as an additional security measure, given that any change to these monitored files, either intentional or not, is always related in some way with the security of the system. These files include for example /etc/ssh and /etc/passwd, which are both tightly related to the system's security.

All of the previous considerations can be applied to SCAPmonitor18, which is the tool that can monitor a Ubuntu 18.04 system. The main functionalities are identical, and only minor differences can be highlighted. All these differences derive from the different rules that are either contained or not in the two SCAP source content. These rules typically refer to the presence of absence of specific services that can only be found in one Ubuntu version rather than the other.

6.3.4 Remote monitoring

Continuous monitoring of a single system is already valuable in itself, but extending this functionality to control remote systems can cover a much wider range of use-cases. Suppose a context in which a single main host acts as orchestrator and controls multiple nodes. Ensuring that all the child nodes are compliant to a policy and continuously hardened from a security perspective is a challenging task, in terms of coordination and management effort.

The introduction of as much possible automation in the process can greatly reduce the time consumed by a system administrator in maintaining the security requirements of the system. Some degree of manual intervention is always needed, at least in the setting up of the IT structure and services, but it will be shown how it is possible to activate and maintain an automatic process for continuous monitoring of multiple sub-systems, all coordinated by a single orchestrator.

The real world deployment of such a structure can be of great utility in a variety of use cases, including:

- A simple company setting where multiple machines assigned to different employees can be continuously checked for security by a security administrator.
- Large or complex network infrastructures that utilize multiple devices.
- Highly virtualized environments where a considerable amount of virtual machines and containers are active simultaneously.
- Monitoring of an NFV infrastructure.

As a first step, a main host can connect to the remote nodes using a secure mechanism, for example an SSH connection, from which it can remotely download and run the desired SCAPmonitor script. All the functionalities described in the previous sections can be applied to the nodes, meaning that any notable event will issue a security scan and subsequent remediation.

A possible issue can emerge here, as the results of SCAPmonitor can be retrieved only by inspecting the reports that are generated after a scan. These reports are created within the machine that is actively being monitored, and without further action remain on that remote system.

The administrator that controls the orchestrator needs access to these reports, if not often at the very least occasionally, to check the actual status of the system. Hence a mechanism is necessary to transfer the reports outside the machine that originally created it.

Bringing the reports outside the context of the node is additionally useful to prevent their malicious modification or removal. If the monitored node is under an attack, it could be easy for an attacker to remove the reports and thus hiding the traces of its presence.

There may be some instances where a copy of the report can simply be transferred back to the main host without further complications. For example, copying files from running containers is relatively easy and can be done with a single command (e.g. by using "podman cp").

Other scenarios can be of more difficult implementation, or can represent a possible security issue that should be avoided. For example transferring files back to the host directly from the nodes, even if executed with secure mechanism, still creates a possible opening to the main host: the node, which is generally less secure than the orchestrator, creates a connection that can be potentially used to transfer malicious files.

Transferring reports to an external resource

The proposed solution to the aforementioned problems is to incorporate a third entity in the process, exploiting an external resource. This additional component should be separately accessible both by the main host and by the nodes, and should serve as a remote database containing the reports. All the different nodes could upload their reports as soon as they are generated, and then the system manager could access all of those reports at a later stage.

This solution solves many problems as the validity of the reports is no longer reliant on the actual security of the nodes, but it's guaranteed by the level of security of the external resource that can be, in principle, much higher than the one of the single nodes. Additionally, the nodes have no direct connection to the host, meaning that even in the worst case scenario the attacked system would be the external database. The actual damage that can be inflicted would be limited to the data integrity and availability of the reports, but no direct harm can be done on the principal host.

There are multiple candidates that can fulfil the role of external resource, and each company should select the most appropriate one for its particular needs. One proposed preferred solution is to utilize a git repository. Although not originally intended to be used in such context, git repositories present all of the needed characteristics and provide additional functionalities and security.

Some of the key aspects of a git repository are:

- It is relatively easy to create and configure;
- it can be accessed by multiple systems simultaneously;
- each node can have its own dedicated branch;
- every change is recorded and the history cannot be manipulated;
- it's a fast and efficient system.

Any company can utilize a git repository either by relying on already available services (e.g. GitHub, Gitlab) or by creating a custom repository. There are many settings that can be configured to create a personalized experience for every different need. This includes security measures such as choosing which users are allowed to commit or view the repository, and which connection types are enabled.

Every single node that needs to be monitored can push its reports to a dedicated branch of the repository, allowing a clear separation between reports that can greatly help the system administrator in navigating and identifying the reports.

Another interesting aspect is the actual functioning of the git version control system: each event that modifies the contents of the repository, being the insertion of new files, their modification or even their removal, is recorded and stored by using a Merkle tree, meaning that each new modification is labelled with the cryptographic hash of the previous contents.

This procedure insures that any past modification cannot be unnoticed, as the generated hashes would not match the current one. Therefore, even if a malicious attacker succeeds in removing or modifying the reports, these changes cannot be hidden and remain tracked in the history of the repository.

The "SCAPmonitor18git" tool is the implementation of the SCAPmonitor script for Ubuntu 18.04 with the additional capability of automatically pushing the reports to a remote git repository. This functionality is provided only if both the repository and the machine which is running the script are properly configured.

Chapter 7

Testing

This chapter presents some real world applications of the proposed solutions. Each tool and new functionality is tested in order to verify its actual capabilities and check whether the execution progresses as intended. Additionally, specific tests are reported that provide details on the performance of the tools, focusing on the specific aspects that are critical in the typical context of application of each tool.

The first section discusses the CVEtoCWE tool and its extensions to CAPEC, analysing in particular how the time consumed by the execution increases with respect to the number of vulnerabilities that affect a system.

The second section focuses on the ContainerRem tool both in term of its performance and compatibility. On the topic of performance the subject of the tests are time spent for installation, evaluation and remediation as well as occupied disk space and image size increase. Further limitations are highlighted in terms of the availability of SCAP content specifically created for container images.

The third section aims at testing the execution time of SCAPmonitor and its impact on CPU utilization on various systems and under different work conditions. It is also presented a practical implementation of remote monitoring that employs an external server storing reports on a dedicated repository.

7.1 CVEtoCWE and CAPEC

The functionalities of the CVEtoCWE tool has been tested with multiple OVAL Result files. These files were generated with a OpenSCAP vulnerability evaluation, taking as input a OVAL Definitions file that checks for the presence of specific vulnerabilities for the desired operating system.

There are some requisites that need verification to ensure consistency of results and to check the correct implementation of the various functionalities. The following numbered list of required cases is used as reference for the rest of the section:

1. A vulnerability was checked and resulted as not present on the system.
2. A vulnerability is not referenced as CVE but uses another standard (e.g. USN, Ubuntu Security Notices).
3. A vulnerability is referenced as a CVE that is not present on the NVD (i.e. it's in the reserved state).
4. A vulnerability relates to one of the two special cases: "NVD-CWE-Other" or "NVD-CWE-noinfo".
5. A weakness is not related to any CAPEC entry.

6. A weakness is related to multiple CAPEC entries.

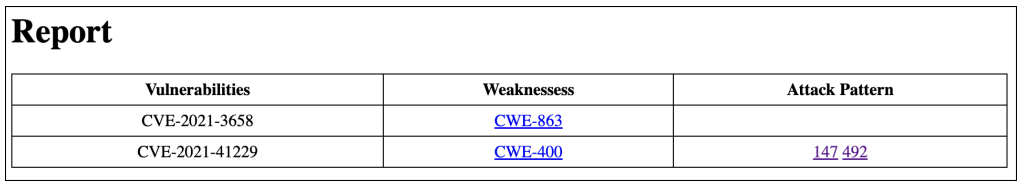
Regarding the specific input for the tests the following three files were used, which primarily differ by number of vulnerabilities and size: a small sample OVAL Result file, a medium size file containing 10 different CVEs, and lastly a bigger file of 20 vulnerabilities.

The smallest file was specifically created from a regular OVAL Results file, removing many of the vulnerability definitions, leaving only a small sample. The following is a fragment of this file:

```
1  <definition id="oval:com.ubuntu.focal:def:51551000000" version="1"
       class="patch">
2          <metadata>
3            <title>5155-1 -- BlueZ vulnerabilities</title>
4            <reference source="USN" ref_id="USN-5155-1"
                 ref_url="https://ubuntu.com/security/notices/USN-5155-1"/>
5            <reference source="CVE" ref_id="CVE-2021-3658"
                 ref_url="https://ubuntu.com/security/CVE-2021-3658"/>
6            <reference source="CVE" ref_id="CVE-2021-41229"
                 ref_url="https://ubuntu.com/security/CVE-2021-41229"/>
7  ...
8  <definition id="oval:com.ubuntu.focal:def:46401000000" version="1"
       class="patch">
9          <metadata>
10           <title>4640-1 -- PulseAudio vulnerability</title>
11           <reference source="USN" ref_id="USN-4640-1"
                 ref_url="https://ubuntu.com/security/notices/USN-4640-1"/>
12           <reference source="CVE" ref_id="CVE-2020-16123"
                 ref_url="https://ubuntu.com/security/CVE-2020-16123"/>
13 ...
14 <definition definition_id="oval:com.ubuntu.focal:def:51551000000"
       result="true" version="1">
15 ...
16 <definition definition_id="oval:com.ubuntu.focal:def:46401000000"
       result="false" version="1">
```

This sample is able to represent a variety of use cases. First, the definition located at line 11 resulted to be false, as it can be seen at line 22. This means that the vulnerabilities that are referenced inside that definition should not be considered (requirement 1). The other definition, which resulted as true, contains one vulnerability at line 7 that isn't referred to as CVE but instead as USN (requirement 2). The CVE of line 8 is related to a weakness that is not linked to any attack pattern (requirement 5) while the vulnerability of line 9 has a related weakness that leads to two attack patterns (requirement 6).

Figure 7.1 shows the report generated when given the sample file as input. As can be seen from the image, the four aforementioned requirements are satisfied.



Report

Vulnerabilities	Weaknessess	Attack Pattern
CVE-2021-3658	CWE-863	
CVE-2021-41229	CWE-400	147 492

Figure 7.1. Report generated by the CVEtoCWE tool from the sample input.

The other two files are the result of two evaluations performed on the same machine at different times. The medium size file was generated on a machine that had not received security updates for approximately one month. The largest file was generated after an additional three months of not updating. In both cases, the target was a Ubuntu 20.04 system.

These larger input files contain instances of the two requirements left, such as CVE-2020-14562[1] that links to the "Insufficient Information" special case (requirement 4) and CVE-2022-26387[2] that is, at the time of testing, in a reserved state (requirement 3).

Table 7.1 shows some performance statistics of the CVEtoCWE tool upon receiving the three testing files as input. These times were acquired on a 2015 personal computer equipped with a 2.7 GHz Intel Core i5 CPU and 8 GB of memory.

Each test was repeated 30 times in order to have values weakly influenced by statistical variations. The average execution time grows as the total number of vulnerabilities increases, but thanks to the asynchronous process by which multiple requests are sent concurrently the relation is not linear. This trend is shown in figure 7.2.

Number of CVEs	Average time	Standard deviation	Max value
2	1.103 s	0.16 s	1.921 s
10	1.648 s	0.21 s	2.268 s
20	1.932 s	0.33 s	3.022 s

Table 7.1. CVEtoCWE tool performance statistics.

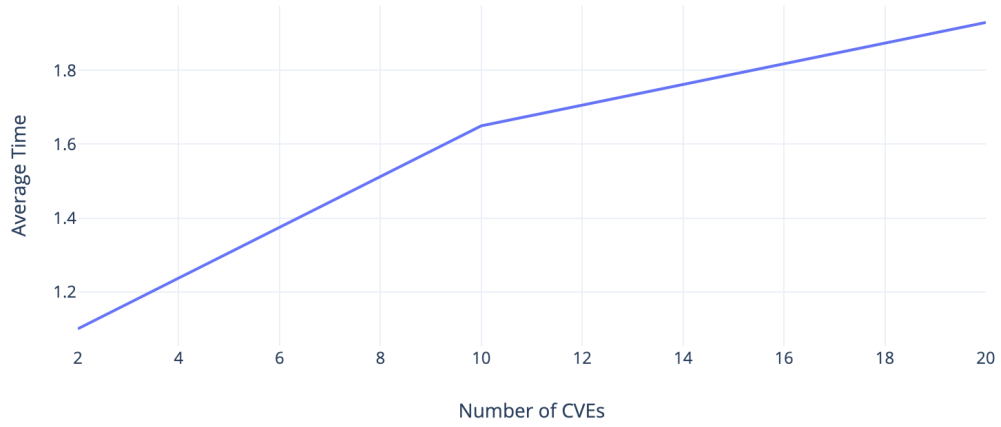


Figure 7.2. Average execution time of the CVEtoCWE tool.

The intrinsic nature of internet connections, which exhibit high volatility in response times, produces relatively high standard deviation values, ranging from 12.74% to 17.08% of the mean value. All things considered, the test that required the highest execution time took a little over three seconds to complete, meaning that the tool is always relatively fast, at least under standard internet conditions

7.2 ContainerRem

The ContainerRem tool can be tested on any operating system that supports Podman. The testing and results shown in this section were performed with a MacOS machine. Podman can only run Linux containers, which means that in order to function on MacOS it must utilize a Linux virtual machine. Podman already includes the necessary commands to automatically manage such environment.

The Podman client for MacOS can be installed via the Homebrew package manager:

[1] https://nvd.nist.gov/vuln/detail/CVE-2020-14562

[2] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-26387

```
$ brew install podman
```

After the installation is completed, the Podman-managed virtual machine can be initialized and started. These operations need to be performed only once:

```
$ podman machine init
$ podman machine start
```

The following table 7.2 reports the total execution time spent by the ContainerRem tool for the various types of container image. Notice the great difference between first and subsequent executions, in which time is significantly reduced thanks to the layer caching mechanism of Podman.

These times were acquired on a 2015 personal computer equipped with a 2.7 GHz Intel Core i5 CPU and 8 GB of memory. The third column shows the average execution time calculated after repeating 20 times each test. These results show that the tool is quite consistent in the amount of time utilized, as the standard deviation constitutes less than 5% of the average value.

Image type	First execution	Following executions (AVG)	Standard deviation
Fedora	3 m 2.180 s	23.871 s	0.979 s
Ubuntu 18.04	41.819 s	8.332 s	0.482 s
Ubuntu 20.04	43.104 s	10.629 s	0.231 s

Table 7.2. Execution time of the ContainerRem tool.

Both the Ubuntu versions spend a large portion of their execution time in updating the package list with *apt update*, a process that is needed to insure the installation of the latest version of the OpenSCAP package.

On the other hand, the Fedora image requires much more time for a variety of reasons. The biggest impact on speed is the installation of updates and packages that are related to Fedora. These packages alone require more than 100 MB of download, which is quite a significant amount compared to the total OpenSCAP download size of 3.8 MB.

Another slowing factor that is also reflected on the subsequent executions time is the actual OpenSCAP evaluation: some rules that are part of the source SCAP content utilized by ContainerRem consume several seconds to be checked, which leads to an evaluation time that is more than double the amount of the Ubuntu evaluations.

The size changes of the images are instead reported in table 7.3. The tool already limits as much as possible the increase in size, managing directly the copy of the source SCAP content in the images, instead of installing the whole SCAP Security Guide package. Nevertheless, the increase is still very much noticeable.

Image type	Size before execution	Size after execution
Fedora	159 MB	508 MB
Ubuntu 18.04	65.5 MB	227 MB
Ubuntu 20.04	75.2 MB	245 MB

Table 7.3. Image size before and after the ContainerRem tool execution.

A considerable amount of space is occupied by the installation of OpenSCAP and its dependencies: approximately 70 MB, with small variations between different operating systems.

7.3 SCAPmonitor

As the SCAPmonitor tool is intended to be constantly running in the background, its performance in terms of CPU utilization has been carefully tested. The aspect of processor time consumed on

average by the tool is of even more importance when considering many real world scenarios, such as highly virtualized environments, that present limited computing power.

The testing has been conducted focusing in particular on the rules that require monitoring via a polling mechanism. The rest of the rules, which are in fact the majority, can be monitored in a highly efficient way by waiting for specific signals, with very minimal impact on the CPU workload.

All of the rules tests that require polling are enclosed in an infinite cycle that repeats itself after a fix amount of seconds. This sleep time can be adjusted to match the specific needs of the various use-cases. The only relevant computing power utilized by the SCAPmonitor tool actually derives from the specific tests that are executed during each cycle (excluding the actual evaluation and remediation performed by the OpenSCAP scanner, which will be discussed later).

For the purpose of testing, a standalone version of this cycle has been produced. This shell script contains all the checks that are performed during a single cycle, thus its execution time is analogous to that required by the original tool on every iteration. Here is a snippet of the code:

```
#!bin/bash
FLAG=0
  # Rule: Ensure the audit Subsystem is Installed
  # Rule: Enable auditd Service
  ps -e | grep -w auditd >/dev/null
  if [ $? != 0 ];
  then
    FLAG=1
    echo "Service auditd not found"
  fi
  ...
  # Rule: Uninstall the ntpdate package
  dpkg -s ntpdate &>/dev/null
  if [ $? = 0 ];
  then
    FLAG=1
    echo "Package ntpdate found"
  fi
  ...
  if [ $FLAG = 1 ];
  then
    echo "Flag is 1"
  fi
```

As shown, the typical operations include the listing of active processes or checking the absence of packages, that are both operations that require a very low processing power. The tests were performed on different machines and with different loads on the processor. Each test has been repeated five times in order to get a more accurate average time.

In particular, table 7.4 reports the tests executed on a physical device (from now on referred to as "PC"), while table 7.5 refers to the tests executed on a virtual machine ("VM"). Each device has been tested both on standard conditions, that is without any specific task performed by the system, and during a high stress situation. Time is always reported in milliseconds.

CPU load	Test 1	Test 2	Test 3	Test 4	Test 5	Average
Low	201	210	225	208	206	**210**
High	314	312	326	328	334	**323**

Table 7.4. Time of execution (in ms) of one SCAPmonitor cycle on PC.

The PC is equipped with a 2.5 GHz Intel Core i5 and 6 GB of RAM, running Ubuntu 20.04 as operating system. The VM is a Ubuntu 18.04 virtual machine with a single processor and 3 GB of memory, running with VirtualBox.

CPU load	Test 1	Test 2	Test 3	Test 4	Test 5	Average
Low	218	205	189	194	193	**200**
High	524	480	477	465	500	**489**

Table 7.5. Time of execution (in ms) of one SCAPmonitor cycle on VM.

The stress tests were performed using the "stress" package, which can be downloaded on various Ubuntu distribution with the following command:

```
$ sudo apt install stress
```

After the installation, it is possible to customize the type and level of stress with the proper options. It is recommended to use these tools with caution and under root privileges, as it is possible to cause harm to the system. For these tests, the issued command took the form shown below. It puts the system under a relatively high load both in term of processing power and memory operations.

```
$ sudo stress --cpu 8 --io 8 --vm 8
```

As shown in the tables, the average time on standard conditions is similar for both machines and hovers around 200 milliseconds. A clearer gap can be seen during the stress tests, when the PC value increased by approximately 50%, as opposed to the VM times that became almost 2.5 times higher. All things considered, the execution time rarely surpassed the half a second mark, making the tool quite efficient and suited for many use-cases.

Another aspect that is needed to get a better perspective is the average time required by an OpenSCAP evaluation. As the SCAPmonitor tool provides continuous monitoring to the system, the frequency of scanning can be very high. Table 7.6 and table 7.7 show the test results executed on the PC and VM respectively. The high load scenario has been achieved with the same stress commands shown before.

CPU load	Test 1	Test 2	Test 3	Test 4	Test 5	Average
Low	2215	2155	2255	2286	2202	**2223**
High	3487	3470	3518	3488	3208	**3434**

Table 7.6. Time of execution (in ms) of one OpenSCAP evaluation on PC.

It is possible to notice the somewhat unexpected standard condition times, where the less powerful VM actually completes an OpenSCAP evaluation faster than the PC. It is possible that this discrepancy is due to the fact that the idle state between the two systems is not the same: the PC may have more processes and applications running in the background, as it's a more complete and refined system compared to the basic VM configuration.

Once again the biggest difference lies in the high stress scenario, where the VM times increase by more than 2.5 times and can take up to five seconds for a single evaluation. It is important to note that these times are not related to SCAPmonitor and depend only on the OpenSCAP scanner. Additionally, while the previous times of a single cycle are repeated every fixed amount of time and can be quite frequent, the OpenSCAP evaluations only occur when some specific events are detected. Typically these events do not occur frequently enough to overload the system.

7.3.1 Remote monitoring

A whole sample structure for remote monitoring with report transfer to an external resource has been deployed for testing. This structure is composed of three main actors:

- A main host, also referred to as "orchestrator";

CPU load	Test 1	Test 2	Test 3	Test 4	Test 5	Average
Low	1661	1719	1730	1647	1669	**1685**
High	3780	3814	3893	4013	4815	**4063**

Table 7.7. Time of execution (in ms) of one OpenSCAP evaluation on VM.

- a child node, which in this case is a virtual machine;
- an external resource that acts as the repository server.

The host is the system from which every other node is monitored. The child node represents one of these monitored systems that can be physical devices, virtual machines, or containers. This structure can support multiple nodes simultaneously. The external server is in this case a second physical machine that hosts the git repository.

Setting up the server

The following steps were executed to set up the git repository on the second machine, which is running Ubuntu 20.04. First, the git package shall be installed on the system:

```
$ sudo apt update
$ sudo apt install git
```

Then, a dedicated user (named "git") is created and the required SSH file is created:

```
$ sudo useradd -r -m -U -d /home/git -s /bin/bash git
$ sudo su - git
$ mkdir -p git/.ssh && chmod 0700 git/.ssh
$ touch git/.ssh/authorized_keys && chmod 0600 git/.ssh/authorized_keys
```

SSH keys are needed to enable the automatic commit and push of reports from the monitored nodes. Finally, the git repository can be initialized:

```
$ git init --bare git/reportserver.git
```

Now the "reportserver" repository is ready and will be stored in this server. From a security perspective, the git user can also be restricted to be able to execute with privileges only git related commands, which guarantees an additional level of security if the server becomes compromised.

Setting up the node

The following steps were executed on the virtual machine that represents the child node, in this case a Ubuntu 18.04 running with VirtualBox. These manual operations are needed to set up properly the machine and take advantage of the external repository. First of all, the node needs a pair of SSH keys, which can be created with the ssh-keygen utility, to be stored in the proper hidden directory:

```
$ mkdir -p $HOME/.ssh
$ chmod 0700 $HOME/.ssh
$ ssh-keygen
```

The public key shall now be copied to the server. In order to do that, the IP address of the server is needed. Let's assume the IP address of the server to be 192.168.1.10:

```
$ ssh-copy-id git@192.168.1.10
```

A message might be shown that states that the authenticity of the host cannot be established, but it is possible to simply type "yes" and continue; then insert the required password. The public SSH key should now be stored on the remote server. To check if this is the case, simply try to connect to the server, that shouldn't ask for a password again:

```
$ ssh git@192.168.1.10
```

Then, after closing the SSH connection, it's required the installation of the git packages:

```
$ sudo apt update
$ sudo apt install git
```

The local repository shall be configured in a dedicated folder, specifying a reference name and email (these fields can be totally arbitrary or random but are required). The remote origin refers to the actual location of the repository on the server:

```
$ mkdir Reports
$ cd Reports
$ git init .
$ git remote add origin git@192.168.1.10:/reportserver.git
$ git config --global user.email "node1@example.com"
$ git config --global user.name "node1"
```

Then a specific branch for the node can be created with an appropriate name and pushed to the repository:

```
$ git checkout -b node1branch
$ git push origin node1branch
$ git commit -m "First commit"
$ git push -u origin node1branch
```

Connecting host and node

The preferred way of connecting the main host to the node is to use an SSH connection. In this case the node acts as the SSH server, which means that it needs the correspondent package installed and the service running:

```
$ sudo apt install openssh-server
$ service ssh status
(if the service is not active, use the following)
$ service ssh start
```

In the context of this testing, the node is a virtual machine running on the main host with VirtualBox. In order to be able to connect via SSH, open VirtualBox and select the desired virtual machine. Then go on `Settings -> Network -> Advanced -> Port Forwarding` and add a new rule specifying the fields as follows:

- Protocol: TCP;
- host IP: 127.0.0.1;
- host port: 2222;
- guest IP: the IP address of the virtual machine, typically 10.0.2.15;
- guest port: 22.

After starting the virtual machine, the host can connect to tit by opening a terminal and typing:

```
$ ssh -p 2222 user@127.0.0.1
```

Where "user" should be replaced with the actual name of the user of the virtual machine. The user can also be root, but it must be the same user that generated the SSH key pair in the set up process, so that it can automatically commit and push to the remote repository without the need of a password.

Now the terminal controls the child node. If the node doesn't have it already, it needs the installation of the OpenSCAP scanner:

```
$ sudo apt install libopenscap8
```

Then the SCAPmonitor18git tool and the SCAP content can be downloaded directly from GitHub (note that the files names need to be changed):

```
$ wget
    https://github.com/maxwhy/SCAPmonitor/blob/main/SCAPmonitor18git.sh?raw=true
$ wget
    https://github.com/maxwhy/SCAPmonitor/blob/main/ssg-ubuntu1804-ds.xml?raw=true
$ mv SCAPmonitor18git.sh?raw=true SCAPmonitor18git.sh
$ mv ssg-ubuntu1804-ds.xml?raw=true ssg-ubuntu1804-ds.xml
```

Now the continuous monitoring process can be started by specifying also the folder where the local repository is located and the dedicated branch of this node:

```
$ sudo bash SCAPmonitor18git.sh ssg-ubuntu1804-ds.xml Reports/ node1branch
```

Each generated report is pushed to the external repository. The contents of the repository can be accessed from the git server directly, but can also be obtained by setting up another local repository on a different system, adding as remote origin the *reportserver* git, and then pulling the desired branch:

```
$ git pull origin node1branch
```

Chapter 8

Conclusions

8.1 Final considerations

The objectives of the thesis were to analyse the state of the art of the SCAP framework and its implementations, pointing out their issues and limitations, and then propose some solutions that can reduce those points of weakness.

Each one of the proposed solutions successfully addresses one of the three selected areas of improvement, solving at least partially some of the intended issues. This chapter provides general considerations on the developed tools and also indicates where further improvement can be done.

8.1.1 CVEtoCWE

The reports generated by the CVEtoCWE tool do provide the intended objective of extending the available information regarding the security of a system. Although simple in structure, these HTML reports allow for a rapid and direct visualization of the connections between vulnerabilities, weaknesses and attack patterns.

All of the aforementioned information is already publicly available in some form, but the retrieval and grouping process can be non-trivial and time consuming, as numerous resources already discussed focusing specifically on CWE and CAPEC [22], or including more general concepts of security [23].

In the particular instance of the CVEtoCWE tool, the information regarding weaknesses can be easily accessed through the NVD, as the page that describes a vulnerability contains the reference to the correspondent weakness.

The same statement cannot be made regarding the inclusion of attack patterns, which requires a more elaborated process. At the actual state of things, only the CAPEC definitions contain the list of related weaknesses, and not vice versa. The work flow of this tool starts from vulnerabilities, links them to CWEs, and only then can be related to attack patterns: this final step required the internal creation of a list that maps each weakness to the related CAPEC entries.

Moving on to the topic of performance, the biggest bottleneck lies in the waiting time of the API responses. These responses arrive through the internet and their travel times can vary greatly based on the connection speed or the current network traffic. The tool allows for asynchronous requests, meaning that it can wait for multiple responses concurrently, drastically reducing the total time spent.

Any performance problem can arise only when evaluating highly vulnerable systems, where hundreds of different CVE needs to be retrieved. In these cases, it is also possible that the NVD server responds to some requests with a “403: Forbidden” code. This behaviour likely consists in some sort of prevention against DoS attacks, when a high number of requests from the same source are generated.

If this issue occurs, simply waiting some time before trying again should fix the problem. Notice that in a typical real world scenario it's difficult to find such a high number of vulnerabilities on a single system. The definitive resolution to this problem lies in the use of a NVD API key, which can be requested at the NVD site[1]. Using this key extends the limits on the number and frequency of API requests that can be sent, and should be requested when dealing with highly vulnerable systems.

8.1.2 Container Remediation

The ContainerRem tool by itself provides the desired functionality of evaluating and remediating container images without generating any issue. The only real limitations and critiques that can be pointed out can be traced to external sources, in particular the OpenSCAP scanner and the source content.

As shown in the testing chapter, both time consumed and disk occupation depend heavily on the download and installation of the OpenSCAP scanner. Given the high variety of functionalities that are provided by the tool, it comes as no surprise that its installation size is quite significant. It seems that only two approaches are suitable to solve the problem: either modify the scanner itself, creating a smaller custom tool or using another scanner entirely, or somehow manage to exploit the tool installed on the host machine that is running the container.

Both these solutions are non-trivial and may require a considerable amount of development and testing in order to match the functionalities and stability of the standard OpenSCAP scanner.

The other limitation concerns the actual input content that is used to evaluate the container images. SCAP-expressed checklists are still not thoroughly adopted as standard in many use cases, especially when it comes to relatively recent concepts of virtualization like, for instance, containers.

It follows that many security checklists that are available, including the ones that are part of the SCAP Security Guide collection, are designed having as a target physical machines and their typical context of application. When using these checklists as base for evaluation of an image, many of the rules contained within the checklist cannot be checked and thus result as "Not Applicable".

This limits the confidence and the scope of the security aspects that are tested during an evaluation. The real solution must come from the SCAP content generators, which should focus on creating security checklists that are well-suited for container images.

As new content becomes available, the ContainerRem tool will be capable of further extending the number of supported images. The structure of the tool allows for a seamless and simple integration of additional image types and the use of diverse security policies.

8.1.3 Continuous monitoring

The various SCAPmonitor tools have demonstrated to be quite fast and efficient in their execution. Section 7.3 analysed the tool performance on different machines and under different stress conditions, and the execution time and processor load were always quite low and manageable.

Nevertheless, the exact work load imposed on the monitored system can further be adjusted by appropriately setting the frequency of polling for those checks that utilize that technique. There may be some scenarios where security is not the top priority, in which case the waiting time can be in the order of minutes. Other much more rigorous scenarios might want to lower that time to really increase the level of security confidence: numbers as low as three to four seconds are still a feasible option.

[1] https://nvd.nist.gov/developers/request-an-api-key

The main consideration that needs to be done regarding the SCAPmonitor tools is that their development is strictly tied to the actual contents of the security policies. Different checklists are used to audit different aspects of a system, and consequently require different modes of monitoring.

Hence, the extension to new use cases can take a considerable amount of work, as every rule contained in the checklist needs a specific implementation that allows the detection of the particular events that may bring the system in a non-compliant state.

8.2 Suggestions on future works

Regarding the incorporation of new specifications in SCAP, the next step for the process of vulnerability assessment after CWE and CAPEC would be MITRE ATT&CK. This standard is a collection of adversary techniques and tactics that can be used to develop threat models and subsequently create appropriate defence mechanisms [24].

The concepts that are categorized and described by CAPEC and ATT&CK, namely attack patterns and adversary tactics, cannot be directly mapped one to one, but it's rather a more complex relation [23]. Typically, attack patterns are used by malicious agents through the specific techniques described by ATT&CK.

These techniques provide additional information regarding the operational phases of the attackers, including pre and post-exploitation. This acquired knowledge is valuable in the deployment of an effective defence, but it can be difficult to include it in the context of automated security, as the specific characteristics and environment of a system heavily influence the actual deployment of countermeasures.

Another area with opportunities for improvement lies in the remote continuous monitoring using an external entity for storing reports. As of now, the SCAPmonitor tool implements this functionality only if using a git repository previously configured in a specific way.

From a theoretical perspective, the external server can be deployed with a variety of technologies as long as it acts as a database that can be accessed automatically. A more complete solution for the tool should be compatible with multiple technologies.

The automation aspect is also important. Some degree of manual intervention in setting up the system and the database may be always required, but there is still room for improvement to reduce this manual operation as much as possible.

Appendix A

User manual

This manual is intended to be a guide for users that want to utilize these tools for the most common use-cases for which they were designed. The manual is split into three sections that reflect the three main tools of the proposed solutions: CVEtoCWE, ContainerRem, and SCAPmonitor.

For each one of these sections, the link to the respective GitHub repository that contains the code is provided. Then follow the instructions for installation and utilization of the tools, alongside examples for some common use-cases.

A.1 CVEtoCWE

The contents of the CVEtoCWE tool can be found and downloaded from:

https://github.com/maxwhy/CVEtoCWE

The tool is a Node.js application, therefore it is required that the host system installs the Node.js package, which is available for the vast majority of systems.[1] The installation of the Node.js source code already includes the default package manager **npm**. The CVEtoCWE tool can be installed globally on the machine with the following command:

```
$ npm install -g
```

Now the tool is ready to be used. First, it is possible to see the help page:

```
$ cvetocwe --help
  Usage: cvetocwe [options] [command]
  Options:
    -h, --help display help for command
  Commands:
    show <OVALpath> Prints to terminal the list of CVE identifiers and their
        relative CWEs
    save [options] <OVALpath> Saves into an HTML file the list of CVE
        identifiers and their relative CWEs
    help [command] display help for command
```

In order to see the available options for the *save* command, the following command can be inserted:

```
$ cvetocwe save --help
  Usage: cvetocwe save [options] <OVALpath>
  Saves into an HTML file the list of CVE identifiers and their relative CWEs
```

[1] https://nodejs.org/it/download/

```
Options:
  --reportname <reportname> The name of the file where the list is saved.
      If omitted, the default name is report.html
```

CVEtoCWE provides two main functionalities when given as input an OVAL Result file. The first ("show") is to print on the screen the list of all the vulnerabilities that affect the system alongside the corresponding CWE identifiers. The second ("save") is to generate an HTML report that contains the list of CVEs, the related CWEs, and finally the correspondent CAPEC identifiers.

When using the save command, it is possible to specify the desired name and location of the generated report with the "reportname" option. The report contains the link to the official MITRE page of each included CWE and CAPEC entry.

It is possible that some OVAL Result files contain references to CVEs that are not present on the NVD, or that the API of some vulnerabilities cannot be retrieved as expected. In these cases, the tool will show a warning message on the terminal, reporting on which CVE identifier caused the issue and showing the problem type.

Some other CVEs do not have a specific weakness assigned to them, as the NVD uses only a slice of all the possible CWEs. In these cases, the output shows the corresponding value of either "NVD-CWE-noinfo" or "NVD-CWE-Other".

A.2 ContainerRem

The contents of the ContainerRem tool can be found and downloaded from:

https://github.com/maxwhy/ContainerRem

In order to run the tool, the host system needs the Podman container engine installed. The instructions for its installation vary for different operating systems, and can be found at the official Podman project page.

Once downloaded, the tool presents itself as a shell script file *ContainerRem.sh* and three folders named "fedora", "ubuntu1804" and "ubuntu2004".

To start utilizing the tool and get the first informations, simply open a terminal and run the shell script without any additional parameter:

```
$ bash ContainerRem.sh
Usage: bash ContainerRem.sh imagename
Where imagename can be 'fedora', 'ubuntu1804', or 'ubuntu2004'.
```

As the output suggests, the user must specify the type of container image that needs to be remediated. At the moment, three types of images are supported:

- Fedora 33 or later, using "fedora";
- any version of Ubuntu Bionic (18.04.x), using "ubuntu1804";
- any version of Ubuntu Focal (20.04.x), using "ubuntu2004";

The tool detects if the referenced image is already present on the system, otherwise it pulls it from the official image registry. Then it shows in the terminal window the various steps of building the image, running the container remediation with OpenSCAP, and finally committing the remediated image.

Note that the first time that the image is built the installation of some packages may take up to three or four minutes, depending on the internet connection speed, in particular when building the Fedora image. Any subsequent use of the ContainerRem tool for that type of image will have a much shorter execution time, typically around a dozen seconds.

At the end of the execution, the user will find two new images and an HTML report. As an example, here is the list of images after the tool has been executed for the remediation of all three versions:

```
$ podman images
REPOSITORY                          TAG
localhost/focal_remediated_image    latest
localhost/bionic_remediated_image   latest
localhost/fedora_remediated_image   latest
localhost/focal_non_remediated      latest
localhost/bionic_non_remediated     latest
localhost/fedora_non_remediated     latest
```

Where the non-remediated version represents the state of the image just before the start of the OpenSCAP evaluation. On the other hand, the remediated version is the final output of the tool, a version of the image with the applied fixes found on the SCAP content.

Lastly, by navigating to the previously mentioned folders, the user can find the HTML reports that were generated by OpenSCAP during the remediations. Each image type has its own folder so that the reports are already differentiated and can be easily accessed. The reports show which rules were evaluated and their status (passed, failed, or not applicable), alongside the results of the applied remediations.

A.3 SCAPmonitor

The contents of the various SCAPmonitor tools can be found and downloaded from:

https://github.com/maxwhy/SCAPmonitor

This repository includes five files: *SCAPmonitor.sh* and *SCAPmonitor18.sh* that are the shell scripts used to monitor a Ubuntu 20.04 and a Ubuntu 18.04 system respectively, two SCAP source data content for those operating systems (*ssg-ubuntu2004-ds.xml* and *ssg-ubuntu1804-ds.xml* retrieved from the SCAP Security Guide collection), and lastly a modified version for Ubuntu 18.04 systems, *SCAPmonitor18git.sh*, that allows the copy of reports on a remote repository.

In order for these tools to function properly, the target system of monitoring must have the Linux inotify utilities installed. They are available by downloading the corresponding package with the following command:

```
$ sudo apt install inotify-tools
```

In the case of the SCAPmonitor18git tool, the git package is also required:

```
$ sudo apt install git
```

The functioning of the tools for the two Ubuntu versions is analogous. First, simply running the scripts with no parameters prints the instructions for the correct usage:

```
$ bash SCAPmonitor.sh
  Usage: bash SCAPmonitor.sh filepath
  Where filepath refers to the ssg-ubuntu2004-ds.xml of SSG
```

As stated above, the tool can be started by providing the path to the correspondent source file. Assuming that the SCAP content is in the same location as the script, the command takes the following form:

```
$ bash SCAPmonitor.sh ssg-ubuntu2004-ds.xml
  SCAPmonitor is starting...
  SCAPmonitor is running.
```

Starting from this moment, the tool is continuously monitoring the system for specific events that can be potentially dangerous. When an event is detected, the tool prints on the terminal window the type and location of the event. Then, an OpenSCAP evaluation and remediation is issued.

For example, this is the expected output if SCAPmonitor detects a change in the permissions of the file *passwd*, located in the *etc* directory:

```
New event detected:
Directory: /etc   events: ATTRIB   file: /etc/passwd
Starting remediation...
...
Remediation completed.
```

Instead of the three dots, any warning or error generated during the security scanning and remediation is shown.Every OpenSCAP evaluation generates an HTML report that contains the rules results and shows if the remediations succeeded.

The tool keeps running until stopped with the abort signal generated by pressing "CTRL+C".

As far as SCAPmonitor18git is concerned, its functioning is partially different and can directly be seen when running the script with no parameters:

```
$ bash SCAPmonitor18git.sh
  Usage: bash SCAPmonitor18git.sh filepath repoPath nodeBranch
  Where filepath refers to the ssg-ubuntu1804-ds.xml of SSG, repoPath is the
      git repository, nodeBranch is the local branch.
```

The two new parameters, "repoPath" and "nodeBranch", refer to the folder where the remote git repository has been locally imported and the name of the dedicated repository branch created for the specific node that is being monitored. Please refer to the developer manual for a detailed description on how to set up the repository and utilize this functionality.

This git version of SCAPmonitor behaves as the previously discussed tools, but stores the reports generated by OpenSCAP during each evaluation in a remote repository.

Appendix B

Developer manual

This manual is intended for developers searching for the implementation details of the tools. The manual is split into three sections that reflect the three main tools of the proposed solutions: CVEtoCWE, ContainerRem, and SCAPmonitor.

This guide extends the contents of the standard user manual by including technical considerations and in-depth analysis of the actual implementation of the tools. In particular, the following sections highlight the software requirements for the installation of the tools and carefully describe the programming choices and the main internal functions.

B.1 CVEtoCWE

The contents of the CVEtoCWE tool can be found and downloaded from:

https://github.com/maxwhy/CVEtoCWE

The tool is a CLI Node.js application that needs the relative package installed on the system to be able to run. The following commands also installs the default package manager **npm**:

```
$ apt install nodejs
$ apt install npm
```

After the installation is completed the tool can be downloaded and installed by opening a terminal and typing:

```
$ npm i -g
```

Internally, this command also installs the various packages that are required by the tool:

- commander 8.3.0;
- conf 10.1.1;
- fs 0.0.1-security;
- xhr2 0.2.1;
- xml2json 0.12.0.

The *index.js* file is the starting point of the tool, which accepts the input parameters and calls one of the two scripts contained in the commands folder.

The *show.js* script takes as input the OVAL Results file and converts it to JSON, so that its contents can be easily accessed. Then the tool searches for every OVAL Definition that checks for the presence of vulnerabilities (which are the definitions of the "patch" class) that are marked as "true", which means that those vulnerabilities are present on the system.

Only the vulnerabilities that refer to a CVE are considered, as some OVAL patch definitions utilize non-SCAP formats such as Ubuntu Security Notice (USN) or other proprietary vulnerability collections. For every found CVE, a REST API is called with the following url:

```
url = "https://services.nvd.nist.gov/rest/json/cve/1.0/" + CVE_ID;
```

The CVEtoCWE tool sends an asynchronous HTTP GET request for each CVE. When the response returns, in JSON format, the CWE field is retrieved and the output is shown in the terminal in the form of "CVE : CWE". If the response generates an error such as "404 Not Found" or doesn't return at all, the tool prints a warning message specifying which CVE generated the error.

On the other hand, the *save.js* script initially follows the same steps but then elaborates more the API responses. Notably, each CVE and the correspondent retrieved CWEs are stored in two dedicated arrays, called CVE_list and CWE_list. Once every API request is completed, excluding the ones that returned as errors, the report can be generated starting from those two arrays.

As last step before the actual creation of the report, CAPEC is also considered. Within the tool files, the one named "1000.json" contains a list of all the CAPEC entries that are associated with one or more CWEs. This file was created starting from the CAPEC-1000 view ("Mechanisms of Attack"), which contains all of the other CAPEC entities, that is available for download as CSV file[1]

The tool takes the file in JSON format and creates an internal array with the list of attack patterns that relate to a specific CWE. Finaly the report can be generated by creating a table that contains the list of all CVEs, their correspondent CWEs, and the related CAPEC entries. The link to each CWE and CAPEC reference MITRE page is provided.

B.2 ContainerRem

The contents of the ContainerRem tool can be found and downloaded from:

https://github.com/maxwhy/ContainerRem

In order to run the tool, the host system needs the Podman container engine installed. The instructions for its installation vary for different operating systems, and can be found at the official Podman project page.

Once downloaded, the tool presents itself as a shell script file *ContainerRem.sh* and three folders named "fedora", "ubuntu1804" and "ubuntu2004".

The main execution steps of the ContainerRem tool are as follows:

1. The shell script receives the image type as argument and sets the value of some specific variables.
2. The current directory is changed to the folder that corresponds to the image type.
3. The image is built using the dedicated Dockerfile.
4. The OpenSCAP evaluation and remediation is executed on a container.
5. The generated report is copied back to the host.
6. The new remediated container is committed as a new image.

[1] https://capec.mitre.org/data/definitions/1000.html

The variables that are set on step 1 identify the name of the various images and containers that are used during the execution, as well as the name of the SCAP content that is used to evaluate the image.

Each folder contains the relative SCAP content from the SCAP Security Guide project and a Dockerfile. Each Dockerfile contains different commands that are needed to copy the right SCAP content in the image and to install the relative OpenSCAP package.

Once the final remediated image is committed, the container that generated it can be removed to clean the environment. The report remains on the host system inside the relative image type folder.

B.3 SCAPmonitor

The contents of the various SCAPmonitor tools can be found and downloaded from:

https://github.com/maxwhy/SCAPmonitor

It is required the installation of the *inotify-tools* package, which can be installed on systems that support the "Inotify" Linux kernel feature. Inotify is compatible with machines that have a kernel version 2.6.13 or later. To check the version of the kernel, type the following command:

```
$ uname -a
Linux nodename 5.3.0-28-generic ...
```

Among the Intofy utilities, the *inotifywait* command is used extensively in the SCAPmonitor tools, as it allows to efficiently wait for specific events using the Inotify interface.

Each rule (or group of similar rules) contained in the source SCAP content has a correspondent section in the SCAPmonitor code which monitors the events that can potentially change the output of that rule. The monitoring of many rules present the following form:

```
while :
do
  inotifywait -q -e EVENT PATH |
  while read -r dir events filename; do
    echo "New event detected:"
    echo "Directory: $dir events: $events file: $filename"
    echo "Starting remediation..."
    oscap xccdf eval --remediate --report compliance-report-new.html \
    --profile standard $FILEPATH &>/dev/null
    echo "Remediation completed."
    echo ""
  done
done &
```

The entire section is contained within an infinite while cycle, and the "&" at the end allows for the concurrent execution of all the sections. The *inotifywait* command blocks the cycle execution until a specific event occurs at the specified location. PATH can be replaced with the reference to a single file or to an entire directory, while EVENT is used to describe the event type. Two event types are used in SCAPmonitor:

- modify: when the monitored file (or a file within the monitored directory) has been modified;
- attrib: when the metadata of the monitored file (or a file within the monitored directory) has changed, for example changes to permissions.

As soon as the target event is detected, the tool outputs the type of event on the terminal and issues an OpenSCAP evaluation. The *remediate* option allows OpenSCAP to actively remediate the system. Once the remediation is completed, the cycle continues and a new *inotifywait* command is called, which monitors again the same rules.

Other rules cannot be checked by monitoring file changes alone. These rules typically check for the presence (or absence) of specific services and packages. In this case, the monitoring is performed with a polling mechanism, repeating the rule verification every fixed amount of time. A dedicated variable, "SLEEPTIME", can be set to the desired value of seconds. The default is ten seconds.

Remote monitoring with external repository

The *SCAPmonitor18git* script is a modified version of the SCAPmonitor tool for Ubuntu 18.04, which allows the upload of the reports generated with each evaluation on a remote git repository. There are some particular steps that must be performed manually before using the tool.

The remote server which stores the repository should have a dedicated user and directory for these git operations. The server shall have both the SSH service active and the git package installed. The repository can be initialized with the following command:

```
$ git init --bare repositoryName.git
```

After that, the system that needs to be monitored requires some passages too. An SSH key pair is required, and the public key should be copied on the server as authorized key. This step can also be performed with the *ssh-copy-id* utility, and guarantees that all the git operations can be performed automatically.

The git package must be installed on the system, and a dedicated directory for the repository shall be created. The location of this directory is the second parameter that is required by the SCAPmonitor git tool. The local git repository can be initialized with the following commands:

```
$ git init .
$ fit remote add origin user@serverIP:/repositoryName.git
$ git config --global user.email "example@example.com"
$ git config --global user.name "example"
```

Where "user" and "serverIP" shall be the aforementioned dedicated git user and the IP address of the server. The monitored system should also have a dedicated branch on the repository, as the typical use case involves several nodes being monitored at the same time:

```
$ git checkout -b exampleBranch
$ git push origin exampleBranch
$ git commit -m "First commit"
$ git push -u origin exampleBranch
```

The tool can now be executed on the target system. The correct syntax is as follows (supposing that the local repository path is "NewDirectory"):

```
$ bash SCAPmonitor18git.sh ssg-ubuntu1804-ds.xml NewDirectory exampleBranch
```

The tool will start monitoring the system until explicitly stopped. The functionalities are the same of the standard SCAPmonitor tool, with the addition of the automatic commit and push of reports to the remote repository.

Bibliography

[1] L.LaBerge, C.O'Toole, J.Schneider, K.Smaje, "How COVID-19 has pushed companies over the technology tipping point and transformed business forever". McKinsey & Company, October 2020. https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever

[2] "2020 State of SecOps and Automation", A Survey of IT Security Professionals. Dimensional Research, sponsored by Sumo Logic, June 2020. https://www.sumologic.com/brief/state-of-secops/

[3] D.Waltermire, S.Quinn, H.Booth, K.Scarfone, D.Prisaca, "The Technical Specification for the Security Content Automation Protocol (SCAP)". National Institute of Standards and Technology, Special Publication 800-126, Rev. 3, February 2018. DOI 10.6028/NIST.SP.800-126r3

[4] D.Waltermire, C.Schmidt, K.Scarfone, N.Ziring, "Specification for the Extensible Configuration Checklist Description Format (XCCDF)". National Institute of Standards and Technology, Interagency Report 7275, Rev. 4, March 2012. https://csrc.nist.gov/publications/detail/nistir/7275/rev-4/final

[5] B.A.Cheikes, D.Waltermire, K.Scarfone, "Common Platform Enumeration: Naming Specification". National Institute of Standards and Technology, Interagency Report 7695, Version 2.3, August 2011. DOI 10.6028/NIST.IR.7695

[6] The OpenSCAP project, https://www.open-scap.org/

[7] Security Content Automation Protocol Validation Program, NIST, July 2021, 142 Red Hat SCAP 1.2 Product Validation Record, https://csrc.nist.gov/projects/scap-validation-program/validated-products-and-modules/142-red-hat-scap-1-2-product-validation-record

[8] The NFV project, European Telecommunications Standards Institute (ETSI), https://www.etsi.org/technologies/nfv

[9] S.Lal, T.Taleb, A.Dutta, "NFV: Security Threats and Best Practices". IEEE Communications Magazine, 2017. DOI 10.1109/MCOM.2017.1600899

[10] The OPNFV project, https://www.opnfv.org/

[11] L.Hinds, November 2016. Security Scanning, wiki.opnfv.org. https://wiki.opnfv.org/pages/viewpage.action?pageId=6826205

[12] B.Baude, May 2016. Introducing atomic scan - Container vulnerability detection, developers.redhat. https://developers.redhat.com/blog/2016/05/02/introducing-atomic-scan-container-vulnerability-detection

[13] M.Jahoda, J.Fiala. Red Hat Enterprise Linux 7 - Security Guide, Chapter 8, access.redhat. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/index

[14] P.Klinker, June 2020. Securing Container Images Using OpenSCAP and Atomic, pklinker.medium https://pklinker.medium.com/securing-container-images-using-openscap-and-atomic-7e3d94322cae

[15] Security Content Automation Protocol Version 2 (SCAP v2), National Institute of Standards and Technology, https://csrc.nist.gov/Projects/security-content-automation-protocol-v2

[16] S.Kinzer, October 2015. Using CPEs for Open-Source vulnerabilities? Think Again, veracode. https://www.veracode.com/blog/managing-appsec/using-cpes-open-source-vulnerabilities-think-again

[17] The Podman project, https://podman.io/
[18] The Common Weakness Enumeration project, https://cwe.mitre.org/
[19] NVD CWE slice, National Institute of Standards and Technology, https://nvd.nist.gov/vuln/categories
[20] Common Weakness Scoring System, cwe.mitre. https://cwe.mitre.org/cwss/cwss_v1.0.1.html
[21] The Common Attack Pattern Enumerations and Classifications project, https://capec.mitre.org/
[22] A.Brazhuk, "Semantic model of attacks and vulnerabilities based on CAPEC and CWE dictionaries", International Journal of Open Information Technologies. ISSN: 2307-8162 vol. 7, no. 3, 2019
[23] E.Hemberg, J.Kelly, M.Shlapentokh-Rothman, B.Reinstadler, K.Xu, N.Rutar, U.O'Reilly "Linking Threat Tactics, Techniques, and Patterns with Defensive Weaknesses, Vulnerabilities and Affected Platform Configurations for Cyber Hunting". In Proceedings of ACM Conference (Conference'17). ACM, New York, NY, USA. DOI 10.1145/ nnnnnnn.nnnnnnn
[24] The MITRE ATT&CK project, https://attack.mitre.org/