BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

OPENSCAP REPORT: A TOOL FOR VISUALIZING SECURITY COMPLIANCE INSPECTION RESULTS

OPENSCAP REPORT: NÁSTROJ PRO VIZUALIZACI VÝSLEDKŮ KONTROLY DODRŽOVÁNÍ BEZPEČNOSTNÍCH PŘEDPISŮ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR   **JAN RODÁK**
AUTOR PRÁCE

SUPERVISOR   **Mgr. JOZEF DRGA,**
VEDOUCÍ PRÁCE

BRNO 2023

Bachelor's Thesis Assignment

Institut:	Department of Intelligent Systems (UITS)
Student:	**Rodák Jan**
Programme:	Information Technology
Specialization:	Information Technology
Title:	**OpenSCAP Report: A Tool for Visualizing Security Compliance Inspection Results**
Category:	Security
Academic year:	2022/23

Assignment:

1. Get acquainted with SCAP standards and explore HTML reports generated by OpenSCAP security scanners or other tools.
2. Suggest content improvements to the report.
3. Create a tool for processing SCAP results of the OpenSCAP scanner and generating an interactive report in HTML format.
4. Create the package available for Fedora (and RHEL 9?).
5. Create a user-testing methodology.
6. Perform user testing according to the methodology and evaluate the results.

Literature:
- The SCAP - Security Content Automation Protocol, https://csrc.nist.gov/projects/security-content-automation-protocol/
- Publication of SCAP: https://csrc.nist.gov/publications/detail/sp/800-126/rev-3/final
- The OpenSCAP portal, https://www.open-scap.org/
- Fedora packaging guidelines, https://docs.fedoraproject.org/en-US/packaging-guidelines/
- Designing with data: improving the user experience with A/B testing, https://primo.lib.vutbr.cz/permalink/f/1roshr/420BUT_Aleph000142118
- Don´t make me think!: a common sense approsach to web usability, https://primo.lib.vutbr.cz/permalink/f/1roshr/420BUT_Aleph000076907

Requirements for the semestral defence:
Completed points 1 and 2.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Drga Jozef, Mgr.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2022
Submission deadline:	10.5.2023
Approval date:	3.11.2022

Abstract

The goal of the thesis is to develop a utility to present the results of OpenSCAP security scans. SCAP scans use standardized input and output formats, and those formats are not consumable by humans.

The utility aims to present the SCAP scan output in the form of an interactive report that helps find the root cause of failed security requirements and enables users to understand the composition of the respective security checks.

The report will allow users of OpenSCAP and developers of security profiles to debug their checks. It will provide insights into relations between individual checks and help understand the context of these checks within SCAP security profiles.

Abstrakt

Cílem práce je vyvinout nástroj pro prezentaci výsledků bezpečnostního skeneru Open-SCAP. SCAP skenery používají standardizované vstupní a výstupní formáty, které nejsou čitelné člověkem.

Cílem nástroje je prezentovat výstup SCAP skeneru ve formě interaktivního reportu, který pomůže najít hlavní příčinu selhání bezpečnostních požadavků. Dovolí uživatelům porozumět složení příslušných bezpečnostních kontrol.

Report umožní uživatelům OpenSCAP i vývojářům bezpečnostních profilů ladit jejich kontroly. Poskytne pohled na vztahy mezi jednotlivými kontrolami a pomůže pochopit kontext těchto kontrol v rámci bezpečnostních profilů SCAP.

Keywords

SCAP, OpenSCAP, ARF, XCCDF, OVAL, security compliance, audit, UI/UX

Klíčová slova

SCAP, OpenSCAP, ARF, XCCDF, OVAL, bezpečnostní předpisy, audit, UI/UX

Reference

RODÁK, Jan. *OpenSCAP Report: A Tool for Visualizing Security Compliance Inspection Results*. Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Mgr. Jozef Drga,

OpenSCAP Report: A Tool for Visualizing Security Compliance Inspection Results

Declaration

I hereby declare that this Semestral project was prepared as an original work by the author under the supervision of Mgr. Jozef Drga. The supplementary information was provided by Evgeny Kolesnikov. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .

Jan Rodák

May 8, 2023

Contents

List of Figures

Chapter 1

Introduction

Cybersecurity has been growing in importance over the last few years. Many organizations such as large enterprises, governments, hospitals, the military, and airports are using computers to organize infrastructure and manufacturing, communicate, process data, and serve customers. These computers might contain sensitive data or might be part of a company's critical infrastructure. Protection and security for that machines are becoming an increasingly important issue for many companies.

In the field of computer security, there are different approaches to protecting computers: antivirus programs, firewalls, attack detection, or automated security audits.

Automated security audits (with accordance to the organization's security policy) can reveal vulnerabilities in systems. These security policies, through a set of rules and recommendations, define what a secure system should look like.

Traditional manual security audits are time-consuming and increase the potential for human error. The Security Content Automation Protocol (SCAP) has been created to reduce the time required for such kind of audits and amount of mistakes. Security scanners that implement SCAP have standardized input and output formats.

The scanner uses a profile. A profile is a machine-readable version of a security policy that the scanner understands. A profile consists of rules. These rules are evaluated. The overall result of the scan is a cumulative score and/or a number of failed rules. It describes the extent to which the system complies with the security policy.

Scanners usually offer several versions of result reports. But, commonly, these reports are in a machine-readable format. And sometimes they lack information about the tests that were performed on the system and the relationship between those tests and the rules. Finding the root cause of the rule's failure is not an easy task: users have to get a grip on large reports that were created for machines.

This work focuses on developing reports that help users understand the logic behind complex security checks. The main standardized output (that would be used as the tool's main input) is the Asset Reporting Format (ARF). This format is based on XML and contains all the information about the security checks performed.

Chapter 2

Security Content Automation Protocol

The Security Content Automation Protocol (SCAP) [25] is a set of specifications that standardize the form in which security configuration requirements are represented to machines and humans. For example, SCAP can be used for automated configuration, vulnerability and patch enumeration, technical control compliance activities, and security measurement.

According to SCAP technical specifications version 1.3 [25]. SCAP components can be divided into several categories: "

- *Languages – The SCAP languages provide standard vocabularies and conventions for expressing security policy, technical check mechanisms, and assessment results. The SCAP language specifications are Extensible Configuration Checklist Description Format (XCCDF), Open Vulnerability and Assessment Language (OVAL), Open Checklist Interactive Language (OCIL), and Common Platform Enumeration (CPE) – Applicability Language.*

- *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. Although Asset Identification is not explicitly a reporting format, SCAP uses it to identify the assets that reports relate to.*

- *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 on 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 – In SCAP this refers to evaluating specific characteristics of a security weakness (for example, software vulnerabilities and security configuration issues) and, based on those characteristics, generating a score that reflects their relative severity. The SCAP measurement and scoring system specifications are the Common Vulnerability Scoring System (CVSS) and Common Configuration Scoring System (CCSS).*

5

- *Integrity – A SCAP integrity specification helps to preserve the integrity of SCAP content and results. The Trust Model for Security Automation Data (TMSAD) is the SCAP integrity specification.*

"

The categories of Language and Report Formats are important for this work. All of the languages or reporting formats used in SCAP are based on XML. All of these documents have an XML schema that can be used to validate the XML document format. The source of content for SCAP scanners is a document called a Data Stream, a structure that incorporates SCAP components into a single file. This is useful for simplifying SCAP content handling. Data Stream mainly contains documents written in the following languages:

- The XCCDF language, used to describe security checklists,

- The OVAL language, used to define checks of rules in security policy and declare logical assertions about the system state,

- The OCIL language, used for checking rules that cannot be fully automated with OVAL, and

- The CPE Applicability Language, that is used to declare automatic applicability tests for benchmark sub-components.

A Data Stream file usually contains several security policies. Because, in many cases, security policies overlap and share rule definitions and checks. Security policies define what a secure computer system should look like, using rules and recommendations. A machine-readable implementation of a security policy in the XCCDF language is called a Profile. A Profile consists of a list of, possibly grouped, rule definitions. Rule definitions describe the policy rules and hold references to OVAL and OCIL checks, applicability definitions (the CPE Applicability Language), and other types of metadata.

The main reporting format is an ARF document that contains all results, such as OVAL and XCCDF check results, with all SCAP components in Data Stream. It is also often called the Result Data Stream because this file contains the original Data Stream information in addition to evaluation results.

2.1 The Open Vulnerability and Assessment Language (OVAL)

The main component of SCAP is the Open Vulnerability and Assessment Language [1]. It is an XML-based language for identifying vulnerabilities and configuration problems in computer systems. OVAL standardizes the representation of automated checks in security policy.

An OVAL Definition defines a logical check of rules and describes the expected state of a computer system. The OVAL Definition contains definition and criteria metadata. The definition metadata are title, version, description, etc. See Listing 2.1 of the OVAL Definition.

```xml
<oval-def:definition class="compliance"
  id="oval:ssg-dconf_gnome_screensaver_idle_delay:def:1"
  version="2">
 <oval-def:metadata>
  <oval-def:title>
   Set GNOME3 Screensaver Inactivity Timeout
  </oval-def:title>
  <oval-def:affected family="unix">
    <oval-def:platform>Red Hat Enterprise Linux 9</oval-def:platform>
  </oval-def:affected>
  <oval-def:description>
    The allowed period of inactivity before the screensaver is activated.
  </oval-def:description>
  <oval-def:reference ref_id="CCE-86510-5" source="CCE"/>
  <oval-def:reference
    ref_id="dconf_gnome_screensaver_idle_delay" source="ssg"/>
 </oval-def:metadata>
 <oval-def:criteria operator="OR">
  <oval-def:extend_definition
    comment="dconf installed"
    definition_ref="oval:ssg-package_dconf_installed:def:1" negate="true"/>
  <oval-def:criteria
    comment="check screensaver idle delay and prevent user from changing it"
    operator="AND">
   <oval-def:extend_definition
     comment="dconf user profile exists"
     definition_ref="oval:ssg-enable_dconf_user_profile:def:1"/>
   <oval-def:criterion
     comment="idle delay has been configured"
     test_ref="oval:ssg-test_screensaver_idle_delay:tst:1"/>
   <oval-def:criterion
     comment="idle delay is set correctly"
     test_ref="oval:ssg-test_screensaver_idle_delay_setting:tst:1"/>
  </oval-def:criteria>
 </oval-def:criteria>
</oval-def:definition>
```

Listing 2.1: Example of OVAL Definition.

The OVAL Definition criteria are a tree structure consisting of several types of nodes that reference other OVAL components. Criteria nodes can be nested and are used as a logical operators. The Criteria node reflects the relationship between child nodes. This relationship is represented by logical operator types such as AND, OR, ONE, and XOR. These operators have defined logical tables[1].

Child nodes can be another Criterion node, an Extended Definition node that references another OVAL Definition, or a Criterion node that references an OVAL Test. OVAL Tests

---

[1]https://oval.mitre.org/language/version5.11/ovaldefinition/documentation/oval-common-schema.html#OperatorEnumeration

define the relationship between the OVAL Objects and OVAL States components. See Listing 2.2 of an OVAL Test.

```
<ind:textfilecontent54_test check="all" check_existence="all_exist"
   comment="screensaver idle delay setting is correct"
   id="oval:ssg-test_screensaver_idle_delay_setting:tst:1"
   version="1">
  <ind:object
     object_ref="oval:ssg-obj_screensaver_idle_delay_setting:obj:1"/>
  <ind:state
     state_ref="oval:ssg-state_screensaver_idle_delay_setting:ste:1"/>
</ind:textfilecontent54_test>
```

<div align="center">Listing 2.2: Example of OVAL Test.</div>

The OVAL State defines the expected value collected by the OVAL Object or refers to the OVAL Variable. The OVAL Variable defines values that can be referenced by the OVAL Elements such as OVAL State. The OVAL Object defines the value to be collected from the system being evaluated. See the Listing of OVAL State 2.3, OVAL Objects 2.4, and OVAL Variable 2.5.

```
<ind:textfilecontent54_state
   id="oval:ssg-state_screensaver_idle_delay_setting:ste:1" version="1">
  <ind:subexpression
     datatype="int" operation="less than or equal"
     var_check="all" var_ref="oval:ssg-inactivity_timeout_value:var:1"/>
</ind:textfilecontent54_state>
```

<div align="center">Listing 2.3: Example of OVAL State.</div>

```
<ind-def:textfilecontent54_object
   id="oval:ssg-obj_screensaver_idle_delay_setting:obj:1" version="1">
  <ind-def:path>/etc/dconf/db/local.d/</ind-def:path>
  <ind-def:filename operation="pattern match">^.*$</ind-def:filename>
  <ind-def:pattern operation="pattern match">
    ^idle-delay[\s=]*uint32[\s]([^=\s]*)
  </ind-def:pattern>
  <ind-def:instance datatype="int">1</ind-def:instance>
</ind-def:textfilecontent54_object>
```

<div align="center">Listing 2.4: Example of OVAL Object.</div>

```
<local_variable id="oval:ssg-inactivity_timeout_value:var:1"
   version="1" datatype="int" comment="inactivity timeout variable">
  <literal_component>900</literal_component>
</local_variable>
```

<div align="center">Listing 2.5: Example of OVAL Variable.</div>

OVAL results section contains information about OVAL evaluation: OVAL Objects that were collected from the targeted system and results of OVAL Objects matching against OVAL States. It also contains evaluation criteria that have the same structure as OVAL Definitions but stripped of node metadata such as comments and names. What they do contain instead is a result attribute with the conclusion of an individual criterion.

The results of Criteria, Extended Definitions, Criterion, and Tests can be negated. The values of the results are True, False, Error, Unknown, Not Evaluated, and Not Applicable. See Listing 2.6 of the OVAL Result.

```xml
<definition
   definition_id="oval:ssg-dconf_gnome_screensaver_idle_delay:def:1"
   result="false" version="2">
  <criteria operator="OR" result="false">
    <extend_definition
     definition_ref="oval:ssg-package_dconf_installed:def:1"
     version="1" result="false" negate="true"/>
    <criteria operator="AND" result="false">
      <extend_definition
       definition_ref="oval:ssg-enable_dconf_user_profile:def:1"
       version="1" result="true"/>
      <criterion
       test_ref="oval:ssg-test_screensaver_idle_delay:tst:1"
       version="1" result="false"/>
      <criterion
       test_ref="oval:ssg-test_screensaver_idle_delay_setting:tst:1"
       version="1" result="false"/>
    </criteria>
  </criteria>
</definition>
```

Listing 2.6: Example of OVAL Result.

A document that contains OVAL components is normalized. This means that all the information is distributed in the document and components cross-reference each other. Just like in a relational database. This makes it difficult for users to understand and follow the logic of tests.

2.2 Extensible Configuration Checklist Description Format (XCCDF)

Extensible Configuration Checklist Description Format (XCCDF) [26] is a language for creating security checklists and benchmarks. XCCDF contains a structured collection of security rules from security policies for a target system. Like other languages in the SCAP standards collection, XCCDF is based on XML. XCCDF defines a data model that consists of several XML elements.

The root element of an XCCDF document is the XCCDF Benchmark, which serves as a container for other XCCDF elements. The XCCDF Benchmark element contains the Group, Rule, Value, Profile, and TestResult elements.

The Group element may contain an additional Group element as a subgroup, Rule, Value, platform reference, and other descriptive Group elements. The Rule element is filled with a reference to the CPE platform, Check, Fix, and other metadata such as description, title, links, etc.

A Fix is an element that contains code of fix in Bash or a different language that is used to modify the system in a way that it would satisfy the requirement. See Listing 2.7 of XCCDF Rule.

```xml
<xccdf-1.2:Rule
  selected="false"
  id="xccdf_org.ssgproject.rule_dconf_gnome_screensaver_idle_delay"
  severity="medium">
 <xccdf-1.2:title>
  Set GNOME3 Screensaver Inactivity Timeout
 </xccdf-1.2:title>
 <xccdf-1.2:description>
  The idle time-out value for inactivity in the GNOME3 desktop
  is configured via the <html:code>idle-delay</html:code>...
 </xccdf-1.2:description>
 <xccdf-1.2:reference
  href="https://www.cisecurity.org/controls/">1</xccdf-1.2:reference>
 <xccdf-1.2:reference href="https://www.fbi.gov">
  5.5.5
 </xccdf-1.2:reference>
 ...
 <xccdf-1.2:rationale>
   A~session time-out lock is a temporary action taken
   when a user stops work...
 </xccdf-1.2:rationale>
 <xccdf-1.2:platform idref="#machine"/>
 <xccdf-1.2:ident system="https://nvd.nist.gov/cce/index.cfm">
  CCE-86510-5
 </xccdf-1.2:ident>
 <xccdf-1.2:fix system="urn:xccdf:fix:script:ansible"
   id="dconf_gnome_screensaver_idle_delay"
   complexity="low" disruption="medium" reboot="false"
   strategy="unknown">...
 </xccdf-1.2:fix>
 <xccdf-1.2:check
    system="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <xccdf-1.2:check-content-ref
    href="ssg-rhel9-oval.xml"
    name="oval:ssg-dconf_gnome_screensaver_idle_delay:def:1"/>
 </xccdf-1.2:check>
 ...
</xccdf-1.2:Rule>
```

Listing 2.7: Example of XCCDF Rule.

The Value element contains data that can be re-defined. For example, timeout value in the rule can be specific to each profile. The Profile is a customization element of the Benchmark. The profile contains references to the Rule, Group, and Value elements. The TestResult element contains results of a scan performed on the target system. TestResult references

Rules, Values, and Checks performed and may reference Profile. See Listing 2.8 of the Rules result.

```xml
<rule-result
  idref="xccdf_org.ssgproject.rule_dconf_gnome_screensaver_idle_delay"
  role="full" time="2023-03-16T13:54:43+01:00"
  severity="medium" weight="1.000000">
 <result>fail</result>
 <ident system="https://nvd.nist.gov/cce/index.cfm">CCE-86510-5</ident>
 <check system="http://oval.mitre.org/XMLSchema/oval-definitions-5">
  <check-content-ref
   name="oval:ssg-dconf_gnome_screensaver_idle_delay:def:1"
   href="#oval0"/>
 </check>
</rule-result>
```

Listing 2.8: Example of XCCDF Rule Result.

The customization element occurs only once in the XCCDF document and contains a Profile element that modifies the behavior of the benchmark. The XCCDF can contain several security policies in the form of profiles. These profiles may share the same Rules and Values.

2.3 Common Platform Enumeration (CPE)

The Common Platform Enumeration (CPE) [24] is a specification that standardizes the way operating systems, hardware devices, and other components of the target system are identified and enumerated. The CPE specification defines two main modules. One of these modules is the CPE Applicability Language, a language for describing the relationships between CPE platforms and checks. The relationship is represented by logical expressions defined in the CPE specification. Another module of the specification is CPE Dictionary, which defines a dictionary of CPE identifiers (names) and references to checks. Both specifications refer to checks written in the OVAL Language. Like other components of SCAP, they are based on XML.

The CPE Applicability Language consists of the platform element, which is a container referenced by the IT platform, for the logical-test element. The logical-test element is the root element for evaluating the CPE Applicability Language. Logical-test has operator and negation attributes. The attributes represent the relationship between the children of the logical-test element. Children of the logical-test element can check references, dictionary references, or nested logical-test elements. See Listing 2.9 of applicability checks written in the CPE Applicability Language.

```
<cpe-lang:platform-specification>
  <cpe-lang:platform id="not_aarch64_arch_and_not_ppc64le_arch">
    <cpe-lang:logical-test operator="AND" negate="false">
      <cpe-lang:logical-test operator="AND" negate="true">
        <cpe-lang:fact-ref name="cpe:/a:aarch64_arch"/>
      </cpe-lang:logical-test>
      <cpe-lang:logical-test operator="AND" negate="true">
        <cpe-lang:fact-ref name="cpe:/a:ppc64le_arch"/>
      </cpe-lang:logical-test>
    </cpe-lang:logical-test>
  </cpe-lang:platform>
  ...
</cpe-lang:platform-specification>
```

Listing 2.9: Example of CPE Aplicibility Language.

The CPE Dictionary maps CPE names to OVAL Checks. The CPE Dictionary can be used with the CPE Applicability Language or on its own. In older versions of the Open-SCAP scanner and content for the OpenSCAP scanner, the relationship between platforms is represented differently. The OVAL Language is used. IT platforms are specified using the CPE name in the Rule, Group, or Profile element with the Platform element. See Listing 2.10 of the CPE Dictionary.

```
<cpe-dict:cpe-list
  xsi:schemaLocation="http://cpe.mitre.org/dictionary/2.0
  http://cpe.mitre.org/files/cpe-dictionary_2.1.xsd">
 <cpe-dict:cpe-item name="cpe:/a:aarch64_arch">
   <cpe-dict:title xml:lang="en-us">
     System architecture is AARCH64
   </cpe-dict:title>
   <cpe-dict:check
     system="http://oval.mitre.org/XMLSchema/oval-definitions-5"
     href="ssg-rhel9-cpe-oval.xml">
       oval:ssg-proc_sys_kernel_osrelease_arch_aarch64:def:1
   </cpe-dict:check>
 </cpe-dict:cpe-item>
 <cpe-dict:cpe-item name="cpe:/a:audit:">
   <cpe-dict:title xml:lang="en-us">
     Package audit is installed
   </cpe-dict:title>
   <cpe-dict:check
     system="http://oval.mitre.org/XMLSchema/oval-definitions-5"
     href="ssg-rhel9-cpe-oval.xml">
    oval:ssg-package_audit:def:1
   </cpe-dict:check>
 </cpe-dict:cpe-item>
 ...
</cpe-dict:cpe-list>
```

Listing 2.10: Example of CPE Dictonary.

Chapter 3

SCAP Scanners

SCAP scanners are the tools used to perform security audits. Usually, these tools are accompanied with security profiles, implementations of specific security policies. The largest open-source implementation of a SCAP scanner is the OpenSCAP project, the core of which is the OpenSCAP library, an implementation of the SCAP standard. It is also the base for different helper components like Anaconda Installer OpenSCAP Add-on.

3.1 OpenSCAP Projects

OpenSCAP [6] is an open-source project that is developing an ecosystem with a range of tools and components to help administrators and auditors assess, evaluate and enforce security baselines.

Development of this project began in November 2008 within Red Hat [23].

The tools that are part of the OpenSCAP ecosystem are:

- OpenSCAP base,

- SCAP Workbench,

- ComplianceAsCode,

- and others.

OpenSCAP Base

OpenSCAP Base [5] is a library and provides a command line tool that can be used to analyze and evaluate the individual components of the SCAP standard. The library-based approach allows rapid creation of new SCAP-based tools. OpenSCAP has received Security Content Automation Protocol (SCAP) 1.2 certification from the National Institute of Standards and Technology (NIST), which means that OpenSCAP is fully SCAP 1.2 compliant. The command-line tool called `oscap` provides a multipurpose tool for evaluating a system based on any SCAP content.

SCAP Workbench

SCAP Workbench [7] is a tool that provides a graphical user interface (GUI) to easily perform scans using `oscap`. This tool allows users to scan local or remote systems, perform system remediation, generate multiple reports in different formats containing system scan

results, and easily edit the XCCDF profile without having to modify the corresponding XCCDF file. The tool provides a graphical way to enable or disable XCCDF elements and save changes to the XCCDF tailoring file.

ComplianceAsCode Project

ComplianceAsCode is a very important part of the ecosystem because a SCAP scanner without SCAP content is useless. SCAP content is standardized input for SCAP scanners in a machine-readable format that determines how the scanner evaluates the system based on the set of rules in a given security policy. ComplianceAsCode [9] is an open-source project that provides a SCAP Security Guide package[1].

As the SCAP Security Guide [22] has grown and the goals of the project have expanded to also making contributions to the SCAP content more accessible to non-programmers, the name became a bit obsolete. As a result, the project was renamed to ComplianceAsCode. The SCAP Security Guide [8] implements security policies recommended by recognized authorities (the Payment Card Industry Security Standards Council (PCI SSC), Security Technical Implementation Guides (STIG), the United States Government Configuration Baseline (USGCB), etc.) for 18 products [21] including Red Hat Enterprise Linux, Fedora, Debian, Firefox, Chromium and others. The SCAP Security Guide translates these security policies into a machine-readable format that SCAP scanners can use to perform audit.

3.2 3-rd Party SCAP-compatible Scanners

There are many NIST-certified SCAP scanners. The list on the NIST website[2] provides an overview of available tools. Despite the visible variety the options are often obsolete or are a part of a paid solutions. The list includes:

- SCAP Compliance Checker (SCC),

- Qualys SCAP Auditor,

- and more.

All implementations are compliant with SCAP standards, but each implementation of the scanner and SCAP content may differ in some details. This can cause compatibility issues.

For example, when using content from the SCAP Security Guide with the SCAP Compliance Checker. The same compatibility issue exists with the visualization of SCAP scanner results. The `openscap-report` is aimed to support reports generated with the OpenSCAP implementation.

SCAP Compliance Checker (SCC)

This scanner was developed by the Naval Information Warfare Center Atlantic (NIWC Atlantic) [19]. SCC performs automatic security configuration checks based on the contents of the SCAP that comes with the scanner. The supplied SCAP content is an implementation of the Security Technical Implementation Guides (STIG) policy. SCC users can install different SCAP content to perform compliance checks according to their policy.

---

[1]https://www.open-scap.org/security-policies/scap-security-guide/
[2]https://csrc.nist.gov/projects/scap-validation-program/validated-products-and-modules

Qualys SCAP Auditor

Qualys SCAP Auditor [2] is a cloud-based solution for SCAP compliance. This scanner is a subscription software and it is delivered as a Qualys Cloud Platform solution. SCAP content is provided for this scanner to determine if the target system meets the United States Government Configuration Baseline (USGCB) requirements. Users can also install different kinds of SCAP content to perform compliance checks according to their policies.

Chapter 4

Reporting Capabilities of SCAP-compatible Tools

Almost every SCAP scanner is equipped with a reporting format specified by the SCAP standard, such as ARF. However, these reports are difficult to read by humans. That is why each scanner implements its own way of reporting in a user-friendly manner.

4.1 SCAP Compliance Checker (SCC)

SCC provides standardized XML files for reporting, such as ARF, XCCDF, and OVAL results. SCC provides text reports containing plain text information about the scan performed. See Figure 4.1 for an example text report.

```
SRG-OS-000073-GPOS-00041
  RHEL 8 must encrypt all stored passwords with a FIPS 140-2 approved cryptographic hashing algorithm. - Pass
SRG-OS-000073-GPOS-00041
  RHEL 8 must employ FIPS 140-2 approved cryptographic hashing algorithms for all stored passwords. - Fail
SRG-OS-000073-GPOS-00041
  The RHEL 8 shadow password suite must be configured to use a sufficient number of hashing rounds. - Pass
SRG-OS-000080-GPOS-00048
  RHEL 8 operating systems booted with United Extensible Firmware Interface (UEFI) must require authentication upon booting
into single-user mode and maintenance. - Fail
SRG-OS-000080-GPOS-00048
  RHEL 8 operating systems booted with a BIOS must require authentication upon booting into single-user and maintenance
modes. - Pass
SRG-OS-000080-GPOS-00048
  RHEL 8 operating systems must require authentication upon booting into rescue mode. - Pass
SRG-OS-000120-GPOS-00061
  The RHEL 8 pam_unix.so module must be configured in the password-auth file to use a FIPS 140-2 approved cryptographic
hashing algorithm for system authentication. - Pass
SRG-OS-000120-GPOS-00061
```

Figure 4.1: Example of a text report generated by SCC scanner.

SCC also provides HTML reports that are generated from XML files. There are several versions of such reports with details of the rules being evaluated. This detail includes OVAL Tests and a lightweight overview of the relationships between OVAL Tests in the OVAL Definition. Another type of report is the summary type, which contains only rule names and results. Users can get both reports in filtered versions that contain only failed rules or all tested rules. See Figure 4.2 for an example of a rule detail.

RHEL 8 operating systems booted with United Extensible Firmware Interface (UEFI) must require authentication upon booting into single-user mode and maintenance.

Rule ID:	xccdf_mil.disa.stig_rule_SV-230234r743922_rule
Result:	Fail
Version:	RHEL-08-010140
Identities:	CCI-000213 (NIST SP 800-53: AC-3; NIST SP 800-53A: AC-3.1; NIST SP 800-53 Rev 4: AC-3; NIST SP 800-53 Rev 5: AC-3)
Description:	If the system does not require valid authentication before it boots into single-user or maintenance mode, anyone who invokes single-user or maintenance mode is granted privileged access to all files on the system. GRUB 2 is the default boot loader for RHEL 8 and is designed to require a password to boot into single-user mode or make modifications to the boot menu. false
Fix Text:	Configure the system to require a grub bootloader password for the grub superusers account with the grub2-setpassword command, which creates/overwrites the /boot/efi/EFI/redhat/user.cfg file. Generate an encrypted grub2 password for the grub superusers account with the following command: $ sudo grub2-setpassword Enter password: Confirm password:
Severity:	high
Weight:	10.0
Reference:	Title: DPMS Target Red Hat Enterprise Linux 8 Publisher: DISA Type: DPMS Target Subject: Red Hat Enterprise Linux 8 Identifier: 2921

Definitions:	Definition ID: oval:mil.disa.stig.rhel8:def:106
	Result: false
	Title: RHEL-08-010140 - RHEL 8 operating systems booted with United Extensible Firmware Interface (UEFI) implemented must require authentication upon booting into single-user mode and maintenance.
	Description: If the system does not require valid authentication before it boots into single-user or maintenance mode, anyone who invokes single-user or maintenance mode is granted privileged access to all files on the system. GRUB 2 is the default boot loader for RHEL 8 and is designed to require a password to boot into single-user mode or make modifications to the boot menu.
	Class: compliance
	Tests: • false (One or more child checks must be true.)
	    ○ false (All child checks must be true.)
	        ■ true (/boot/efi/EFI/redhat/grub.cfg:superusers exists and has a name.)
	        ■ false (/boot/efi/EFI/redhat/user.cfg:GRUB2_PASSWORD exists and has a PBKDF2/SHA512 password assigned.)
	    ○ false (/boot/efi/EFI/redhat/grub.cfg exists.) (negated)

Tests:	Test ID: oval:mil.disa.stig.rhel8:tst:10600 (textfilecontent54_test) Result: true Title: /boot/efi/EFI/redhat/grub.cfg:superusers exists and has a name. Check Existence: All collected items must exist. Check: All collected items must match the given state(s). Object ID: oval:mil.disa.stig.rhel8:obj:10600 (textfilecontent54_object) Object Requirements: • filepath must be equal to '/boot/efi/EFI/redhat/grub.cfg' • pattern must match the pattern '^\s*set\s+superusers\s*=\s*"(\w+)"\s*$' • instance must be greater than or equal to '1'

Figure 4.2: Example of HTML report with detail of rule generated by SCC scanner.

4.2 Qualys SCAP Auditor

According to the Qualys SCAP Auditor documentation – Qualys SCAP Auditor provides several types of [2] reports. The SCAP Scorecard Report provides a summary of the current status of SCAP compliance. See Figure 4.3 for an example.

Figure 4.3: Example of SCAP Scorecard Report from documentation of Qualys SCAP Auditor. Taken from [2].

Users can also create SCAP policy reports that can be generated in XML or CSV formats. The content of SCAP Policy reports is based on the XCCDF report. Using the API, users can download the full results as an ARF report. Qualys SCAP Auditor provides two types of interactive SCAP reports. The rule pass/fail report identifies the SCAP compliance status for a specific rule. See Figure 4.4 for an example of an interactive Pass/Fail report.

Figure 4.4: Example of the Rule Pass/Fail Report from documentation of Qualys SCAP Auditor. Taken from [2].

And the Individual Host Report (figure 4.5) identifies the SCAP compliance status for a particular host.

Figure 4.5: Example of the Individual Host Report from documentation of Qualys SCAP Auditor. Taken from [2].

4.3 OpenSCAP Scanner

The `oscap` scanner provides several types of reports that are designed to be inspected by users. Regular scanner output is a plain text, that contains only information about the rules being tested and their results. See figure 4.6.

```
--- Starting Evaluation ---

Title    Prefer to use a 64-bit Operating System when supported
Rule     xccdf_org.ssgproject.content_rule_prefer_64bit_os
Ident    CCE-90839-2
Result   pass


Title    Install AIDE
Rule     xccdf_org.ssgproject.content_rule_package_aide_installed
Ident    CCE-90843-4
Result   fail
```

Figure 4.6: Example of the command line output of oscap.

XML-based Reports

Also, the scanner generates a report, which is a standard ARF report that contains all the information about system evaluation. Usually, this report is very large, so the user has an option to select a subset of result types, such as XCCDF or OVAL results, and generate a smaller XML report to save space. These XML reports are very complex and not human-friendly. See Listing 4.1 with a small section of an ARF Report.

```xml
<arf:reports>
<arf:report id="xccdf1">
 <arf:content>
  <TestResult
    xmlns="http://checklists.nist.gov/xccdf/1.2"
    id="xccdf_org.open-scap_testresult_xccdf_org.profile_cis_workstation_l1"
    start-time="2023-03-16T13:54:43+01:00" end-time="2023-03-16T13:55:03+01:00"
    version="0.1.66" test-system="cpe:/a:redhat:openscap:1.3.6">
   <benchmark
    href="#scap_org.open-scap_comp_ssg-rhel9-xccdf.xml"
    id="xccdf_org.ssgproject.content_benchmark_RHEL-9"/>
   <title>OSCAP Scan Result</title>
   <identity authenticated="false" privileged="false">jrodak</identity>
   <profile
    idref="xccdf_org.ssgproject.content_profile_cis_workstation_l1"/>
   <target>rhel90-2</target>
   <target-address>10.0.2.15</target-address>
   <platform idref="#grub2"/>
   <set-value idref="xccdf_org.ssgproject.value_var_ssh_client_rekey_limit_size">
      512M
   </set-value>
   ...
```

Listing 4.1: Example of small part of the ARF report.

HTML Reports

The ARF XML report can be converted to an HTML representation. This version of the report contains limited details of the scan. And XSLT transformation used to produce takes a lot of processing time. The HTML report contains information about the rule and the OVAL Test result, but does not explain tests relationship in the OVAL Definition that is associated with the rule. Also, it does not contain information about the applicability of the rule to the target system.

This HTML report, produced by the OpenSCAP scanner, is more sophisticated than the others, but it still does not contain all the useful information for content developers and OpenSCAP users. See figure 4.7 with an example of the rule details in the report.

Ensure PAM Enforces Password Requirements - Minimum Digit Characters   ✕

Rule ID	xccdf_org.ssgproject.content_rule_accounts_password_pam_dcredit
Result	**fail**
Multi-check rule	no
OVAL Definition ID	oval:ssg-accounts_password_pam_dcredit:def:1
Time	2022-12-28T16:17:46+01:00
Severity	medium
Identifiers and References	**Identifiers:** CCE-83566-0 **References:** BP28(R18), 1, 12, 15, 16, 5, DSS05.04, DSS05.05, DSS05.07, DSS05.10, DSS06.03, DSS06.10, CCI-000194, 4.3.3.2.2, 4.3.3.5.1, 4.3.3.5.2, 4.3.3.6.1, 4.3.3.6.2, 4.3.3.6.3, 4.3.3.6.4, 4.3.3.6.5, 4.3.3.6.6, 4.3.3.6.7, 4.3.3.6.8, 4.3.3.6.9, 4.3.3.7.2, 4.3.3.7.4, SR 1.1, SR 1.10, SR 1.2, SR 1.3, SR 1.4, SR 1.5, SR 1.7, SR 1.8, SR 1.9, SR 2.1, 0421, 0422, 0431, 0974, 1173, 1401, 1504, 1505, 1546, 1557, 1558, 1559, 1560, 1561, A.18.1.4, A.7.1.1, A.9.2.1, A.9.2.2, A.9.2.3, A.9.2.4, A.9.2.6, A.9.3.1, A.9.4.2, A.9.4.3, IA-5(c), IA-5(1)(a), CM-6(a), IA-5(4), PR.AC-1, PR.AC-6, PR.AC-7, FMT_MOF_EXT.1, Req-8.2.3, SRG-OS-000071-GPOS-00039, SRG-OS-000071-VMM-000380
Description	The pam_pwquality module's `dcredit` parameter controls requirements for usage of digits in a password. When set to a negative number, any password will be required to contain that many digits. When set to a positive number, pam_pwquality will grant +1 additional length credit for each digit. Modify the `dcredit` setting in `/etc/security/pwquality.conf` to require the use of a digit in passwords.
Rationale	Use of a complex password helps to increase the time and resources required to compromise the password. Password complexity, or strength, is a measure of the effectiveness of a password in resisting attempts at guessing and brute-force attacks. Password complexity is one factor of several that determines how long it takes to crack a password. The more complex the password, the greater the number of possible combinations that need to be tested before the password is compromised. Requiring digits makes password guessing attacks more difficult by ensuring a larger search space.

Remediation Shell script ⌄

Remediation Ansible snippet ⌄

OVAL test results details

check the configuration of /etc/pam.d/system-auth | **oval:ssg-test_password_pam_pwquality:tst:1** | **true**

Following items have been found on the system:

Path	Content
/etc/pam.d/system-auth	password requisite pam_pwquality.so try_first_pass local_users_only retry=3 authtok_type=

check the configuration of /etc/security/pwquality.conf
oval:ssg-test_password_pam_pwquality_dcredit:tst:1 | **false**

No items have been found conforming to the following objects:

Object **oval:ssg-obj_password_pam_pwquality_dcredit:obj:1** of type **textfilecontent54_object**

Filepath	Pattern	Instance
/etc/security/pwquality.conf	^\s*dcredit[\s]*=[\s]*(-?\d+)(?:[\s]|$)	1

Figure 4.7: Example of HTML report with rule detail.

Chapter 5

Proposed Report Improvements

All reports, generated by SCAP scanners, have several problems. Some reports are more useful than others, but, in general, the content of the reports is a summarized result of the scan performed. In some use cases, users only need to know the scan scores and rule results with descriptions. However, when developing a security profile or hardening system according to a guideline, users want to know what was tested and how, or whether the rule is applicable.

If the target system has been remediated, users need information on how the remediation was performed. User has to look for this information in a huge XML file, which is very confusing, as it was designed primarily for machines.

5.1 OVAL Results

The main component that determines what is being tested and how is the OVAL. The OVAL Definitions establish the relationship between different tests. This relationship affects the results of the rules. For example, some tests or operator nodes can be negated, which can change the result of a branch in an OVAL Definition.

An OVAL Definition can be described by a tree structure, which I'll call an OVAL tree. This tree structure represents the relationships between tests. OVAL Tests define which objects are collected from the target system to be evaluated against reference states. See Figure 5.1 for an example graphical representation of an OVAL tree.

Figure 5.1: Visualization of OVAL.

5.2 Applicability of a Rule

The applicability of a rule is very important information for the user. If the rule is not applicable, the OVAL Check is not performed and the result of the OVAL Check is not evaluated.

The applicability of a rule is determined by the CPE platform. [4] "*The CPE platform of a rule or a group can contain a boolean expression, which describes the relationship of a set of individual CPE platforms, which would later be converted by the build system into the CPE AL definition.*"

This conversion to CPE Applicability Language definitions is under development. Earlier versions of the content and scanner exclusively exploited the CPE Dictionary mechanism.

The CPE Applicability Language directly refers to OVAL Definitions that check if a rule makes sense for current environment. The relationship between these applicability checks can be represented as a tree structure. See figure 5.2 for an example visualization of the CPE Applicability Language.
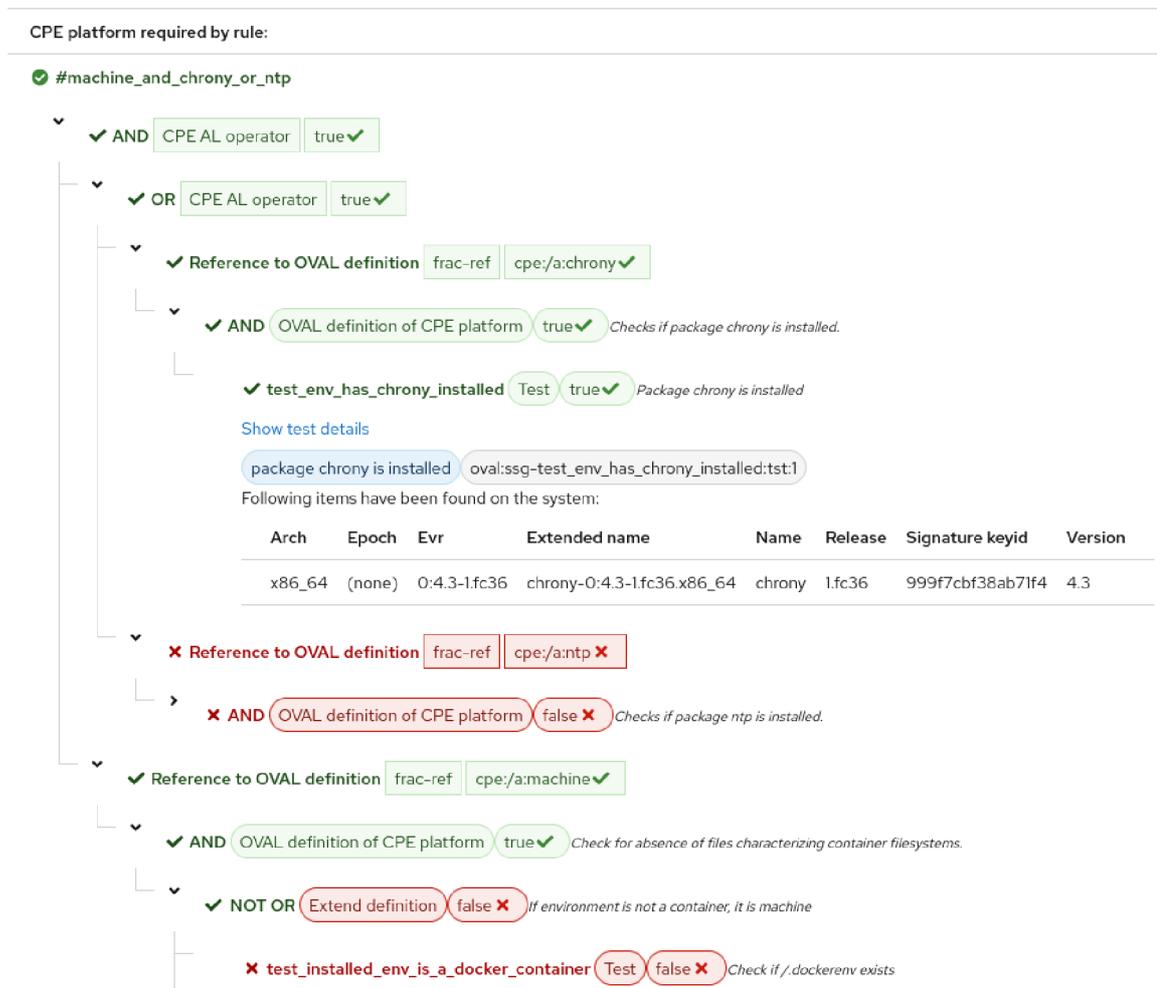

Figure 5.2: Visualization of CPE Applicability Language.

5.3 Post-Remediation Rule Result

This improvement introduces two new results types, that are available when the target system has been remediated. The result `fix unsuccessful` means that the remediation was made, but the subsequent OVAL Check was unsuccessful. The result `fix failed` means that the fix failed in the execution phase. See figure 5.3 of a rule with a `fix failed` result.

Figure 5.3: Rule with the result of the fix failed.

Chapter 6

Implementation

This thesis deals with the development of a tool for generating interactive HTML reports. This tool is implemented mostly in Python, but HTML with JavaScript are used to make the generated reports interactive.

A clean code approach is used to implement the tool along with object-oriented programming. The tool is divided into several parts, which are:

- Command line interface,

- SCAP results parser,

  - Data structures of SCAP results,

- Report generators.

These parts are implemented as sub-packages of the collection package `openscap-report`. It is available as a stand-alone Python package as well as an RPM package. This approach allows developers to reuse the code and integrate parts of the tool into other projects. See the 6.1 package structure diagram.
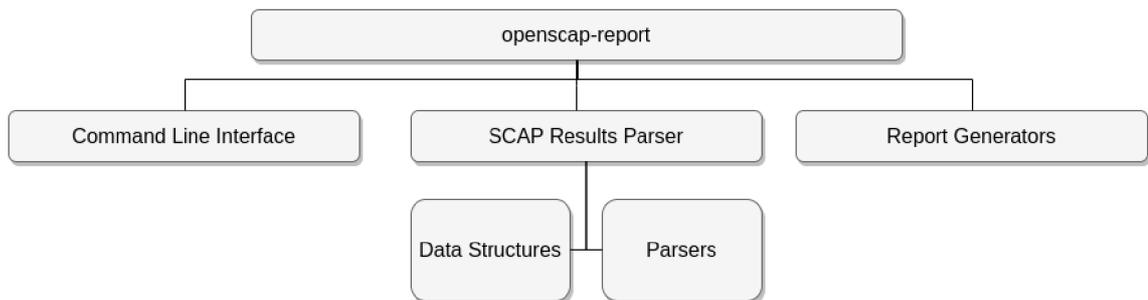

Figure 6.1: Structure of openscap-report package.

6.1 Command Line Interface

The command line interface (CLI) is the first entry point that users interacts with. This entry point is called `oscap-report`. The implementation of the command line interface is

in the file `cli.py`. In this file, there is a class called `CommandLineAPI` that is used as the application programming interface (API). This API provides basic methods for loading input data, storing output data, generating reports, etc. The `CommandLineAPI` class internally parses arguments using the `argparse` library. The `argparse` library is part of the Python Core Libraries. API methods are affected by the arguments being parsed.

The user can view information about the use of `oscap-report` by using `oscap-report --help` or by using man page[1] with `man oscap-report` (as part of the `openscap-report` RPM package).

The tool expects the input file to be an ARF file. The `FILE` positional argument allows the user to specify the path to the file. If no file path is specified, the tool expects a file on standard input. Users can specify the path where to save the generated reports using the `--output OUTPUT` parameter. If no output path is specified, the standard output is used.

For debugging Python code, the user can set the logging level using the parameter the `--log-level LEVEL`. The logs are output to standard error output using the parameter the `--log-file LOG_FILE`, the file to which the logs should be saved can be specified.

To debug the generated HTML report or the generation process, user can apply additional debug settings using the `--debug DEBUG_FLAGS` option. Currently available debug flags would modify HTML reports for report development and testing, but if desired, the debug flags can be extended to include flags that affect input processing and generation. For example, for some specific logging formats, generating only OVAL visualizations, etc. Available debug flags include disabling HTML report minification or enabling online CSS resources. To propagate debug flags to report generators, the `DebugSettings` data class is used.

The tool supports several output types, which the user can specify using the parameter the `--format FORMAT` option. Default output format is HTML. That is, the tool will generate an HTML report. For backward compatibility, the `OLD-STYLE-HTML` format is available, which generates an HTML report that resembles the report created by OpenSCAP using XSLT transformations (original XSLT files from OpenSCAP are used for this process).

On top of that, as an experimental feature, the tool can generate reports in `JSON` format.

6.2   SCAP Results Parser

This sub-package is responsible for analyzing the ARF file and generating the object-oriented data model of the report. The `lxml` library is used to process XML files. The `lxml` [13] library provides an object-oriented approach to processing XML files. The library is also quite good performance-wise, having critical parts of processing routines implemented in C.

The `scap_result_parser` sub-package consists of two sub-packages for better clarity and code orientation. One is the `parsers` sub-package, which contains parser classes for processing specific parts of ARF files, such as:

- Information about the performed scan,

- Used profile,

- Identifiers of rules and groups that are used in the profile,

---

[1]Online man page: https://openscap-report.readthedocs.io/en/latest/oscap-report.1.html

- Definition of individual rules, groups, rule checks in OVAL Language and rule applicability checks,

- Results of rules, rule checks in OVAL Language and rule applicability checks.

The second sub-package named `data_structures` is used to represent and store processed information. This sub-package contains class definitions that represent the object-oriented data model that is used to generate the HTML report or a variant of the `JSON` report.

The Data Model

The implementation of the data model is in the `data_structures` sub-package. Classes in this sub-package use the `@dataclass` [17] decorator, which automatically adds special methods to classes like `__init__()`. The entire data model is divided into several classes, that represent different information structures from the ARF file. These classes together define the main structure called `Report`. A report can be represented as an HTML report, `JSON` structure, or a Python dictionary.

Classes that constitute the report have many associations between them. This diagram 6.2 represents the relationship within `data_structures` sub-package. Additionally, some classes may contain multiple instances of referred classes. This kind of relationship is marked with an „X" symbol on the scheme.
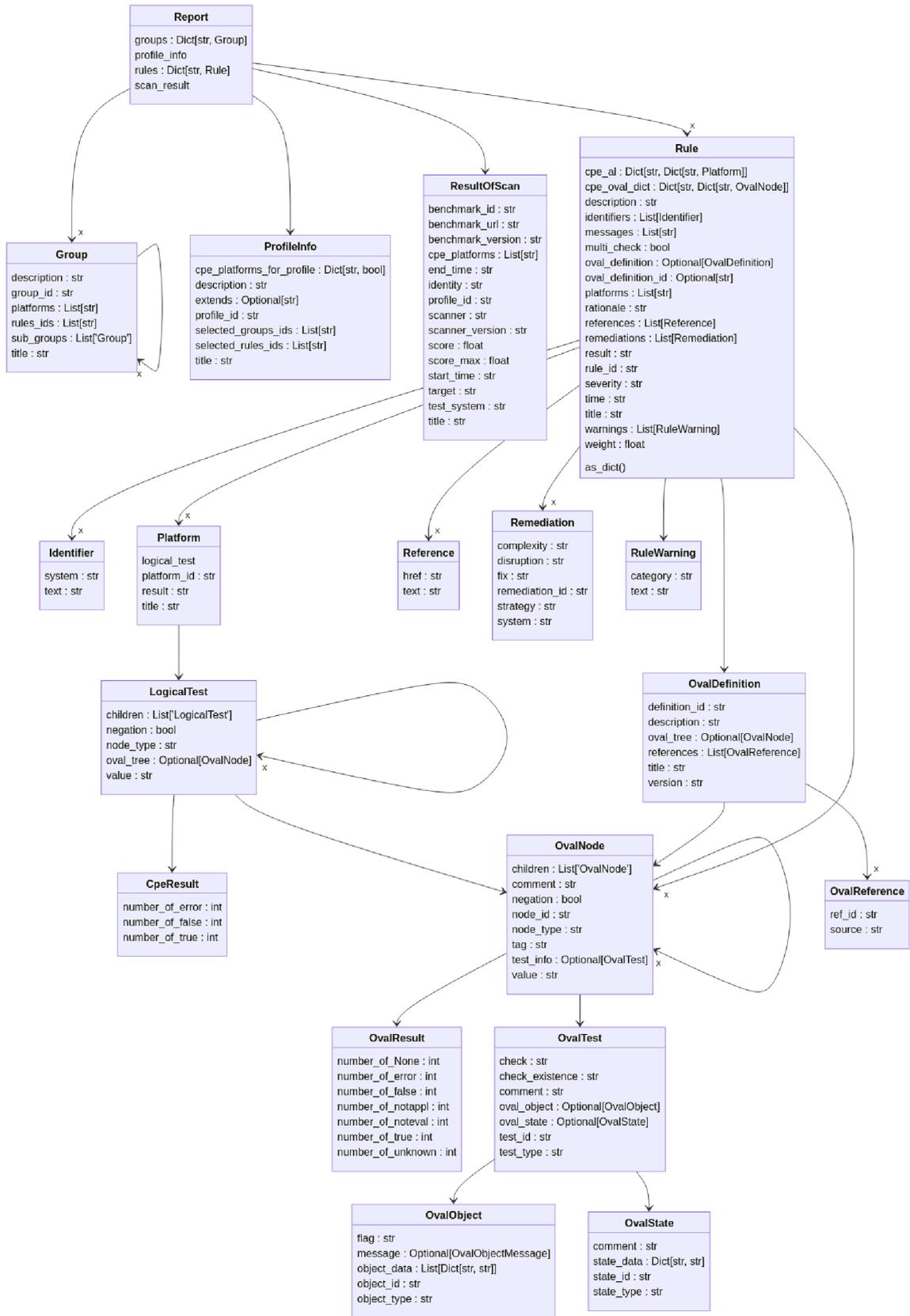
Figure 6.2: Diagram of the Report data structure

The `Report` class contains instances of the `ProfileInfo` and `ResultOfScan` classes, these instances contain information about selected profile and scan results.

Information about the selected profile is stored in an instance of `ProfileInfo`, which holds profile elements such as the description, name, profile identifier, identifiers of selected rules and groups, and the CPE platform, with information about whether the platform is applicable to the target system.

Information about the scan is stored in an instance of the `ResultOfScan` class, which contains information about the scan that was performed, such as the name of the target machine, the name and version of the scanner used, the name of the user who performed the scan, start and end time, the resulting score, and so on.

The `Report` class contains a dictionary of instances of the `Group` class that are indexed by the group identifier in the profile. An instance of the `Group` class contains information such as a group description, an identifier, a list of CPE platform identifiers, a list of rule identifiers that are listed in the group, and a list of subgroups that are instances of the `Group` class.

The most important part of the `Report` class is the dictionary, which collects instances of the `Rules` class, that is indexed by rule identifiers. This dictionary contains all the rules of the Data Stream. An instance of the Rule class contains information about the rule, such as description, name, rule identifier, OVAL Definition identifier, list of platform CPE identifiers, security policy identifier and reference, rule result, list of remedies, list of warnings, dictionaries with the applicability definitions, instances of the OvalDefinition class, etc.

The dictionaries, that determine whether a rule is applicable to the target system and can be part of an instance of the `Rule` class, are very similar. These two dictionaries are based on the CPE Applicability Language and the CPE Dictionaries. The attribute named `cpe_al` contains an instance of the Platform class that contains the CPE Applicability Language tree. The `cpe_oval_dict` attribute contains a dictionary with instances of the `OvalNode` class, which is a class of the OVAL Definition criterion structure, representing a CPE Dictionary entity. Both dictionaries are ordered according to the position of the CPE that defines rule applicability.

An instance of the `OvalDefinition` class in the `Rule` class contains information about the OVAL Definition, such as description, references, name, version, and criteria, represented by the `OvalNode` class, a recursive tree structure. An instance of the `OvalNode` class contains information about criteria such as identifier, node type, value, and commentaries. For the leaf nodes, there is also information about the OVAL Test. The OVAL Test is represented by an instance of the `OvalTest` class. The `OvalNode` class uses an instance of a special class, that can evaluate operators according to the OVAL specification.

An instance of `OvalTest` in the `OvalNode` contains information about the OVAL Test, such as the test identifier, the check and check attributes, metadata, an instance of `OvalObject` that represents the OVAL Object, and an instance of `OvalState` that represents the OVAL State.

The `OvalObject` class uses wrapping to extend the OVAL Object abstraction. The `OvalObject` class has a `object_data` attribute that stores a list containing dictionaries of attributes and values. These dictionaries are stored in a list, functioning as a shared database, because they are used to populate multiple instances of the collected objects. In this case, common object-oriented approach has been modified to make it easier to use different types of OVAL Objects and to store multiple instances of OVAL Objects.

The `OvalState` class uses similar wrapper to extend the OVAL State abstraction. The `OvalState` class has a `state_data` attribute that stores a dictionary of attributes and values, representing the state of the OVAL Object.

The CPE Applicability Language is represented by an instance of the `Platform` class. The `Platform` class is used as a wrapper for the `LogicalTest` class, a class that represents the logical test from the CPE Applicability Language. It defines a relationship between other instances of `LogicalTest`. For the leaf logical test node, it also refers to OVAL criteria. This reference is handled by an instance of the `OvalNode` class. The `LogicalTest` class uses an instance of a special class, used to evaluate the relationship between applicability checks.

ARF File Decomposition

The `SCAPResultParser` class handles the XML ARF file as a string. This string is passed as an argument to the `SCAPResultParser` instance. When the object is initialized, validation is performed according to the XSD schema provided by NIST[2]. Then the parser is ready to create an object of class `Report`. If the input file is not valid, the ARF parser will warn about this situation. The parser can process files in XCCDF result format, but the result is an incomplete report.

The `SCAPResultParser` has a method called `parse_report` that returns an instance of the `Report` class from the `data_classes` sub-package. This method uses additional instances of specialized parsers from the Parsers sub-package to build a complete report object.

The sub-package named `Parsers` is extended with several specialized parsers to ease code navigation, but introduces many dependencies between them. This is key element of the performance improvement. The slowness of the process was caused by references to various elements in the XML, that are deep in the XML file hierarchy. For example, in an ARF report, a variable can replace certain text in descriptions or define a value that is being evaluated. This is caused by the reuse of rules for several profiles with different requirements. For example, password length or idle timeout. There are many other non-trivial cases where different elements are referenced in ARF reports. For example, rule elements referencing the OVAL Definition, the CPE Dictionary, or the CPE Applicability Language definition are checked. All these elements can be encountered in different parts of the ARF file, including the results.

This problem of searching and cross-referencing in a large XML file has been solved by bottom-up processing and limiting the parser scope dive. The parser processes a small portion of the parsed XML and limits the depth of the search. The bottom-up processing means that the parsers use another parser to extract the information to be parsed. This information is returned as dictionaries, that are indexed by identifiers. The values in these dictionaries can be XML elements or object instances from the `data_structures` package. For example, a parser that processes OVAL results prepares a dictionary that is indexed by the identifiers of the OVAL Definitions and whose values are OVAL criteria, represented by the `OvalNode` class. The extension definitions are represented by the class `OvalNode`, which contains the OVAL Definition identifier that is resolved later. If the `OvalNode` reference to OVAL Test is used, the parser that contains preprocessed parts of the OVAL Tests is used.

---

[2]https://csrc.nist.gov/Projects/Security-Content-Automation-Protocol/Specifications/arf

6.3 Report Generator

The `report_generator` sub-package is responsible for generating reports in different formats. The `report_generator` sub-package contains classes that are used to generate specific report formats. Report generators use an instance of the `Report` class from the `scap_report_parser` sub-package, created by an instance of the `SCAPReportParser` class. In the case of JSON, the report can be transformed into a dictionary that matches the JSON format. In the case of the default JSON version, filtering and dictionary transformation is performed. If the `JSON-EVERYTHING` format is selected, the filtering and transformation are skipped.

A template with macros from the `Jinja2` library is used to generate an HTML report. These macros add the data from the input report to the output report template. However, not all information is rendered as plain text. Some information is inserted as JSON. Also, JavaScript code, fonts, and CSS files are incorporated in the final report to make it self-sufficient.

The reason for embedding JSON in the HTML element instead of generating proper HTML is to minimize the size of the generated report and to save time in generating complex HTML structures. JSON in the HTML element is used to represent the OVAL Language and the CPE Applicability Language. The graphical representation of the OVAL Language and the CPE Applicability Language as HTML elements is generated by the browser using asynchronous JavaScript code.

Some browsers have trouble displaying huge HTML reports with thousands of HTML elements because the browser renders the entire HTML document. That is why displaying a report containing hundreds of rules can cause performance problems. To optimize the performance lazy loading of rules in the report has been implemented to reduce the rendering time when the user scrolls through the report.

The HTML report provides a full-text search mechanism for rule names and rule identifiers. The list of rules can be also filtered by rule results and by rule severity.

6.4 Report Structure

The final report consists of several parts. For the main report style, the Patternfly[3] framework is used. An example of the generated report is contained on the storage medium as a file named `report_example.html`. The first part displays the description and name of the selected profile. See figure 6.3 for an example of the first section.
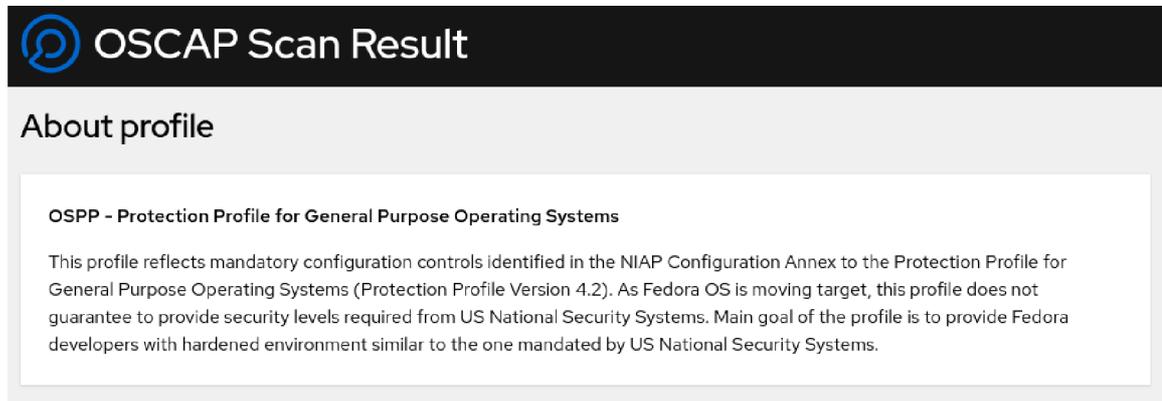

Figure 6.3: Example of the first section in report

The second part focuses on compliance and scoring. This section contains overall statistics on performed tests, such as the number of passing and failing tests and the rules for scoring and summarizing the tests. See the example figure 6.4 with the Compliance and Scoring section.
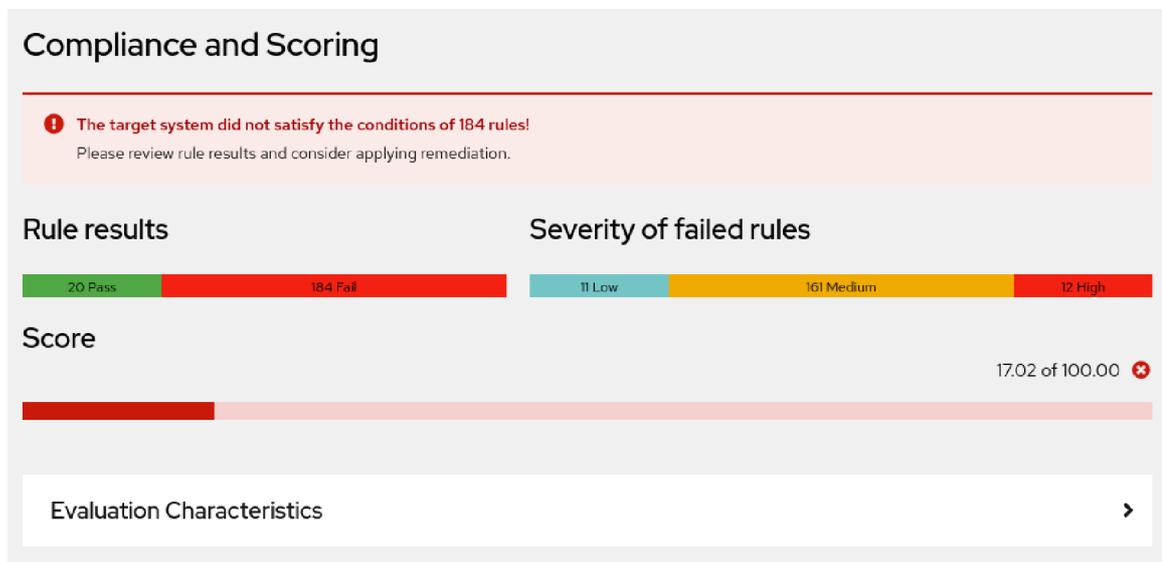
Figure 6.4: Example of the compliance and scoring section in report

The section on compliance and scoring includes a subsection on the Evaluation Characteristics of the scan. This subsection contains general information about the scan, such

---

as the name and version of the scanner, the name of the user who performed the scan, etc. See figure 6.5 for an example of an Evaluation Characteristic in the report.

Figure 6.5: Example of the Evaluation Characteristics section in report

Below the Compliance and Scoring section is the Rule Overview section, which contains a list of rules and a search bar with filtering. See figure 6.6 of the filterable search bar.
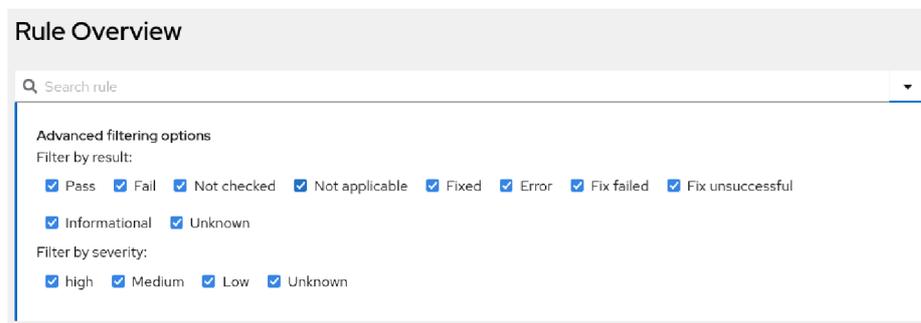
Figure 6.6: The search bar with filtering options in report

Each rule is interactively expandable with the rule details: result, check time, severity, description, rationale, remediation, warning, visualization of the rule check and applicability check, etc. See figure 6.7 for an example of a rule detail.

Figure 6.7: Detail of rule in report

In the rule details there is a section with the rule remediation code. If user is interested in manual remediation of the system, they can click on the correction and get the correction code that can be used to modify the system to conform to the rule. See figure 6.8 for an example of remediation.

Figure 6.8: The remediation of rule in report

Figure 6.9: The visualization of the OVAL Definition with criteria

In the rule details, users can see the visualization of the rule checks and the applicability of the rule, which is defined in OVAL and the CPE Applicability Language. The OVAL

Language visualization is demonstrated using an oval definition with criteria. See Figure 6.9 for a visualization of the OVAL Definition with criteria.

If the criterion node is a leaf node, it is a test node that can populate the OVAL Object and the OVAL State. The OVAL Test optionally contains two attributes that define how the OVAL Test is evaluated. These attributes are called check and `check_existence`. The check attribute specifies how many collected objects must meet the requirements specified in the OVAL State for the OVAL Test to evaluate to `true`. The `check_existence` attribute specifies how many objects defined by the OVAL Object must exist for the OVAL Test to evaluate to `true`. See figure 6.10 as an example of an OVAL Test that populates the OVAL Object in the OVAL State.
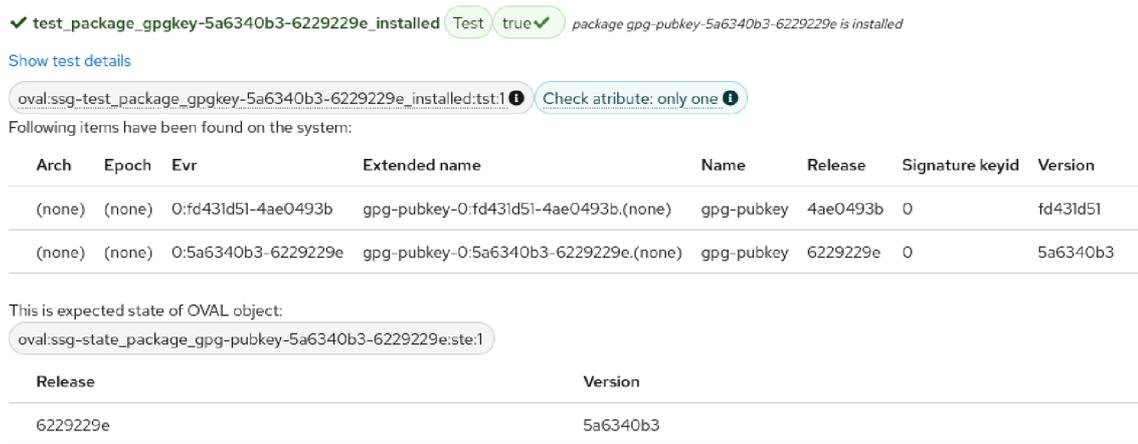
Figure 6.10: Detail of the OVAL Test

The visualization of the applicability rule is very similar to the visualization of the OVAL Language. When using the CPE Dictionary-based approach, the criteria that are related to the CPE Dictionary are displayed.

When using the CPE Applicability Language, the node hierarchy that is defined by the CPE Applicability Language is displayed. These nodes have square labels to distinguish them from rule's own OVAL Criteria. See figure 6.11 for a visualization of the CPE Applicability Language.
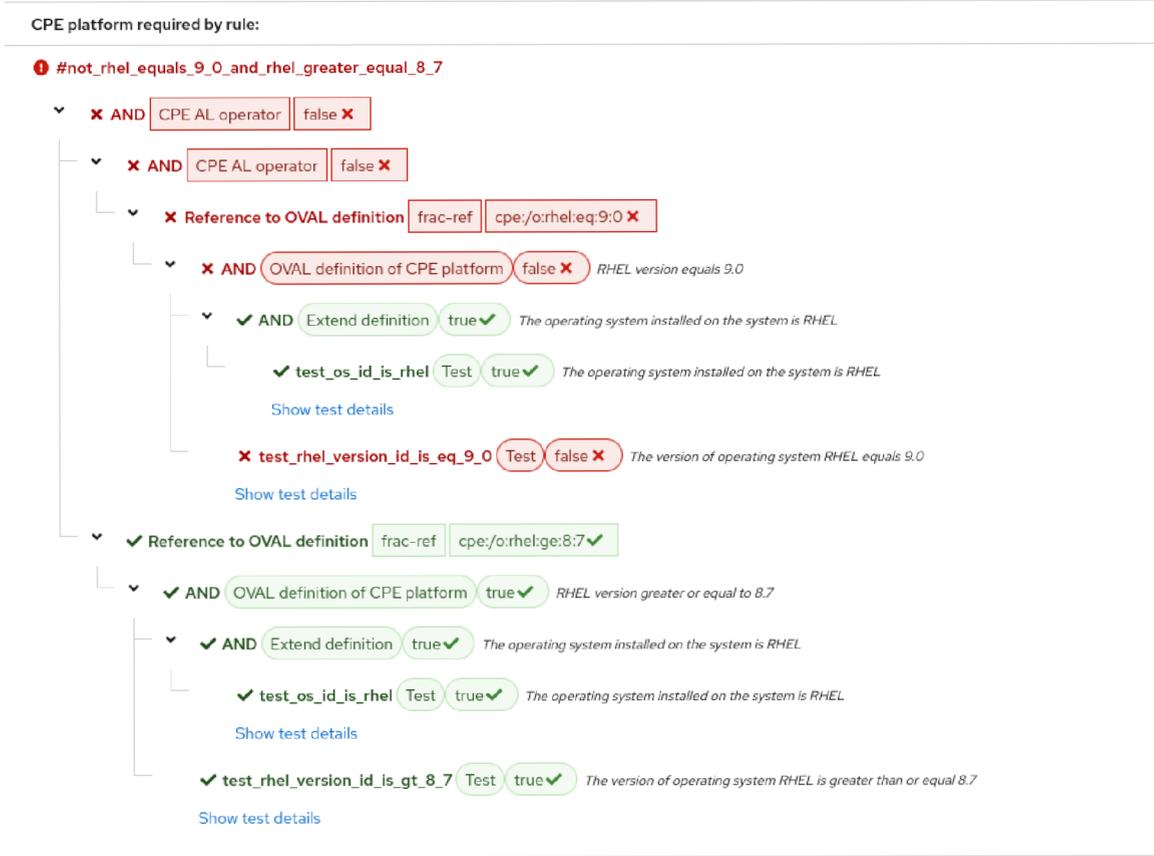
Figure 6.11: Visualization of CPE Applicability Language

6.5 Benchmark

The `pref` [3] tool was used to benchmark the report generation process. This tool serves as an interface to monitor the performance of the Linux kernel. It is used to evaluate the execution time, CPU usage, and other metrics of the inspected application. The `oscap-report` tool will be compared to the `oscap` tool. The versions of `openscap-report` `0.2.2` and `openscap-scanner` `1.3.7` are being compared. Both commands were executed ten times, and the resulting time was calculated as the average of these runs [20].

The following command was used for the benchmarking of `oscap-report`:

```
$ perf stat -e cpu-clock -r 10 oscap-report arf.xml > report.html
```

The results:

```
Performance counter stats for 'oscap-report arf.xml' (10 runs):

   19 969,05 msec cpu-clock:u # 9,995 CPUs utilized ( +- 9,62% )


       1,9979 +- 0,0113 seconds time elapsed ( +- 0,57% )
```

Listing 6.1: Output of oscap-report benchmark.

The following command was used for the benchmarking of `oscap`:

```
$ perf stat -e cpu-clock -r 10 oscap xccdf generate \
  report arf.xml > report.html
```

The results:

```
Performance counter stats for 'oscap xccdf generate report arf.xml' (10 runs):

   40 250,06 msec cpu-clock:u # 9,998 CPUs utilized ( +- 9,64% )


       4,0258 +- 0,0323 seconds time elapsed ( +- 0,80% )
```

Listing 6.2: Output of oscap benchmark.

According to the benchmark, `oscap-report` generates reports in half of the time that is needed for `oscap`.

Chapter 7

The openscap-report Package

The upstream repository of the `openscap-report` project is available at GitHub.com[1], where the development process is ongoing.

Two files are required to create an RPM package. The first file is a specification file that describes the process of installing the application as a cookbook. There are special macros and guidelines [12] for Python that simplify the process of creating an RPM package and using Python's packaging tools. The spec file used in the Fedora and EPEL 9 RPM packages is included on storage media under the name `openscap-report.spec`.

However, there is one catch with the `openscap-report` package dependencies. The biggest issue, that required a resolution, was the Patternfly dependency. This problem was caused by offline CSS as a source for reports, generated on air-gaped systems. The Patternfly XStatic package set was not a good fit because these packages were abandoned (they do not receive updates and are not available on RHEL systems). Therefore, the `openscap-report` archive contains internally packaged Patternfly CSS files and the package requirements include only the fonts that are in use by Patternfly. Fonts are installed separately, from their own package, as a dependency according to the packaging guideline [11]. The fonts are not present in the PyPI Python package (available for the pip tool), but the user can install these fonts manually, or ignore their absence as it is not an important part of the `openscap-report`.

With a complete spec file, only the tool source archive is needed. Which can be generated using the Python build tool, used to build a Python package compatible with the PyPI repository.

The `setup.py` bootstrap script is required to create a Python package [14]. The script contains information about the package such as entry points, dependencies, versions, etc. For a more complex project, one also need to provide a `MANIFEST.in` file, that specifies all the extra files the Python package must deliver: templates, CSS files, an so on.

Packaging `openscap-report` into the RPM package and PyPI package (Python package) is facilitated by the release process. Thanks to Github's Actions[2] service, almost all tasks are performed in an automated fashion. The release is done using a bash script named `release.sh`. This script expects a version number as the parameter and performs tasks such as checking if the release is possible, retrieving the version from `setup.py` and pushing the changes with the version tag to the repository.

---

[1]https://github.com/OpenSCAP/openscap-report
[2]https://github.com/OpenSCAP/openscap-report/actions

After that, two independent Github actions are performed. The first commits the Python package to PyPI[3] and the second creates a release on GitHub[4].

The Github release is a trigger for the PackIt [15] service, which is used to build and test RPM packages in the upstream pull requests. In the course of the release process, it is used to create a pull request with the changes to the downstream repository. The downstream repository is used to release the RPM package for Fedora and Extra Packages for Enterprise Linux [10] (can be enabled on Red Hat Enterprise Linux 9). Finally, the build and release submission to the downstream repository actions are performed manually. This step concludes the RPM package release process.

The developer must become a Fedora contributor to request permission to release the package for Fedora Linux distribution. And all new packages must be reviewed and approved by other Fedora contributors to be released for Fedora Linux under Fedora Guidelines [11].

7.1 Building Packages

To build an RPM package, you must install the PackIt CLI tool. Instructions for installing the PackIt CLI tool are available at webpage[5]. Other RPM tools, such as `rpmbuild` or `mock` can be used to build, but PackIt simplifies the process of building an RPM package for your version of Fedora. To build an RPM package, you can run the following command in the project root:

```
$ packit build locally
```

The build process for PyPI repository and installation process using the pip tool are described in the manual. Running code from source files and installing using the RPM package manager are also described in the manual. The manual itself is built using `Sphinx`. The description of the build process for the manual is available in the appendix B.

7.2 Continuous Integration and Tests

A test suite was created to test the functionality of `openscap-report`. The way to run the test suite locally is described in the manual. The main test suite consists of several tools. The Tox [18] tool, which is a Python test tool for automating tests. This tool can run test suites with different versions of Python in virtual environments. The Tox configuration enables `openscap-report` to be tested in different Python environments without using containers or virtual machines.

Pytest [16] is the framework used for testing the package. Tests are divided into two categories: unit and integration tests. Unit tests test each part of the `openscap-report` independently. Integration tests test the `openscap-report` package as a whole.

The `openscap-report` code is also lint-checked to be aligned with PEP8[6], a set of requirements and guidelines to keep the code is healthy, clean as readable.

---

[3]https://PyPI.org/project/openscap-report/
[4]https://github.com/OpenSCAP/openscap-report/releases
[5]https://packit.dev/docs/cli/
[6]https://peps.Python.org/pep-0008/

Each pull request in the upstream repository triggers a continuous integration that runs a suite of tests and other testing services, such as scrutinizer-ci.com[7] and CodeQL[8], which scan the code to check if the pull request introduces security holes or vulnerabilities. There is also an RPM build test with changes to the pull request using the smoke test and a dependency check to ensure that the change does not break the release process with unexpected dependencies or broken code.

A weekly integration test is created to check that `openscap-report` keeps up with other components such as OpenSCAP Scanner and SCAP Security Guide. The weekly integration test uses the latest released version of `openscap-report` along with ARF reports generated from the latest released packages `openscap-scanner`, `scap-security-guide` and content that is created from the ComplianceAsCode repository development branch.

---

[7]`https://scrutinizer-ci.com/g/OpenSCAP/openscap-report/`
[8]`https://codeql.github.com/`

Chapter 8

User Testing

User testing is an important part of project development and is one of the main sources of user feedback for the application being developed. Following the user feedback, `openscap-report` can be improved to the satisfaction of the customers.

The `openscap-report` tool is developed for very narrow group of users such as security content developers, OpenSCAP scanner developers, and users who use OpenSCAP to verify compliance with their company's security policy. These users usually are system administrators or auditors. Hallway or shadow testing approaches are therefore not viable because these users are spread out across the globe. Therefore, I decided to reach out to users using a form where users would answer questions, try out `openscap-report`, and also would be able to leave extended feedback on `openscap-report`.

8.1   Methodology

The user testing form consists of two main parts. Each part focuses on a different part of the `openscap-report`. The first part focuses on using the command line interface and generating reports. The second part focuses on report controls and report contents. All files that were used for user testing are contained on the storage medium in a directory named `user_testing`.

Environment Setup

The user must install `openscap-report` before the testing process can begin. This section is excluded from the evaluation because the use of package managers such as DNF is not in the focus of user testing.

In this section, users are provided with installation instructions or RPM packages and the necessary files for user testing. The evaluation files needed for testing, such as security scan results in the form of ARF XML, can be generated by the user, but for convenience, these files are also provided.

Report Generation

This section focuses on the command line tool called `oscap-report`, which is provided by the `openscap-report` package. Users are introduced to the command line capabilities of the `oscap-report` tool and prompted to use it to generate an interactive HTML report

from an ARF results file. The user was asked to perform the following tasks and answer questions:

1. Task: Display help for the `oscap-report` tool via the man pages (`man oscap-report`) or with the command `oscap-report --help`.

   (a) Can you generate an HTML report with `oscap-report` after reading the man pages or help messages?

   (b) Do you have enough information that describes the functionality of the tool? If you don't, what would you like to be explained in more depth?

   (c) Do you have any other problems with the display or content of the man page or help message?

2. Task: Please generate an HTML report from the ARF results XML file using the `oscap-report` command. You can use the file that was provided in the setup section. Alternatively, you can use your own ARF report generated by the SCAP-compatible scanner.

   (a) Was the generation of the HTML report successful? Please, don't hesitate to create an issue for the project here[1] if you had problems with the tool (attach the ARF file you've used).

Report Contents

This section focuses on the content of the HTML report and its interactive capabilities. User would work with the HTML report generated in the previous section. Users had to complete the following tasks and answer the following questions:

1. Task: Please open the generated HTML report in your web browser.

   (a) Do you have any trouble opening the HTML report? If you do, please write them down.

2. Task: Please explore the contents of the HTML report.

   (a) Do you find the About Profile and Compliance and Scoring sections useful?

   (b) Missing any scoring or profile information? If yes, write what is missing.

   (c) Do you find the Evaluation Characteristics section useful?

   (d) Do you miss any information in the Evaluation Characteristics section? If yes, write down what is missing.

3. Task: Please try filtering rules by result or severity to find the rule you want or filter rules that have the word „account" in the title or id, which have failed and are of medium severity.

   (a) Was it easy to use the filter? (If you have a problem, click the arrow on the right side in the search bar)

   (b) Do you miss any filtering options? If yes, write down what is missing.

---

[1] https://github.com/OpenSCAP/openscap-report/issues

4. Task: Please see the rule detail. (Click on the title of a rule or the arrow on the left side next to the rule's title)

   (a) Do you miss any information about a rule? If yes, write down what is missing.

   (b) Do you miss any information in the rule's check visualization (OVAL Language)? If yes, write down what is missing.

   (c) Are you missing any information in the rule applicability check visualization? If yes, write what is missing.

   (d) Why did the rule
   `xccdf_org.ssgproject.content_rule_dconf_gnome_screensaver_idle_delay`
   fail? The report was provided via the link.

5. Conclusion

   (a) Do you have improvement ideas?

   (b) Did you find any bugs or problems?

   (c) Anything else?

8.2 Outcome

The feedback was positive in general, and I also received several suggestions for improving the report. I have had the opportunity to speak personally with some users and discuss their needs. I have received feedback from users not only from Red Hat but also from other companies or institutions such as Canonical, Fermilab, and SUSE.

Based on the responses, I will point out the main issues and suggestions for improvement. For any ideas or issues that were mentioned in the responses, I plan to create issues on GitHub. So that I'd follow up on them and could be able to improve `opnescap-report` in the future. I've divided the answers into several categories, such as Documentation, Report Informativeness, and Report Capabilities.

Users liked the new Report and asked if this Report will replace the old OpenSCAP scanner HTML Report. Users were surprised at the speed of generating the new HTML Report compared to the old HTML Report. Users are also interested in generating OVAL visualizations from the OVAL Results file or other report formats.

Documentation

According to the feedback, users are missing the examples of tool usage in the man page. One user suggested that only options without too much text should be shown in the help and that some flags not related to normal use should be moved to the man pages only. The flags for debugging was given as an example. Users also noted the lack of documentation on manual build and installation procedures for `openscap-report` package.

Report Informativeness

Users want the Report to include a link to the original security policy, the version of the SCAP Security Guide, and the IP address of the target system.

Some users were interested in an explanation of how the scan score is calculated. How OVAL operators are evaluated because they confused them with boolean operators. In general, they wanted to have an explanation readily available in the report to avoid switching to the SCAP specification.

Users also wanted to be able to visually distinguish platform usability checks from configuration checks.

Report Capabilities

Users suggest a two-layered view of the Evaluation Characteristics, with the most useful items visible after the first expansion and the rest of it after the second. Users were not able sort rules by title, severity, and outcome.

Users are interested in being able to filter rules using STIG identifiers.

Users missed buttons to uncheck all filters, clear the search box, filter only matching or non-matching rules, and a button to move to the top of the report. Users would like to have links in rules organized by security policy, for example.

Chapter 9

Conclusion

The goal of this work was to learn about SCAP standards and examine the reports generated by the OpenSCAP security scanner or other SCAP scanners, propose improvements to the HTML report content, and develop a tool that generates an interactive HTML report from the SCAP results.

The tool was packaged into an RPM package available on Fedora Linux and other distributions such as Red Hat Enterprise Linux 9 or Centos Stream 9 via Fedora EPEL packages.

The main part of this thesis is to learn about the SCAP standard report formats and implement the tool, which is packaged in a package called `openscap-report`, which provides a `oscap-report` tool that can generate an interactive HTML report.

This thesis begins by explaining the main components of the SCAP standard. In this Chapter 2, the author draws on his next three years of experience with the OpenSCAP project. Chapter 3 introduces the OpenSCAP project and other SCAP scanners. Chapter 4 provides an overview of the reporting capabilities of SCAP scanners. In Chapter 5, the author presents his reporting enhancements. In Chapters 6 and 7, the author describes the implementation and what tools were used to develop and package the tool to generate interactive HTML reports. In Chapter 8, the author develops the User Testing Methodology and presents the results of the user testing.

The author successfully developed the `oscap-report` tool and created an RPM package called `openscap-report`, which is available for Fedora Linux and other distributions such as RedHat Enterprise Linux 9 or Centos Stream 9 via Fedora EPEL packages. Also, `openscap-report` has been bundled into openSUSE Linux by SUSE engineers. It is possible that in the future, `openscap-report` will also be included in Ubuntu Linux by Ubuntu developers.

You can check the current availability of `openscap-report` on Linux distributions at pkgs webpage[1].

User testing provides some ideas and reveals issues that the author plans to implement and fix in future development. Users have also expressed interest in other versions of the reports, such as the report generated from OVAL results or formats like JSON.

---

[1] https://pkgs.org/download/openscap-report

Bibliography

[1] *OVAL Content Creation Tutorial.* Center for Internet Security, 2017 [cit. 2022-12-18]. Available at: https://ovalproject.github.io/getting-started/tutorial/.

[2] *Qualys PC/SCAP Auditor – Getting Started Guide.* Getting Started Guide. QUALYS, INC, november 2017 [cit. 2022-12-28]. Available at: https://www.qualys.com/docs/qualys-scap-getting-started-guide.pdf.

[3] *Getting started with the perf command* [online]. 2021 [cit. 2023-4-12]. Available at: https://www.ibm.com/docs/en/linux-on-systems?topic= performance-getting-started-perf-command.

[4] *ComplianceAsCode Developer documentation.* Red Hat Security Compliance Team, 2022 [cit. 2022-12-20]. Available at: https://complianceascode.readthedocs.io/en/latest/manual/developer/ 06_contributing_with_content.html#applicability-of-content.

[5] *The OpenSCAP Base* [online]. 2022 [cit. 2022-11-15]. Available at: https://www.open-scap.org/tools/openscap-base/.

[6] *The OpenSCAP tools* [online]. 2022 [cit. 2022-11-15]. Available at: https://www.open-scap.org/tools/.

[7] *The OpenSCAP Workbench* [online]. 2022 [cit. 2022-11-15]. Available at: https://www.open-scap.org/tools/scap-workbench/.

[8] *SCAP security guide* [online]. 2022 [cit. 2022-11-15]. Available at: https://www.open-scap.org/security-policies/scap-security-guide/.

[9] *The Security compliance content in SCAP, Bash, Ansible, and other formats* [online]. 2022 [cit. 2022-11-15]. Available at: https://github.com/ComplianceAsCode/content.

[10] *Extra Packages for Enterprise Linux (EPEL).* The Fedora Project, 2023 [cit. 2023-4-12]. Available at: https://docs.fedoraproject.org/en-US/epel/.

[11] *Fedora Packaging Guidelines.* The Fedora Project, 2023 [cit. 2023-4-12]. Available at: https://docs.fedoraproject.org/en-US/packaging-guidelines/.

[12] *Fedora Packaging Guidelines: Python.* The Fedora Project, 2023 [cit. 2023-4-12]. Available at: https://docs.fedoraproject.org/en-US/packaging-guidelines/Python/.

[13] *The lxml XML toolkit for Python.* Contributors of The lxml library, 2023 [cit. 2023-4-12]. Available at: https://lxml.de/.

[14] *Packaging and distributing projects*. Python Software Foundation, 2023 [cit. 2023-4-12]. Available at: https://packaging.python.org/en/latest/guides/distributing-packages-using-setuptools/.

[15] *Packit*. Contributors of The Packit, 2023 [cit. 2023-4-12]. Available at: https://packit.dev/.

[16] *Pytest: helps you write better programs*. Holger Krekel and pytest-dev team, 2023 [cit. 2023-4-12]. Available at: https://docs.pytest.org/en/7.3.x/.

[17] *The Python Standard Library – Data Classes*. Python Software Foundation, 2023 [cit. 2023-4-12]. Available at: https://docs.python.org/3/library/dataclasses.html.

[18] *Tox – automation project*. The Tox Developers, 2023 [cit. 2023-4-12]. Available at: https://tox.wiki/en/latest/.

[19] Atlantic, N. *Security Content Automation Protocol (SCAP) Compliance Checker (SCC)* [online]. 2022 [cit. 2022-11-20]. Available at: https://www.niwcatlantic.navy.mil/scap/.

[20] Eranian, S., Gouriou, E., Moseley, T. and Bruijn, W. de. *Linux kernel profiling with perf*. 2023 [cit. 2023-4-12]. Available at: https://perf.wiki.kernel.org/index.php/Tutorial.

[21] Haičman, M. *SCAP Security Guide intro pitch* [Red Hat internal slides]. 2018 [cit. 2023-4-12].

[22] Lysoněk, M. *System for Automatic Filtering of Tests*. Brno, CZ, 2020. [cit. 2023-4-12]. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/23098/.

[23] Černý, J. *Nástroj pro tvorbu definic OVAL v projektu OpenSCAP*. Brno, CZ, 2016. [cit. 2023-4-12]. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/18235/.

[24] Waltermire, D., Cichonski, P. and Scarfone, K. *Common Platform Enumeration: Applicability Language Specification Version 2.3*. NIST Interagency Report 7698. National Institute of Standards and Technology, august 2011 [cit. 2022-12-18]. Available at: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7698.pdf.

[25] Waltermire, D., Quinn, S., Booth, H., Scarfone, K. and Prisaca, D. *The Technical Specification for the Security Content Automation Protocol (SCAP)*. NIST Special Publication 800-126, 3rd ed. National Institute of Standards and Technology, february 2018 [cit. 2022-11-28]. Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-126r3.pdf.

[26] Standards, N. I. of and Technology. *Extensible Configuration Checklist Description Format (XCCDF)* [online]. 2016 [cit. 2022-12-18]. Available at: https://csrc.nist.gov/Projects/Security-Content-Automation-Protocol/Specifications/xccdf#resource-1.2.

Appendix A

Contents of the Included Storage Media

The storage medium contains the following structure[1]:

```
/ .................................................. Root directory of storage media
├── openscap-report ......................... Source files of openscap-report project
│   ├── .github ....................................... Configuration files for GitHub
│   ├── docs ........................................ Source codes of documentation
│   ├── openscap_report ............................ Source codes of implementation
│   │   └── cli.py ............................................. Command line API
│   ├── plans ................................... TMT plans for the TMT test suite
│   ├── tests .......................................... Source codes of test suite
│   ├── openscap-report.spec .......................................... Spec file
│   ├── setup.py ............................................ Python setup script
│   ├── MANIFEST.in ......................... Specification of files in Python package
│   ├── release.sh .............................................. Realase script
│   ├── LICENSE ...................................................... License
│   ├── LICENSE.spdx ....................................... SPDX version of license
│   ├── requirements.txt ............................. Python package requirements
│   └── README.md
├── manual ............................................. Compiled documentation
│   ├── oscap-report.1 ................................... Compiled manual page
│   └── HTML_manual ..................................... Compiled HTML manual
├── these_source_code .............................. Source codes of this document
├── RPM ...................................................... Built RPM packages
├── user_testing .......................................... Files for user testing
├── report_example.html ............................. Example of generated report
├── xrodak00_openscap-report.pdf ..... Electronic version of this document for print
├── xrodak00_openscap-report_IS.pdf ....... Electronic version of this document for
    submission to the IS VUT
```

---

[1] The entire storage medium contains about 67 directories and 355 files.

Appendix B

Build Manual

To build a manual page and a manual that contains all the information and usage of the `openscap-report` package and the `oscap-report` command line tool, you need to install the `python3-sphinx`, `python3-sphinx_rtd_theme` packages. The built versions of the manual page and manual are contained on storage media in a directory named `manual` or provided with the online version of this documentation[1].

To build a man page open directory `docs` and run this command:

```
$ sphinx-build -b man . TARGET_DIR
```

To build HTML manual open directory `docs` and run this command:

```
$ sphinx-build . TARGET_DIR
```

Optionally files in the `modules` directory can be regenerated using a command executed in directory `openscap-report`:

```
$ sphinx-apidoc openscap_report -o docs/modules
```

---

[1]https://openscap-report.readthedocs.io/en/latest/