



BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

**AUTOMATED SECURITY COMPLIANCE SCANNING
OF MS WINDOWS OPERATING SYSTEM
USING OPENSCAP PROJECT**

AUTOMATIZOVANÉ OVĚŘOVÁNÍ KONFIGURACE OPERAČNÍHO SYSTÉMU

MS WINDOWS POMOCÍ PROJEKTU OPENSCAP

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR

AUTOR PRÁCE

Bc. JAN ČERNÝ

SUPERVISOR

VEDOUCÍ PRÁCE

Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů

Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Černý Jan, Bc.**

Obor: Informační systémy

Téma: **Automatizované ověřování konfigurace operačního systému MS Windows pomocí projektu OpenSCAP**

Automated Security Compliance Scanning of MS Windows Operating System Using OpenSCAP Project

Kategorie: Operační systémy

Pokyny:

1. Nastudujte standard SCAP, který popisuje automatizované ověřování bezpečné konfigurace systémů. Soustřeďte se na jazyk OVAL a jeho části specifické pro MS Windows. Seznamte se s implementací OpenSCAP.
2. Navrhněte změny projektu OpenSCAP potřebné pro přidání plné podpory pro operační systém MS Windows. Navrhněte způsob implementace objektů jazyka OVAL specifických pro operační systém MS Windows.
3. Implementujte navržená řešení jako rozšíření projektu OpenSCAP.
4. Ověřte řešení na reálných bezpečnostních politikách pro operační systémy MS Windows ve formátu SCAP.

Literatura:

- S. Quinn, D. Waltermire, C. Johnson, K. Scarfone, J. Banghart. 2009. The Technical Specification for the Security Content Automation Protocol (SCAP): Scap Version 1.0. NIST Special Publication 800-126. ISBN: 9781437934878
- Domovské stránky iniciativy SCAP, National Institute of Standards and Technology, <http://scap.nist.gov/>
- Domovské stránky projektu OpenSCAP, <http://www.open-scap.org/>

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese

<http://www.fit.vutbr.cz/info/szz/>

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D., UITS FIT VUT**

Datum zadání: 1. listopadu 2017

Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta informačních technologií

Ústav inteligentních systémů

612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstract

This work deals with security compliance of computer systems, namely operating systems, applications and system services. Concept of security policies, their evaluation and their enforcement is described. Security compliance automation and the SCAP standard are presented. OpenSCAP project, which is used as an SCAP scanner, is described together with its tools and its usage. An idea to add support of Microsoft Windows within OpenSCAP, which was previously unsupported, is presented. The core part of the thesis is to identify necessary changes of OpenSCAP and to design an extension of this project. All these modifications are implemented. The solution is demonstrated on security policies for Windows. The solution is evaluated and further improvements are discussed.

Abstrakt

Tato práce se zabývá problematikou bezpečné konfigurace výpočetních systémů, jako jsou operační systémy, aplikace a služby. Seznamuje čtenáře s konceptem bezpečnostních politik a jejich ověřováním. Soustředí se na problematiku automatizace bezpečné konfigurace s důrazem na standard SCAP. Popisuje projekt OpenSCAP, který se používá jako SCAP scanner, jeho aplikace a jejich použití. Navrhuje rozšířit OpenSCAP i na operační systém Microsoft Windows, který doposud nebyl podporován. Těžištěm práce je identifikace nutných změn projektu OpenSCAP a návrh jeho rozšíření. Všechny navržené úpravy jsou implementovány. Implementované řešení je demonstrováno s využitím bezpečnostních politik pro Windows, vyhodnoceno a také jsou diskutovány možnosti jeho budoucího vylepšení.

Keywords

SCAP, OpenSCAP, OVAL, Windows, open-source, Security Audit, Security Compliance, Configuration

Klíčová slova

SCAP, OpenSCAP, OVAL, Windows, open-source, bezpečnostní audit, ověřování bezpečné konfigurace

Reference

ČERNÝ, Jan. *Automated Security Compliance Scanning of MS Windows Operating System Using OpenSCAP Project*. Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Aleš Smrčka, Ph.D.

Rozšířený abstrakt

Tato práce se týká počítačové bezpečnosti. Jedním z prvků počítačové bezpečnosti je vhodné nastavení operačního systému a vhodná konfigurace běžících služeb a aplikací. Takové nastavení by mělo například omezit otevřené porty, povolit pouze vybrané aplikace, nastavit používané sady šifer nebo vyřešit přihlašování uživatelů a uživatelská práva.

Správná a bezpečná konfigurace může být popsána nějakou bezpečnostní politikou. Příkladem mohou být politiky typu DISA STIG používané v USA. Pro zápis těchto politik ve strojově zpracovatelné podobě lze využít standard SCAP (Security Content Automation Protocol, protokol pro automatizaci bezpečnostního obsahu). Politika zapsaná ve SCAP formátu poté popisuje konkrétní požadavky na stav systému formou jednoduchých pravidel.

Pro automatické ověření zda systém je v souladu s bezpečnostní politikou a vyhovuje jejím požadavkům existují nástroje, jejichž vstupem je soubor ve formátu SCAP a výstupem přehledná zpráva, která informuje o míře splnění těchto požadavků.

Příkladem nástroje, který implementuje standard SCAP, je projekt OpenSCAP. Jedná se o software s otevřeným zdrojovým kódem (tzv. open source software). Tento produkt je vyvíjen převážně ve společnosti Red Hat. Je to populární nástroj používaný na operačním systému Linux. Bohužel, OpenSCAP nepodporuje Microsoft Windows nebo další oblíbené operační systémy.

Cílem práce je rozšířit již existující projekt OpenSCAP o podporu operačního systému Microsoft Windows. To znamená, že bude možné pomocí programu OpenSCAP ověřit, zda konkrétní instance Windows vyhovuje pravidlům bezpečnostních politik zapsaných ve formátu SCAP.

Jedním z důvodů vedoucích k této práci byly časté požadavky uživatelů. Uživatelé hledali schopný nástroj pro vyhodnocování SCAP politik pro Windows. Je důležité si uvědomit, že ostatní nástroje implementující SCAP jsou drahé komerční produkty. Dále je nutné zmínit, že uživatelé grafického programu SCAP Workbench pro Windows si stěžovali, že tento nástroj umí pracovat pouze se vzdálenými unixovými servery a nikoli s lokální instancí Windows. Také se objevila potřeba oživení projektu a získání více přispěvatelů.

Nejtěžší částí práce bylo změnit silně Linuxově specifické záležitosti ve zdrojovém kódu OpenSCAPu na multiplatformní kód a umožnit vývoj pod oběma systémy.

Pro snadnou konfiguraci, nalezení závislostí a sestavení na různých platformách byl sestavovací systém přepsán s využitím programu CMake. To mimo jiné urychlilo kompilaci a umožnilo programovat ve Visual Studiu 2017. Závislosti na Windows byly vyřešeny pomocí Vcpkg, nového nástroje Microsoftu pro sestavení potřebných knihoven třetích stran.

Kód v jazyce C, který byl použitelný pouze v překladači GCC na Linuxu, byl upraven tak, aby šel přeložit a spustit i na Windows pomocí Visual Studia 2017. To zahrnovalo rozsáhlé změny v kódu, náhradu některých funkcí a maker jinými nebo dokonce programování nových částí s využitím aplikačního programového rozhraní (API) Windows.

Dále byla provedena rozsáhlá změna v logice OpenSCAPu. Procesy, zvané sondy, které doteď fungovaly jako oddělené procesy, byly sloučeny a vloženy do hlavního procesu. OpenSCAP je nyní jednoprosesová aplikace, což umožnilo odstranit velmi složitý způsob meziprosesové komunikace, který byl dosud používán. Tato změna přinesla větší přenositelnost, snadnější ladění programu a teoreticky i menší chybovost a možnost zrychlení.

Těžištěm práce byla implementace Windows součástí jazyka OVAL (Open Vulnerability and Assessment Language, otevřený jazyk pro hodnocení a zranitelnosti). Úkolem bylo navrhnout implementaci některých vybraných testů jazyka OVAL. Tyto testy byly implementovány jako nové moduly v rámci OpenSCAPu. Ačkoli se jedná o implementaci

standardu, konkrétní řešení je netriviální. Řešení využívá pokročilých volání aplikačního programovaného rozhraní Windows.

Nakonec se úspěšně podařilo vytvořit spustitelný funkční program. Sestavený program je zabalen jako instalační MSI balíček nesoucí všechny závislosti. Tento balíček se sám vygeneruje z CMake.

Práce probíhala ve spolupráci s hlavním vývojovým proudem OpenSCAPu. Všechny zmiňované změny byly podrobeny kontrole ze strany vývojářů tohoto produktu a všechny byly začleněny do vývojové verze kódu.

Výsledkem práce je spustitelná verze OpenSCAPu běžící na Windows. Tato verze zvládá většinu operací. Je schopná provádět skenování systému s využitím reálných bezpečnostních politik zapsaných ve SCAP formátu. Bylo ověřeno, že OpenSCAP pro Windows dokáže například pracovat s politikami DISA STIG a USGCB na Windows 7 a 10. OpenSCAP v současnosti umí vyhodnotit 69 % pravidel v DISA STIG pro Windows 10 a 80 % pravidel v USGCB pro Windows 7. Výsledky jsou prezentovány prostřednictvím HTML reportu, tedy způsobem obvyklým v OpenSCAPu.

Během práce byly implementovány pouze nejdůležitější testy definované ve Windows části jazyka OVAL. Implementace dalších v budoucnosti je ale možná. Budoucí vývojáři mohou postupovat podle popisu, který je obsažen v této práci.

Kromě možnosti přidání plné podpory OVAL má práce celou řadu možných budoucích rozšíření. Zmiňme například možnost integrace s grafickou aplikací SCAP Workbench nebo podporu pro tvorbu bezpečnostních politik pro Windows uvnitř projektu SCAP Security Guide. Nejzajímavější se zdá být možnost integrace s nástroji pro hromadnou správu systémů v heterogenních infrastrukturách.

Vzhledem k dosaženým výsledkům a možnostem budoucího rozšíření by práce mohla být užitečná i v praxi.

Automated Security Compliance Scanning of MS Windows Operating System Using OpenSCAP Project

Declaration

Hereby I declare that this master's thesis was prepared as an original author's work under the supervision of Ing. Aleš Smrčka, Ph.D. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Jan Černý
May 20, 2018

Acknowledgements

I would like to thank my advisor Ing. Aleš Smrčka, Ph.D. for his professional advice and my consultant Ing. Peter Vrabc for his great support. I would like to thank Mgr. Martin Preisler and other people from OpenSCAP team for inspiring code reviews, useful feedback and valuable pointers. I also thank my family and friends who supported me during my studies.

Contents

1	Introduction	3
2	Security Compliance	5
2.1	Security Policies	5
2.2	Security Content Automation Protocol	8
2.3	Open Vulnerability and Assessment Language	9
2.4	Extensible Configuration Checklist Description Format	11
3	OpenSCAP Project	13
3.1	OpenSCAP	13
3.1.1	OpenSCAP Library	13
3.1.2	OpenSCAP Command Line Interface	14
3.1.3	SCAP Security Guide	14
3.1.4	SCAP Workbench	15
3.1.5	Other Tools from the OpenSCAP Ecosystem	15
3.2	Typical OpenSCAP Use Cases	16
3.3	OpenSCAP Alternatives	18
3.4	OpenSCAP Internal Implementation	18
4	Extending OpenSCAP to Windows	20
4.1	Requirements	20
4.2	Extension of an Existing Project	23
4.3	XML Changes	23
4.4	Addressing Portability Issues	24
4.4.1	Moving to CMake Build System	24
4.4.2	Dependencies	27
4.4.3	Portable C Code	27
4.5	OpenSCAP Architecture Changes	30
4.5.1	Offline Mode Changes	33
4.6	Retrieving Windows Operating System Properties	34
4.6.1	Adding a New Probe in OpenSCAP	35
4.6.2	OpenSCAP System Info Probe	37
4.6.3	OVAL Independent Family Test	38
4.6.4	OVAL Windows Registry Test	38
4.6.5	OVAL Windows Access Token Test	41
4.6.6	OVAL WMI Test	42
4.7	Packaging	44

5	Testing, Verification, and Assessment of the Project	46
5.1	Scanning Windows Systems Using Real World Security Policies	46
5.2	Testing	47
5.3	Future Plans	50
5.3.1	Additional Probes	50
5.3.2	SCAP Workbench	51
5.3.3	PowerShell Remediation	52
5.3.4	Windows Support in SCAP Security Guide	52
5.3.5	Automatic Rights Elevation	53
5.3.6	Porting Test Suite	53
6	Conclusion	54
	Bibliography	55
A	OpenSCAP Upstream Pull Requests	58
B	Contents of the Attached Media	60

Chapter 1

Introduction

Information technology has expanded to all areas of our lives. Computer systems control critical processes and store sensitive data. Our civilization depends on them and we expect them to be fast, safe, reliable, faultless and secure.

However, computer systems are prone to misuse and are targeted by attackers. People and organizations are concerned about protection and security of their systems and data privacy. They realize that breaking the systems or stealing valuable data has serious consequences. In general, we can say that computer security has become a very important topic.

A very important part of computer security is proper configuration of the systems, namely operating systems, their services and applications. The industry has defined certain recommendations, rules, regulations and policies which set requirements on secure configuration. In some organizations it is required that every device meets these requirements. For others it is very beneficial to follow these policies. The challenge is to deploy a suitable policy and to continuously verify that systems comply with the policy. This area is often denoted as security compliance.

To facilitate security compliance management, Security Content Automation Protocol (SCAP) has been developed and standardized. SCAP enables verification of secure configuration in an automated way. It also provides a way to express security policies in a machine-readable form.

One of popular implementations of SCAP is OpenSCAP¹. It is an open source project which is used frequently to automate security audit on Linux machines. Unfortunately, OpenSCAP works only on Unix systems and lacks support for other operating systems. Currently, the most missing platform is Microsoft Windows, because OpenSCAP users frequently have both machines running Linux and Windows in their infrastructure. Moreover, there is no other open source tool that would manage compliance of the Microsoft Windows operating system. It would be appreciated to use the same tool with same output formats across different systems. Also, the support for other operating systems can be leveraged in systems management tools that OpenSCAP integrates with.

The goal of this work is to enable scanning of Microsoft Windows operating system using OpenSCAP. This goal has been achieved by contributing to the existing project, which includes both adding new code and improving the existing parts. A Microsoft Windows extension is important for the OpenSCAP project and helps with promoting the product.

¹<https://www.open-scap.org>

The thesis begins with a brief introduction to the area of security compliance. The text mentions frequently used security policies and methods of computer security evaluation. It describes the most common requirements and their reasoning. Then, it gives a rationale to automate verification using scanners. Finally, the chapter focuses on SCAP which is a standard used in this area. A description of formats within the SCAP standard will be provided and their usage will be explained.

Chapter 3 illustrates OpenSCAP, an open source security compliance solution. OpenSCAP provides both high-quality SCAP content and certified SCAP scanner. The chapter gives an overview of the project and explains typical usage of the applications and tools that are part of OpenSCAP. Due to the goal of this thesis, it also describes its internal design and implementation. Moreover, a comparison of OpenSCAP and its competition will be provided.

Chapter 4 describes the core part of this thesis where OpenSCAP support for Windows is designed and implemented. Portability issues and other difficulties arising from differences of the new platform are discussed. The problems with current design and implementation of the tool are identified and many changes are proposed. Then, the text shows design and implementation of Windows specific parts of the SCAP standard, namely Open Vulnerability and Assessment Language (OVAL) Windows Definition Schema. The chapter describes how the thesis goal was achieved step-by-step. It focuses on various problems that were encountered during the implementation and how they were solved.

In the last chapter, the solution is evaluated by using real world security policies for Microsoft Windows written in SCAP format available on the Internet, e.g. Security Technical Implementation Guides (STIGs). The chapter also mentions how the changes in OpenSCAP project were tested and accepted into the upstream. The text assesses achieved results and evaluates the benefits of the project. Finally, further improvements of the created work are suggested as well.

Chapter 2

Security Compliance

Computer security is a broad topic and is related to many areas including system administration, cryptography, networking, even hardware, but also legal requirements and ethics.

Computer security can be improved by a large variety of measures. We can install security updates, use encryption, communicate using suitable network protocols, follow good programming practices, detect malware, run access control systems, etc. The measures also include establishing business processes, physical measures or prevention and user training.

A very important part of computer security is secure configuration of operating system, its services and also installed applications. It is necessary to correctly configure user authentication, set-up logging, forbid old protocols and broken ciphers, shut down unused services, install various utilities and services that improve security, verify access rights, and many other things.

In fact, correct and secure configuration is a complicated task that requires deep knowledge and experience. It is not clear what exactly the secure configuration should look like. It can differ depending on used technologies, business area or regulatory and legal requirements.

To define the correct secure system configuration many security policies have been developed. The policies provide a guidance on tasks that should be done to improve and maintain effective configuration settings focusing primarily on security. These policies are often published in a form of an official document.

2.1 Security Policies

In 1970s, U.S. military started to widely use mainframe computers to support their operations. They faced problems of processing classified information securely. They started a research of technical issues associated with securing computer systems and they defined policies for the systems. In 1983, U.S. Department of Defense (DoD) published Trusted Computer System Evaluation Criteria (TCSEC), frequently referred as The Orange Book, due to the colour of its cover [7].

The TCSEC was used to evaluate commercial products. The TCSEC defines four divisions: D, C, B and A where division D has the lowest and division A has the highest security. Each division puts more requirements on the trust we can place on the evaluated system. Additionally, divisions C, B and A are broken into hierarchical classes: C1, C2, B1, B2, B3 and A1.

The TCSEC was followed by other security standards which were also published in books of different colours, therefore these publications are often referred as *The Rainbow Series*. The impact of TCSEC was to motivate vendors to implement security controls into their products and increase awareness of computer security in general.

Nowadays, the TCSEC is abandoned [7], but to evaluate the security of information technology products we may follow The Common Criteria for Information Technology Security Evaluation (CC) [16], which is an international standard (ISO/IEC 15408). There are seven Evaluation Assurance Levels (EALs) that specifies level of meeting security functional requirements. A list of CC certified products can be found on the Common Criteria portal [16].

However, the CC does not provide a configuration guidance. It serves to evaluate level of security of the product, but the product can be used in different ways in different environments.

The requested configuration of computer systems is described in various other policies, that are specific for type of organization or the character of data they work with. They define a policy that the system should follow. We may think of these policies as of specific implementations for the CC.

In some areas, for example government, military or financial sector it is obligatory to follow these policies. All systems in an organization have to comply with the prescribed policy in such case. Otherwise the system cannot be used. Therefore, security audits are performed to verify compliance with the policies.

Let us mention a few examples of policies that are widely used and enforced.

- The United States Government Configuration Baseline (USGCB) creates security configuration baselines for IT products deployed in government agencies in The United States [26].
- The Security Technical Implementation Guides (STIGs) are configuration standards published by Defense Information Systems Agency (DISA). These policies are used on systems within U.S. Department of Defense [5]. There are STIGs for a large amount of software products.
- The Payment Card Industry Data Security Standard (PCI DSS) is an information security standard for organizations that handle credit card payments.
- The Health Insurance Portability and Accountability Act of 1996 (HIPAA) enforces data protection for health care providers, health plans, payers, and similar in USA.
- The U.S. Government Approved Protection Profile—Protection Profile for General Purpose Operating Systems, known as OSPP, describes the security functionality of operating systems in terms of CC and defines functional and assurance requirements for such products [24]. This is prepared by The National Information Assurance Partnership (NIAP), who is responsible for U.S. implementation of the Common Criteria.
- and many others.

Most of these policies are very detailed and contain a large amount of requirements. It is difficult and costly to configure the systems in the required way. The policies frequently put requirements on installed services, time synchronization, password length or expiration, authentication, ciphers, etc.

Examples of configuration rules on Linux (adopted from Guide to Secure Configuration of Red Hat Enterprise Linux 7):

- Ensure `/var/log` Located On Separate Partition.
- Verify Permissions on `shadow` File.
- Enable Randomized Layout of Virtual Address Space.
- Ensure `Logrotate` Runs Periodically.
- Set Password Hashing Algorithm in `/etc/login.defs` .
- Enable the NTP Daemon.
- Set SSH Idle Timeout Interval.
- Set Password Strength Minimum Uppercase Characters.
- Disable the `selinuxuser_use_ssh_chroot` SELinux Boolean.
- Record Events that Modify the System's Discretionary Access Controls.
- Enable GNOME3 Screensaver Lock After Idle Period.

Examples of configuration rules on Windows (adopted from Windows 10 DISA STIG):

- The TFTP Client must not be installed on the system.
- Windows 10 account lockout duration must be configured to 15 minutes or greater.
- Passwords must, at a minimum, be 14 characters.
- Indexing of encrypted files must be turned off.
- Kerberos encryption types must be configured to prevent the use of DES and RC4 encryption suites.
- PowerShell script block logging must be enabled.
- Windows 10 must be configured to disable Windows Game Recording and Broadcasting.
- User Account Control must automatically deny elevation requests for standard users.

Organizations also need to verify if the given system is compliant with the policy to pass the audits and fulfil legal constraints.

Although this verification might seem simple, it is a task that requires to check contents of many configuration files, browse large file systems, check running processes, check network interfaces, inspect access rights of all files, etc. This takes a few days of work of a person. Moreover this verification or audit needs to be performed repeatedly, because systems evolve, new applications are deployed, new software vulnerabilities are discovered and requirements change.

Nowadays a typical IT infrastructure can consist of hundreds of servers and thousands of virtual machines and workstations. It is obvious that it is not possible to perform security audit manually in such an environment. Therefore, we need to automate the process. We

want to simply scan the given system and get a report which says which requirements are fulfilled.

This can be achieved by a big set of scripts, or various proprietary systems that claim to solve the security compliance problem. The problem of these solutions is that they are not standardized and interoperable [30]. To address these problems the Security Content Automation Protocol (SCAP) has been created.

2.2 Security Content Automation Protocol

Security Content Automation Protocol (SCAP) is a standard used in security automation, configuration management and evaluation. It was published by National Institute of Standards and Technology (NIST) in the Special Publication 800-126 [35]. The current version is SCAP 1.3 which was released in February 2018.

The standard includes a set of multiple components or more precisely languages and schemas. Each component is based on Extensible Markup Language (XML) and specifies machine-readable documents that are used to express expected configuration, define benchmarks and store results.

The SCAP components are:

1. Common Vulnerabilities and Exposures (CVE).
2. Common Configuration Enumeration (CCE).
3. Common Platform Enumeration (CPE).
4. Common Weakness Enumeration (CWE).
5. Common Vulnerability Scoring System (CVSS).
6. Extensible Configuration Checklist Description Format (XCCDF).
7. Open Vulnerability and Assessment Language (OVAL).
8. Open Checklist Interactive Language (OCIL).
9. Asset Identification (AI).
10. Asset Reporting Format (ARF).
11. Common Configuration Scoring System (CCSS).
12. Trust Model for Security Automation Data (TMSAD).
13. Software Identification Tags (SWID).

The key components are XCCDF and OVAL. Typically, the policy is expressed in a form of an XCCDF document (XCCDF benchmark). For a detailed description of the XCCDF, see Section 2.4.

Each rule references a check written in the OVAL language, called an OVAL definition. The OVAL definition describes a specific state of a specific system object. For example, an OVAL definition could define that configuration file `/etc/selinux/config` must exist and must contain the string `SELINUX=enforcing`. The OVAL definition must be satisfied in order to fulfil the XCCDF rule requirements.

The XCCDF document can be passed to an SCAP scanner, which is a tool that interprets the SCAP document, scans the system to get its current state, compares the current state with state expected by OVAL definition, computes results and shows a report. An example of such an SCAP scanner is OpenSCAP, which will be described in Chapter 3.

The OVAL definitions are usually found in a separate file. To facilitate distribution, it is possible to wrap multiple types of SCAP documents into a SCAP datastream, which is a container that allows to store everything in a single file.

Some checklists can be downloaded from National Checklist Program Repository website [25]. Another favourite resource is the SCAP Security Guide, which is described in Subsection 3.1.3. Other checklists are developed by consultants or security organizations.

Nowadays SCAP is widely used, some organizations are even required to use SCAP. Most important users are government organizations, but also army contractors, organizations that work with U.S. public sector and financial sector. SCAP is started to be used worldwide.

On the other hand, many commercial businesses think SCAP is too complicated. Also the tools providing SCAP for their platform are expensive. This group of users is afraid to start using SCAP to automate their security.

One of the aims of this work is to bring SCAP to larger amount of users by introducing free, open-source solution on a major platform, which Windows is. The users can benefit from previous work on SCAP standard and existing experience of the OpenSCAP project.

2.3 Open Vulnerability and Assessment Language

Open Vulnerability and Assessment Language (OVAL) is a part of the SCAP standard. OVAL is used to define the expected system state and report the actual state. OVAL is a domain-specific language (DSL) designed exclusively for purpose of security audit, therefore its abilities are limited. But it perfectly fits the use case of security compliance.

The language used to be maintained by MITRE [34] and now it is preserved by Center for Internet Security [2], that also keeps up a repository of OVAL definitions.

The specification is open and information security community can propose changes and improvements. The current version is 5.11.1.

The language describes the three main steps of the assessment process [34]:

1. Representing configuration information of systems for testing.
2. Analysing the system for the presence of the specified machine state (vulnerability, configuration, patch state, etc.).
3. Reporting the results of this assessment.

Similar to other SCAP components, the language is derived from XML. OVAL consists of three parts that correspond to the aforementioned steps.

1. OVAL Definition schema for expressing a specific machine state.
2. OVAL System Characteristics schema for representing system information.
3. OVAL Results schema for reporting the results of an assessment.

OVAL definitions describe desired configuration of a system. A definition is the most high level logical unit of the OVAL language. A definition consists of one or more OVAL

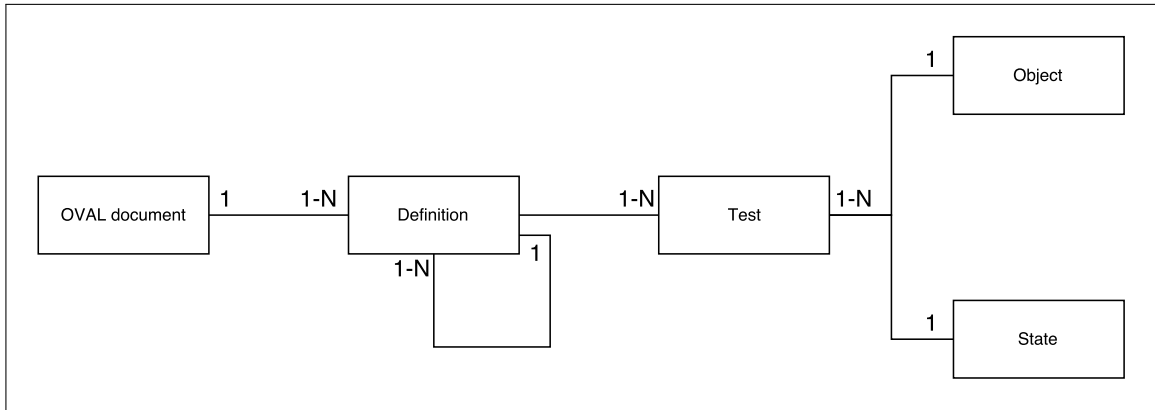


Figure 2.1: Structure of OVAL elements in OVAL Definitions.

```

<objects>
  <linux:systemdunitdependency_object id="oval:x:obj:1" comment="list of
    dependencies of multi-user.target" version="1">
    <linux:unit>multi-user.target</linux:unit>
  </linux:systemdunitdependency_object>
</objects>

<states>
  <linux:systemdunitdependency_state id="oval:x:ste:1" comment="is chronyd
    listed at least once in the dependencies" version="1">
    <linux:dependency entity_check="at least one">
      chronyd.service
    </linux:dependency>
  </linux:systemdunitdependency_state>
</states>
  
```

Listing 2.1: Responsibilities of OVAL objects and states.

tests that need to be passed to satisfy the definition. The OVAL test compares an OVAL object with an OVAL state. Relation of OVAL elements is shown in Figure 2.1.

OVAL object is an XML element describing particular object that exists on a system—a file, a process, an environment variable, a kernel parameter, a value in a configuration file, an entry in SQL database, an entry in Windows registry and many others. Each type of object has a name (e.g. `process_object` or `file_object`) and a specific set of child elements and attributes. The child elements and attributes differ depending on object type and purpose. For example, a `textfilecontent_object` serves to describe a text string in a file.

OVAL state specifies features of an object that the object has to conform to fulfil the requirements of respective test. For every type of OVAL object, there exists a corresponding state with same prefix in OVAL definition schema. For example, a `file_state` corresponds to a `file_object`. States are optional elements, in OVAL documents we can find a lot of tests that do not contain any state, they only check whether a particular object exists or does not exist. We can see an example of related OVAL object and state in Listing 2.1.

The basic concept of creating definition by combining a test, an object and a state can be extended by more advanced constructions. For example, instead of using any test, a definition can be extended by another definition within the same OVAL document. Another important concept is using *filters* that more restrict the values of an OVAL object. OVAL also provides some functions that convert data or compute basic arithmetic operations. In fact, OVAL allows creating very complex logic. Therefore, OVAL definitions are usually difficult to write [3].

2.4 Extensible Configuration Checklist Description Format

The Extensible Configuration Checklist Description Format (XCCDF) is a format to define configuration checklists. XCCDF documents are high-level XML documents that add additional data and metadata on the top of OVAL definitions.

An XCCDF document is called a *Benchmark*. The XCCDF benchmark is a collection of rules. Each rule has an unique identifier, description and various metadata. Rules can also have severity, which means how a finding is important if the rule is violated. We can see an example of an XCCDF rule in Listing 2.2.

The Benchmark can contain multiple profiles. A profile is a set of rules. These sets can overlap, which means a rule is typically member of multiple profiles. The profiles specify a policy and they have their own title and a description.

The OVAL checks are referenced using `check-content-ref` element which references the concrete OVAL definition. It is possible to reference not only OVAL checks, but also OCIL questionnaires, and checks in other formats as well. That means in theory the rule can have multiple ways to evaluate, but in practice we prefer OVAL as a standard way.

Multiple XCCDF rules usually form an XCCDF group that groups together related rules. Groups can be e.g. auditing, updating, permissions, services. Groups also can be nested which means that a group can be inside a different group. A group can contain any amount of groups and rules.

XCCDF rules can also include a remediation script, which is a short code snippet written in any scripting language. This script modifies the system in order to be compliant with the rule. Users can run remediation scripts from the whole XCCDF document at once, which will lead to make their system compliant with the policy described in that XCCDF.

```

<Rule id="install_antivirus" selected="false" severity="high">
  <title xmlns:xhtml="http://www.w3.org/1999/xhtml" xml:lang="en-US">
    Install Virus Scanning Software
  </title>
  <description xmlns:xhtml="http://www.w3.org/1999/xhtml" xml:lang="en-US">
Install virus scanning software, which uses signatures to search for the
presence of viruses on the filesystem. Ensure virus definition files are
no older than 7 days, or their last release. Configure the virus scanning
software to perform scans dynamically on all accessed files. If this is
not possible, configure the system to scan all altered files on the system
on a daily basis. If the system processes inbound SMTP mail, configure the
virus scanner to scan all received mail.
  </description>
  <reference href="http://nvlpubs.nist.gov/nistpubs/SpecialPublications/
NIST.SP.800-53r4.pdf">SC-28</reference>
  <rationale xmlns:xhtml="http://www.w3.org/1999/xhtml" xml:lang="en-US">
Virus scanning software can be used to detect if a system has been
compromised by computer viruses, as well as to limit their spread
to other systems.
  </rationale>
  <platform idref="cpe:/a:machine"/>
  <ident system="https://nvd.nist.gov/cce/index.cfm">CCE-27140-3</ident>
  <check system="http://oval.mitre.org/XMLSchema/oval-definitions-5">
    <check-content-ref name="oval:ssg-install_antivirus:def:1"
      href="ssg-rhel7-oval.xml"/>
  </check>
</Rule>

```

Listing 2.2: An XCCDF Rule from SCAP Security Guide (shortened).

Chapter 3

OpenSCAP Project

OpenSCAP is a collection of open source tools for managing system security and configuration compliance [20]. OpenSCAP implements SCAP standard and enables its enforcement. The OpenSCAP ecosystem provides multiple tools to assist administrators and auditors with assessment, measurement and enforcement of security policies and baselines. The tools enable automated evaluation of security policies to reduce costs of performing security audits. In this chapter, we will describe OpenSCAP, its components and its usage.

3.1 OpenSCAP

OpenSCAP is an open source software project. The source code is available on GitHub¹, where the development of the project happens and where bugs can be reported. The project is released under open source licenses, most of the code is licensed by GNU Lesser General Public License (GNU LGPL).

The development was started in 2008 by Red Hat, Inc. Now people from other companies and from the community contribute to the project, but Red Hat still has a leading position. OpenSCAP 1.2.13 has been certified by NIST [27] in 2017, which is an important fact for users that are required to use only certified scanners.

OpenSCAP is packaged in various Linux distributions, including Red Hat Enterprise Linux, Debian, Fedora, Ubuntu, and OpenSUSE. There is also a limited availability on legacy systems, e.g. Solaris.

OpenSCAP strictly follows the SCAP standard [20], and supports most of SCAP components in all their versions. As of version 1.2.16, it can not evaluate OCIL questionnaires, though.

OpenSCAP is not a single monolithic application. Instead, it is an ecosystem of multiple tools, scripts, documentation and security policies in SCAP formats, that work together. It is not required to use all of the tools. Some of the tools have a specific usage—they extend SCAP capabilities or make their use easier.

3.1.1 OpenSCAP Library

The core part of the project is `libopenscap`, a shared library. The library implements majority of the OpenSCAP functionality, e.g. SCAP documents processing, system scanning, evaluation and reporting.

¹<https://github.com/openscap>

The library provides stable and documented application programming interface (API), which enables independent developers to create various applications working with SCAP capabilities. The library also has bindings for Ruby, Python and Perl languages.

3.1.2 OpenSCAP Command Line Interface

The main tool that OpenSCAP offers is the command-line application `oscap`. It is an SCAP scanner that runs in a terminal. It adds a text user interface on the top of the OpenSCAP shared library.

The main purpose of the `oscap` tool is to perform scanning of a local machine on which the tool is installed. Also, it can validate SCAP files, merge and split datastreams, generate remediation scripts, generate HTML guides and reports, evaluate separate OVAL files, print metadata from SCAP content and other use cases. The tool offers a large number of arguments and options. It is therefore a powerful tool, but from the author's experience, most of the functions are rarely used by normal users.

The application is hard to get familiar with. It does not work interactively, at the same time a lot of options must be provided by the user on the command line. The user experience is complicated because the command-line interface strictly follows logic and terminology of the SCAP specification. To illustrate this, the most frequent use case, scanning of the system, is hidden under `oscap xccdf eval` command. Worse, to get a meaningful report user has to add another 3 command line options. Therefore, the application is difficult for beginners.

3.1.3 SCAP Security Guide

It is important to realize that the main input for the SCAP scanner are SCAP documents, in other words security policies in SCAP format, often referred as *SCAP content*. They are interpreted and evaluated by the scanner tool.

The SCAP documents, mostly XCCDF and OVAL, define what should be evaluated and prescribe the expected state of the system. We can say that the result of scanning highly depends on the SCAP content quality and the tools will not do anything useful without appropriate inputs. Therefore, there was a need for an open source SCAP content, that would allow users to fully consume it without paying a high price for third-party SCAP content.

To offer an open source SCAP content, finally SCAP Security Guide (SSG) has been created. SSG is a collection of SCAP documents. For Red Hat Enterprise Linux, it implements most of the policies mentioned in Chapter 2.1. It also offers policies for Debian and other Linux distributions, for Firefox, Java Runtime Environment and other applications. Unfortunately, SSG does not provide Windows content, but Windows is more covered by SCAP content available on NIST website [25].

SSG uses SCAP standards in the following way: For every product (operating system or application) there is an XCCDF checklist. A specific policy is only a profile in this XCCDF checklist. A profile is a subset of rules contained in the XCCDF checklist. This approach enables sharing of rules across multiple policies, which is convenient, because they often overlap. The actual checks are written in OVAL—each rule references a check which is in a form of OVAL definition.

As of version 0.1.39, SCAP Security Guide contains a lot of rules. The rules are structured and logically organized. Every rule is accompanied by a detailed description and

rationale. Rules are marked by identifiers and references directly to the specific requirements in the policy (policies) the rule implements.

The rule also contain short pieces of code, called “remediations”, which can fix the configuration of the system so that it satisfies the rule. They are usually written in Bash or Ansible. The remediations for example edit the configuration files, disable a system service or install a missing software package. Remediations can be run either during the scan or any time later. It is also possible to extract the remediations from the SCAP files to generate an Ansible playbook or a bash script. They can be run separately in order to put the system in line with the given security policy. SSG is not only a SCAP resource.

It is of course possible to use different SCAP content than SCAP Security Guide in OpenSCAP products.

3.1.4 SCAP Workbench

SCAP Workbench is a GUI application that allows users to scan the local system or remote machines using SSH. It is implemented in C++ using Qt library and uses `libopenscap`. It is a simple and intuitive application and is intended for inexperienced users [20]. However, it provides only the most common functions, for advanced work it is necessary to use command line. Also, many servers do not have graphical stack installed at all.

After SCAP Workbench is started, it automatically offers to select SCAP Security Guide content and choose the appropriate profile. It can run the scan by clicking on a single button. We can see the scan process in Figure 3.1. Then, it can show the result, run remediations or generate a fix script. SCAP Workbench also allows to customize existing security policies to the user’s needs, e.g. to change minimal password length. The customization is called tailoring.

It is important to mention that there exists a Windows version for SCAP Workbench. However, as of version 1.1.6, SCAP Workbench for Windows can be used only to view or customize SCAP content and to scan remote Linux servers via network using SSH. It is practical for users who run Windows on their laptop and want to scan easily a Linux server and see the HTML report without copying the files back and forth. It is not possible to scan Windows machines, the Local button is greyed out. That is because the underlying `libopenscap` library does not implement Windows scanning. Users who download SCAP Workbench frequently report this as a bug and request Windows scanning to be implemented. Their requests were one of the reasons to work on this thesis.

3.1.5 Other Tools from the OpenSCAP Ecosystem

The OpenSCAP project also provides other tools that improve the experience or add new possibilities.

First of them, `oscap-docker`, can scan Linux containers and container images. The scan is performed from outside of the container, it mounts the container system as read-only. Therefore it isn’t required to install anything in the container and its filesystem remains untouched. An utility to scan virtual machines, `oscap-vm`, works in a similar way.

A small wrapper script `oscap-ssh` can scan remote machines over network using SSH. Unfortunately it requires OpenSCAP installed on the target machine, because OpenSCAP does not have agent-less scanning.

A progressive tool is OpenSCAP Daemon, a system service that runs in background and continuously verifies the state of the system. It also provides an easy and interactive command line interface.

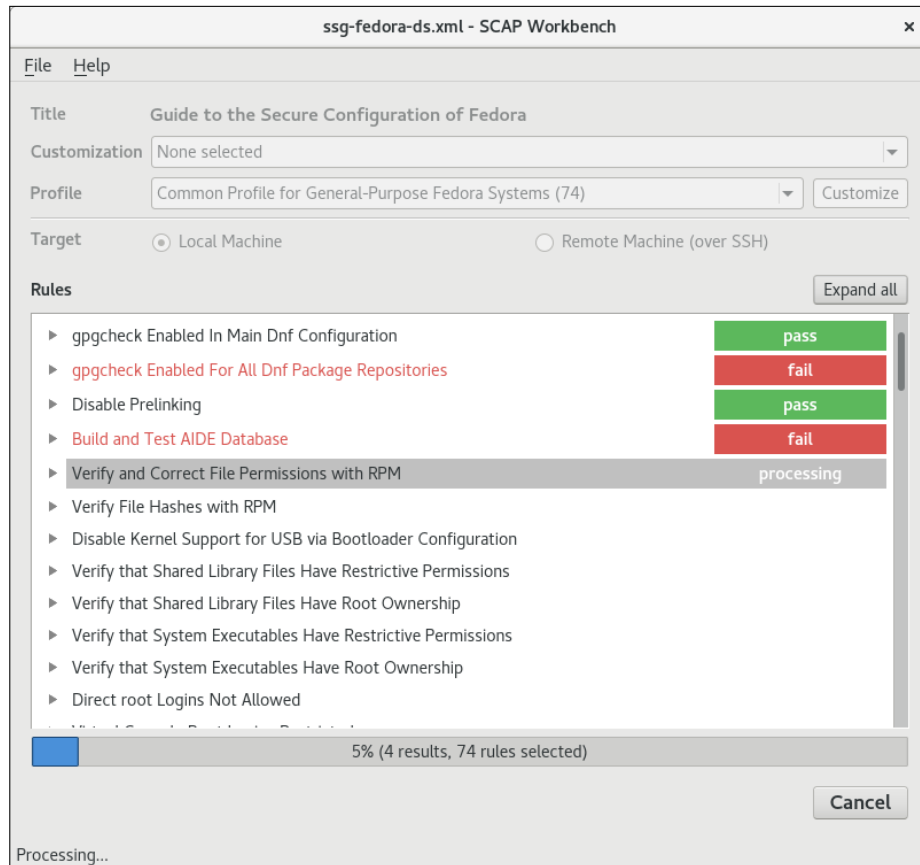


Figure 3.1: SCAP Workbench 1.1.6 running on Fedora.

The Red Hat Enterprise Linux installer, Anaconda, contains a plug-in, called OSCAP Anaconda Addon [29], which can install the operating system in a way it is compliant to the given policy from the very first boot.

Another part of the ecosystem is Libswid. Libswid is a new library that allows interacting with Software Identification (SWID) tags. SWID is a new component introduced in SCAP 1.3 standard.

OpenSCAP is used in Project Atomic [1], a container management tool, to scan containers and container images. OpenSCAP is integrated in Red Hat Satellite [8] and ManageIQ, tools for systems management.

3.2 Typical OpenSCAP Use Cases

The most frequent use case is to scan in order to verify whether the given systems complies with a selected security policy. In the following example we will scan RHEL 7 machine against DISA STIG using SCAP Security Guide.

```
# oscap xccdf eval \
--profile xccdf_org.ssgproject.content_profile_stig-rhel7-disa \
--results-arf results.xml --oval-results --report report.html \
/usr/share/xml/scap/ssg/content/ssg-rhel7-ds.xml
```

The progress of scanning can be seen in the terminal, and a report is generated to `report.html`. The HTML report contains an overview of the evaluated rules. As can be seen on Figure 3.2, for each rule HTML report contains detailed information about the rule and why it passes or fails.

The screenshot shows a detailed view of a failed rule evaluation. The rule ID is `xccdf_org.ssgproject.content_rule_set_firewalld_default_zone`. The result is **fail**. The time of the scan is `2018-05-14T17:53:14` and the severity is `medium`. The description states that the default zone should be set to `drop` in `/etc/firewalld/firewalld.conf`. The rationale explains that in `firewalld`, the default zone is applied only after all applicable rules are examined, and setting it to `drop` is a proper design for a firewall.

The OVAL details section shows a failed check: `Check /etc/firewalld/firewalld.conf DefaultZone for drop` failed because the items were missing. The object is `oval:ssg-obj_firewalld_input_drop:obj:1` of type `textfilecontent54_object`. The following table summarizes the details of the failed check:

Filepath	Pattern	Instance
<code>/etc/firewalld/firewalld.conf</code>	<code>^DefaultZone=drop\$</code>	1

Figure 3.2: A result of a single rule evaluation in OpenSCAP HTML report.

Machine readable rules are stored in ARF format in `results.xml`. We can use this ARF for example to fix the system to put it in line with selected security policy.

```
# oscap xccdf remediate results.xml
```

This command runs a remediation script for rules that were evaluated as fail during the scan. If we run the scan in a same way as in previous example again, all the rules should be evaluated as pass. In practice, this might not be completely true, because some rules do not contain any remediation script. The reason of omitting the script might be that the script is not possible to be written in a generic way. For example, a rule that requires remote logging to be activated, it is necessary to configure a specific IP adress of the remote log server, which obviously vary across organizations.

Before we run the `remediate` command it is recommended to check the scripts that are going to be run. We can extract them from the ARF results file.

```
# oscap xccdf generate fix --fix-type bash --output fix.sh results.xml
```

Those were the 3 most frequent use cases on a command line. Users can do the same tasks using SCAP Workbench GUI application.

3.3 OpenSCAP Alternatives

There are more products that implement SCAP or its subset. However, most of them are not open source, majority of them are commercial software.

- Space and Naval Warfare Systems Command (SPAWAR) has developed SCAP Compliance Checker (SCC) [19]. SPAWAR SCC can only be downloaded by employees of U.S. government or its agencies, it is not available to the general public. It was used to evaluate SCAP Security Guide by some users before OpenSCAP was certified.
- Joval [18] is a multi-platform configuration compliance scanner, implemented in Java. It provides an agentless sensor. The problem of Joval is that it supports only the latest version of OVAL.
- IBM BigFix Compliance [17] is a SCAP scanner by IBM.

We can clearly see that OpenSCAP is comparable with other products that implement SCAP. The disadvantage of OpenSCAP is enormous complexity of the project from both the user experience and inside implementation points of views.

3.4 OpenSCAP Internal Implementation

As we have already explained in Subsection 3.1.1, most of the code creates a shared library. The shared library is divided into modules that conform to logical structure of the SCAP specification. There are modules for XCCDF, OVAL, etc.

The implementation of scanning follows algorithms specified in SCAP specification.

Retrieving data from the system is done using probes. Probes, in terms of OpenSCAP, are small separated executable binaries. Although they run as separated processes, probes are started and terminated by the library code and communicate with the library using IPC mechanisms. Probes are strongly related to the OVAL language. Each probe implements a single OVAL object according to OVAL Definitions specification. They produce OVAL items according to OVAL System Characteristics specification. That means each probe takes care of a single type of object. Most used probes on Linux are `file` probe, `textfilecontent54` probe, `rpminfo` probe and `systemdunitdependency` probe.

As we can see in Figure 3.3, the probes scan the operating system and applications and they send their findings to the OpenSCAP shared library. The OpenSCAP shared library used from SCAP Workbench, or `oscap` tool, or via Python bindings. The `oscap` tool can be used either directly or via wrapper utilities, e.g. the `oscap-docker` utility. As an input, they use SCAP Security Guide or any other SCAP content resource.

The probes were designed as a separated executable programs because there was a requirement to cover each probe by a SELinux policy. Later, it turned out that writing a SELinux policy for probes would be very complicated, so the requirement for separate

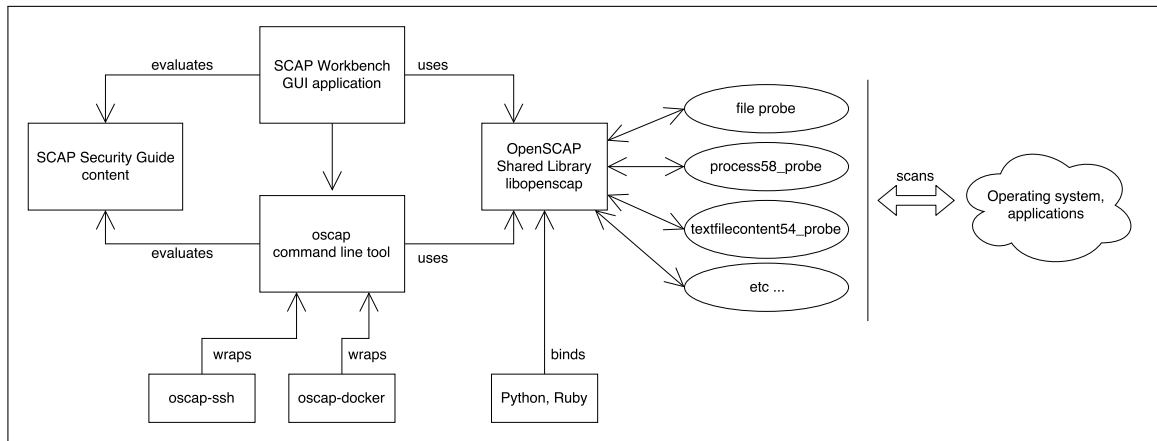


Figure 3.3: OpenSCAP structure.

processes was dropped. There is no need to run probes as separated processes anymore. Nevertheless, due to separation of probes there is a complicated mechanism for communication between probes and library. Data are serialized using a custom implementation of Lisp serial expressions, which is called SEXP, are send through a socket and then decoded again. The communication protocol is not documented anywhere.

In my opinion, the solution is over-engineered. For example, there are 4 distinct ways of communication and both asynchronous and synchronous commands. Therefore this code is big, hard to understand and debug, but that would not be necessary at all if probes were not designed as separate processes.

All probes use a common interface, but each probe is implemented in a different way. General rule is that they are forbidden to call external processes. For example, it is not possible to get information about network interfaces by calling a network tool (such as `ip`) and parsing its output. Instead the data are retrieved by using system calls, API of libraries and very often reading from files or directories.

OpenSCAP library, probes and `oscap` tool are written in C language. The language choice is now seen very inappropriate in regards of the implemented features. A lot of code is dealing with XML parsing and building, large data structures processing, searching or regular expression matching. Some programming languages provide large facilities that would be very useful for working on these tasks. Instead, a lot of code in OpenSCAP is written from scratch. Also, using object oriented programming could be useful. From OpenSCAP code review we can see it often tries to mimic object design. Due to C language the development is slower, more complicated and more error prone. On the other hand, the code might be faster sometimes and the libraries have smaller footprint.

The HTML outputs are generated from XML results (e.g. XCCDF result documents) afterwards by eXtensible Stylesheet Language Transformations (XSLT). That causes some information can never be displayed in HTML report because they are not allowed in XML results by SCAP specification. On the other hand, this solution allows users to create their custom designs and new formats of reports.

The OpenSCAP repository contains XML schemas (XSD) of all versions of all supported SCAP documents. Users can validate their SCAP documents using OpenSCAP. The validation is also performed implicitly each time an XCCDF or OVAL are processed.

Chapter 4

Extending OpenSCAP to Windows

OpenSCAP is available in most Linux distributions and is widely used to perform security audits of Linux machines. OpenSCAP users and the community of auditors and system administrators often request to add support for Microsoft Windows. They want to scan their Windows machines for security compliance using OpenSCAP in the same way as their Linux machines. The Windows version would help organizations where both Linux and Windows are deployed. OpenSCAP could also be a good alternative to proprietary SCAP scanners for Windows users.

Extending OpenSCAP to Windows will be also beneficial for the project itself. The new features will improve visibility of the project and could attract new users. People looking for a security compliance solution will have more reasons to choose OpenSCAP. With a larger community, bugs and feature requests are more likely to be reported. New contributors might be interested in working on developing OpenSCAP. This work is also an opportunity to refactor the code and to fix outstanding issues.

The goal of this work is to enable SCAP scanning of Microsoft Windows operating systems using OpenSCAP. That means to port the existing parts and implement Windows parts of the SCAP standard, namely the Windows tests defined in the OVAL specification.

In this chapter, we will define requirements on the solution and we will design changes and extensions of the OpenSCAP project necessary to achieve the goal. We will describe the way they have been implemented and which problems had to be solved.

4.1 Requirements

First we need to define features that should be present in the designed solution and identify requirements on the final product.

Extension of an Existing Project The Windows part of OpenSCAP should not be a new separate tool but rather a feature extending the existing program.

The implementation should not re-implement anything that is already done. Instead, it should reuse as much existing code as possible. To be able to process real-world SCAP content, we need an implementation of common OVAL parts, XCCDF and other SCAP standards. It is a very huge task to implement these features from scratch, and all this is already implemented in OpenSCAP and tried by many users. If some part of code cannot be used, it should be refactored and common parts should be shared.

The user interface (GUI, HTML reports and command line interface) needs to be kept the same. People expect them to be used on all platforms in the same way. Therefore, current documentation, website, tutorials and presentations should be applicable to the Windows version as well.

At the same time, it is not preferred to fork OpenSCAP. The changes introduced by this work should be public and should be merged into the upstream repository. Only this way the project can benefit from the work done while preparing this thesis.

The patches should be submitted upstream using GitHub pull requests. They should be reviewed by upstream developers and the feedback should be always reflected. Sharing the work with upstream is much harder than working privately on its own fork, but OpenSCAP already knows the author.

This requirement is set because this thesis is done in cooperation with Red Hat.

Portability OpenSCAP is written in C but it often calls POSIX-only functions or even contains Linux specific code. This does not affect only Linux probes, where it is expected and acceptable, but it is a problem of the whole code base. Certain parts of code have to be changed in a more portable way. Both the shared library and the `oscap` tool should be able to build natively on Windows. Therefore, the code should be made as much platform independent as possible. At the same time it needs to build on various Linux systems, so the build system should automatically configure the project based on the platform. The OpenSCAP shared library API should be the same on Windows as on Linux.

Visual Studio Support There are multiple developer environment options on Windows. For a project that comes from Unix world, we could try using Cygwin or MSYS, which bring Linux environment to Windows. They would make the transition easier at the first sight. But it would cause problems in future, because these options provide only a limited subset of Linux features, the programs are not native, do not provide convenient tools and are not preferred by Windows community.

Rather, native Windows applications are often developed and built using Microsoft Visual Studio. According to Stack Overflow Developer Survey 2017 [33] responded by 64000 developers, 38.8% of developers use Microsoft Visual Studio, which makes it the most popular Integrated Development Environment (IDE). Also, Visual Studio provides a lot of convenient tools, debugger, documentation, etc. Using it as a developer environment should make development easier. A full Visual Studio support will make it easier for new people to start contributing, because it is a tool that most people are familiar with.

At the beginning of this work, it was not possible to configure and build the OpenSCAP code in Visual Studio 2017. We need to enable importing the project into Visual Studio and all the code must be compiled by Visual Studio 2017 compiler without any problems. The procedure to develop on Windows should be straightforward and also documented for future contributors.

Avoiding Software Regression An obvious requirement is to not break existing use cases and avoid introducing regressions. OpenSCAP has a test suite in the upstream repository which will help us. The tests are run by Jenkins, a continuous integration (CI) server. The tests are triggered periodically every week, on every change of upstream code and on every pull request opened on GitHub. However, it will be necessary to verify regression tests to the project before major changes. Later, we might also need to add Windows node to the Jenkins server and create tests to test the new code.

OVAL Object	Windows 7 USGCB	Windows 10 STIG	Windows 7 STIG	Windows 2016 STIG	Total	%
registry	208	175	288	141	812	74.22%
accesstoken	23	21	20	59	123	11.24%
fileeffectiverights53	0	12	0	23	35	3.20%
wmi57	0	8	6	14	28	2.56%
wmi	1	5	4	3	13	1.19%
file	3	3	5	1	12	1.10%
regkeyeffectiverights53	0	0	12	0	12	1.10%
passwordpolicy	6	1	1	1	9	0.82%
lockoutpolicy	5	1	1	2	9	0.82%
sid_sid	0	4	0	3	7	0.64%
sid	4	0	2	0	6	0.55%
user	4	0	2	0	6	0.55%
auditeventpolicysubc...	1	1	1	3	6	0.55%
family	2	1	1	1	5	0.46%
group	0	4	0	0	4	0.37%
variable	2	0	1	0	3	0.27%
user_sid55	0	2	0	1	3	0.27%
wuaupdatesearcher	1	0	0	0	1	0.09%

Table 4.1: Most used OVAL objects on Windows.

Partitioning OVAL Objects As we have discussed in Section 3.4, data from the system are retrieved using probes. Each probe implements a single OVAL object according to the OVAL specification.

The OVAL implementation needs to include not only the Windows specific OVAL objects, but also objects that are defined in OVAL Independent Definitions Schema. OVAL 5.11.1 specification defines 14 types of objects in Independent Definition Schema and 48 types of objects in the Windows Definitions Schema (including deprecated objects). All of them are applicable on Windows.

In real-world policies some OVAL objects are used more often than others. From a data set obtained by merging Windows 7 USGCB [26] and STIGs (Windows 7 STIG version 1.35, Windows 10 STIG version 1.11, and Windows 2016 Server STIG version 1.5) [5], we can get the statistics in Table 4.1. The author has made a script that counts frequency of occurrences of OVAL object types in those SCAP documents. The script can be found on the attached CD.

We can see that from 62 total types only 18 were used and remaining 44 OVAL object types are not used there at all.

The probes can be implemented one by one, because without probe the scan is still possible, only rules that need the probe would be evaluated as “unknown”. Therefore, the design and implementation of probes should start with the most used OVAL object.

It should not be a problem if we implement only a few probes in this thesis. Much more important would be that any contributor will be able to add new probes in future. They should not solve any problems with the build system, the common part, the internal logic,

and other things unrelated to the probe implementation. Omitting the rarely used objects will have lower impact and can be fixed later.

SCAP Scanning OpenSCAP on Windows should be able to process and evaluate SCAP content, namely the real world policies in XCCDF and OVAL formats. It should be able to generate the report and display the results. Most of the key features provided by `oscap` interface have to work as expected.

4.2 Extension of an Existing Project

As we have discussed in Section 4.1, we have a requirement to extend the OpenSCAP and include all the work into the OpenSCAP source code Git repository.

Due to nature of the work that changes existing and mature project, the Windows support will be developed in small incremental steps. The changes will be done in a series of small patches, which are commits in Git terms. Each commit has to have a proper description. To make the code review and the inclusion of the code easier the set of patches will be split into distinct pull requests, depending on the area changed or introduced. Each pull request has to have a proper description, rationale, labels, milestone and particularly it has to be reviewed and accepted by the OpenSCAP developers community.

The OpenSCAP project has 2 branches. The implementation can be based either on master branch or on maint-1.2 branch. The maint-1.2 branch is a branch binary compatible with OpenSCAP 1.2.0 release. It contains only small features and bug fixes. Major changes are not accepted into this branch due to upstream versioning policy [21]. Since we plan to introduce some major changes, the work has to be based on OpenSCAP master branch.

Although it seems we will not change the maint-1.2 branch, it is not true, because some patches might fix bugs that are present in maint-1.2 branch, and so it is required to include them in maint-1.2 first to follow the upstream versioning policy [21].

Even if this workflow means that the two branches will diverge a lot after this work will be finished and it will be more difficult to fix bugs that appear in both branches, stability of the stable branch must be preserved. Moreover, the development should be easier because the versioning policy allows to change more things in master branch, namely to change the application programming interface (API) and application binary interface (ABI), which will be needed. This way is also in line with the requirement to keep all existing use cases.

It is not planned to change command line options or user interface. To work on Windows most likely we will not need to add any new options that would be Windows specific.

4.3 XML Changes

To evaluate Windows SCAP content we need to be able to validate the input files to make sure all input data are valid and can be processed by OpenSCAP correctly and without errors. But the OVAL Windows XML schemes are already present in OpenSCAP, so it is possible to validate the XML documents with Windows specific content. It is possible to validate the Windows content also on Linux, because OpenSCAP uses XSD schemes and XML schematrons imported from the official OVAL website and it does not perform any additional checks.

However, we need to be able to determine if the checks are applicable on Windows. To do that we should add new Common Platform Enumeration (CPE) definitions to OpenSCAP

CPE dictionary. That will enable OpenSCAP to recognize it runs on Windows and evaluate XCCDF rules that are explicitly marked as applicable only to Windows.

The OpenSCAP CPE dictionary defines CPEs for many supported platforms and describes them using OVAL definitions.

4.4 Addressing Portability Issues

OpenSCAP was originally designed and developed only on Linux. Portability of the source code to Windows has not been a priority for the project so far. At the beginning of the work, it was not possible to build it natively on Windows or run it on Windows at all. Although it was possible to build a limited subset of the code using Cygwin or using cross-compilation, the process was very complicated and the resulting program was not able to scan.

4.4.1 Moving to CMake Build System

First problem that we need to solve is to choose a build system that we will use to build OpenSCAP on Windows. We need to be able to process the configure step smoothly. The build system should automatically find dependencies, generate a Visual Studio 2017 project, build and link the code and install the executables.

OpenSCAP uses GNU Automake as a build system now. It is used to build the library, the `oscap` application and other utilities. The build system also provides test harness to run the upstream test suite.

Nevertheless, the Automake build system is not used correctly in OpenSCAP. Currently, it builds everything into static libraries and does much unnecessary linking. During the build of probes it builds some parts multiple times. The result is that the probes, which are supposed to have about 10 kB in size, have grown up to 1.5 MB, because the probe binaries contain a lot of duplicated symbols and duplicate code sections from the shared library. Due to these problems, the build is also quite slow, which is demonstrated in Table 4.2.

Moreover, Visual Studio 2017 does not support Automake. One can install the Cygwin or MSYS system to use Automake on Windows, but Visual Studio 2017 does not support them as well.

OpenSCAP uses Automake from the very beginning. Makefiles are convoluted and do not follow the build system development practices. In current shape it would be difficult to adapt the build system to a different platform. Therefore, we need to investigate possibilities to use a different, multi-platform and widely supported build system.

We have identified CMake build system [10] as a suitable solution of the problem. Its advantages are platform and compiler independent configuration files that are easy to understand. CMake is widely used in large C or C++ projects. It provides a test framework (CTest) and a packaging tool (CPack). It is relatively easy to find the dependencies and set compiler options. CMake generates Unix Makefiles, Microsoft Visual Studio project files, and other development environments. It works with multiple compilers, including MSVC. Moreover, Visual Studio 2017 has direct support for CMake [9], which means it is enough to open a directory containing a CMake project and everything is processed automatically.

Some of the projects in OpenSCAP ecosystem already use CMake. CMake is used as a build system in SCAP Workbench. SCAP Security Guide have been ported from Makefile to CMake as well. The project has a good experience with it. The choice of CMake is logical then.

Git branch	Build system	real [s]	user [s]	sys [s]
maint-1.2	Automake	107.13	97.09	13.09
master	CMake	38.68	32.29	7.46

Table 4.2: Speed of OpenSCAP build using old GNU Automake and new CMake.

```

67/84 Test #68: probes/rpmverify/all.sh ..... Passed 8.02 sec
      Start 69: probes/rpmverifyfile/all.sh
68/84 Test #69: probes/rpmverifyfile/all.sh .....***Failed 7.43 sec
TEST: rpmverifyfile probe test with OVAL 5.11.1
Definition oval:x:def:1: true
Evaluation done.
Failed: expected count: 1, real count: 0, xpath: 'oval_results/results/
system/oval_system_characteristics/system_data/lin-sys:rpmverifyfile_
item/lin-sys:capabilities_differ[text()="not performed"]'
+ rpmverifyfile probe test with OVAL 5.11.1 [ FAIL ]
RESULT: FAILED

```

Listing 4.1: Example of a CTest fail.

However, it will be necessary to change Jenkins configuration when those changes will be implemented, which is potentially risky, because we can introduce a regression in the build project. We need to verify if CMake build system produces all expected artefacts by comparing the build with the previous build system. After we are sure the build system is able to build every target as the old one, we can reconfigure Jenkins.

Therefore, the implementation of OpenSCAP for Windows started by moving the project from GNU Automake to CMake build system. This has been done on Linux.

It is important to mention that the project has many build targets—the shared library, utilities, 40 separate probe executables, SWIG bindings, Doxygen documentation etc.

Some of build dependencies could not be covered by commonly shipped CMake modules, so some new modules were either implemented or obtained from other projects and tailored to find dependencies. For example, it was necessary to provide a module that checks for PCRE library.

Also, the author has provided many CMake options to enable developers adjust the build configuration. The options allow to disable or enable tests, probes, documentation and SWIG bindings. It is also possible to turn off or on building of a specific probe or whole group of probes. The options can be chosen on command line, and CMake also provides an user-friendly GUI to customize the build, which can be seen in Figure 4.1.

Finally, the upstream test suite was converted to CMake, using CTest framework. The CTest framework enables better test fail reporting, allows to select only particular tests, and can be used on Windows as well. Moreover, the CTest testing is faster and its output is better than the output of the old test suite. It is easier to find the failed test in the log, as we can see in Listing 4.1. The move of the test suite to CTest could be used as a starting point to improve the test suite of the OpenSCAP project, which now e.g. lacks unit tests.

The CMake Build System has been implemented successfully¹ and has been merged into upstream master branch. The new build commands were updated in upstream README

¹<https://github.com/OpenSCAP/openscap/pull/890>

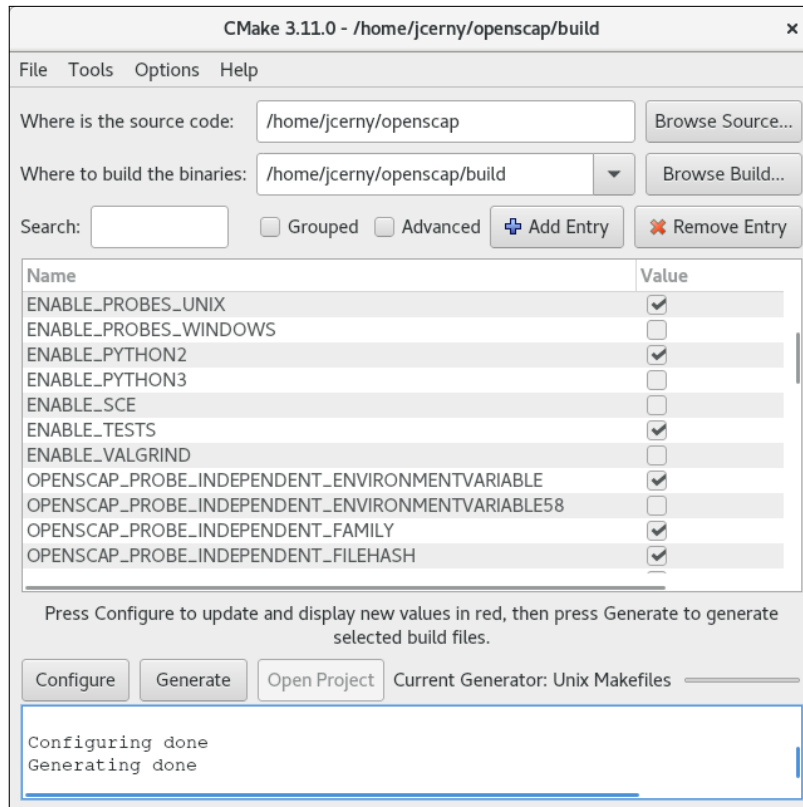


Figure 4.1: Configuring the OpenSCAP master branch using CMake GUI on Fedora 27.

and in OpenSCAP User Manual. After a few small bug fixes² it has started to be used as the preferred build system. Finally, the old GNU Automake system has been removed from the OpenSCAP master branch. It is very likely that OpenSCAP will start to use the CMake build system to build Fedora packages once OpenSCAP version 1.3.0 is released.

The build is with CMake approximately 2-3 times faster than before. 10 measurements have been made on a laptop. Average values can be seen in Table 4.2. Also the build can be easily parallelized.

After the problems of CMake were resolved on Linux, the focus has moved to Windows, and the author started to work on importing the project into Visual Studio 2017.

It turned out that even though Visual Studio 2017 has a direct support for CMake, it has performance problems when using it on such a large project. Visual Studio 2017 was not able to load the project from CMake and always froze.

Fortunately, CMake is able to generate a Visual Studio 2017 Solution which can be easily opened by Visual Studio 2017. Then, it was able to regenerate the build if any string changed in any `CMakeLists.txt`. This additional step is not considered a major problem because Visual Studio versions older than 2017 have not supported CMake before. The steps to generate and use a Visual Studio 2017 Solution have been documented³ in the OpenSCAP User Manual in the Developer Operations Section [23].

²<https://github.com/OpenSCAP/openscap/pull/915>

³<https://github.com/OpenSCAP/openscap/pull/926>

4.4.2 Dependencies

OpenSCAP depends on other open-source libraries, namely libxml2, zlib, curl, pcre, libxslt and libexslt [23]. Fortunately, all the libraries that are required to build OpenSCAP are compatible with Windows.

In past, Windows developers usually downloaded the libraries and compiled them manually or they downloaded compiled binaries. Then, they bundled the libraries into their application. This is not convenient.

There is a new open source tool by Microsoft that helps developers to acquire and build C or C++ libraries, called *Vcpkg* [15].

Vcpkg is similar to package managers known from Linux distributions, but it downloads source code of libraries and compiles them on user's machine. We only need to ensure that the CMake scripts in OpenSCAP can always find libraries obtained by Vcpkg on both platforms.

After we have adopted Vcpkg, we can get all dependencies for OpenSCAP by issuing these commands into the Windows command prompt:

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
.\bootstrap-vcpkg.bat
.\vcpkg install curl libxml2 libxslt bzip2 pcre pthreads
.\vcpkg integrate install
```

Then, we add this option to cmake command:

```
-D CMAKE_TOOLCHAIN_FILE=c:/devel/vcpkg/scripts/buildsystems/vcpkg.cmake
```

These instructions have been added to the OpenSCAP User Manual [23].

4.4.3 Portable C Code

OpenSCAP is written in C. The problem with portability is that different compilers support different C language features, different standard libraries, data types and they provide a different set of functions. The adoption of standards also differs among C compilers.

So far OpenSCAP was compiled only by GCC on Linux and rarely with CLang. It has not been tested regularly with different compilers or on different platforms. The consequence is the code heavily uses GCC extensions and Linux specific features.

Due to the requirement to enable Visual Studio 2017 for our future contributors, we need to be able to compile OpenSCAP on Windows using Microsoft Visual Studio 2017 C Compiler (MSVC). Unfortunately, the MSVC 2017 compiler does not fully support C99 standard. It fully supports only C89.

The most painful point is a missing support for variable length arrays, because they are used frequently in the code base. Variable length arrays are arrays on stack whose size cannot be determined at the compile time. An example is shown in Listing 4.2. It has been necessary to replace variable length arrays by dynamic memory allocation using `malloc`. Then, all this memory has to be freed, so we cannot replace all the occurrences of variable length arrays automatically, but we have to think about their removal.

SEXP parser uses dynamically computed goto labels which are a special feature of the GCC and Clang compiler [22]. It serves to get the address of a label defined in the current function with the unary operator `&&`, as we can understand from Listing 4.3. SEXP parser needs to be rewritten because SEXP is used at many places in OpenSCAP. It is a function

```

int do_something(char *s1, char *s2)
{
    char buffer[strlen(s1) + strlen(s2) + 1];
    ...
}

```

Listing 4.2: C99 Variable Length Array.

```

const void *labels[] = {
    &&L_BRACECLOSE,
    &&L_CHAR,
    &&L_INVALID,
    ...
}
...
L_BRACECLOSE:
    goto labels[i];
...
L_CHAR:
    if (x >= 1)
        goto labels[x - 1];

```

Listing 4.3: Dynamically computed goto labels compilable by GCC.

that has 1049 lines. In order to solve the issue with dynamically computed goto labels, the SEXP parser has been redesigned and changed⁴ to a deterministic finite automaton. The jump labels have been replaced by states of this automaton.

OpenSCAP uses many functions and system calls that are not available in Microsoft C Runtime. Windows uses the Win32 API instead of the usual API found on Linux. Those functions that are not available on Windows have to be replaced by other functions or by a custom implementation. We will try to use functions available on both systems everywhere possible.

Another problem is that OpenSCAP sometimes uses memory alignment. Windows has functions for memory aligned allocations, but memory allocated by these functions needs to be freed by corresponding functions that perform aligned free [14].

The memory alignment is used to achieve a memory saving strategy in SEXP module. The SEXP can have 4 types, which are specified by flags. Normally, there would be a variable that would store these flags. But that variable would take a few bytes from memory, and since SEXP is used extensively in OpenSCAP, that could be megabytes in worst case. Instead, OpenSCAP aligns memory, it makes sure the lowest 4 bits of the pointer to aligned memory block are zeros. Then, it stores the flag to these lowest 4 bits using bitwise operations. Every time before such a pointer is dereferenced, those flags are masked out. The masked 4 bits then serve to determine the type of value stored at the location the pointer points to. That way it saves the memory. The drawback of this solution is that it is less readable, harder to understand, and slow. Also, this is not the biggest memory problem of OpenSCAP.

⁴<https://github.com/OpenSCAP/openscap/pull/979>

The problems with memory alignment have been successfully fixed by freeing the memory correctly.

Another portability issue is that OpenSCAP extensively uses macros with variable amount of arguments, so-called *variadic macros*. Unfortunately, Microsoft Visual Studio 2017 does not fully support them. The problems with *variadic macros* have finally been solved⁵ by removing them or replacing them by conventional code. As a side effect, some of the checks and assertions that were performed only in debug mode will now be performed always, also in the released builds of OpenSCAP.

Sometimes, there is no simple replacement for a Linux function on Windows. An example is `strptime`. To solve this the author has proposed to create a compatibility module in OpenSCAP that will contain fallback code of some functions. This module is named `compat.h`.

The code from the compatibility module will be compiled only if the function was not found by CMake on the system. To avoid reinventing, functions can be copied from other open source projects if their code is licensed under a license compatible with GNU LGPL 2.1. If no suitable implementation can be found it will be needed to implement it from scratch, though.

Smaller differences between the platforms can be resolved using conditional compilation. If CMake detects that some function or library is available on the system, it will add a new macro into `configure.h` and the code will contain two options depending on this macro. Sometimes a problematic function can be found at multiple places across the code. For these cases we have proposed to always create a new function that has two different implementations and put this function to the `common.h` header file. The advantage of this approach is that the actual implementation is only at one place and all function calls are simply replaced.

A big difference is the file path queries, because Windows uses different organization that has no common root directory. We will define separate data structures and functions to handle paths on Windows.

We need to create a multi-platform shared library [32], which means to build a Dynamic Linked Library (DLL) on Windows. With GCC, all symbols are automatically exported. The private headers are marked using a *pragma*. On Windows or in Visual Studio 2017 we have to mark every symbol that is going to be exported explicitly. Every symbol is private by default. CMake has a variable that can make every symbol in a library public, but that would expose also the symbols from private header files. We have a requirement to have the API on Windows same as on Linux, which we have discussed in Section 4.1.

The API symbols have been all marked by the `OSCAP_API` macro⁶, which is defined based on used compiler. This is a common approach [32]. In MSVC, if a shared library is compiled, it means the symbol is exported to the library. In GCC, it means that the symbol changes its visibility. As a result, we could remove the GCC visibility pushes and pops, because they were replaced by this macro. Now, we can see from every function declaration if the function is a member of public API, we do not have to search for pushes and pops.

Originally, the API conversion has been made automatically, but incorrectly, because it put the macro to some places where it should not put it, namely in other macros or in comments. Therefore, it is not a good idea to make this type of changes blindly automatically.

⁵<https://github.com/OpenSCAP/openscap/pull/984>

⁶<https://github.com/OpenSCAP/openscap/pull/934>

To make the transition easier, we first have used cross compilation on Fedora using MinGW. MinGW, a contraction of “Minimalist GNU for Windows”, is a minimalist development environment for native Microsoft Windows applications [6]. All the probes have been disabled and we started with the core part, that should be multi-platform.

There were problems with functions that are not available in MinGW runtime, which means they are usually not available in Windows as well. For example `stpcpy()`, `waitpid()`, `realpath()`. We have either used different, more common, functions or used Windows API functions. Sometimes, a GNU library (gnulib) has been downloaded and adopted to the compatibility module.

A wrapper function `oscap_dirname` has been introduced. This function wraps `dirname` on Linux and `splitpath_s` on Windows. Additional logic has been added on Windows to align the behaviour with its POSIX counterpart. For example, the returned string should never end with a slash or a backslash.

Similar solution has been done in `stpcpy`, `vasnprintf` and `basename`⁷.

After the compatibility fixes have been done⁸ to enable a successful cross-compilation, we have moved to Visual Studio 2017 support. This was more difficult, because this is a different platform and a different, more restrictive compiler.

After some minor changes needed in the CMake build system, the code have been able to be imported into Visual Studio 2017.

However most of the work included making the code building. The variable length arrays have been completely replaced and removed⁹. Complicated variadic macros have been completely replaced and removed as well¹⁰. These changes are very gigantic patches. Also, small amendments to the `CMakeLists.txt` files have been made—e.g. checking for some libraries that were available only on Linux or fixing quotes. Then, the probe subsystem has been added to the build. That required other set of changes, including the aforementioned SEXP parser rewriting.

Finally, we have been able to build OpenSCAP on Windows using Visual Studio 2017, but without the OVAL part, which means it has not been useful at all. To make the OVAL part work, we need more fundamental changes, which will be described in Section 4.5.

4.5 OpenSCAP Architecture Changes

As we have mentioned in Section 3.4, data from the scanned system are retrieved by probes, which are separate processes. Probes have to communicate with the shared library.

But probes communicate with the core library using a pair of `AF_UNIX` sockets which is created by `socketpair` function. The `socketpair` is POSIX function and is not available on Windows. And `AF_UNIX` sockets will be a new feature in Windows 10 [4]. Since we need to support other versions, we need to find a different way of using probes and communicating with them.

The easiest way would probably be to replace the `AF_UNIX` socket by a pipe or a named pipe. Windows API provides functions for this (e.g. `CreatePipe`) and they could be used, but they cannot check the state of processes they communicate with. Handling signals and checking the probes process statuses would have to be done in a different way.

⁷<https://github.com/OpenSCAP/openscap/pull/932>

⁸<https://github.com/OpenSCAP/openscap/pull/925>

⁹<https://github.com/OpenSCAP/openscap/pull/931>

¹⁰<https://github.com/OpenSCAP/openscap/pull/984>

Another option is to rework the communication with probes using a cross-platform IPC library, for example Apache Portable Runtime (APR). APR is available both on Linux and Windows, and can be obtained by Vcpkg tool for the Windows build as well. We could also find many other cross-platform IPC libraries. However, the drawback is that it introduces a new dependency also on Linux. It is not convenient in Linux distributions if a footprint of a package is growing.

The third option is to try to not use separate processes at all, because we have mentioned that the mechanism of separate processes is not needed anymore. It should be possible to change the architecture and include the probe code into the main process. This way we would avoid inter-process communication completely. The application without IPC is less error prone and easier to debug.

This approach also enables us to get rid of SEXP and SEAP in the future. However, the code contains useful type checks and data conversions. It also supports evaluating OVAL variables, OVAL sets, OVAL filters and other advanced OVAL features. Therefore it is not easy to remove the SEXP and SEAP completely, because we would have to do a completely new implementation of all these OVAL features. Also, the current implementation is not designed to be able to extract these from the code. Due to aforementioned reasons, we are not going to remove SEXP and SEAP now.

We can do the change for all probes, including the Linux probes, since this is handled by a common code shared by every probe. All the probes share a lot of common code. This code implements communication with the base library, handling SEXP, input and output caches, initialization and cleanup. The actual probe code that is different is smaller in comparison with the common code.

As we have mentioned in previous section, communication between `oscap` process and probe processes is encapsulated in SEAP protocol. SEAP protocol already has multiple implementations in OpenSCAP source code. To solve the transition, we can add another implementation of SEAP, that would not use pipes or socket operations, but rather would be build around a shared queue.

The entry point `main` function is same for all the probes, and does not consume any parameters. The main function of the probe process can easily be reused as a thread function spawned from parent `oscap` process. OpenSCAP already uses threads for handling probe inputs and caches, so it would not introduce a new dependency.

Nowadays, OpenSCAP uses `AF_UNIX` socket to send data between processes. When we convert the processes into threads, they could use a different way of communication. Threads can share a common variable. If this variable would be a structure containing 2 queues, first queue could be a queue of messages from the core to the probe, the second one would be used for the opposite direction.

The parent thread needs to have a way to inform the probe thread that it added some data into the queue and vice versa. The two threads need to be synchronized. We can use two conditions, each for one queue, signalling that the queue is not empty. The reader blocks until the writer puts some data into the queue. This will simulate the current implementation.

To be able to convert the probes to threads, we need to rename the probe functions, because now all the probes have a function called `probe_main`. We also need a table that maps OVAL object types to probe functions.

The biggest change in terms of program logic was the designed shift from multiprocess architecture to a single-process architecture. We have developed this on Linux, because there are tests from the upstream test suite and it is easier to verify.

We have started by implementing a simple queue. Then, this queue was used as a base for the new SEAP scheme, which was called simply a `sch_queue`.

Meanwhile, processes have been converted to threads. The `fork`, `execve`, `signal` calls have been removed.

A new table that maps OVAL object to probe functions has been added. This table replaced the old table that is not relevant any more. This table contains an OVAL test the probe implements, a pointer to probe initialization function, a pointer to probe main function, a pointer to probe clean up function and a pointer to the probe offline mode function. The functions from this table have been used as callbacks in the shared probe code.

The conversion of probes into the main process involved renaming all the probe functions because they conflicted. Also, a new header file has been added for each probe C file, because they had no header files. We had to rename the C files, because names like `file.h` or `password.h` can clash with header files from other projects or system libraries. A suffix to each header file and C file in the same format has been added.

During these changes, some problems have occurred.

One of the problems was that the probes used global variables to pass some parameters to their local functions. This was very often used in RPM and file probes. When OpenSCAP evaluates an XCCDF that references multiple OVAL files, multiple instances of the same probe can run at the same time. That was not a problem when probes were separate processes, but threads share their global variables. Using global variables will lead to crashes when multiple threads will run at the same time, namely double frees or data inconsistency. Therefore, we have decided to get rid of global variables, and pass all the data as function parameters.

This change also involved adapting CMake build system. The probe targets were removed and their sources were added as new modules of the `libopenscap` shared library. That means the build system produces much less artefacts than before.

The original implementation build only the probes for which all the dependencies (libraries, header files) were installed, because not all dependencies are available on every operating system. We need to not even keep this behavior, but also improve it, because a large amount of probes will not be built on Windows. OpenSCAP dynamically determined which probe binaries are build after its start. When the compiled probes are no longer separated binary files, this has to be reworked. Dependencies have to be determined at the compilation time. We have used the variables produced by CMake together with C preprocessor directives to use only the probes that were build.

After the socket has been removed, it was no longer needed to convert data from SEXP to a string and then back from string to SEXP. Instead, it is possible to save a pointer to SEXP structure to queue directly. This removed time consuming SEXP parsing and serialization, which was now an unnecessary overhead.

Also, the generic SEAP scheme has been replaced by the specific SEAP queue scheme. Then, all the other implementations of the SEAP protocol have been removed. That allows further refactoring.

The author expected performance improvements, because we removed a lot of overhead, mainly we have removed inter-process communication and serializations, which are time intensive algorithms. However, we have not seen any speed up. The author suggests performing audit of existing threads and usage of synchronization primitives, because it might turn out that mutexes and barriers were used at places where they are not needed anymore.

The author at least discovered an existing performance problem caused by unneeded extensive usage of `getenv` function when compiled with debugging symbols and reported it to the upstream¹¹.

After this change¹², OpenSCAP still works in all its use cases and the change did not introduce any major regression.

4.5.1 Offline Mode Changes

OpenSCAP probes can run in the so-called offline mode. The offline mode means that the probe scans a different (guest) filesystem mounted in a host filesystem. This is used to scan virtual machine images, Linux container images and containers without any need to install OpenSCAP inside them. Users can easily scan virtual machines or containers from the host, if they mount a container or an image in some directory on their host. OpenSCAP provides convenient utilities like `oscap-docker` or `oscap-vm` that leverage offline mode.

The offline mode can be implemented either using `chroot` call or by consistently adding a prefix to each file path that is manipulated by the probe.

Adding a prefix to paths is preferred solution over `chroot`, because on Linux user have to be granted `CAP_SYS_CHROOT` capability, which is not usually granted to normal users.

The first option is very easy to implement and was used in all probes in past. However, users have to have `CAP_SYS_CHROOT` capability granted to be able to call `chroot`. It leads also to other problems like loading libraries from the mounted guest system. Limitation of this approach is that some probes cannot work in offline mode, because they either depend on library calls that do not support it, or their read from runtime environment which is not present in mounted filesystem.

To access the virtual machine or container, their filesystem is mounted in a directory on the host machine, usually in `/tmp`. Then OpenSCAP switches to offline mode. Offline mode means that probes operate on the mounted filesystem, not on the host filesystem.

The second option is more complicated, but avoids the problems of the former, so it is a preferred way. Some of the probes have already been migrated to this option in past. But during work on this thesis, the author has discovered a serious bug¹³ that the probes mixes data from the host and guest operating system. An example of this bug was `textfilecontent54` probe.

The offline mode is widely used by OpenSCAP users. After changes implemented in Section 4.5 we have to make sure offline mode still works. We wanted to move to the second option, which was successfully done by the author for many probes. This included fixing the aforementioned bug.

However, at the same time, we have discovered it is not possible to not use `chroot` in RPM probes, because there is a bug¹⁴ in RPM. Therefore the author has implemented a mechanism to return back to original probe directory after the probe finishes the scan. This is a temporary solution until the RPM bug is fixed.

The offline mode will not be supported on Windows at the time of writing this text. Nevertheless, the fix of offline mode was necessary to get the architecture change described in Section 4.5 accepted and merged by upstream.

¹¹<https://github.com/OpenSCAP/openscap/issues/995>

¹²<https://github.com/OpenSCAP/openscap/pull/981>

¹³<https://github.com/OpenSCAP/openscap/issues/1001>

¹⁴https://bugzilla.redhat.com/show_bug.cgi?id=1566985

```
Administrator: Command Prompt
C:\>oscap info c:\devel\scap_gov.nist_USGCB-Windows-7.xml
Document type: Source Data Stream
Imported: 2018-03-06T07:20:38

Stream: scap_gov.nist_datastream_USGCB-Windows-7-2.0.5.1.zip
Generated: 2015-04-07T10:00:00
Version: 1.2
Checklists:
  Ref-Id: scap_gov.nist_cref_USGCB-Windows-7-2.0.5.1-xccdf.xml
  Status: accepted
  Generated: 2015-04-07
  Resolved: false
  Profiles:
    Title: United States Government Configuration Baseline 2.0.5.1
    Id: xccdf_gov.nist_profile_united_states_government_configuration
baseline_version_2.0.5.1
  Referenced check files:
    USGCB-Windows-7-2.0.5.1-oval.xml
      system: http://oval.mitre.org/XMLSchema/oval-definitions-5
    USGCB-Windows-7-2.0.5.1-OCIL.xml
      system: http://scap.nist.gov/schema/ocil/2
    USGCB-Windows-7-2.0.5.1-patches.xml
      system: http://oval.mitre.org/XMLSchema/oval-definitions-5
Checks:
  Ref-Id: scap_gov.nist_cref_USGCB-Windows-7-2.0.5.1-OCIL.xml
  Ref-Id: scap_gov.nist_cref_USGCB-Windows-7-2.0.5.1-oval.xml
  Ref-Id: scap_gov.nist_cref_USGCB-Windows-7-2.0.5.1-patches.xml
  Ref-Id: scap_gov.nist_cref_USGCB-Windows-7-2.0.5.1-cpe-oval.xml
Dictionaries:
  Ref-Id: scap_gov.nist_cref_USGCB-Windows-7-2.0.5.1-cpe-dictionary.xml
C:\>
```

Figure 4.2: OpenSCAP `oscap info` running on Windows 10.

At this point, we are able to successfully build OpenSCAP in Visual Studio, and run simple tasks, as we can see in Figure 4.2. However, no Windows probe is implemented yet. That means no data can be retrieved from the system and therefore the results of all XCCDF rules are “unknown” at this point. Their design and implementation will be discussed in Section 4.6.

4.6 Retrieving Windows Operating System Properties

System objects and their properties in SCAP are described by the OVAL language. The OVAL language specifies two groups of elements that we have to deal with in this work.

First group is group of independent objects, defined by OVAL Independent Definitions Schema, and corresponding independent items, defined by OVAL Independent System Characteristics Schema. Although they have the word independent in their name, they are implemented in OpenSCAP using POSIX functions.

Second group is the Windows objects and Windows items defined by OVAL Windows Definitions Schema and OVAL Windows System Characteristic schema. We need to create new probes that will implement these objects. We will most likely use the basic Windows API calls. Windows API provides a large amount of functions. The concept of configuration files is not so often used on Windows, so we need to query the system registry. According to Table 4.1, the `registry` probe is the most used and therefore it is preferred to be implemented first.

Another problem is that the Windows API uses different conventions. The API documentation uses a specific coding style, which is completely unlike from the Linux Kernel coding style used in OpenSCAP. When adding patches with Windows API calls we need to be careful to not break conventions used in OpenSCAP project.

OpenSCAP needs to work correctly in localized systems. That is a problem because Windows API functions usually work with wide character strings (UTF-16), but whole OpenSCAP code uses standard C strings, or multibyte strings in UTF-8 optionally. The OpenSCAP API does not provide functions to work with wide character strings. To process the data correctly, the strings have to be converted to wide character strings just before passing them into any Windows API function. Similarly, the wide character strings returned by Windows API functions have to be converted immediately to standard C strings [31]. To solve this, the author has added functions that easily convert the strings and should be used everywhere in OpenSCAP source code.

In case of errors we also need to get the error codes of the Windows API calls and corresponding error messages. We would like to inform the user about the problem. Windows has `GetLastError` and `FormatMessage`. They are difficult to use, so a wrapper function has been created, has been put into the common module in OpenSCAP and it is going to be used across all the code that cooperates with Windows API. The error messages need to be converted to C strings and then used in various places in OpenSCAP, namely verbose mode for informational and debug messages, OVAL message elements and of course standard error output in case of serious errors.

4.6.1 Adding a New Probe in OpenSCAP

We will describe the process of developing and adding a new probe into OpenSCAP. We will assume we develop a probe in master Git branch after the architecture changes described in Section 4.5 have been done.

Each probe implements an OVAL test, its main input is an OVAL object and its output is an OVAL item. We have to read the OVAL specification carefully, because the probe shall strictly implement the OVAL specification and should not produce invalid results. First, we need to understand the test, object, and state specification in OVAL Definition Schema to realize what inputs we can have. Then, we need to study OVAL System Characteristics Schema to understand which data should be collected from the system and what they mean.

After we have studied the OVAL specification, we need to find a suitable API or a good library that can give us these data. Sometimes an option is to read data from some file instead. On Linux, we can probably find a file where we can read the data from, because there are many useful files in `/proc`, but on Windows this is less likely. It is forbidden to call external commands or execute any other processes. The probe cannot be implemented in a way that it calls some utility and then parses its output. This is a very important concept of OVAL language that ensures the scan is always read only.

When we have found a suitable API we have to think about the algorithm and its implementation. This is specific to each probe. As a general advice, we try to do everything in low complexity, because the probes usually collect large data.

Important task is to prepare the infrastructure of the probe. Each probe has some overhead that allows CMake to build the probe and OpenSCAP to communicate with the probe and list it in various outputs, e.g. `oscap -V`.

```

cmake_dependent_option(OPENSAP_PROBE_LINUX_RPMVERIFYPACKAGE
  "Linux rpmverifypackage probe"
  ON "ENABLE_PROBES_LINUX; RPM_FOUND" OFF)
cmake_dependent_option(OPENSAP_PROBE_LINUX_SELINUXBOOLEAN
  "Linux selinuxboolean probe"
  ON "ENABLE_PROBES_LINUX; SELINUX_FOUND" OFF)

```

Listing 4.4: CMake Dependent Options in OpenSCAP.

First, we add CMake logic for libraries that the probe requires, if they are new dependencies. Sometimes, this includes writing a new CMake module or looking for a suitable module on the Internet.

Second, we add a CMake dependent option to root `CMakeLists.txt` that lets developer to enable or disable building the probe. The dependent option is a special construction that differs from a normal CMake option that it is a macro to provide an option dependent on other options. This macro presents an option to the user and allows him to set the option to true only if a set of other conditions are true. In our case, the option should be dependent on finding all the libraries and on the operating system. This way we will easily avoid building Windows code on Linux and vice versa. The dependent option is presented in Listing 4.4. We need to add the same variable also to `config.h.in` in order to access this option from the C code. Each C file in OpenSCAP includes `config.h`, which is generated by CMake from `config.h.in`.

Third, we create a header file and C file in a subdirectory under `src/OVAL/probes`, depending on the schema the probe belongs to, e.g. `windows` or `unix`. We add both files to the `CMakeLists.txt` in that directory. That must be guarded by the probe option.

The developer can be inspired by existing probes. The C file must contain at least one function with `main` as a suffix of its name. This function must take a probe context and a void argument as parameters.

Then, we add the probe functions as a new entry in the OVAL probe table, which is located in `src/OVAL/probes/probe-table.c`. In this table, we map the OVAL type to probe functions. Therefore, we need to include the probe header file in this file as well. The probe functions could be: initialization, clean-up, main function and offline mode function. The functions will be used as callbacks in common probe code. It is not needed to implement all of them, in fact that would be rare. If some of them is not implemented, we should insert a `NULL` to the probe table in place of callbacks instead of defining some dummy functions in the probe.

Now we can start programming the probe logic itself. It is really inconvenient that the probe has to work with SEXPs and serialize the data to and from this format. The input data are obtained from the probe context.

To debug, it can be useful to use verbose mode and add messages of debug, informational, warning, and error level. If these messages provide valuable information, we recommend to keep them in the code, so that users can run `oscap` in verbose mode. Users can also include verbose output to the bug report if they encounter an issue. It will greatly help engineers that investigate and fix this bug if the verbose messages are useful.

Thanks to changes described in Section 4.5, debugging of the probe is now easier, particularly because the probes are now in the same process. In the old architecture, developers had to capture the messages sent through the socket and inject them into the

probe standard input, which was complicated and annoying. Now, they can set a breakpoint in the probe and run `oscap` in a debugger. It is very easy in Microsoft Visual Studio 2017 as well.

We highly recommended to create example OVAL definitions for testing the probe and submit them into the upstream test suite, under `tests/probes` directory. It would be nice to add unit tests, but at the time of writing this thesis, OpenSCAP does not have any unit tests and does not use any unit test framework.

4.6.2 OpenSCAP System Info Probe

All OVAL Results documents produced by a SCAP scanner shall contain `system_info` element. This element provides basic information about the evaluated system. The OVAL specification requires this element to contain the operating system name, its version, processor architecture, primary host name of the machine and IP and physical (MAC) addresses of all network interfaces of the machine.

For this purpose, OpenSCAP has a special probe, which we call `system_info` probe. We had to implement a Windows version of this probe prior to implementation of other probes, because it must be run during every system scan.

The OVAL specification [34] is very vague and does not specify what data should be exactly retrieved. We can use intensively all the data obtained by the API calls. The author has decided to include both IPv4 and IPv6 addresses including the network mask. According to specification, the `interface` element can only have one `ip_address` child element. That means if some interface has both IPv4 and IPv6 addresses or more addresses, there will be multiple `interface` elements in the output. This is consistent with the output that OpenSCAP generates on Linux.

```
<system_info>
  <os_name>Windows</os_name>
  <os_version>10.0.16299</os_version>
  <architecture>x64</architecture>
  <primary_host_name>DESKTOP-VIRTUAL</primary_host_name>
  <interfaces>
    <interface>
      <interface_name>Ethernet Connection 1</interface_name>
      <ip_address>fe80::bcae:e6bc:c42e:312a/64</ip_address>
      <mac_address>08-00-27-34-F1-A1</mac_address>
    </interface>
    ...
  </interfaces>
</system_info>
```

Listing 4.5: OVAL System Info produced by OpenSCAP.

We have used Windows API calls to get these information from the system, and Windows Sockets 2 library to get information about network interfaces. The implementation¹⁵ might look easy, but it had various pitfalls.

First, the obvious function to get Windows version, `GetVersionEx`, does not return the real Windows version. It lies to applications, because this function is meant to be used to check compatibility of an application with the system. On the author's Windows 10 virtual machine, it provided a version corresponding to the first release of Windows 7. Moreover,

¹⁵<https://github.com/OpenSCAP/openscap/pull/1004>

this function is deprecated. Microsoft recommends to use Version Helpers API instead, but that allows the application to check if the operating system is compatible with a specific version, and does not return the exact version string. The Windows version is stored in the Windows registry, but the author wanted to know if there is some other API that could retrieve the version. Fortunately, it is possible to load `ntdll.dll` dynamic library and access symbol `RtlGetVersion` from there. This function gives us the result we need for our purposes. The same approach is used in Glib.

Second, if we decide to use Windows API and conversion functions from Windows Sockets 2 API we get many build conflicts, because `windows.h` includes `winsock.h` which redefines symbols from `winsock2.h`, which we want to use. Fortunately, for the purpose of this probe it has been enough to define `WIN32_LEAN_AND_MEAN` macro and the problem has disappeared.

These API calls have been used: `GetNativeSystemInfo` to get the processor architecture, `GetAdaptersAddresses` to get the network adapters and their IP and MAC addresses and `WSAAddressToStringW` to convert the IP addresses to strings.

As can we see from the Listing 4.5, the contents of the generated `system_info` element is not exciting, but its purpose is to distinguish the reports from the different systems. Even if host names in the organization's network are not set up correctly, we can identify reports from different systems in the report by their IP and MAC addresses.

4.6.3 OVAL Independent Family Test

The OVAL Independent Family Test is used to check the family of the operating system—whether it is Unix, Windows, OS X, etc. The implementation of the probe is extremely trivial. The probe just returns a hard-coded string "Windows" and populates it into the corresponding OVAL item. The purpose of this is to provide base building block for CPE dictionaries. In this work it has been used as a simple input for integration testing of OpenSCAP on Windows, specifically at the time no other probes have been implemented yet.

4.6.4 OVAL Windows Registry Test

The Windows registry is a hierarchical database that stores low-level settings for the Microsoft Windows operating system and for applications that have chosen to use the Windows registry [36]. The database entries are called keys, and one key consist of multiple named values of different data types. The keys form a tree, in fact multiple trees, where each tree root is called a *hive*. The key is a path in this tree.

Many applications, including Windows itself, prefer using the registry over text configuration files. This is a big difference from Linux systems, where text configuration files are traditionally preferred. That also means on Linux, a majority of the OVAL definitions describe contents of text files, but on Windows, most of the OVAL definitions query registry values.

The OVAL Windows registry test is used to check values and metadata associated with Windows registry key. In OVAL Definitions document, the `registry_object` element can be used to identify the registry key or specific values under the key. The object consists of the following elements:

1. `hive`, a root registry key, which is limited to this set of options:
 - (a) `HKEY_CLASSES_ROOT`

- (b) HKEY_CURRENT_CONFIG
 - (c) HKEY_CURRENT_USER
 - (d) HKEY_LOCAL_MACHINE
 - (e) HKEY_USERS
2. **key**, which specifies the key by its path. The hive part should not be included, which is against convention, but that makes sense, because there is a dedicated element for the hive. The key can be specified not only by an exact string, but also by using a regular expression.
 3. **name**, the name of the registry value, which can be also specified as a regular expression.
 4. **behaviors**, an optional element, that can specify if the registry tree should be walked recursively or not, in which direction, and maximal depth of recursion. It also specifies a Windows View, which denotes if the registry should be viewed as a 32 bit application or as a 64 bit application.

The features required by **behaviors** element complicate design and implementation of the probe. We have to implement the probe by an in-order tree search algorithm.

The possibility of regular expressions in the **key** element implies that the OVAL object could match multiple keys, including `.*` which matches all keys on all levels under the given hive. Unfortunately, the author has discovered that OpenSCAP does not have any limits on count of collected items, and could be possibly killed because it runs out of memory. This happens often in Unix File probe, which can check all the files on the system, including mounted remote filesystems. A good solution would be to enforce limits on counted objects, and optimize the memory usage by writing the collected data to the output file continuously. However, this should be solved for all OpenSCAP probes, which is closely related to how OpenSCAP serializes data with libxml2, so this optimization is out of scope of this thesis.

Unfortunately, Windows API does not provide any function for recursive traversals of registry. It also does not have any function that could match registry keys or values by a regular expression. Therefore it is needed to implement these functions in OpenSCAP.

However, in practice, the registry key is usually specified exactly and the key operation is equals. This fact can be leveraged in optimization—if the exact registry key is specified the probe does not have to search whole registry tree, but it can directly open the given registry key. A similar optimization can be used also for operation case insensitive equals, because registry keys are not case sensitive, so it is sufficient to convert the key to lowercase.

OVAL specification also defines 2 special cases. First is that the **key** element is allowed to set `xsi:nil="true"`, which means the root hive should be collected only. This means we can skip recursion, collect only data about the root, and make sure no **key** element appears in collected OVAL item. Second special case is that **name** element is allowed to set `xsi:nil="true"`, which means the object is intended only to check the existence of a particular key. This means the probe has to collect the key, but cannot collect any values under that key. Therefore we need to check for those 2 possibilities each time an item is going to be collected. We need to add a possibility to collect only the registry key without its values, though.

The recursive implementation is quite time consuming, because in worst case it has to search the whole registry including all the values. However, practical use cases restrict the search on a very small subtree of the registry.

```

<win-sys:registry_item id="10229616" status="exists">
  <win-sys:hive>HKEY_LOCAL_MACHINE</win-sys:hive>
  <win-sys:key>
    System\CurrentControlSet\Services\Netlogon\Parameters
  </win-sys:key>
  <win-sys:name>MaximumPasswordAge</win-sys:name>
  <win-sys:last_write_time datatype="int">
    131511664238376178
  </win-sys:last_write_time>
  <win-sys:type>reg_dword</win-sys:type>
  <win-sys:value datatype="int">30</win-sys:value>
  <win-sys:windows_view>64_bit</win-sys:windows_view>
</win-sys:registry_item>

```

Listing 4.6: Snippet from OVAL Results Document produced by OpenSCAP by evaluating an OVAL Windows Registry Object.

The collected data are represented by OVAL `registry_item` element and should contain `hive`, `key`, `name`, `last_write_time`, `type`, `value`, and `windows_view` child elements. All these child elements are optional.

There is a problem with data representation and data type of the registry values. The Windows API returns only a pointer to an untyped buffer. It is up to the application to typecast and understand the data in the buffer correctly. The OVAL language also has data types, which are different from the Windows registry types. Therefore it is needed to implement in OpenSCAP all the conversions to C data structures, and then implement a conversion to internal SEXP representation, so that the collected data elements have a correct datatype according to OVAL specification, and also this type is in line with `type` element of the collected item. Also, we should be careful because some types of strings contain the terminating null character and others do not.

It is not clear from the OVAL specification how to process the `REG_EXPAND_SZ` type, which is a null-terminated string that contains unexpanded references to environment variables. First possibility would be to replace the variables by their values, second possibility would be to collect original unexpanded string. The author has chosen the second option, because it is more clear where the string comes from.

New registry data types (e.g. `REG_RESOURCE_LIST`, `REG_FULL_RESOURCE_DESCRIPTOR`, `REG_RESOURCE_REQUIREMENTS_LIST`) were added in OVAL 5.11.1, but they existed in Windows registry before. It is not defined how a scanner shall process these types if it evaluates content written in older versions of the OVAL language. The author has decided to treat these data types as binary data and print them out in a hexadecimal form, because they do not occur much often.

The implementation of the probe is built on the top of these API calls: `RegEnumValueW`, `RegOpenKeyExW`, `RegQueryInfoKeyW`. We prefer using wide character versions of the calls to fully support localized values, although we have to use conversions everywhere.

During the testing of this probe, a problem with integer data representation has arose. The NIST validation test suite tests if the boundary values of the unsigned 64-bit registry `REG_DWORD` data type are collected correctly. It turned out that the OVAL specification requires the integers in OVAL shall have unlimited size. But OpenSCAP represents integers

```

<accesstoken_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" id="oval:mil.disa.fso.windows:obj:437001" version="3"
comment="Guests">
  <security_principle datatype="string" operation="equals">
    Guests
  </security_principle>
</accesstoken_object>
<accesstoken_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" id="oval:mil.disa.fso.windows:obj:437100" version="3"
comment="Domain Admins">
  <security_principle datatype="string" operation="pattern match">
    ^.*\\Domain Admins$|^Domain Admins$
  </security_principle>
</accesstoken_object>

```

Listing 4.7: Examples of Access Token objects adopted from Windows 10 STIG.

by a signed 64-bit number internally. It is not an easy task to implement integers of unlimited size in C, unless we use GMP library. It also requires large changes across OpenSCAP codebase. Therefore the author decided to not solve this issue, keep the integers as they are and do not support very large numbers. This hopefully will not affect typical use cases.

Finally, the probe has been implemented completely¹⁶ and is able to collect the registry keys and their values, as can be seen in Listing 4.6. We might find the `last_write_time` value suspicious, but the OVAL specification requires the date to be represented as 64 bit unsigned integer.

4.6.5 OVAL Windows Access Token Test

The OVAL `accesstoken_test` is used to check the properties of a Windows access token as well as individual privileges and rights associated with it. The `accesstoken_object` contains only a single security principle that identifies user, group, or computer account that is associated with the token. An example of the `accesstoken_object` is shown in Listing 4.7. The trustee string has format `computer_name\trustee_name` or just `trustee_name` for local accounts.

On the other hand, the corresponding OVAL item (`accesstoken_item`) has 45 child elements. Each element represents rights or privileges which the trustee can get. For example `SeAuditPrivilege` or `SeBackupPrivilege`. The values are boolean.

According to the OVAL specification, the OVAL Access Token Test has been deprecated in OVAL 5.11, will be removed in OVAL 6.0 and has been replaced by User Rights Test. However, in the specification, we can not find any definition of the User Rights Test. And in Table 4.1 we can see that existing SCAP content widely uses it. Also, the SCAP scanners should support every version of OVAL language due to backwards compatibility.

The important concept is a SID, which stands for Security Identifier. A SID is an unique value of variable length used to identify a trustee [12]. We need to convert the value of

¹⁶<https://github.com/OpenSCAP/openscap/pull/1007>

`security_principle` element to a SID. Then, we can use the `LsaEnumerateAccountRights` to get the privileges and rights granted for this SID.

The security principle can be specified by a regular expression, as demonstrated in the second example in Listing 4.7. The regular expression is a problem, because Windows API does not provide any function that accepts a regular expression as a parameter. To perform the match, we first need to get a set of all possible values of the `security_principle` element that this element could gain on the given operating system. Then, we have to choose the items that match this regular expression. Again, there is no function for that. To solve the problem, we have to obtain lists of all existing local users, local groups, global users and global groups and merge those lists together. We also need to generate so-called well-known SIDs, which are special SIDs that are built in each Windows system. We need to add the list of well-known SIDs to the pattern match search as well.

It is clear that the possibility of a regular expression again makes the implementation harder than expected. On the author's Windows 10 machine, which was not connected to a domain, the list contained 114 items. And we have to implement many helper functions to get all the members of this list.

To get the privileges the `LsaEnumerateAccountRights` was used. However, the privileges can be inherited from parent group(s). Because of that, we also need to get names of the parent group and get their privileges and merge them together.

The solution has been implemented¹⁷ and is working properly on Windows 10 where it has been tested. However, the OVAL specification says that the privileges may differ among Windows versions and the documentation for that version of Windows should be consulted for more information. Therefore to implement this probe properly we will have to test it also on older versions of Windows and cover the possible differences in the probe code.

4.6.6 OVAL WMI Test

Windows Management Instrumentation (WMI) is an infrastructure for management of data and operations on Windows operating systems [13]. It can be used for automated administration of remote computers.

The OVAL Windows Definition Schema defines 2 tests that are related to WMI—a `wmi_test` and a `wmi57_test`. The former has been deprecated and replaced by the latter in OVAL 5.7. Because the 5.7 version was released in 2010, we are going to focus on the currently used `wmi57_test`, but the final product should implement both of them, because both can appear in real-world SCAP policies.

The `wmi57_object` has only 2 elements: `namespace` and `wql`. The latter contains a query in the WMI Query Language (WQL) format. WQL is a subset of standard Structured Query Language (ANSI SQL) with minor semantic changes. This language lets users to query specific fields from WMI tables. OVAL specification forbids to query all fields (using `SELECT * from`). We can see two examples of `wmi57_object` in Listing 4.8.

Windows provides an API to interact with WMI via Component Object Model (COM). COM is an object-oriented system for creating binary software components that can interact with each other [11]. It can be used from various languages, including C and C++. OpenSCAP is written in C. Unfortunately, it is easier to use the COM API from C++ than from pure C, because C++ provide programming mechanisms that simplify the implementation of COM objects, which are object-oriented. Their documentation is also in C++, so

¹⁷<https://github.com/OpenSCAP/openscap/pull/1009>


```

<wmi57_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions
-5#windows" id="oval:mil.disa.stig.windows:obj:3501" version="1">
  <namespace datatype="string">root\cimv2</namespace>
  <wql datatype="string" operation="equals">
    SELECT installstate FROM win32_optionalfeature
      WHERE name = 'IIS-HostableWebCore'
  </wql>
</wmi57_object>

<wmi57_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions
-5#windows" id="oval:mil.disa.stig.windows:obj:3800" version="6"
comment="McAfeeFramework State">
  <namespace>root\cimv2</namespace>
  <wql>
    SELECT state FROM Win32_Service WHERE Name="McAfeeFramework"
  </wql>
</wmi57_object>

```

Listing 4.8: Examples of WMI objects adopted from Windows 10 STIG.

```

C++: hr = service->ExecQuery(...);
C: hr = service->lpVtbl->ExecQuery(service, ...);

```

Listing 4.9: Using COM methods from C++ in pure C.

we cannot blindly follow the documentation, but we need to translate classes and methods in a way that is demonstrated in Listing 4.9.

Obtaining the data from WMI is very simple. First, we need to initialize the COM library for use by the calling thread using `CoInitializeEx` and initialize security attributes using `CoInitializeSecurity`.

Second, we create an instance of `IWbemLocator` interface using `CoCreateInstance`. Then, we use the `IWbemLocator` interface to obtain the initial namespace pointer to the `IWbemServices` interface for WMI by calling `ConnectServer` function. The namespace pointer is specified by `namespace` element in the OVAL object.

Finally, we execute the WQL query, defined by the `wql` element of the `wmi57` OVAL object using `ExecQuery`. It returns an enumerable list of `IWbemClassObject` interface, which we then process and we then get the fields specified by the WQL query. To be able to identify the fields from the WQL query string, we could write a small parser on our side, but we can use the `GetAnalysis` function that parses the WQL string. At the end, the obtained data must be properly converted and populated to OVAL items.

This probe should produce OVAL record elements according to the OVAL specification. It turned out that support for OVAL record elements was incomplete in OpenSCAP. Therefore, the remaining part of the OVAL record element has been implemented¹⁸ first.

¹⁸<https://github.com/OpenSCAP/openscap/pull/1018>

Comparison of OVAL records with OVAL states has been implemented as well¹⁹. Finally, the probe for OVAL Windows wmi57 test has been implemented²⁰ successfully.

4.7 Packaging

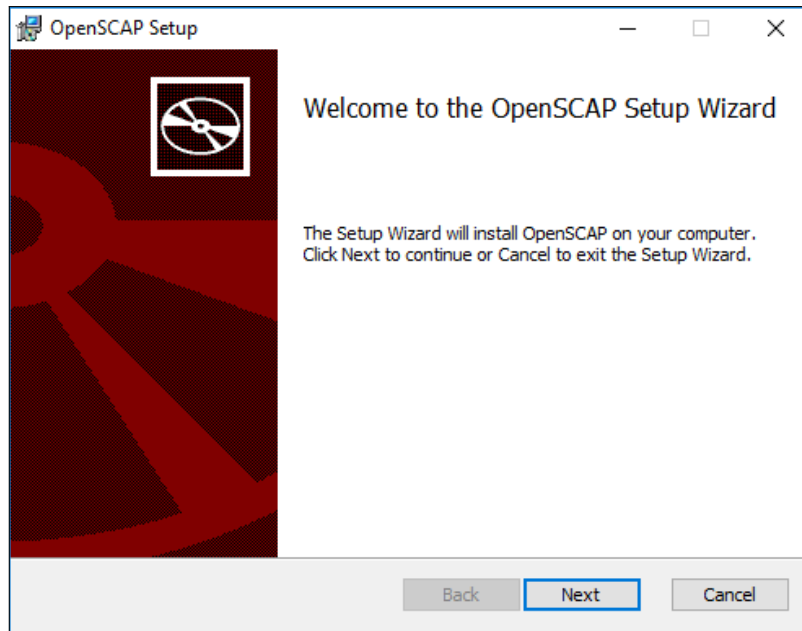


Figure 4.3: Windows Installer.

The Windows applications are usually distributed as executable packages which provide a graphical installer wizard. To make OpenSCAP for Windows easy to install, we have decided to create a Windows Installer.

Making an installer has been easy thanks to CMake and CPack. CPack provides generators to create various types of packages, including DEB, RPM, and namely Windows Installer. There are 2 options in CMake: Creating a standalone executable file (EXE) using Nullsoft Scriptable Installation System (NSIS) or creating MSI packages using WiX toolset (WIX). We have chosen the latter, because it is more customizable.

Since OpenSCAP is a command-line application, it would be useful to add to the system PATH target directory where `oscap.exe` is located, to let the users run `oscap` any time without having to remember and type the full path. Unfortunately, we cannot achieve this by CMake directly, but fortunately we can create a WIX patch. The WIX patch is a simple XML file.

The main problem is to pack all the dependent libraries into the installer. CMake has useful module `BundleUtilities`. This module analyses the installed target binary and finds all the dependent libraries, including indirect recursive dependencies. The recursion is very useful in case of OpenSCAP, because `oscap.exe` depends only on `OPENSAP.DLL`, but `OPENSAP.DLL` depends on many other libraries, which also depend on other libraries, e.g. `curl` depends on `zlib`.

¹⁹<https://github.com/OpenSCAP/openscap/pull/1024>

²⁰<https://github.com/OpenSCAP/openscap/pull/1019>

When BundleUtilities identify all dependent DLLs, they copy all the DLLs from the paths specified in the CMakeLists to the MSI package. We provide only the path to the Vcpkg build directory, where all the libraries are located.

The MSI package is generated automatically when we run `cpack` command in the command line or when we build `PACKAGE` project in Microsoft Visual Studio 2017. The resulting file is about 9 MB large, because it contains all XML schemes. A file with SHA512 hash of the installer is produced as well.

The Installer successfully installs OpenSCAP, including all the dependencies, XSD and XSLT files and adds it to `PATH` for all users. We can see the OpenSCAP Windows Installer running in Figure 4.3.

The changes in CMake have been submitted to upstream²¹. The built installer package has been published on OpenSCAP GitHub in Releases section and a link to download has been added into Download page on the OpenSCAP Portal.

²¹<https://github.com/OpenSCAP/openscap/pull/1020>

Chapter 5

Testing, Verification, and Assessment of the Project

In this chapter, we will demonstrate the results of the project. We will show that OpenSCAP can scan Windows systems and we will discuss results of these scans. Furthermore, we will focus on testing and evaluation. Finally, we will propose future improvements of this work.

Since OpenSCAP is an existing project and follows the specification, we need not to evaluate its existing parts. Rather, we should demonstrate that we successfully provided an extension of the project to enable Windows support, and we are able to scan Windows machines using OpenSCAP, which was the goal of this thesis.

5.1 Scanning Windows Systems Using Real World Security Policies

OpenSCAP evaluates security policies written in SCAP format, which are often referred simply as SCAP content. We will demonstrate that SCAP content describing secure Windows configuration can be consumed and evaluated by OpenSCAP.

A lot of SCAP content for Windows already exist. For example, DISA STIG for Windows 10 [5] or USGCB for Windows 7 [26] can be used. These benchmarks are available to download from the Internet.

The DISA STIG for Windows 10 version 1.35 contains multiple profiles. They define different settings according to classification level of the system in question—mission critical, mission administrative, mission support. Each of them has three subcategories: classified, public and sensitive. We can see OpenSCAP scanning Windows with one of the DISA STIG profiles in Figure 5.1.

The Windows 10 STIG version 1.35 consists of 222 XCCDF Rules. With OpenSCAP, it is possible to evaluate 154 of these rules, which is 69 %. The remaining 68 rules are evaluated as “unknown” because the probes that could evaluate these rules are not implemented. Until those probes are implemented, auditors will have to check these rules manually.

OpenSCAP is able to produce results in SCAP formats and human readable HTML report on Windows, as current users expect. We can see an example of HTML report of a scan of a Windows 10 machine in Figure 5.2.

OpenSCAP is able to evaluate DISA STIG in about 2 minutes on a standard laptop. The slowest part is evaluation of Access token tests, which could be improved by adding

```
Administrator: Command Prompt - oscap xccdf eval --profile MAC-1_Classified --results results.xml --oval-results --report report.h...
C:\>oscap xccdf eval --profile MAC-1_Classified --results results.xml --oval-results --report report.html
c:\devel\U_Windows_10_V1R10_STIG_SCAP_1-2_Benchmark.xml
Title  Domain-joined systems must use Windows 10 Enterprise Edition 64-bit version.
Rule   xccdf_mil.disa.stig_rule_SV-77809r3_rule
Ident  CCI-000366
Result unknown

Title  Users must be prevented from changing installation options.
Rule   xccdf_mil.disa.stig_rule_SV-77811r1_rule
Ident  CCI-001812
Result fail

Title  The Windows Installer Always install with elevated privileges must be disabled.
Rule   xccdf_mil.disa.stig_rule_SV-77815r1_rule
Ident  CCI-001812
Result fail

Title  Users must be notified if a web-based program attempts to install software.
Rule   xccdf_mil.disa.stig_rule_SV-77819r1_rule
Ident  CCI-000366
Result pass

Title  Automatically signing in the last interactive user after a system-initiated restart must be disabled.
Rule   xccdf_mil.disa.stig_rule_SV-77823r1_rule
Ident  CCI-000366
Result fail

Title  The Windows Remote Management (WinRM) client must not use Basic authentication.
```

Figure 5.1: OpenSCAP evaluates a Windows 10 machine against Mission Critical Classified profile from DISA STIG.

a global cache for the trustees SIDs, to prevent collecting them repeatedly. If we remove these rules, the evaluation is shortened to approximately 10 seconds.

The detailed results can be seen after clicking on the rule title in HTML report. It will show us OVAL details. We can see an example of a rule result detail in Figure 5.4. The report has the same format as on Linux. It contains the IP and MAC addresses to distinguish reports from different systems. Example reports from various machines can be found on the attached CD.

A similar case is USGCB. To show you that we have not tested only Windows 10, but also older Windows 7, we have successfully installed OpenSCAP on Windows 7 SP1 and we have run USGCB scan. OpenSCAP running on Windows 7 is shown on Figure 5.5. The USGCB for Microsoft Windows 7 contains only one profile, which consists of 258 rules. With OpenSCAP, it is possible to evaluate 207 of these rules, which is 80 %. Again, the remaining rules could not be evaluated because some OVAL tests are not supported by OpenSCAP so far.

5.2 Testing

OpenSCAP upstream contains a large test suite, which is run regularly. The Jenkins CI server runs the tests automatically for every change in upstream code and also for every pull request opened on GitHub.

Unfortunately, the test suite cannot be run on Windows because it is implemented in Bash. The test suite lacks unit tests. Most of the tests could be categorized as feature or integration tests.

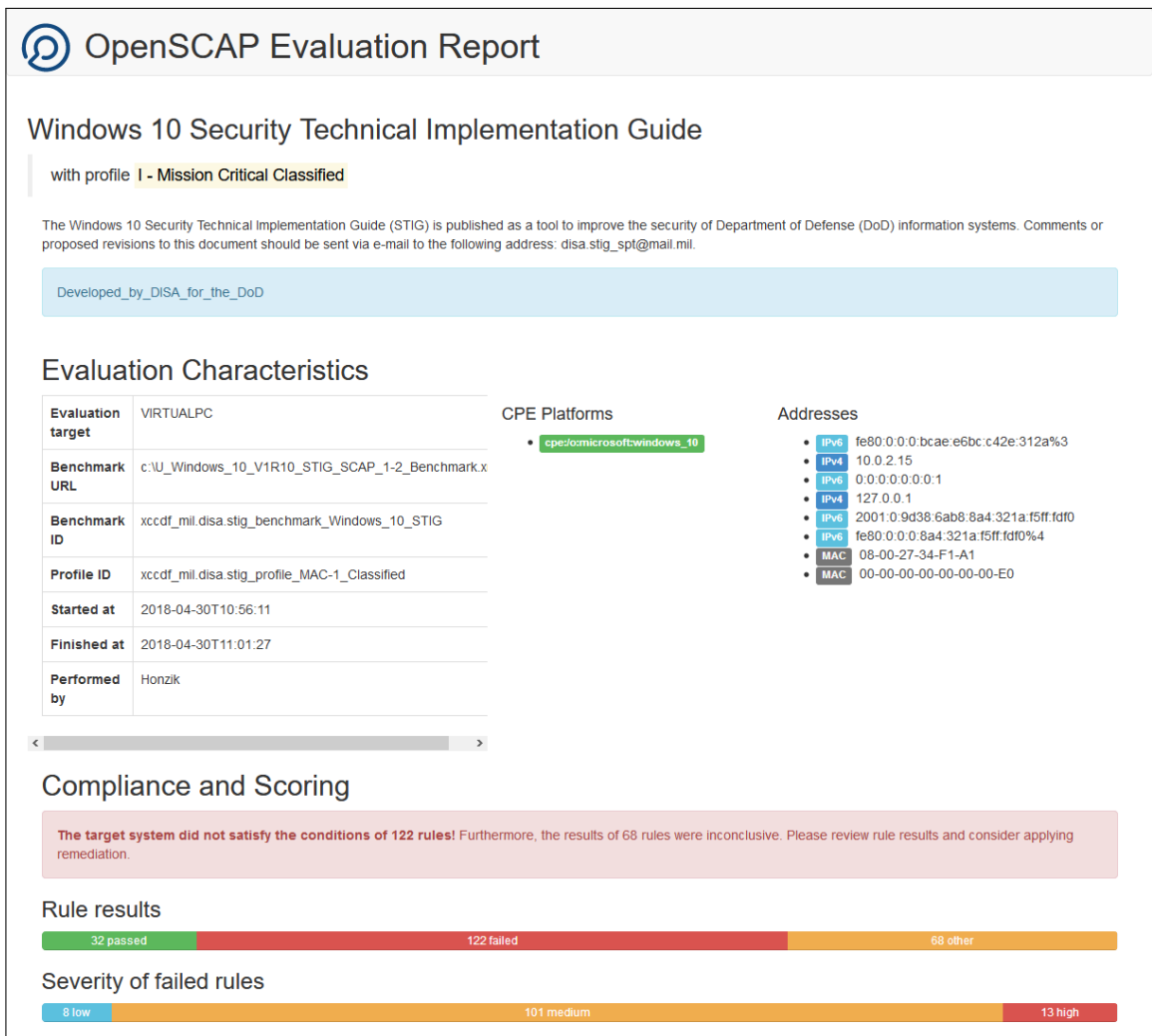


Figure 5.2: OpenSCAP HTML report presenting results of Windows 10 evaluation against DISA STIG.

Rewriting the test suite is a large task, so it is out of scope of this thesis. But increasing test coverage and introducing unit tests could be a task that the project will focus on.

For purposes of this work, we will only make sure we will not break tests and introduce regressions on Linux. Then, we will introduce a few basic tests that will be runnable on Windows.

To test Windows probes, we could use SCAP validation test suite [28] provided by NIST. The validation test suite is used in process of SCAP certification. Their test suite provides input OVAL files and expected results. Unfortunately, as of version 1-2.2.0.0 the test suite is not automated and results have to be compared manually. But OpenSCAP received certification in past, and some parts of this suite were automated by the OpenSCAP team and include in OpenSCAP upstream in past. Based on that, it will be possible to extend the upstream test suite by Windows tests form the NIST Validation Test Suite.

The author has provided exemplary content that tests the OVAL Windows probes.

Originally, we wanted to add a Windows node in Jenkins, to make sure the new pull requests will not break the Windows builds or destroy the work done in this thesis. The

▼ WN10-SO-000040		
Outgoing secure channel traffic must be encrypted when possible.	medium	pass
▼ WN10-CC-000145 (1x fail)		
Users must be prompted for a password on resume from sleep (on battery).	medium	fail
▼ WN10-SO-000045		
Outgoing secure channel traffic must be signed when possible.	medium	pass
▼ WN10-CC-000150 (1x fail)		
The user must be prompted for a password on resume from sleep (plugged in).	medium	fail
▼ WN10-CC-000155 (1x fail)		
Solicited Remote Assistance must not be allowed.	high	fail
▼ WN10-SO-000050		
The computer account password must not be prevented from being reset.	low	pass
▼ WN10-CC-000165 (1x fail)		
Unauthenticated RPC clients must be restricted from connecting to the RPC server.	medium	fail
▼ WN10-CC-000170 (1x fail)		
The setting to allow Microsoft accounts to be optional for modern style apps must be enabled.	low	fail
▼ WN10-SO-000055		
The maximum age for machine account passwords must be configured to 30 days or less.	low	pass
▼ WN10-CC-000175 (1x fail)		
The Application Compatibility Program Inventory must be prevented from collecting data and sending the information to Microsoft.	low	fail
▼ WN10-SO-000060		
The system must be configured to require a strong session key.	medium	pass
▼ WN10-CC-000180 (1x fail)		
Autoplay must be turned off for non-volume devices.	high	fail
▼ WN10-SO-000070 (1x fail)		
The machine inactivity limit must be set to 15 minutes, locking the system with the screensaver.	medium	fail

Figure 5.3: Rule results in OpenSCAP HTML report of evaluating Windows 10 machine against DISA STIG.

Windows builds would be scheduled regularly on Jenkins and OpenSCAP maintainers would be notified if build fails.

We have found it would be convenient to use AppVeyor for Windows builds instead of Jenkins. AppVeyor is a CI solution oriented on Windows builds. AppVeyor provides pre-defined images with Windows and Visual Studio 2017 and is fully integrated with GitHub. At the time of writing this thesis, AppVeyor is free for open source projects. We have configured AppVeyor¹ to build Windows version of OpenSCAP. AppVeyor runs the Windows build on every upstream pull request on master branch and AppVeyor reports the build status in the GitHub pull request.

The author has checked that evaluating of datastreams, composing datastreams and decomposing datastreams works as expected. That means OpenSCAP can handle temporary files and directories on Windows, because these operations work due to a temporary directory.

¹<https://ci.appveyor.com/project/OpenSCAP/openscap>

Outgoing secure channel traffic must be encrypted when possible. ✖

Rule ID	xccdf_mil.disa.stig_rule_SV-78133r1_rule
Result	pass
Time	2018-04-30T09:51:16
Severity	medium
Identifiers and References	Identifiers: CCI-002418 , CCI-002421 References:
Description	<VulnDiscussion>Requests sent on the secure channel are authenticated, and sensitive information (such as passwords) is encrypted, but not all information is encrypted. If this policy is enabled, outgoing secure channel traffic will be encrypted.</VulnDiscussion><FalsePositives></FalsePositives><FalseNegatives></FalseNegatives><Documentable>>false</Documentable><Mitigations></Mitigations><SeverityOverrideGuidance></SeverityOverrideGuidance><PotentialImpacts></PotentialImpacts><ThirdPartyTools></ThirdPartyTools><MitigationControl></MitigationControl><Responsibility></Responsibility><IAControls></IAControls>

OVAL details

'Domain member: Digitally encrypt secure channel data (when possible)' is set to 'Enabled' passed because of these items:

Hive	Key	Name	Last write time	Type	Value	Windows view
HKEY_LOCAL_MACHINE	System\CurrentControlSet\Services\Wetlogon\Parameters	SealSecureChannel	131511664238376178	reg_dword	1	64_bit

'Domain member: Digitally encrypt or sign secure channel data (always)' is set to 'Enabled' passed because of these items:

Hive	Key	Name	Last write time	Type	Value	Windows view
HKEY_LOCAL_MACHINE	System\CurrentControlSet\Services\Wetlogon\Parameters	RequireSignOrSeal	131511664238376178	reg_dword	1	64_bit

Figure 5.4: Rule detail presented in OpenSCAP HTML report.

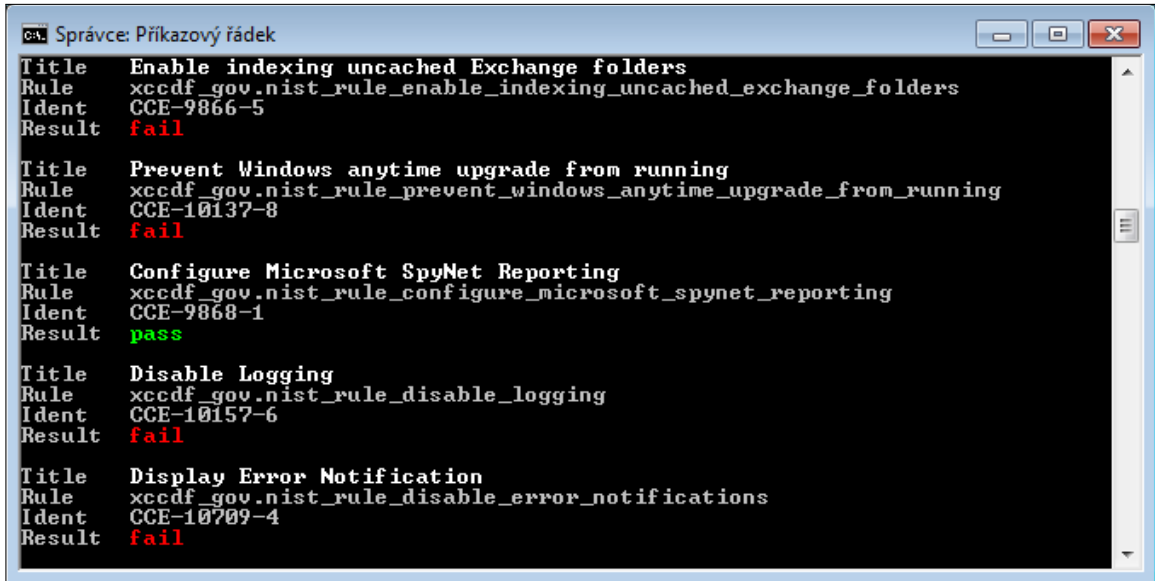
5.3 Future Plans

This work can be extended in many ways. It can be improved or integrated with other tools. We might want to port on Windows other tools from the OpenSCAP project. There is a potential in the project to leverage the work done during this thesis. We will mention a few possible future extensions that are enabled by this work.

5.3.1 Additional Probes

The OVAL Windows Definitions Schema defines 48 OVAL Tests. The author has not implemented all the OVAL tests that are applicable to Microsoft Windows. But the implemented part of OVAL language is enough to cover majority of the rules used in the practical use cases on Windows, as we have described in Section 5.1. The remaining parts of OVAL can be implemented later. The author suggests starting with the Windows file probe, because properties of files can be interesting for the system audit.

Thanks to changes made during work on this thesis, it will be now easy to add implementations of remaining OVAL tests for Windows, because the infrastructure is ready. The developers who will implement new probes should follow Subsection 4.6.1 and could get



```
ca: Správce: Příkazový řádek
Title      Enable indexing uncached Exchange folders
Rule       xccdf_gov.nist_rule_enable_indexing_uncached_exchange_folders
Ident      CCE-9866-5
Result     fail

Title      Prevent Windows anytime upgrade from running
Rule       xccdf_gov.nist_rule_prevent_windows_anytime_upgrade_from_running
Ident      CCE-10137-8
Result     fail

Title      Configure Microsoft SpyNet Reporting
Rule       xccdf_gov.nist_rule_configure_microsoft_spynet_reporting
Ident      CCE-9868-1
Result     pass

Title      Disable Logging
Rule       xccdf_gov.nist_rule_disable_logging
Ident      CCE-10157-6
Result     fail

Title      Display Error Notification
Rule       xccdf_gov.nist_rule_disable_error_notifications
Ident      CCE-10709-4
Result     fail
```

Figure 5.5: OpenSCAP running on Windows 7 SP1 evaluating USGCB.

inspired by existing code. Also, some of probes could reuse the existing code, e.g. probe that will implement OVAL `sid_test` can share many code with the Access Token Probe.

Moreover, the OVAL Independent Definitions Schema specifies more tests that are applicable to any operating system, including Microsoft Windows. If we implement all of the OVAL Windows tests and OVAL Independent tests, OpenSCAP could aim for the SCAP Certification by NIST as a certified scanner on Windows.

The implementation should start by implementing probes for the most used OVAL object according to Table 4.1. The implementation of independent probes should reuse the platform independent parts of the existing code and only provide a Windows alternative to Linux-specific code. It will be important to realize which parts of the code are generic.

5.3.2 SCAP Workbench

As we have discussed in Subsection 3.1.4, SCAP Workbench is a GUI application that wraps OpenSCAP. We have already mentioned that SCAP Workbench can run on Windows, but it can not scan the Windows machine, it can only scan remote Linux servers via SSH. As of version 1.1.6, the local scan option is inactive on Windows, because OpenSCAP has not been able to run on Windows before. After we have implemented OpenSCAP for Windows we could enable this option.

Integrating the native OpenSCAP into SCAP Workbench for Windows would make this work more accessible for users. We should not forget that the requests for Windows support often came from SCAP Workbench Windows users who expected it to be able to scan Windows machines.

The Windows scans will be enabled in the following way: We will bundle OpenSCAP for Windows into the SCAP Workbench installer, fix the CMakeFiles to link SCAP Workbench with the native `OPENSCAP.DLL` and enable the Local scan button in the GUI. We can also generate the Windows installer on Windows using CPack the same way as we have done for OpenSCAP. Then, we would not have to build the installer manually and we would be able to update it with every OpenSCAP release.

We can also associate SCAP XML files with SCAP Workbench, which would mean if a user double-clicks on an SCAP file in Windows Explorer, it will automatically open SCAP Workbench, and offer the user to scan his machine.

5.3.3 PowerShell Remediation

A *Remediation* is in SCAP terminology a short script, which purpose is to fix the system to put it in compliance with a rule. These scripts are usually part of XCCDF Rules, they are hard-coded inside XCCDF `fix` elements. Remediations are usually written in Python, Bash, and Perl. It has been very popular to write them in Ansible recently.

On Linux, OpenSCAP is able to run those scripts automatically, during evaluation, to fix rules that fail, if the user provides `--remediate` option. We could implement a similar feature on Windows as well. OpenSCAP would check if an interpreter of the script language is installed on the system, and then it would run it directly. The most useful thing will be to run Microsoft PowerShell scripts directly from OpenSCAP.

5.3.4 Windows Support in SCAP Security Guide

SCAP Security Guide is a very popular project where the security policies are defined and maintained. It already contains a lot of SCAP content for various platforms:

- Fedora Linux
- Red Hat Enterprise Linux 6 and 7
- Red Hat Enterprise OpenStack Platform 7
- CentOS 6 and 7
- Scientific Linux 6 and 7
- Debian 8
- Ubuntu 14.04 and 16.04
- Wind River Linux
- Chromium
- Firefox
- Java Runtime Environment
- Webmin
- JBoss Fuse
- SUSE Linux Enterprise and OpenSUSE

We can see that we could add Windows to the list. Regarding the possible Windows support in this project, we can think about 2 different features:

First, we could enable building SCAP Security Guide on Windows. That will be possible because the build system is a combination of CMake and Python, which are both available on Windows. The SCAP Security Guide build system uses a small amount of Linux-specific

things, e.g. paths, which will be needed to be replaced by platform-independent alternatives. The port will be much easier than OpenSCAP, because in Python most things are platform-independent. This effort will make life easier for contributors who use Windows on their workstations, but they develop SCAP content for Linux servers there.

Second, we could start developing Windows SCAP content in SCAP Security Guide. That would mean to add a new product folder in the project and outline a structure. By engaging with the community, we could make SCAP Security Guide the preferred resource of SCAP policies, as we already do on Linux. All Windows SCAP content development efforts across the industry could be concentrated in the repository, the same way it has already happened for Linux SCAP content.

5.3.5 Automatic Rights Elevation

In order to perform a complete scan of a Windows machine, OpenSCAP needs to have administrator privileges. Currently, the user must run the Administrator command prompt and must make sure he has all the rights to run the scan. OpenSCAP does not check the privileges. If it does not run as Administrator, the scan results are incomplete.

It would be better if OpenSCAP checked privileges before the scan, and if they are not sufficient, it would automatically display the User Account Control (UAC) elevation prompt. The UAC will require that the user provides valid administrator credentials. Then, OpenSCAP would continue with an administrator access token.

5.3.6 Porting Test Suite

The upstream test suite is written in Bash. It would be useful to port it into some high-level test framework, so that we could run the tests on any platform. This will be needed for contributors working on Windows to test their work. If this is not done, they will have to install Linux, e.g. using Vagrant, to run the tests or wait for Jenkins.

Chapter 6

Conclusion

This work described the problem of evaluating secure configuration of computer systems, including various security policies. The concept of SCAP, the OVAL language and its objects was studied. The OpenSCAP project was described in detail, including its implementation. A goal to extend OpenSCAP on a currently unsupported platform, Microsoft Windows, was presented.

The problematic parts in regards of portability to Windows were identified. Based on review of the current implementation, multiple changes in the OpenSCAP project were proposed. New probes in OpenSCAP project were identified as strongly needed.

The author worked closely with the upstream open source project and continuously improved his code based on code reviews and other feedback provided by the project community.

The author has provided 34 pull requests that altogether add 17405 and remove 65319 lines of code. The pull requests together consist of 735 commits. All the patches related to this work have finally been accepted into the OpenSCAP upstream repository. The changes are described in the project documentation.

Many improvements that were made during author's work on this thesis could be beneficial for the whole OpenSCAP project, not only for the Windows support activity. Namely architecture changes, the build system rewrite and offline mode fixes are useful in general. Also, the code portability was improved a lot, therefore it is now easier to port OpenSCAP to other platforms (Mac OS X, Android, etc.) as well.

The improvements enabled to scan Windows operating systems using OpenSCAP with popular SCAP security policies, e.g. DISA STIG. OpenSCAP reports the results and can be used on Windows the same way as on previous platforms.

The work described in this thesis is part of OpenSCAP master branch and will be part of upcoming major upstream release OpenSCAP 1.3.0.

Bibliography

- [1] Baude, B.: *Introducing atomic scan—Container vulnerability detection*. Blog post. May 2016. [Online; retrieved 7.12.2017]. Retrieved from: <https://developers.redhat.com/blog/2016/05/02/introducing-atomic-scan-container-vulnerability-detection/>
- [2] Center for Internet Security: *Open Vulnerability and Assessment Language*. Web site. 2018. [Online; retrieved 3.1.2018]. Retrieved from: <https://oval.cisecurity.org/>
- [3] Černý, J.: *A Tool for Development of OVAL Definitions within OpenSCAP Project*. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. 2016.
- [4] Cooley, S.: *AF_UNIX comes to Windows*. Blog post. December 2017. [Online; retrieved 14.2.2018]. Retrieved from: https://blogs.msdn.microsoft.com/commandline/2017/12/19/af_unix-comes-to-windows/
- [5] Defense Information Systems Agency: *Security Technical Implementation Guides (STIGs)*. Web site. 2017. [Online; retrieved 2.1.2018]. Retrieved from: <https://iase.disa.mil/stigs/Pages/index.aspx>
- [6] Jones, R.: *MinGW: Compile software for Windows without leaving your Fedora machine*. Blog post. October 2008. [Online; retrieved 1.12.2017]. Retrieved from: <http://camltastic.blogspot.cz/2008/10/mingw-compile-software-for-windows.html>
- [7] Lipner, S. B.: The Birth and Death of the Orange Book. *IEEE Annals of the History of Computing*. vol. 37, no. 2. April 2015: pp. 19–31. ISSN 1058-6180.
- [8] Lukašík, Š.: *Compliance Audit of Linux Environments*. Masters's thesis. Masaryk University, Faculty of Informatics. Brno. 2013.
- [9] Luparu, M.: *CMake support in Visual Studio*. Blog post. October 2016. [Online; retrieved 6.12.2017]. Retrieved from: <https://blogs.msdn.microsoft.com/vcblog/2016/10/05/cmake-support-in-visual-studio/>
- [10] Martin, K.; Hoffman, B.: *Mastering CMake*. Kitware, Inc.. 2013. ISBN 978-1930934269.

- [11] Microsoft: *Component Object Model (COM)*. Web site. [Online; retrieved 29.3.2018]. Retrieved from: [https://msdn.microsoft.com/en-us/library/windows/desktop/ms680573\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680573(v=vs.85).aspx)
- [12] Microsoft: *Security Identifiers*. Web site. [Online; retrieved 16.3.2018]. Retrieved from: [https://msdn.microsoft.com/en-us/library/windows/desktop/aa379571\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379571(v=vs.85).aspx)
- [13] Microsoft: *Windows Management Instrumentation*. Web site. [Online; retrieved 27.3.2018]. Retrieved from: [https://msdn.microsoft.com/en-us/library/aa394582\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/aa394582(v=vs.85).aspx)
- [14] Microsoft: *Aligned Free*. Web site. September 2016. [Online; retrieved 10.2.2018]. Retrieved from: <https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/aligned-free>
- [15] Mitteleite, E.: *Vcpkg: a tool to acquire and build C++ open source libraries on Windows*. Blog post. September 2016. [Online; retrieved 8.11.2017]. Retrieved from: <https://blogs.msdn.microsoft.com/vcblog/2016/09/19/vcpkg-a-tool-to-acquire-and-build-c-open-source-libraries-on-windows/>
- [16] Multiple Authors: *Common Criteria Portal*. Web site. [Online; retrieved 16.12.2017]. Retrieved from: <https://www.commoncriteriaportal.org/>
- [17] Multiple Authors: *IBM BigFix Compliance*. Web site. [Online; retrieved 18.4.2018]. Retrieved from: <https://www.ibm.com/us-en/marketplace/bigfix-compliance>
- [18] Multiple Authors: *Joval Continuous Monitoring*. Web site. [Online; retrieved 17.4.2018]. Retrieved from: <https://jovalcm.com/>
- [19] Multiple Authors: *Security Content Automation Protocol (SCAP) Compliance Checker (SCC)*. Web site. [Online; retrieved 12.3.2018]. Retrieved from: <http://www.public.navy.mil/spawar/Atlantic/Technology/Pages/SCAP.aspx>
- [20] Multiple Authors: OpenSCAP portal. Web site. 2016. [Online; retrieved 27.2.2018]. Retrieved from: <https://www.open-scap.org/>
- [21] Multiple Authors: *OpenSCAP Versioning*. Web site. January 2016. [Online; retrieved 4.1.2018]. Retrieved from: <https://github.com/OpenSCAP/openscap/blob/maint-1.2/docs/contribute/versioning.adoc>
- [22] Multiple Authors: Using the GNU Compiler Collection (GCC): Labels as Values. Web site. 2016. [Online; retrieved 15.2.2018]. Retrieved from: <https://gcc.gnu.org/onlinedocs/gcc/Labels-as-Values.html>
- [23] Multiple Authors: *OpenSCAP User Manual*. User manual. 2018. [Online; retrieved 11.1.2018]. Retrieved from: <https://github.com/OpenSCAP/openscap/blob/master/docs/manual/manual.adoc>

- [24] National Information Assurance Partnership: *Protection Profile for General Purpose Operating Systems*. Web site. [Online; retrieved 10.1.2018].
Retrieved from: https://www.niap-ccevs.org/MMO/PP/pp_os_v4.1.pdf
- [25] National Institute of Standards and Technology: *National Checklist Program Repository*. Web site. [Online; retrieved 2.1.2018].
Retrieved from: <https://nvd.nist.gov/ncp/repository>
- [26] National Institute of Standards and Technology: *The United States Government Configuration Baseline (USGCB)*. Web site. May 2009. [Online; retrieved 2.1.2018].
Retrieved from: <https://usgcb.nist.gov/>
- [27] National Institute of Standards and Technology: *Security Content Automation Protocol (SCAP) 1.2 Product Validation Record*. Web site. August 2017. [Online; retrieved 29.12.2017].
Retrieved from: <https://nvd.nist.gov/scap/validation/142>
- [28] NIST Computer Security Resource Center: *Security Content Automation Protocol Validation Program - SCAP 1.2 Validation Resources*. Web site. 2018. [Online; retrieved 30.1.2018].
Retrieved from: <https://csrc.nist.gov/Projects/scap-validation-program/Validation-Test-Content>
- [29] Podzimek, V.: *SCAP policy compliance configuration in Linux installations*. Masters's thesis. Masaryk University, Faculty of Informatics. Brno. 2013.
- [30] Quinn, S.; Scarfone, K.; Barrett, M.; et al.: *Guide to Adopting and Using the Security Content Automation Protocol (SCAP) Version 1.0*. Technical report. National Institute of Standards and Technology. July 2010.
- [31] Riesgo, Á.: *Using UTF-8 as the internal representation for strings in C and C++ with Visual Studio*. Blog post. May 2011. [Online; retrieved 1.5.2018].
Retrieved from: <http://www.nubaria.com/en/blog/?p=289>
- [32] Ronacher, A.: *Beautiful Native Libraries*. Blog post. August 2013. [Online; retrieved 6.12.2017].
Retrieved from: <http://lucumr.pocoo.org/2013/8/18/beautiful-native-libraries/>
- [33] Stack Overflow: *Developer Survey Results*. Web site. May 2017. [Online; retrieved 6.1.2018].
Retrieved from: <https://insights.stackoverflow.com/survey/2017>
- [34] The MITRE Corporation: *Open Vulnerability and Assessment Language*. Web site. May 2014. [Online; retrieved 3.1.2018].
Retrieved from: <https://oval.mitre.org/about/>
- [35] Waltermire, D.; Quinn, S.; Booth, H.; et al.: *The Technical Specification for the Security Content Automation Protocol (SCAP): SCAP Version 1.3*. Technical report. National Institute of Standards and Technology. 2018.
- [36] Wikipedia: *Windows Registry*. Web site. 2018. [Online; retrieved 3.1.2018].
Retrieved from: https://en.wikipedia.org/wiki/Windows_Registry

Appendix A

OpenSCAP Upstream Pull Requests

1. <https://github.com/OpenSCAP/openscap/pull/890> Porting OpenSCAP to CMake build system
2. <https://github.com/OpenSCAP/openscap/pull/901> Fixing header files includes
3. <https://github.com/OpenSCAP/openscap/pull/907> Minor problems caused by CMake
4. <https://github.com/OpenSCAP/openscap/pull/915> Minor problems caused by CMake
5. <https://github.com/OpenSCAP/openscap/pull/918> Minor problems caused by CMake
6. <https://github.com/OpenSCAP/openscap/pull/925> Enabling cross-compilation for Windows on Fedora using MinGW
7. <https://github.com/OpenSCAP/openscap/pull/926> Describe project import to Visual Studio 2017 in documentation
8. <https://github.com/OpenSCAP/openscap/pull/927> Fixes to build OpenSCAP without probe subsystem
9. <https://github.com/OpenSCAP/openscap/pull/930> Compatibility function for `realpath`
10. <https://github.com/OpenSCAP/openscap/pull/931> Remove variable length arrays
11. <https://github.com/OpenSCAP/openscap/pull/932> Compatibility function for `basename`
12. <https://github.com/OpenSCAP/openscap/pull/934> Mark public library symbols
13. <https://github.com/OpenSCAP/openscap/pull/961> Add a comment about 1 macro
14. <https://github.com/OpenSCAP/openscap/pull/965> Enable Ninja build system
15. <https://github.com/OpenSCAP/openscap/pull/979> Rewrite serial expression parser
16. <https://github.com/OpenSCAP/openscap/pull/978> Build OpenSCAP probe subsystem on Windows
17. <https://github.com/OpenSCAP/openscap/pull/984> Remove variadic macros

18. <https://github.com/OpenSCAP/openscap/pull/994> Remove unused code that was difficult to build on Windows
19. <https://github.com/OpenSCAP/openscap/pull/999> Change offline mode in system info probe to not use chroot
20. <https://github.com/OpenSCAP/openscap/pull/1000> Improve offline mode
21. <https://github.com/OpenSCAP/openscap/pull/981> Make OpenSCAP a single process
22. <https://github.com/OpenSCAP/openscap/pull/1004> Windows version of OVAL System Info Characteristics
23. <https://github.com/OpenSCAP/openscap/pull/1006> Create and remove temporary directory on Windows
24. <https://github.com/OpenSCAP/openscap/pull/1007> Add Windows Registry Probe
25. <https://github.com/OpenSCAP/openscap/pull/1009> Add Windows Accesstoken Probe
26. <https://github.com/OpenSCAP/openscap/pull/1010> Update developer's documentation
27. <https://github.com/OpenSCAP/openscap/pull/1012> Fix `oscap_dirname` on Windows
28. <https://github.com/OpenSCAP/openscap/pull/1018> Fix OVAL record element
29. <https://github.com/OpenSCAP/openscap/pull/1019> Add wmi57 Windows probe
30. <https://github.com/OpenSCAP/openscap/pull/1025> Remove `'\r'` characters from help output
31. <https://github.com/OpenSCAP/openscap/pull/1020> Windows installer
32. <https://github.com/OpenSCAP/openscap/pull/1024> Support comparing state record elements with OVAL state
33. <https://github.com/OpenSCAP/openscap/pull/1027> Add IP addresses in XCCDF results on Windows
34. <https://github.com/OpenSCAP/openscap/pull/1032> Add AppVeyor CI badge and configuration

Appendix B

Contents of the Attached Media

- `oval/` – Example OVAL definitions
- `reports/` – Example HTML reports
- `src/` – Source code for this text in \LaTeX
- `count.py` – Script counting OVAL object elements
- `openscap-1.3.0.tar.gz` – OpenSCAP source code
- `OpenSCAP-1.3.0-win32.msi` – OpenSCAP for Windows Installer
- `thesis.pdf` – This text in PDF format