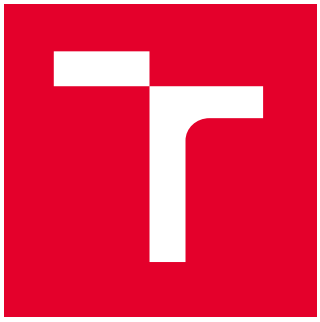


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

SECURE DEPLOYMENT AND TESTING OF OVIRT PLATFORM

NÁSTROJ PRO OVĚŘENÍ BEZPEČNÉHO NASAZENÍ PLATFORMY OVIRT

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Vojtěch Vágner

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. Petr Dzurenda, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Vojtěch Vágner ***ID:*** 216975

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Nástroj pro ověření bezpečného nasazení platformy oVirt

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku zabezpečení cloudové platformy oVirt pomocí bezpečnostních standardů s ní spojených (FIPS, DISA STIG, Common Criteria). Analyzujte aktuální standardy a možné zlepšení zabezpečení platformy oVirt. Navrhněte proces ověření kompatibility cloudové platformy oVirt s posuzovanými standardy a kroky k případnému dalšímu zlepšení zabezpečení. Navrhnuté řešení implementujte a otestujte na nasazených cloudových platformách kompatibilních s bezpečnostními standardy se zaměřením na nedostatky nasazených platforem k dosažení daných standardů. Zhodnoťte výsledky a diskutujte možná rozšíření.

DOPORUČENÁ LITERATURA:

[1] Product Documentation for Red Hat Virtualization 4.4. Red Hat Customer Portal [online]. Raleigh: Red Hat, 2021 [cit. 2021-9-7]. Dostupné z: https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.4

[2] OVirt User Documentation. OVirt [online]. World: oVirt Community, 2021 [cit. 2021-9-7]. Dostupné z: https://ovirt.org/documentation/

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 31.5.2022

Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

Konzultant: Jiří Macku

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRACT

The oVirt virtualization platform offers a wide range of different configurations. All of those configurations are by no means inherently secure when deployed out of the blue. The secure configurations are defined by security standards which the configurations claim conformance to. Since oVirt is a community project, it is hard to get such a software certified in the field of security standards. Certifications in the field of security standards are expensive. Luckily, oVirt shares common ground with Red Hat Virtualization to which certain security standards are applicable (FIPS 140-2, DISA STIG, Common Criteria). The link between Red Hat Virtualization and oVirt enables secure configurations also for oVirt. The focus of this thesis is then to determine which configurations are supported by the standards, therefore secure, and how to verify that the secure configurations are present. This is done through the use of a script in the form of an Ansible Playbook incorporating Ansible Roles that manage compliance evaluation for each of the presented security standards.

KEYWORDS

oVirt, Red Hat Virtualization, FIPS, DISA STIG, Common Criteria, bezpečné nasazení, bezpečná konfigurace, Ansible, bezpečnostní standard

ABSTRAKT

Virtualizační platforma oVirt nabízí široké spektrum možných konfigurací. Avšak žádná z těchto konfigurací není bezpečnou bez předchozího zásahu v kontextu slepého nasazení platformy. Bezpečné konfigurace jsou definovány bezpečnostními standardy, se kterými je daná konfigurace v souladu. Jelikož oVirt je komunitní projekt, není lehké tento typ softwaru certifikovat v oblasti bezpečnostních standardů. Certifikace v oblasti bezpečnostních standardů je drahou záležitostí. Naštěstí, oVirt sdílí stejný základ s produktem Red Hat Virtualization, vůči kterému jsou určité bezpečnostní standardy aplikovatelné (FIPS 140-2, DISA STIG, Common Criteria). Most mezi Red Hat Virtualization a oVirt dává možnost bezpečných konfigurací i pro oVirt. Záměr této práce je následně určit, které konfigurace jsou podporovány danými standardy, tedy bezpečné, a jak ověřit, že jsou přítomné v dané nasazené platformě. To je realizováno pomocí skriptu ve formě Ansible Playbook, který zahrnuje Ansible Role. Každá role v rámci skriptu obhospodařuje evaluaci shody pro daný bezpečnostní standard.

KLÍČOVÁ SLOVA

oVirt, Red Hat Virtualization, FIPS, DISA STIG, Common Criteria, secure deployment, secure configuration, Ansible, security standard

ROZŠÍŘENÝ ABSTRAKT

Úvod

Každý software typu virtualizační platformy oVirt poskytuje obrovské množství různých konfigurací. Avšak ne všechny konfigurace takového software jsou vhodné pro produkční nasazení z hlediska kybernetické bezpečnosti. Za předpokladu, že dojde k slepému nasazení virtualizační platformy oVirt, nemusí být konfigurace takto nasazené platformy automaticky bezpečná. Aby bylo možné zajistit, že dané nasazení platformy je bezpečné, je nejprve nutné zvolit správnou konfiguraci. Pro tuto situaci je vhodné využít relevantních bezpečnostních standardů. Bezpečnostní standardy deklarují, které konfigurace daného software jsou bezpečné, a to za předpokladu shody s bezpečnostními požadavky daných bezpečnostních standardů. Toto je primární zaměření této práce, a to určit které konfigurace virtualizační platformy oVirt jsou bezpečné a zdali jsou ve shodě s bezpečnostními požadavky vybraných standardů. Dalším cílem práce je vytvořit nástroj pro automatizované ověření této shody.

Vybrané bezpečnostní standardy jsou analyzovány v průběhu této práce a jejich architektura a definice jsou uvedeny. Týká se to bezpečnostních standardů FIPS 140-2 (Federal Information Processing Standard 140-2), DISA STIG (Defense Information Systems Agency Security Technical Implementation Guide) a Common Criteria. Vysvětlení implementace daných bezpečnostních standardů platformou oVirt je následně podáno v kapitole o architektuře platformy oVirt. V této kapitole je zprvu vysvětlen koncept technologie virtualizace a následně implementace daných standardů platformou oVirt je také vysvětlena.

V kontextu implementace daných standardů platformou oVirt se vyskytuje problém absolutní shody se zejména procedurálními aspekty bezpečnostních požadavků těchto standardů. Jelikož platforma oVirt je open-source komunitní projekt, neexistuje schůdný způsob, jak takový software formálně certifikovat. Nelze jej však vyloučit. Toto se na první pohled zdá jako neřešitelný problém. Pokud daná konfi -gurace striktně využívá platformy oVirt a nikoliv platformy Red Hat Virtualization, danou konfiguraci nelze považovat v souladu s danými bezpečnostními standardy. V tomto kontextu lze však hovořit o kompatibilitě s danými standardy, nikoliv o shodě. Termínem kompatibility je autorem označena takzvaná měkká shoda se standardem. Jednoduše řečeno, platformy oVirt a Red Hat Virtualization sdílí stejný implementační základ. Pokud je konfigurace platformy Red Hat Virtualization považována za souladnou s danými standardy, lze říci, že stejnou konfiguraci akorát pro platformu oVirt lze považovat za kompatibilní s těmito standardy v kontextu téměř shodných funkcionalit a implementačního aspektu obou platforem. V kon-

textu virtualizační platformy Red Hat Virtualization se taktéž nejedná o triviální problém. Základ pro souladnou konfiguraci s danými standary lze nalézt ve vrstvené architektuře platformy Red Hat Virtualization. Jedná se o komplexní problém a tato práce si klade za cíl jej nastínit a pokusit se o jeho alespoň částečné řešení.

V kontextu vytvoření mechanismu kontrolních seznamů pro ověření souladné konfigurace virtualizačních platforem oVirt a Red Hat Virtualization je možné využít systémově nativních prostředků a nástrojů poskytovaných samotnými platformami. Nicméně pro bezpečnostní standard Common Criteria žádný systémově nativní prostředek či nástroj pro ověření souladné konfigurace neexistuje. Tento naskýtající se problém je řešen v kapitole vysvětlující implementaci modulu pro ověření shodné konfigurace s tímto standardem.

Technický počin této práce je implementace nástroje určeného pro ověření shody konfigurací virtualizačních platforem oVirt a Red Hat Virtualization s bezpečnostními kritérii sledovaných standardů. Architektura tohoto nástroje je též vrstvená a modulární. Pro každý sledovaný bezpečnostní standard je vytvořen separátní modul obhospodařující funkcionalitu ověření shody.

Popis řešení a shrnutí

V rámci řešení práce bylo nejprve nutné nastudovat problematiku jednotlivých standardů a pochopit jejich obecné fungování. Následně bylo potřeba vůbec porozumět technologii úplné virtualizace, která je implementována virtualizační platformou oVirt. Na základě tohoto pochopení bylo možné rozebrat jednotlivé funkcionality platformy oVirt a zaměřit se na jejich bezpečnostní aspekty. Z dokumentace pro platformu bylo čitelné, ve kterých částech lze uplatnit sledované bezpečnostní standardy, avšak kontext odkud vychází aspekt souladnosti s danými standardy musel být vydedukován. Tento kontext mohl být vydedukován na základě korektního pochopení fungování jednotlivých aspektů.

V další fázi bylo nutné stanovit proces ověření kompatibility platformy oVirt s jednotlivými standardy. Stanovení tohoto procesu vycházelo jak z obecného fungování sledovaných standardů, ale i z implementačního hlediska v rámci platformy. Zde bylo využito vrstvené architektury platformy oVirt založené na operačním systému RHEL (Red Hat Enterprise Linux).

V poslední fázi byla vytvořena architektura pro nástroj ověřující souladnost konfigurací platformy se sledovanými standardy. Tato architektura byla založena na současném stavu techniky v oblasti vývoje software. Pro následnou implementaci architektury byla zvolena technologie Ansible, a to zejména z důvodu toho, že je tato technologie primárně určena pro automatizované nasazování počítačů s předem určenými systémovými konfiguracemi.[68].

Zhodnocení výsledků

Tato práce vytvořila teoretický přehled o sledovaných standardech a usilovala o uvádění pouze důležitých a relevantních informací pro celkové pochopení presentované problematiky. Došlo k vysvětlení architektury a fungování série standardů FIPS 140, zejména tedy standardu FIPS 140-2. Vysvětlila fungování standardu DISA STIG a jeho navázání na SRG (Security Requirements Guide). Následně byl prezentován obecný pohled na fungování standardu Common Criteria.

Práce pokračovala s deklarováním definice technologie úplné virtualizace a její zasazení do kontextu virtualizační platformy oVirt a její downstream odnože Red Hat Virtualization. Zde se naskytlo několik problémů týkající se problematiky striktního souladu a pouhé kompatibility se standardem, pro což byla uvedena dostačující definice. Pro každý sledovaný standard došlo k uvedení jejich implementace platformou oVirt. Práce konstatovala důležité napojení mezi platformami oVirt a Red Hat Virtualization a operačním systémem RHEL. Toto nalezené propojení umožnilo otevřít možnost určení souladných konfigurací pro platformu Red Hat Virtualization.

Také došlo k prezentování kontrolních seznamů pro ověření souladné konfigurace pro sledované bezpečnostní standardy. V kontextu standardu FIPS 140-2 se jednalo o využití systémových kryptografických politik implementovaných na straně operačního systému RHEL a kritérií vycházejících z relevantních dokumentů typu *security policy* pro konkrétní kryptografický modul. Pro standard DISA STIG byl využit systémově nativní nástroj pro ověřování bezpečnosti systémových konfigurací vycházející z projektu OpenSCAP. Pro bezpečnostní standard Common Criteria došlo k využití souboru funkčních testů poskytnutého společností Red Hat jakožto kontrolního seznamu pro ověření souladné konfigurace.

V rámci této práce došlo k navrhnutí zlepšení určitých aspektů nakládání se sledovanými bezpečnostními standardy platformou Red Hat Virtualization. V kontextu standardu FIPS 140-2 se jednalo o návrh k začlenění podporovaných kryptografických modulů do certifikačního procesu nově vzniklého pokračování série standardů FIPS 140, a to konkrétně FIPS 140-3. Co se týče standardu DISA STIG, zde došlo k návrhu spolupráce s institucí DISA na vytvoření oficiálního STIG pro platformu Red Hat Virtualization. V poslední řadě autor práce doporučil frekventovanější zapojení do procesu certifikace pro standard Common Critieria, a to z důvodu novějších funkcionalit nabízených novějšími verzemi platformy.

V části technické implementace práce byl vysvětlen důvod pro automatizované nasazování a testování platforem oVirt a Red Hat Virtualization. Automatizované nasazování a testování bylo zajištěno formou CI/CD. Všechny funkční prvky automatizace byly vysvětleny a popsány. Následně práce ukázala přehled a architekturu

skriptu pro ověření souladné konfigurace. Došlo také k vysvětlení modulárního aspektu daného skriptu. Pro každý standard byl vytvořen jednotlivý modul obhospodařující funkcionalitu ověření shody. Jakožto správnou technologií pro implementaci skriptu byla zvolena technologie Ansible. Skript využívá aspektů playbooku a role vycházející z technologie Ansible pro jeho implementaci. Implementované moduly využívají architektury takzvaných Ansible rolí.

V kontextu implementace jednotlivých modulů jakožto Ansible rolí je využíváno vytvořených kontrolních seznamů pro ověření souladné konfigurace. Tyto kontrolní seznamy byly formalizovány takovým způsobem, aby byla zajištěna strojová čitelnost pro ověřovací skript implementovaný v rámci této práce. Byly vytvořeny dva typy rolí, a to role, které obhospodařují funkci ověřování shody, a role, které dodávají podpůrné funkcionality předchozím rolím. Tyto podpůrné funkcionality jsou například řešení závislostí na ostatním software nebo počáteční získávání informací o nasazených platformách oVirt a Red Hat Virtualization.

Na základě této práce došlo tedy k úspěšnému naimplementování a otestování nástroje s funkcí ověření souladné konfigurace nasazených platforem oVirt a Red Hat Virtualization se sledovanými standardy. V poslední části práce je vyobrazen testovací scénář pro tento nástroj.

Autor deklaruje, že úspěšně splnil zadání bakalářské práce.

VÁGNER, Vojtěch. *Secure deployment and testing of oVirt platform*. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2021, 92 p. Bachelor's Thesis. Advised by Ing. Petr Dzurenda, Ph.D.

Author's Declaration

Author:	Vojtěch Vágner
Author's ID:	216975
Paper type:	Bachelor's Thesis
Academic year:	2021/22
Topic:	Secure deployment and testing of oVirt platform

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the paper and listed in the comprehensive bibliography at the end of the paper.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to thank the supervisor of the bachelor's thesis Ing. Petr Dzurenda, Ph.D. for his guidance, consultations, tolerance and constructive proposals for the thesis.
Without proper technical supervisory of Ing. Jiří Macků and also without proper theoretical supervisory in terms of security standards of Ing. Jaroslav Řezník I would not be able to successfully finish this thesis. That is why I thank them very much for their professional supervisory.
Lastly I would like to thank other people from the company Red Hat and from the Government CERT of the Czech Republic that supported me in finishing the text of the bachelor's thesis.

Contents

Listings

Introduction

Every software, like the oVirt platform, bares a wide range of possible configurations. Not all configurations of the particular software are plausible for a production environment in terms of cybersecurity. When the oVirt virtualization platform is deployed without proper consideration, it might not be *secure* by default. To ensure that a given deployment is secure, the right configuration needs to be chosen first. That is where security standards are of help. Security standards declare that a given software configuration is secure[1] under the condition that it complies with the security requirements defined by the standards. This is the primary focus of this thesis, to determine which configurations of the oVirt platform are secure and to whether they comply with the particular security standards and to provide a tool to verify them.

The particular security standards shall be analyzed through the course of this thesis and their architecture and definition shall be provided. The provided general explanation of the standards shall then be specified in the chapter about the oVirt architecture. There first the overall concept of the full virtualization technology shall be laid out and then the implementation of the respective standards shall be shown.

In the context of the sole implementations of the standards, a problem of compliance and compatibility occurs. Since oVirt is an open-source community project, there is a very low probability of it undergoing a certification process for the above mentioned standards. Now this comes off as a very hard problem to solve. If a configuration strictly uses oVirt and not Red Hat Virtualization, the configuration can not be compliant with the standards. In this case compatibility applies. Simply put, Red Hat Virtualization and oVirt share common grounds. If a configuration of Red Hat Virtualization is compliant with a standard, then the same configuration of oVirt is therefore compatible with the standard in the pure functional context. Neither it is a trivial problem in the case of Red Hat Virtualization. The grounds for compliance with the standards actually lie in many cases in the architecture of a layered product that Red Hat Virtualization employs. It is a complex issue and this thesis tries to unravel it.

In terms of creating a checklist mechanism for evaluating compliance of a deployed configuration, native tools provided by the product itself exist. In the case of Common Criteria, no native tools are provided and therefore such an issue shall be resolved in the implementation part of this thesis.

The thesis then shows the implementation of a scanning tool that is capable of evaluating compliance of oVirt/RHV configurations to the particular standards.

[1] in the context of the standard

The scanning tool is divided into modular pieces and each of the piece implements the functionality of assessing compliance for the given standard it is intended for.

1 Security standards

The trend in security, and in cyber security in particular, is to standardize the processes of implementing security measures. It is useful for a set of similar products to have the same or close to the same level of security which is the goal of many security standards. For a security standard a way of certification is also crucial for the correct employment of the standard. The process of certification is usually delegated to an independent laboratory that tests the compliance of the evaluation target with the standard. These laboratories are strictly selected by the respective authorities[25].

The predominant doctrine in the architecture of security standards today is a threat model. The threat model tries to balance the risk of potential threats on assets by implementing countermeasures that ought to mitigate the risk of the threats. Threats to the assets occur in three possible domains - integrity, availability and confidentiality. This classification is known as the CIA triad[25].

The standards that this thesis focuses on are linked to the oVirt platform, or to be more precise to the Red Hat Virtualization platform. Federal Information Processing Standard (FIPS), Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) and Common Criteria shall be introduced and their architecture explained in their respective sections.

1.1 FIPS

FIPS is an abbreviation for Federal Information Processing Standards. FIPS, hence the name, is a set of security standards and guidelines for computer systems[35] in the field of cybersecurity.[5] These standards and guidelines were produced in accordance with the Federal Information Security Management Act (FISMA) and issued by the National Institute of Standards and Technology (NIST). The approval process goes to the Secretary of Commerce.[35]

The FIPS security standards are primarily mandatory for government organizations of the US but can be of help for non-government institutions that seek out reference for their policies regarding cybersecurity.[35] This might change a bit in the case of the FIPS 140 standards, since the FIPS 140-3 aligns with the ISO/IEC 19790:2012(E)[10]. This makes it more applicable towards worldwide use.

As stated previously, FIPS is a set of security standards. The one that this thesis shall focus on more deeply is the FIPS 140-2 which the core cryptographic components from the RHEL operating system are certified for[1]. The FIPS 140-2 defines "security requirements for cryptographic modules" as it states in the name of the standard.[8] Other FIPS standards include[2]:

- FIPS 180-4 Secure Hash Standard
- FIPS 186-4 Digital Signature Standard
- FIPS 197 Advanced Encryption Standard
- FIPS 198-1 The Keyed-Hash Message Authentication Code
- FIPS 199 Standards for Security Categorization of Federal Information and Information Systems
- FIPS 200 Minimum Security Requirements for Federal Information and Information Systems
- FIPS 201-2 Personal Identity Verification (PIV) of Federal Employees and Contractors
- FIPS 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
- FIPS 140-3 Security Requirements for Cryptographic Modules

FIPS 140-2

The FIPS 140-2 security standard falls under the Cryptographic Module Validation Program (CMVP)[11] which is a joint effort between NIST and the Canadian Centre for Cyber Security[7]. The CMVP program strives for the use of validated cryptographic modules and tries to help federal agencies in choosing the secure implementations of cryptographic modules.[7]

This is achieved through the FIPS 140 certification process. The certification process under the CMVP is done by Cryptographic and Security Testing (CST) Laboratories[1]. These laboratories do the actual testing of the cryptographic modules. The testing procedure involves validating whether the cryptographic module meets the requirements defined in the standard. The other two important documents are the Derived Test Requirements (DTR) and the Implementation Guide. The DTR works as a reference point for the validation requirements of the cryptographic modules that are to be used by the testing laboratories or the vendor.[11] The Implementation Guide, as stated on the CMVP page on FIPS 140-2, "is intended to provide clarifications of CMVP programmatic guidance, FIPS 140-2, FIPS 140-2 Derived Test Requirements, testing guidance, and guidance related to the implementation of Approved or non-Approved security function"[11].

The certification process alone involves the accredited laboratory to rate the cryptographic module on a Security Level scale of one to four in eleven domains defined by the standard. These separate ratings are then used for an overall rating of the cryptographic module[11].

[1]Accredited by NIST through the National Voluntary Laboratory Accreditation Program (NVLAP)[4]

When it comes to the continuity of FIPS 140 standards, FIPS 140-2 superseded FIPS 140-1 and now has been superseded by FIPS 140-3[3]. The transition to the newer version of the standard is still an ongoing effort, so the FIPS 140-2 certifications are still considered as valid[6].

In essence, FIPS 140 standards are applicable anywhere where cryptographical protection of information or data is needed[7]. FIPS 140-2 itself defines security requirements for *cryptographic modules* that were developed and intended to protect unclassified sensitive information in computer and telecommunication systems, voice systems not excluded.[8] A cryptographic module is defined as "the set of hardware, software, and/or firmware that implements approved security functions and is contained within the cryptographic boundary"[8]. A cryptographic boundary is then the physical layer of a cryptographic module.[8]

An important addition to the FIPS 140-2 standard is that it sets a requirement on documenting the implementation process of the cryptographic module with the security requirements provided in the standard. The documentation then can be found in the Security Policy which describes how a particular cryptographic module meets the requirements specified in the standard and documents the Approved mode[2] of operation[12].

As stated previously, the standard defines requirements in eleven different domains for four different Security Levels. The higher the Security Level, the more demanding requirements[3]. The eleven domains are as follows[8]:

- cryptographic module specification;
- cryptographic module ports and interfaces;
- roles, services and authentication;
- finite state model;
- physical security;
- operational environment;
- cryptographic key management;
- electromagnetic interference/electromagnetic compatibility (EMI/EMC);
- self-tests;
- design assurance;
- mitigation of other attacks

[2]e.g. a mode of the cryptographic module that employs only NIST-approved security functions[8]

[3]This is to be demonstrated by the explanation of the Security Levels later in the text.

The standard also lays out functional security objectives which shall be achieved by means of implementing security requirements of the presented domains in a cryptographic module. The general picture of the functional security objectives is to have a cryptographic module that uses NIST-approved cryptography in a proper way for data protection and prevents unauthorized disclosure or modification of cryptographic keys and CSPs (Critical Security Parameters). Other objectives touch on the subject of being able to detect operational errors and for a cryptographic module to indicate operational state[8].

Important requirements in the context of this thesis are the one residing in the operational environment domain, since FIPS 140-2 specifies certain conformance claims of the particular Target of Evaluation (here being the operational environment that the cryptographic module operates in). For Security Level 2 and above the standard specifies Common Criteria Protection Profiles (PPs) and Evaluated Assurance Levels (EALs) to which the Target of Evaluation must claim conformance to[8]. This goes to show that FIPS 140-2 and Common Criteria have an overlap. This overlap though is there in the case of the Target of Evaluation being a general purpose operating system[8].

In terms of FIPS 140-2 security levels, Level 1 is the lowest one with the least number of requirements. Security Level 1 mandates that a NIST-approved algorithm or security function has to be implemented in a cryptographic module. The requirement then states that at least one of those has to be used. No other security precautions such as physical security has to be implemented. This can be of use for software-implemented cryptographic modules that do not have to implement requirements regarding physical security[8].

Security Level 2 adds on top of the Level 1 the requirement for an opaque tamper-evident or pick-resitant locks to be added to the physical layer of the cryptographic module for enhanced physical security and protection against unauthorized access to plaintext cryptographic keys and CSPs. Level 2 also requires implementation of a role-based authentication. This means that the cryptographic module should be able to authorize an operator, grant the operator a role with a respective set of services defined for that role. The operational environment needs to be compliant with NIAP[4] Approved GPOSPP (Protection Profile for General Purpose Operating Systems)[9] or NIAP Approved Protection Profile for Mobile Device Fundamentals[9] and EAL2[8].

Level 3 enhances the introduced tamper-evident to use a circuitry that sets all plaintext critical security parameters to zeros if the physical layer of a cryptographic module is intruded. Regarding the way of handling plaintext critical security pa-

[4]National Information Assurance Partnership

rameters, the Level 3 also states that obtaining or putting plaintext critical security parameters has to be executed through physically or logically isolated ports or interfaces sent through a trusted path. In the case of handling encrypted critical security parameters, these can be sent through an intervening system. Also the role-based authentication required in Level 2 has to be upgraded to an identity-based authentication. The operational environment must meet the requirements listed in Level 2 with additional requirements of compliance to EAL3 with FTP_TRP.1[5] and ADV_SPM.1[6]extensions[8].

Level 4 in terms of physical security of the physical layer of the cryptographic module‘s intrusion detection states that potential intrusions should be detected in all cases with a very high probability. This detection mechanism then would result in setting the plaintext critical security parameters to zero. This mode of operation is very suitable for physically unprotected environments. Level 4 also requires a cryptographic module to implement a protection against fluctuations of normal operating ranges for voltage and temperature either by detecting such fluctuations and shutting down the cryptographic module in order to prevent further manipulation or immediately setting critical security parameters to zero or by verifying through a use of tests that the cryptographic module is resistant to such fluctuations. These modes of detecting fluctuations and responding to them or implementing a way of testing resistance to such fluctuations are called environmental failure protection (EFP) and Environmental failure testing (EFT) and both modes of operation are a valid implementation for a cryptographic module at Level 4. For optimal EFP and EFT implementation the normal operating ranges have to be specified. The operational environment needs to be compliant with requirements from Level 3 and also be compliant with EAL4[8].

1.2 DISA STIGs

The so called Security Technical Implementation Guides (STIGs) are security guides provided by the Defense Information Systems Agency (DISA) under the U.S. Department of Defense (DoD)[19]. These STIGs are based on Security Requirements Guides (SRGs), which are a general implementation of applied security measures for mitigating sources of security risks posed on different parts of IT systems and applications[42].

[5] "... requires that a trusted path between the TSF and a user be provided for a set of events defined by a PP/ST author. The user and/or the TSF may have the ability to initiate the trusted path"[26]

[6] e.g. Formal (having a specification language and a theorem prover) TOE security policy model[28]

SRGs provide general security requirements for various technologies and organisations, the STIGs on the other hand provide detailed implementation of product-specific measures to be taken in order for the product to be SRG compliant in the specific product's technological branch. The main use of SRGs is security control implementation in specific systems or technology types such as general purpose operating systems or network switches[42].

SRGs are based on a myriad of security-oriented principles but the baseline is constructed from established security standards and since DoD is a federal agency, there are certain security standards the DoD has to implement. The security baseline of SRGs include DoD Instructions such as DoD 8500.01 implementing cybersecurity measures, DoD Directives, NIST special publications (NISP SP 800-53, NIST SP 800-37), NIST FIPS standard, NIST Cybersecurity Framework, DoD Risk Management Framework and other DoD cybersecurity policies[42].

SRGs and STIGs are published in a form of eXtensible Markup Language (XML) files in the Extensible Configuration Checklist Description Format (XCCDF)[42] and are intented to be viewed via a Security Content Automation Protocol (SCAP) validated tool while evaluating the compliance of a system[21].

The security requirements in a STIG contain a description of the need to implement the specific requirement, severity (how severe is implementing or not implementing the specific requirement), a check box providing a text description of what one has to check for compliance with this requirement and a fix text providing a text description of what one has to do in order to remediate the particular system's defect in the context of STIG compliance[20].

1.3 Common Criteria

The Common Criteria for Information Technology Security Evaluation, Common Criteria or CC in short, is a set of requirements for security conformity of IT products. The Common Criteria is a joint effort within an international agreement called the Common Criteria Recognition Arrangement. IT products are to be evaluated by licensed laboratories to a certain assurance level. For the specific segments of IT products supporting documents are made which define the process of application of evaluation methods. A certificate confirming a Common Criteria compliance is issued by an authorized certificate issuer and these certificates are recognized by the signatories of the agreement[29].

The architecture of Common Criteria

The Common Criteria standard builds up on the protection of assets known as the CIA triad. The components included in the protection of assets are confidentiality, integrity and availability. Thereof the CC standard focuses primarily on the security of an IT product in its full operational environment, be it software, firmware and/or hardware. On the other hand the CC standard does not address security requirements on the implementation of cryptographic modules, thus does not pose any requirements whatsoever on the qualities of cryptographic algorithms used in the IT product, though if such a security requirement is raised, it has to be assessed as well[25]. In such case a combination of Common Criteria and FIPS 140-2 makes sense and is suitable. The Common Criteria consists of three main parts[25].

- Part 1, Introduction and general model;
- Part 2, Security functional components;
- Part 3, Security assurance components

Key concepts of Common Criteria

The entity that is being evaluated is called the Target of Evaluation (TOE), which in the words of the standard is "a set of software, firmware and/or hardware possibly accompanied by guidance"[25]. The entity that the standard marks as a TOE can refer to "an IT product, a part of an IT product, a set of IT products, a unique technology that may never be made into a product, or a combination of these"[25]. The list of examples of a TOE provided in the standard is rather demonstrative. The TOE can then represent an operating system, a software application (isolated and/or in combination with an OS and/or a workstation) and others. A software TOE can be represented as "a list of files in a configuration management system"[25] but also as "an installed and operational version"[25]. The context of the TOE is always important in order to determine its representation.

However, an IT product can support many different configurations, in the context of a TOE rather strict constrains can be set on the configuration to meet certain security requirements, because of this a dichotomy between all the configurations supported by the IT product and only one or a few configurations supported by the TOE can exist[25].

Another important terminology connected to the TOE is the Security Target (ST). The ST provides a definition of the given security problem and countermeasures to potential threats imposable on an asset in the context of the defined security problem and demonstrates the sufficiency of these countermeasures in the form of Security Objectives (SOs)[25]. The SOs then split based on the object of interest and whether the objective shall be evaluated into the security objectives of[25]:

- the TOE, which shall be evaluated;
- the Operational Environment, which shall not be evaluated

This division is made because TOE is focused solely on the IT related countermeasures and although an operational environment can include an IT related entity (such as an operation system), it can also encompass non-IT related security countermeasures (physical security for example). The IT-related countermeasures of the TOE that are to be evaluated are then formulated into Security Functional Requirements (SFRs). The Security Target of the TOE must meet Security Assurance Requirements (SARs) that if met provide assurance for the TOE to not contain potentially exploitable vulnerabilities that would get intentionally or unintentionally (through an error) integrated into the TOE in the process of development[25].

SARs provide a set of activities which then "determine correctness of the TOE"[25]. These activities comprehend "testing the TOE, examining various design representations of the TOE[25] and "examining the physical security of the development environment of the TOE"[25].

The CC standard presumes fully secure operational environment, since it is not covered in the evaluation process. The evaluation process takes into account whether STs of the TOE are sufficient and whether SARs imposed by the STs are met[25]. The STs can be constructed through defined operations by the standard, which are:

- iteration: "allows a component to be used more than once with varying operations"[25];
- assignment: "allows the specification of parameters"[25];
- selection: "allows the specification of one or more items from a list"[25];
- refinement: "allows the addition of details"[25]

In other words, the Common Standard uses a formal language for describing its respective units. All Common Criteria requirements follow this structure - class, family, component, element; where element is the most isolable unit of structure, since a greater division would not bear any meaning[26].

STs can also have dependencies on each other. Because STs are more of a single isolable unit, although they can have dependencies as said previously, they can be grouped into hypernyms. In the context of the CC standard, these hypernyms are referred to as Packages and Protection Profiles (PPs)[25]. As defined in the standard, a package "is a named set of security requirements"[25] and is either "a functional package, containing only SFRs"[25] or "an assurance package, containing only SARs"[25]. Here "or" has to be understood as exclusive disjunction.

However, a PP, as stated in the standard, "PP is intended to describe a TOE type"[25]. So a logical interpretation of this statement might be that a PP accompanies more general security requirements than a package which is composed out of more specific STs tailored to the particular TOE. So for the purpose of this thesis, one could understand that a PP is incorporated by a "government or large corpora-

tion specifying its requirements as part of its acquisition process"[25] (US government PP) and a package by the developer of the specific TOE (Red Hat creating a set of STs incorporating the US government PP for the company's product)[25].

As for the conformity of the the PPs and packages, the standard "allows PPs to conform to other PPs, allowing chains of PPs to be constructed"[25]. The PPs and STs have to provide a conformance claim, which can either be conformant (claiming conformance to the respective CC parts or other PPs) or extended (claiming conformance to respective CC parts or other PPs with extended features)[26]. PPs can also be subdivided into Base-PPs and PP-Modules. The PP-Module use already certified Base-PPs and either builds up on them or tailors the specific requirements. A PP-Configuration is the result of the combination of a PP-Module with the Base-PPs[25].

One could point out the importance of a specific type of already pre-defined packages, which are the Evaluation Assurance Levels (EALs). EALs consist of seven evaluation levels hierarchically ordered, where each greater EAL provides greater assurance of security[28]. EALs are used for instance in the FIPS 140-2 standard.

2 The oVirt virtualization platform

2.1 oVirt Full Virtualization Architecture

Full Virtualization

Before addressing the actual architecture of the oVirt platform, a few words on the full virtualization technology, which the oVirt platform builds upon, should be laid out. Virtualization is a type of technology that can take a hold of the bare metal resources one has and distribute it across different users or environments. One could define virtualization as a simulation of software and/or hardware upon which other software runs. The term virtual machine (VM) is connected to virtualization and widely used to virtualize operating systems. Virtualization is being used in three major domains. These domains include application virtualization, operating system virtualization and full virtualization. The architecture of the oVirt platform incorporates the full virtualization technology[43].

Full virtualization is based on virtual hardware. Virtual hardware includes physical and virtual resources. This virtual hardware is then used to run virtual machines which usually take the form of virtualized operating systems. Virtualized operating systems in full virtualization are called guest operating systems. These guest operating systems are managed by a hypervisor. Hypervisor acts as an intermediary between physical resources such as CPU, memory, etc and the guest operating systems. Hypervisor can be run either on top of the bare metal - this means installed directly on hardware - or on top of a host operating system - this means that the hypervisor is installed on an operating system such as Windows, Linux or MacOS. In the context of the operating system the hypervisor is just another software that is running its own processes[43].

Hardware's physical platform provides interfaces that are needed for a non-virtualized operating system to run. These physical interfaces are being also used in virtualized environments. Main use of the virtualization of hardware is in networking and storage. In the context of networking, the hypervisor can provide network interfaces to the guest operating systems which can be virtual, physical or both and can also provide them with different forms of network access (bridging, NAT, host only). Hypervisors are able to implement virtual network devices such as switches. These virtual network components can then be assembled into a virtual network that can also be used for general networking purposes of the guest operating systems. Generally speaking the guest operating systems communicate within the virtual network created by the hypervisor but they can be provided with a physical network interface and communicate outside of the virtual network. Hypervisors also have to

virtualize disk storage. This is accomplished through the use of a disk image. Full virtualization also has two major use-cases - server and desktop virtualization[43].

The full virtualization architecture of the oVirt platform

The oVirt virtualization platform (oVirt in short) is an open-source community project that implements full virtualization technology[22]. The difference between Red Hat Virtualization (RHV) and oVirt is that RHV is a stable release and oVirt is the upstream. In other words when oVirt gets updated or any change occurs, this change also propagates to the corresponding downstream product, in this case RHV[18]. This is because RHV is an open-source project, that means that the source code of the product is freely available and the product as well, but with no support from the company that develops it (Red Hat in this case)[39].

Furthermore the Red Hat Virtualization builds on Red Hat Enterprise Linux (RHEL)[15]. Here another important clarification needs to be done. The oVirt platform supports deployment with either CentOS, oVirt node or RHEL hosts[17]. The CentOS operating system used to be a downstream product of the RHEL operating system. Nowadays before RHEL gets a feature pushed into production, this feature is implemented in the CentOS Stream project. Either way, because CentOS is an operating system based on RHEL, and the same can be said about the oVirt node too, in the context of this thesis both shall be declared as RHEL based systems. The term RHEL based system is an umbrella term describing operating systems that build up on top of the RHEL project or are an upstream project[24].

The oVirt platform's architecture comprises of four main components, which in a demonstrative manner are[23]:

- virtualization manager; which according to the documentation is a "service that provides a graphical user interface and a REST API to manage the resources in the environment"[23] and in the case of the thesis is installed on a physical or virtual machine running CentOS Stream;
- hosts; CentOS stream and oVirt node hosts are the supported types of host and as specified in the documentation: "hosts use Kernel-based Virtual Machine (KVM) technology and provide resources used to run virtual machines"[23];
- shared storage; which as specified in the documentation is "used to store the data associated with virtual machines"[23];
- data warehouse; which as specified in the documentation is "a service that collects configuration information and statistical data from the Manager"[23]

In the context of the full virtualization technology, which was presented in the previous section, an oVirt host is nothing more than a hypervisor that manages the resources of the physical hardware and hosts guest virtual machines from those re-

sources. The oVirt hosts are usually put into a cluster, so in other words the whole cluster behaves as a hypervisor. The individual hosts or the cluster itself are managed by the Manager[23]. The Manager can run in slightly different environments depending on the type of deployment of the oVirt platform[16].

The oVirt platform maintains two types of deployments; as a self-hosted engine or as a standalone Manager. In the case of the self-hosted engine deployment, the Manager as according to the documentation "runs as a virtual machine on self-hosted engine nodes in the same environment it manages"[15].

There are a few conditions on the minimal setup of a self-hosted engine deployment - a Manager hosted as a virtual machine on the self-hosted engine nodes, a cluster of at least two self-hosted engine nodes (hosts) and a storage server. In the case of the standalone Manager deployment, the Manager runs on baremetal or a virtual machine in a completely different environment[15].

A few conditions are also accompanied with this type of deployment - a Manager that is deployed as stated previously and also there is a need for the Manager to run on RHEL 8 (CentOS Stream 8 in the case of this thesis), at least two hosts and a storage device[15].

If one desired to understand the oVirt technology more deeply, one can consult the Technical Reference of the Documentation to learn all the different parts running inside the host, the layered structure of the host, virtual networking in oVirt and such. For the purpose of this thesis the main focus shall not be the whole architecture of the host but the baseline of the host's layered architecture, which is the RHEL operating system.

2.2 Implementations of security standards in oVirt

As stated in the previous paragraphs, oVirt is an upstream product of the Red Hat Virtualization. In the context of security standardization, the standards that RHV is certified to be compliant with are only applicable to the product itself. According to this fact a very distinctive statement has to be made - just because RHV is certified to be compliant with certain security standards, it by no means guarantees that the oVirt platform is also certified to be compliant with these specific standards. For the purpose of this thesis, two terms for security standard compliance shall be used - strict compliance and soft compliance.

In the terms of the strict compliance if an entity is ought to implement a full virtualization technology platform on their infrastructure and is ought to be compliant with a certain security standard that the full virtualization platform is compatible with therefore certified with, the entity has to use the same operational environment as stated in the security documentation in order to be strictly compliant.

In the terms of the soft compliance if an entity is ought to implement a full virtualization technology platform on their infrastructure and is ought to be compliant with a certain security standard that the full virtualization platform is compatible with therefore certified with, if the entity chooses to use an upstream product of the particular product that guarantees the needed compliance, then the entity's configuration can be evaluated only as soft compliance, since the product shares the same security countermeasures as the downstream product, but because it is not the same operational environment, thus it is not strictly compliant.

If the above definitions were to be used with the Red Hat Virtualization, then if the product is certified to be compliant with certain security standards, then the particular product with the particular operational environment (Red Hat Enterprise Linux) has to be used in order to achieve full compliance. In other words deploying oVirt with CentOS Stream 8 hosts means that this configuration is not strictly compliant, albeit the configuration shares the same source code and is of the same implementation, but because an independent laboratory did not certify this configuration, it can only be evaluated as soft compliance.

One also has to distinguish between compliance and compatibility with a security standard. And in the context of compliance there are two different possible states that the product and the security standard can be in. First being that the product was certified, meaning that the whole process of certification had undergone. In this case the whole product claims compliance. This is different to the second state, where only a certain configuration of a product had undergone certification, meaning only a specific configuration of a product claims compliance. And this is only the case of compliance. Compatibility with a security standard means that the product is compatible with the standard, meaning the product implements the security requirements defined in the standard, but is not officially certified. Compatibility can also be measured, since there is no need in strict compliance, one can claim that a product is n% (n is a positive integer on the scale of 0-100) compatible with a specific standard. The two terms are sometimes used interchangeably and in specific cases compatibility and compliance with a certain security standard might mean the same. This is the case of self-assessed conformance to a security standard by a vendor. When it comes to self-assessed conformance, this claim by the vendor will not be verified by an independent laboratory.

The oVirt platform shall be analyzed in the context of the three mentioned standards - FIPS 140-2, DISA STIG and Common Criteria. The focus here shall be on the compliance and compatibility of the platform with these standards. Since a very important link between RHV and RHEL exists, the compliance/compatibility problematic shall address this link as well. In some cases it acts as a ground for compliance/compatibility with the particular standard.

FIPS 140-2 and oVirt

Red Hat, as a vendor, has submitted five cryptographic modules to the CMVP validation program for the FIPS 140-2 certification. These cryptographic modules include GnuTLS, NSS, Kernel Crypto API, libgcrypt and OpenSSL[34]. All of these cryptographic modules are certified to be compliant with the security level 1[40]. All of the modules were validated with this following configuration:

Operational Environment[41]			
Manufacturer	Model	Processor	Operating System
Dell	PowerEdge R430	Intel(R) Xeon(R) E5	RHEL 8

One could argue that since these cryptographic modules were validated on the RHEL 8 operating system and not the Red Hat Virtualization platform, then the Red Hat Virtualization platform is inherently not compliant with the standard because of this fact. And this is not true. Although the cryptographic modules were validated on the RHEL 8 operating system and not the RHV platform, the RHV platform is still compliant with the FIPS 140-2 standard[41]. This is because the RHV platform is a layered product built on the RHEL operating system. In other words the RHV platform and the RHEL operating system share the same binaries of the validated cryptographic modules. The compliance with the standard is therefore preserved. This is defined in all security policies of the validated cryptographic modules, specifically in the section on operational environment and applicability, where it states that "The Red Hat Enterprise Linux operating system is used as the basis of other products which include but are not limited to ..."[41] and then it continues with the list of the specific products where RHV is one of the listed products. The preservation of the compliance is then supported by declaring that "compliance is maintained for these products whenever the binary is found unchanged"[41].

It is also important to explain that the security policies exist for the specific versions of the binaries of the cryptographic modules[41]. Therefore if one would run different versions of the binaries on their RHEL 8 system, then one would find themselves not being compliant with the standard. The entities that need to be compliant with the FIPS 140-2 standard and ought to use the RHEL operating system or any other operating system in this case, need to take this issue into account and perform a risk analysis on this issue.

Now here comes the problem of the difference between compliance and compatibility with a security standard. As stated previously, since the cryptographic modules that RHEL 8 uses were validated through the CMVP program, thus are FIPS 140-2 compliant, and since the same binaries of the validated cryptographic modules are used in the RHV platform as well, the RHV platform is also compliant with FIPS 140-2. Now this can not be extrapolated to the oVirt platform. The

oVirt platform is an open-source community product and despite the fact that oVirt and RHV share basically the same source code, the compliance with the FIPS 140-2 standard is only limited to the cryptographic modules deployed in the RHEL operating system and the products that build their architecture on RHEL as explained earlier on. So the conclusion here is that the oVirt platform is not compliant with the FIPS 140-2 standard.

Notwithstanding the fact that the oVirt platform is not FIPS 140-2 compliant, it does not mean that both the oVirt and RHV do not share the same binaries of the same validated cryptographic modules. In fact, they do share the same binaries. And since the configuration of the oVirt platform used in this thesis is oVirt deployed on CentOS Stream 8 hosts, the CentOS Stream operating system shares the same binaries of the validated cryptographic modules with the RHEL operating system. This means that the oVirt platform and CentOS operating system are both compatible with FIPS 140-2 standard. Compatible in the sense that they should provide similar security in terms of the cryptographic modules.

Coming back to the fact that the configuration of the system is compliant only if the configuration contains exactly the same version of the binaries as declared in the respective security policies, the configuration that would include the oVirt platform with CentOS hosts and exactly the same version of the binaries as declared in the security policies, such a configuration is not FIPS 140-2 compliant, but can be evaluated as being compatible with the FIPS 140-2 standard, thus providing similar level of security in the context of the cryptographic modules used in the system. So if an entity has to be compliant with the standard, then the entity should not employ such a configuration. Though if an entity does not in fact have to be compliant with the standard, the entity can be provided with a similar level of security through employing a configuration containing oVirt and CentOS.

Since use of the validated cryptographic modules in the configuration of this thesis relies on the bridge between oVirt and CentOS, which then relies on the bridge between RHV and RHEL, the following section shall focus on the implementation and use of the validated cryptographic modules in the RHEL operating system.

FIPS 140-2 and Red Hat Enterprise Linux 8

The RHEL operating system provides a special interface for the management of the validated cryptographic modules. The documentation refers to this special interface as the system-wide cryptographic policies. The validated cryptographic modules are a part of a particular component of the RHEL operating system called the core cryptographic components which is a component responsible for everything concerning cryptography. The core cryptographic components also contain non-validated

cryptographic modules. These core cryptographic components, which the validated cryptographic modules are a part of, can be manipulated with by the Crypto Officer through the use of system-wide cryptographic modules. If not specified, the RHEL's core cryptographic components do not operate in a mode that would satisfy the FIPS 140-2 requirements in terms of the usage of non-NIST-approved cryptographic algorithms[36].

To get the system to set the validated cryptographic modules to use the NIST-approved cryptographic algorithms, the Crypto Officer has to change the system-wide cryptographic policy. The system-wide cryptographic policies include four modes of operation - LEGACY, NORMAL, FUTURE and FIPS. The LEGACY mode is used for backwards compatibility with the older versions of the RHEL operating system. The NORMAL mode is designed to respond to the threats that are possible today, meaning the mode is using cryptography setting that is resistant to current threats. The FUTURE mode sets the mode of operation of the cryptographic modules to be resistant to possible threats in the future. This for example includes setting the length of the RSA keys to 4096 bits. Lastly the FIPS mode provides the employment of the security requirements on the use of NIST-approved cryptographic algorithms in the validated cryptographic modules[36]. The particular changes in the cipher suites and protocols that the FIPS system-wide cryptographic policy employs are described in the following table:

RHEL FIPS system-wide cryptographic policy[36]	
cipher suite or protocol	usage
IKEv1	no
3DES	no
RC4	no
DH	min. 2048-bit
RSA	min. 2048-bit
DSA	no
TLS v1.0	no
TLS v1.1	no
SHA-1 in digital signatures	no
CBC mode ciphers	yes
Symmetric ciphers with keys $<$ 256 bits	yes
SHA-1 and SHA-224 signatures in certificates	yes

In addition to the table, the FIPS policy also disables the use of RSA key exchange, the use of X25519, X448, Ed25519 and Ed448 eliptic curves and the Chacha20-Poly1305 algorithm[40].

In order to allow the FIPS policy, the designated user has two possible ways of executing such task. That is either by installing RHEL 8 with FIPS policy already enabled or enabling FIPS policy post-installation. The first one is achieved by changing the value of the fips kernel parameter to 1. The seconds is achieved via a comman-line tool called fips-mode-setup. The fips-mode-setup cli tool simply sets the system-wide cryptographic policy to the desired FIPS policy[36]. This is done through the use of the enable parameter like so:

Listing 2.1: Enabling FIPS policy via fips-mode-setup

```
# fips-mode-setup --enable
Setting system policy to FIPS
FIPS mode will be enabled.
Please reboot the system for the setting to take effect.
```

As can be seen from the output of the designated cli tool, the system needs to be rebooted in order for the change to take effect. Whether the FIPS policy had been set can be checked via the check or is-enabled parameter like so.

Listing 2.2: Checking whether FIPS policy had been enabled

```
# fips-mode-setup --check
FIPS mode is enabled.
# fips-mode-setup --is-enabled
1
```

The is-enabled parameter ouputs 0 if FIPS policy has not been set or 1 for the enabled state.

For the strict compliance's purpose, versions of the particular binaries of the validated cryptographic modules can be checked by the use of the rpm cli tool.

Listing 2.3: Checking binary's version of the validated cryptographic modules

```
# rpm -qa gnutls
gnutls-3.6.16-4.el8.x86_64
```

Proposal for FIPS policy enhancements

The FIPS 140-2 standard has been superseded by the FIPS 140-3[3]. That does not mean that the superseded version of the standard is now deprecated. The cryptographic modules that had been certified for the FIPS 140-2 standard shall, as stated by NIST, "remain active for 5 years after validation or until September 21, 2026"[6]. As of the mentioned date, all the validated cryptographic modules shall be moved to the historical list[6]. NIST then states that "even on the historical list, CMVP supports the purchase and use of these modules for existing systems"[6].

The main proposal for the improvement of the RHEL FIPS system-wide cryptographic policy is therefore for the RHEL core cryptogprahic components to implement the FIPS 140-3 standard in order to tackle new security threats to cryptographic modules.

Evaluation criteria for FIPS 140-2 compliant/compatible configuration

The evaluation criteria might take many forms depending on the goal such evaluation strives to achieve. A possible way of assessing compliance is to take the steps laid out in the DTR and follow them. Although this is a very rigorous way of executing such assessment, it has already been done by the NVLAP accredited laboratory that tested the cryptographic modules, used in the RHEL operating system, against those criteria. In other words the goal of assessing whether an oVirt or RHV configuration is FIPS 140-2 compliant, is not to re-test or re-validate the cryptographic modules but rather to verify that they are used by the system in a manner supported by the documentation and so supported by the security policy. This approach then results into these criteria for determining whether an oVirt environment is FIPS 140-2 compliant:

1. the platform installed is RHV
2. hosts run RHEL
3. FIPS mode is enabled on the hosts
4. FIPS system-wide cryptographic policy is enabled on the hosts
5. FIPS system-wide cryptographic policy is applied on the hosts
6. kernel parameter fips_enabled is set to 1 on the hosts
7. versions of the cryptographic modules' binaries are equal to the ones used in their security policies

The first and second items correspond to the operational environment that cryptographic modules were certified in. In order to claim strict compliance, these two conditions must be true. If the configuration runs oVirt on CentOS Stream hosts, the configuration shall not be strictly compliant but can be assessed further for compatibility or soft compliance.

The third item indicates that the cryptographic modules are in the FIPS mode which means that they are using NIST-approved cryptographic algorithms, use cryptographic keys of length defined in the standard and so on.

The fourth and the fifth item ensures that the system-wide cryptographic policy is also set to the FIPS mode and is applied, meaning it is in operation.

The sixth item is there to assess whether the cryptographic modules are running power-on self-tests as defined in the standard.

The last item on the list is to verify that the versions of the cryptographic modules' binaries are the ones which were FIPS 140-2 certified, therefore compliant.

DISA STIG and oVirt

The implementation of the DISA STIG in the oVirt platform is quite problematic, since there is no official DISA STIG for oVirt or RHV[1]. The whole concept of DISA STIG-inspired security hardening of the RHV platform is based on the possibility of deploying the platform with RHEL hosts. This is because an official DISA STIG exists for the RHEL operating system[1]. Since RHV can also be deployed standalone with RHEL hosts which "serve as an ordinary host in a Red Hat Virtualization cluster"[23], the official RHEL STIG can be applied to the hosts. This combination is a supported hardening method for Red Hat Virtualization[38].

The RHEL STIG is integrated into the NIST National Checklist for Red Hat Enterprise Linux 8.x. When it comes to enforcing the official RHEL STIG on the RHV hosts, the strict compliance of the host to the STIG can be evaluated and is therefore preserved. The documentation for security hardening of Red Hat Virtualization also specifies a possible way of hardening its hosts with a custom security profile called *[DRAFT] DISA STIG for Red Hat Virtualization Host (RHVH)*[38]. If this profile is chosen for the RHV hosts, then the deployed infrastructure cannot be considered as compliant with the RHEL STIG. The DISA STIG for RHVH is tailored to the specific flavor of the RHEL operating system called RHVH. The RHVH is a minimal operating system developed solely for the purpose of running on virtualization hosts for RHV[23]. Since there is no official RHVH STIG published on the DISA's website, no claim for compliance can be made. In view of this fact, only the combination of RHEL hosts deployed in the RHV infrastructure can be considered as compliant with the RHEL STIG.

Strict compliance can not be evaluated for the oVirt with CentOS Stream hosts configuration. The NIST RHEL Checklist states in the Target Audience section that the "content is applicable for Red Hat Enterprise Linux 8.x"[44] and that the "content has not been tested, approved, or supported, on derivative operating systems such as CentOS"[44]. Here only a compatibility with the standard can be determined since CentOS Stream shares the same source code, packages, binaries and so on with RHEL.

To verify that the RHEL hosts in a RHV deployment are compatible with the RHEL STIG security profile, auditing tools from the OpenSCAP project provided by Red Hat should be used. OpenSCAP shall be introduced in the coming subsection.

OpenSCAP workflow for RHEL STIG

The OpenSCAP project consists of system security configuration settings auditing and vulnerability assessment tools and SCAP profiles[45]. Most importantly OpenSCAP has been validated through NIST SCAP Validation Program and is therefore a SCAP validated tool[38]. OpenSCAP consists of four main components[37].

- SCAP Workbench; a graphical utility for performing configuration and vulnerability scans capable of generating reports,
- OpenSCAP library; a command-line utility also providing a way of executing configuration and vulnerability scans capable of conducting reports and guides pointing out steps which can be taken in order to be compliant with a certain security requirement,
- SCAP Security Guide (SSG); a package containing latest security policies as a SCAP content designated to Linux machines implemented in accordance with Payment Card Industry Data Security Standard (PCI DSS), STIG and United States Government Configuration Baseline (USGCB),
- Script Check Engine; a tool capable of of transforming SCAP content into scripting languages like Python or Bash

In order to check compliance of a system with certain security requirements the scanning tool provided by OpenSCAP library shall be used together with the SSG which provides the actual SCAP content that is a necessary component to be used for the compliance check[37]. Both of these components have to be installed in the first step.

Listing 2.4: OpenSCAP installation

```
# yum install -y openscap-scanner scap-security-guide
```

After installation of the SSG, the SCAP content is to be found in a dedicated directory[1] in the form of SCAP source data stream. The data streams contain the actual checklists (security profiles) that are to be used in the evaluation. The OpenSCAP audit tool can be found under oscap and is installed into the user's binaries. For compliance evaluation against a chosen security profile, the respective data stream needs to be provided as input to the oscap tool together with the particular security profile specified via the profile parameter in the oscap tool[45].

The RHEL STIG is to be found in the ssg-rhel*-ds.xml[2]data streams. The official STIG is of course integrated into the profile in this data stream. Besides this data stream, the RHEL STIG can be accessed from a xccdf file also stored in the same directory and other formats supported by the SCAP protocol.

[1] /usr/share/xml/scap/ssg/content

[2] The * character here symbolizes an asterisk sign.

Listing 2.5: OpenSCAP RHEL STIG

```
# cd /usr/share/xml/scap/ssg/content
# oscap info ssg-rhel8-ds.xml
Document type: Source Data Stream
...
    Status: draft
    ...
    Profiles:
      ...
      Title: [DRAFT] DISA STIG for Red Hat Enterprise Linux 8
        Id: xccdf_org.ssgproject.content_profile_stig
...
```

For a RHEL host to be scanned for compliance, other necessary subcommands and parameters must be provided. The oscap tools provides eight subcommands - ds, oval, xccdf, cvss, cpe, cve, cvrf and info. For this moment, the xccdf and oval subcommands are important for the actual execution of the compliance scan. The xccdf subcommand specifies that a checklist in xccdf format shall be provided. The eval subcommand indicates that a system configuration shall be evaluated against the provided security profile. As mentioned previously the profile parameter is used for security profile specification. There can also be other optional arguments provided to the oscap tool such as the results parameter which specifies the path of where the results from the compliance scan should be saved. Finally, the data stream is to be provided as input to the oscap tool[45]. The final command for compliance scanning against the RHVH-DRAFT STIG could look like this:

Listing 2.6: OpenSCAP compliance evaluation against the RHEL STIG

```
# oscap xccdf eval \
--profile xccdf_org.ssgproject.content_profile_stig \
ssg-rhel8-ds.xml
```

The above listed command executes an evaluation against an XCCDF benchmark. The results of the evaluation scan are "printed to standard output stream"[45] for "each XCCDF rule within an XCCDF checklist[45].

Possible results for a single XCCDF rule are[45]:

- pass,
- fail,
- error,
- unknown,
- notapplicable,
- notchecked,
- notselected,
- informational,
- fixed

The result of an evaluation scan against the RHEL STIG would then look like this:

Listing 2.7: OpenSCAP RHEL STIG evaluation results

```
...
Title   Disable the secure_mode SELinux Boolean
Rule    xccdf_org.ssgproject.content_rule_sebool_secure_mode
Result  pass

Title   Enable the fips_mode SELinux Boolean
Rule    xccdf_org.ssgproject.content_rule_sebool_fips_mode
Result  pass

Title   Set SSH Client Alive Max Count
Rule    xccdf_org.ssgproject.content_rule_sshd_set_keepalive
Result  fail
...
```

When addressing the issue of system's compliance with certain security requirements, the remediation process of the potential defects that are to be found in a compliance check can be of use. The OpenSCAP project provides an options to generate remediation scripts of the found defects (fails) in the form of a Bash script or a Ansible playbook provided by the SSG.

The remediation scripts can be generated automatically and tailored to the specific security requirements that are not implemented on the system. To achieve this the oscap tool shall be used together with the generate and fix subcommands[37]. Lastly the fix type parameter should be specified, depending on whether one wants to use Ansible or Bash as the means of remediation.

Listing 2.8: OpenSCAP remediation process

```
oscap xccdf generate fix --fix-type ansible \
--profile stig --output stig-remediations.yml \
stig-results.xml
```

Proposal for STIG related enhancements

A relevant proposal for enhancement would be for Red Hat to collaborate with DISA on the creation of an official STIG that could then be listed on the DISA's website just like the RHEL STIG. This STIG would apply to the whole virtualization platform and could be called RHV STIG. This would make more sense since similar STIGs relevant to virtualization platforms have already been made[19].

Evaluation criteria for STIG compliant/compatible configuration

Since an official checklist for STIG compliance is provided RHEL in the OpenSCAP project, this checklist shall be used for assessing compliance/compatibility for the given oVirt/RHV configuration. The configuration of standalone RHV engine with RHEL hosts is equivalent to standalone oVirt engine with CentOS Stream hosts. This configuration shall be tested against the RHEL STIG. This approach then results into these criteria for determining whether an oVirt/RHV environment is STIG compliant:

1. the platform installed is RHV
2. hosts run RHEL depending on the deployment type
3. the oscap system configuration scan with the respective profile is successful

The first and the second items correspond to the requirements given by the Target Audience of the NIST National Checklists of the given profile from the SSG[44]. In other words if the configuration is found to be running oVirt on CentOS stream hosts, the configuration shall not claim compliance, but can be assessed further for compatibility with the given profiles if the third item turns out to be successful.

Common Criteria and oVirt

At present, the oVirt platform has not been issued a Common Criteria certificate. Since oVirt is a community project, it will most probably never get certified for Common Criteria. Though there is a certain assurance of soft conformance to the standard because RHV is Common Criteria certified. The RHV underwent the Common Criteria certification process in Europe under OCSI (Organismo di Certificazione della Sicurezza Informatica) as one of the Certificate Authorizing Schemes[30]. In order to proceed with the certification process, a Security

Target has to be made. The Security Target for Red Hat Virtualization is also available on the website of OCSI.

The RHV ST claims conformance to, as specified in the ST itself, "CC Part 2 extended and CC Part 3 conformant, with a claimed Evaluation Assurance Level of EAL2, augmented by ALC_FLR.3"[27]. The conformance claim specifies that the ST conforms to to EAL2 and does not make any conformance claims to any Protection Profile[27]. The EALs (Evaluation Assurance Levels) are intended to provide assurance that certain procedural security measures were or are being taken when developing the TOE. This as opposed to the PPs (Protection Profiles) cannot be evaluated by the means of a simple checklist. The EALs are controlled and verified by the accredited laboratories or related entities that inspect the procedural security of the company developing the specific piece of software. The PPs on the other hand are more of a checklist that can be easily automated[26]. The procedural approach is rather demanded in the European Union, whereas the so called checklist approach is rather demanded in the US where NIAP is the regulatory body[13].

The RHV ST is divided into five important sections. The first one being the introduction that primarily identifies what is the TOE of this specific Security Target. The second one is reserved for conformance claims which were explained previously. The third section specifies the security problem definition. This section is intended for threat modeling where the threat environment with the identified assets and threat agents is presented. It also defines the threats that are to be countered, assumptions of the TOE and organisation security policies. The fourth part explains the Security Objectives of the ST. The fifth part is there to explain which extended components the Security Target uses[27]. This is a space for vendors to add something specific to the Security Target of their product which may distinguish it from other products or make it more secure in a defined way.

Evaluation criteria for Common Criteria compliant/compatible configuration

The evaluation criteria for assessing compliance of a given oVirt/RVH deployment to the Common Criteria standard are based on the Security Target dedicated to the Red Hat Virtualization. The Red Hat Virtualization Security Target specifies all necessary security requirements that need to be met in order to claim compliance. The security requirements in the Security Target are represented as SFRs and SARs[27]. The following criteria for determining whether an oVirt/RHV environment is Common Criteria compliant were created:

1. the platform installed is RHV
2. the version of RHV is 4.3.17
3. hosts run RHEL

4. the version of RHEL is 7.9
5. all tests from the test suite must pass

The first, second, third and fourth item corresponds to the identification of the TOE that the Security Target specifies. In order to claim strict compliance, these to items must be true. If the configuration runs oVirt on CentOS Stream hosts, the configuration shall not be strictly compliant but can be assessed further for compatibility or soft compliance. The same goes for configurations with different versions of RHV and RHEL. If the version of RHV is different from 4.3.17, the configuration shall not be compliant. If the version of RHEL is different from 7.9, the configuration shall not be compliant.

The last item on the list is to verify that the SFRs specified in the RHV ST are implemented in accordance with the ST itself. This is done through the means of functional testing. In order to execute such functional testing, a testing suite provided by Red Hat shall be used. More on this shall be explained in the subsection explaining the implementation of the role that manages the compliance evaluation process to the Common Criteria standard.

Proposal for Common Criteria related enhancements

The only available proposal for Common Criteria related enhancement is for Red Hat to schedule a more frequent schedule for Common Criteria certification. Since the current and only ST for RHV has its TOE identified as RHV of the version 4.3.17 with RHEL hosts of the version 7.9[27], it misses a lot of new features, bug fixes and other betterments that the newer versions of RHV and RHEL provide.

3 Practical implementation

3.1 Automated oVirt deployment

The manual process of oVirt deployment is not suitable for the fast-paced approach to software development. Because the goal of this work is to determine through the use of a software tool the state of the deployed oVirt infrastructure and based on the state to evaluate whether the infrastructure is adhering to the security requirements it is being tested against, it requires a large number of re-configurations, re-installations and other changes to the sole infrastructure of the platform. These changes would create a large over-head if they were to be done manually. Nowadays the software development process accompanies a discipline called Development & Operations (DevOps) which integrates this idea of an automation-first approach to continuous integration, continuous delivery, and continuous deployment (CI/CD)[60].

The sole implementation of the CI/CD process is not managed only by a single technology but rather is distributed across a number of technologies all working in an orchestrated manner.

Automation setup

For the purpose of this thesis, for the orchestrator of the CI/CD process the Jenkins project is used.[1] Because Jenkins in this case is used as an executor of automation, one needs another layer of automation which would allow for an automated oVirt deployment and the configuration of the operating systems upon which the platform shall be deployed. This is done through the Ansible technology.[2] The main goal for Ansible in the Automation setup of the oVirt platform is to prepare the hosts before the actual installation of the platform which in the case of this thesis is to enable FIPS 140-2 compliance mode or the DISA STIG security profile. The whole process of oVirt installation is then also fully automated by Ansible with the use of oVirt Ansible Collection.

The oVirt Ansible Collection is a collection[3] of Ansible modules[4] in which each of

[1] *Jenkins is a self-contained, open source automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.* More on Jenkins.[73]

[2] *Ansible is a radically simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs.* More on Ansible.[68]

[3] *Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. As modules move from the core Ansible repository into collections, the module documentation will move to the collections pages.* More on Ansible collections.[65]

[4] *Modules (also referred to as "task plugins" or "library plugins") are discrete units of code that*

the module represents a separate domain of actions applicable on the oVirt platform, i.e. a Module is capable of managing a certain aspect of the configuration and management of the oVirt platform's components. For instance the module called ovirt.ovirt.ovirt_user is used to "manage users in oVirt/RHV"[52].

Because the oVirt platform can be deployed in two ways (referenced in The full virtualization architecture of the oVirt platform), both of them have to be supported by the Automation setup and also by the verification script. The individual options are to be viewed and used in the user interface of Jenkins when triggering the automation Pipeline.[5] The configuration files for the automation pipeline are stored in the YAML[6] file format. The Ansible playbooks[7] and roles[8] which configure the oVirt setup and the hosts are stored in a Source Code Manager (SCM) software. In the case of this thesis the Gerrit[9] tool incorporating the Git[10] SCM is used.

Last component of the Automation setup is a way of managing provisionable iso files for the RHEL based operating systems running on the hosts or in the virtual environment provided by the oVirt platform. This is done trough the use of Foreman as a manager of both the iso files and the physical machines that get provisioned.

3.2 Architecture of the compliance verification script

High level overview

The basic principle behind the script is to check a given deployed oVirt/RHV infrastructure for compliance with the particular standards that are covered in this

can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote target node, and collects return values. More on Ansible modules.[62]

[5] *A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released.* More on pipelines.[72]

[6] *YAML is a human-friendly data serialization language for all programming languages.* More on YAML.[67]

[7] *Ansible Playbooks offer a repeatable, re-usable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications.* More on Ansible playbooks.[63]

[8] *Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure. After you group your content in roles, you can easily reuse them and share them with other users.* More on Ansible roles.[64]

[9] *Gerrit provides a framework you and your teams can use to review code before it becomes part of the code base.* More on Gerrit.[66]

[10] *Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency[61].*

thesis. In order to check for compliance the script has to check for specific parameters. These parameters are represented in system configurations of the machines deployed in the oVirt/RHV infrastructure. For the script to determine whether a certain parameter is in accordance with the given standard a database with formal definitions of the parameters has to be provided. The formal definitions have to be thought of as rules that specify under which conditions a given parameter is correct. A formally defined rule has to include 3 variables, these are:

- a variable defining which precise system configuration is to be checked,
- a variable defining precisely how to check that given system configuration,
- a variable defining the desired output of the system configuration check

When such a database with the formal definitions is provided, the script can use the variables to determine compliance. Depending on the architecture of the part of the script that deals with the particular standard, all three rules can either be defined only in the database or partially in the database and the remaining parts in the script itself. In the second scenario the what to check and how to check variables would usually be defined in the script and only the variable containing the desired output would be defined in the database of the formal definitions.

The basic flow of the script would then consist of three main entities. The first one being the script, the second one being the database with the formal definitions and the last part being the deployed oVirt/RHV infrastructure. The flow starts with the script since the script is the executor and the orchestrator of the compliance evaluation process. The script will use the what to check variable to determine what specific system configuration it has to check within the oVirt/RHV infrastructure. Then it will use the how to check variable to determine how it should execute such an action. The script will then establish communication with the deployed oVirt/RHV infrastructure and will start querying it for the system configuration parameters in the formally defined way. Once the script receives the system configuration parameters, it will compare the values of those parameters with the desired output values stored in the database. If the compared values are equal, then the rule passes and is recognized as correct. Otherwise the rule does not pass and is recognized as incorrect. The script then repeats this process for every rule for each of the given standards defined in the database until the list gets exhausted.

Once the list gets exhausted the script evaluates the compliance to the given standard for the deployed oVirt/RHV infrastructure. The evaluation process does not take into account partial compliance. If the deployed infrastructure fails on any rule, it is then considered as not compliant. The script in its final stage generates a report containing the final results with all the evaluated rules that it was checking. This basic flow is graphically depicted in this diagram.3.1

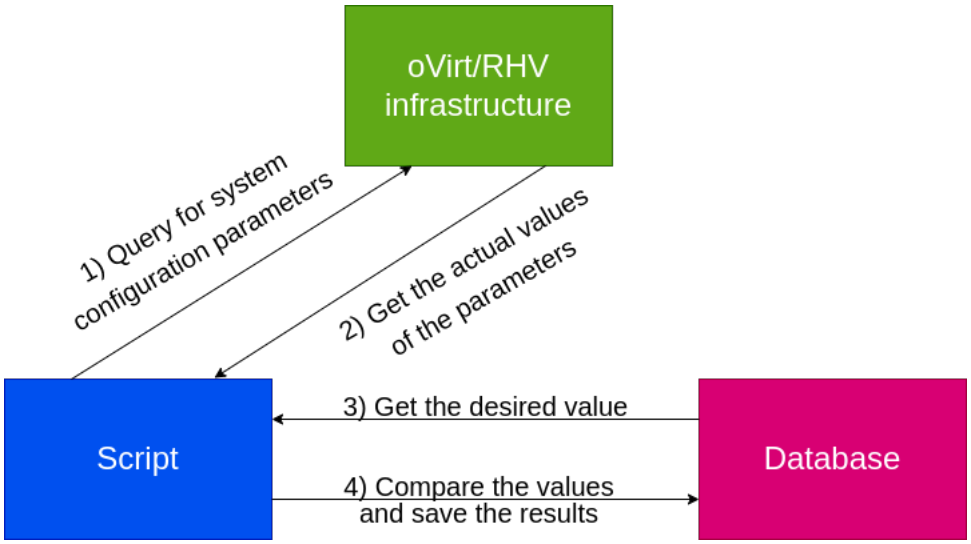


Fig. 3.1: Basic flow of the application

High level architecture

The architecture of the compliance verification script is modular. This means that for each of the security standards a separate module is designated. The purpose of these modules is to implement the functionality that enables the process of compliance verification to the specific standard. This means that every module implements the basic flow and holds its own database with the formal definitions that are needed for the evaluation process. There are also other modules that the script uses which are designated to manage processes that are not strictly related to the compliance verification process. These processes would include managing communication with the oVirt/RHV infrastructure, basic information gathering about the deployed infrastructure and checking and resolving basic dependencies. The modular aspect of the high level architecture is best depicted in the following diagram. 3.2

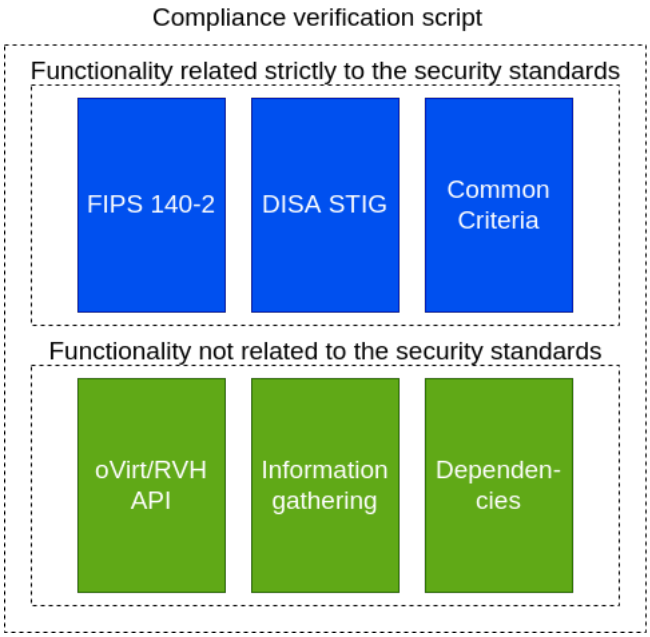


Fig. 3.2: High level architecture of the script - modular aspect

Since the architecture of the script is modular, it needs an orchestrator and an executor of those modules. Such a part of the script handles the internal communication between the modules and parses the output. It also delegates and determines on which part of the oVirt/RHV infrastructure the how to check variable will be executed. This layer also provides an outer interface for communication which would include an actual user that accesses this script. In other words the modules implement the specific functionality and the execution and orchestration layer only includes those modules, places them in the desired order, most importantly executes them in that order and lastly provides an interface for customizing the execution process of the script. The execution and orchestration layer is not the outer most layer that would provide an interface for the use of actual users of the script (evaluators). The layer could be used directly by users but would by no means provide enough abstraction from the technical implementation.

The outer most layer that is designated to provide possibly enough abstraction from the technical implementation is the user interface layer. The user interface layer gives the user a set of defined states that customize the execution process of the script. These defined states correspond to the modules that the script makes use of for the compliance verification process. This means that the user interface layer provides four states, these are:

- verify compliance only to the FIPS 140-2 standard,
- verify compliance only to the DISA STIG standard,
- verify compliance only to the Common Criteria standard,

- verify compliance to all three standards

The high level architecture of the script is modular and also layer based. Every layer provides a defined functionality to the next layer. The execution and orchestration layer takes up the functionality provided by the modules and the user interface layer provides an interface the change the course of execution of the script. The layer based aspect of the script is depicted in the following diagram. 3.3

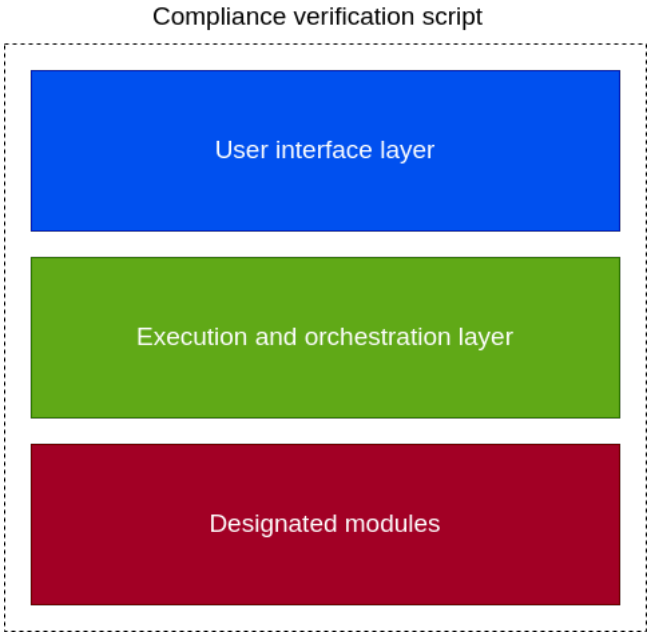


Fig. 3.3: High level architecture of the script - layer based aspect

Architecture of the roles

The modules developed for the use in the compliance verification script follow a standardized skeleton in order to make them re-usable. The re-usability aspect of those modules is key and gives an option to much broader integration possibilities, even outside the use of the script. This standardized skeleton is provided by the Ansible technology. The modular aspect of Ansible is given by the use of Ansible roles. Ansible roles are designed to manage a strictly defined functionality and nothing outside of this strictly defined functionality. Ansible roles can also be made to customize the execution process or to enforce a desired state on the entity that the strictly defined functionality manages. For the purpose of this work Ansible roles are not used to enforce states on the entities it manages. This is because for the implementation purposes of the security standards related modules, only a comparative operation on the collected states shall be executed. Firstly the state is fetched by the particular Ansible role and secondly this state is compared to the

desired state that is stored in the database with the formal definitions. Therefore no states have to be enforced.

The stated skeleton of Ansible roles is also the applied architecture for the security standards related modules. The skeleton follows this specific structure.

```
/ ..... root directory of the role
├── defaults .....
├── files .....
├── handlers .....
├── library .....
├── meta .....
├── tasks .....
├── templates .....
├── tests .....
└── vars .....
```

Not all of the parts of the skeleton is used by the modules of the script nor it is always desired to implement all of the parts in the modules. Nevertheless it is always on the developer of those modules to decide which of these parts shall be implemented. It is important to keep the whole structure of the skeleton, even though not all parts are used. This is done purely for formal purposes for other potential developers to be able to understand the module. Ansible will be able to work with the role, if at least one of those parts is included. The parts of the skeleton are instantiated as directories on a given file system.

The most important part of the skeleton is the tasks part. When Ansible attempts to work with the role, it tries to look for the tasks directory. This directory is intended as a container for executable parts of the role. The executable parts of the role provide the actual strictly defined functionality. The strictly defined functionality is represented by the functional pieces that the roles makes use of. These functional pieces are integrated into Ansible and are called modules. Ansible modules create yet another layer of abstraction from the underlying technology. They are even more granular than roles and provide an even more strictly defined functionality. Here the strictly defined functionality manages only a very narrow part of a sub-entity. Since roles manage whole entities, modules manage only parts of those entities. Here an entity refers to a host. In terms of Ansible, a host is a machine. A machine is either a bare metal one (a physical computer) a or a virtual one (a virtual machine). A sub-entity refers to a part of the host. This part of a host can be thought of a configurable part of an operating system that is installed on the machine. This configurable part is for instance a package manager.

The tasks directory is intended to hold files that make use of the modules integrated into Ansible and enriches them with logical functionality that Ansible itself provides. These files are expected to follow a specific format. This specific file format is the YAML format. Ansible always tries to find a file in the YAML format called

the *main.yml* which defines and provides the strictly defined functionality provided by the role. There can be more than one file in the tasks directory but if these files provide any functionality, they have to be imported or included in the *main.yml* file. If they are used in that file, Ansible can access them and execute them. These files (main.yml file included) are intended to hold and work with Ansible modules. The syntax language of Ansible is YAML, just like the file format that the files in this directory have to follow. Ansible can make use of logical, conditional and other operators that create a control mechanism for the strictly defined functionality of the role.

In the context of the script, the tasks directory is where a large part of the database with the formal definitions is implemented. The files in the tasks directory contain the what to check and most importantly the how to check variables. The what to check and how to check variables are represented as Ansible modules grouped into Ansible tasks together with logical and conditional operators that provide the strictly defined functionality needed for the compliance check. Every rule in the database with the formal definitions represent one particular security requirement for the given standard. Here the particular security requirement is represented as Ansible task. The task pulls the what to check variable either from the vars or the defaults directory or it's already contained in the task itself. The how to check variable is represented by the combination of a particular Ansible module together with logical and conditional operators and other functionality provided by Ansible again grouped into an Ansible task. The Ansible tasks are then grouped into a list of tasks which represent the set of security requirements for the given standard. The task list has a reserved name which is just *tasks*. An example of tasks is provided by Listing 3.1.

Listing 3.1: Example list of tasks

```
tasks:                                                          1
    - name: Check if ssh uses safe ciphers                      2
      module_to_check_safe_ciphers:                             3
        service: ssh                                            4
        check_safe_ciphers: yes                                 5
      register: safe_check                                      6
                                                                7
    - name: Determine if ssh uses safe ciphers                  8
      set_fact:                                                 9
        ssh_is_safe: "{{                                        10
            (safe_check == save_ciphers)|                       11
            ternary(true, false)                                12
        }}"                                                     13
```

As can be seen from the example, there is a dummy module called *module_to_check_safe_ciphers*. Every module, just like this dummy one, can be executed with a set of defined parameters that are represented by the keys that reside in the different suite in terms of indentation. In this example, these are the service and the check_safe_ciphers parameters. The register key only saves the output of this dummy module. It is a native functionality provided by Ansible and is used quite frequently in the implementation of the modules for the compliance verification script. In the following task of the example another module is used. This one in particular is not a dummy one but an actual module from Ansible. The second example task is just to show how a condition would be used. The condition here is provided by the ternary operator marked with the *ternary* keyword in the example.

The vars and defaults directories are intended to provide an option to customize the execution of the role. This is done through the use of variables. Depending on which container the variable is put into, different goal is being achieved. A variable put into the vars directory is expected not to have its value changed by an external entity. Usually the role depends on the preserved structure and value of the variables put into the vars directory. Ansible also assigns higher priority on those variables[69]. On the other hand the variables put into the defaults directory are expected to have its value changed by an external entity. The values of those variables are only defaults that can be changed depending on the needs of the entity working with the module. The variables put into the defaults directory can be viewed as optional variables. Both of those directories store files in the YAML format. And the same rules applies to them as for the tasks directory. The *main.yml* file is expected to exist. Other files with different names can reside in the directory but for them to be used by Ansible, they have to be included into the main.yml file.

When it comes to the script, the vars and defaults also implement the database with the formal definitions as introduced in the high level overview. In the contrast with the tasks directory, the vars and defaults are reserved to store the variables defining the desired output of the system configuration check. In some cases these directories also store the what to check variables. In this particular case where these directories store the what to check variables, they store just a value of the entity that is supposed to be checked. For example the variable could hold a name of a cipher that is to be evaluated. The tasks directory then pulls this information and actually uses it in a task that then defines the how to check variable. The variables that hold the desired output are used to be compared with an output of an action executed by the module within a task. The module executing such an action could for example be a module that fetches contents of a certain configuration file. The output of this action is then saved into memory and compared to the value stored

in the database.

In the context of the script, it does not make sense to hold any variables in the default directory, since the user's choices will not affect the sole execution of the role. The choices that the user can make are handled by the user interface layer and do not interfere with the roles themselves. In other words only the vars directory will be populated with the needed variables for the role to be able to properly execute the verification process. The following are contents of an example *main.yml* file inside the vars directory represented by the Listing 3.2.

Listing 3.2: An example of variables

```
allowed_ciphers:                                                    1
    - RSA                                                           2
    - AES                                                           3
allowed_key_lengths_for_allowed_ciphers:                            4
    -                                                               5
        name: RSA                                                   6
        length: 3072                                                7
    -                                                               8
        name: AES                                                   9
        length: 256                                                 10
```

In this example the concept of what to check variables and variables containing the desired output can be explained. Here the variable called *allowed_ciphers* represents a list of values containing names of first an asymmetric cryptographic system (RSA) and second a cryptograhic standard for symmetric encryption (AES). The idea behind this is that the how to check variables contained in the tasks directory will fetch these values (RSA and AES in this example) and use a module that provides the functionality to check whether these values are present. If they are present, then another module will fetch the contents of
the *allowed_key_lengths_for_allowed_ciphers* that is capable of checking key lengths of the cryptographic functions used by the system. In this example
the allowed_key_lengths_for_allowed_ciphers variable represents the variables that hold the desired output that were presented in the high level overview. The comparative operation gets executed and the security requirement gets evaluated.

The meta directory of the role's structure is supposed to hold metadata about the role itself. Usually the meta directory is used for external dependencies that the role has. The meta directory is also expected to hold files in the YAML format with the particular file called *main.yml*. The meta directory is not implemented by any of the modules used by the compliance verification script.

The files and templates directories have both a similar functionality but they

slightly differ. These directories are both expected to hold external files in non-specified format that are supposed to be transferred onto the entity (host) that the role manages. The difference is that the files directory contains files in its final state. These files are not expected to change, they are static. The files that reside in the templates directory are expected to change since they are considered only as templates. The contents of these files is rather dynamic and can be changed, depending on the template, during the course of execution of the role. This is accomplished by a templating engine that Ansible uses.

In terms of the compliance verification script, it does not use the files directory but it does use the templates directory. Here the templates directory serves the purpose of containing templates of the report that each of the module generates. These reports state whether the deployed oVirt/RHV infrastructure is compliant with that particular standard and then lists the criteria that it used for verifying its compliance. In terms of failure in some of these security requirements, the report shall contain the specific requirement and a hint on how to remediate this issue. The Listing 3.3 is an example of a template that is used to generate a report.

Listing 3.3: An example of a report template

```
rule: System uses RSA key length of 3072 bits                    1
{% if rsa_uses_allowed_key_length %}                             2
pass: true                                                       3
hint: null                                                       4
{% else %}                                                       5
pass: false                                                      6
hint: Use key length of 3072 bits                                7
{% endif %}                                                      8
                                                                 9
rule: System uses AES key length of 256 bits                     10
{% if aes_uses_allowed_key_length %}                             11
pass: true                                                       12
hint: null                                                       13
{% else %}                                                       14
pass: false                                                      15
hint: Use key length of 256 bits                                 16
{% endif %}                                                      17
```

This example shows two rules that correspond to the previous example that showed how the modules that the script uses work with the vars directory. The rule here is defined by three variables - rule, pass and hint. The rule variable holds the name of the particular rule. The pass variable hold the a binary value indicating whether the security requirement defined by this rule was satisfied or

not. Lastly the hint variable contains a possible way of remediating the failed rule in order for it to pass. The curly brackets together with the percentage symbol are used for the templating engine to determine which parts of the text is supposed to contain logical operators. In this example, an if else statement is written in the syntax of the Jinja templating engine that is used by Ansible. Let's assume that the *rsa_uses_allowed_key_length* is true and the *aes_uses_allowed_key_length* is false. The templating engine would then generate the following report represented by the Listing 3.4.

Listing 3.4: An example of a generated report from a template

```
rule: System uses RSA key length of 3072 bits
pass: true
hint: null

rule: System uses AES key length of 256 bits
pass: false
hint: Use key length of 256 bits
```

The handlers directory is supposed to contain handlers. Handlers are specific parts of the Ansible code that listen to defined events. When the defined event occurs, handler gets executed. The handlers directory is not implemented by any of the modules used by the compliance verification script.

The last directory of the role structure is library. The library directory is reserved for custom modules that the developer of the role chooses to develop for the use in the role. Although Ansible offers a wide range of already developed modules, in some cases it is desirable to create custom modules to handle a specific sub-entity. The library directory is expected to hold files in the .py format. These files are used by the Python programming language and it is the language that Ansible modules are written in. There can be more than one files, all of them representing a specific custom module. The name of the file in the .py format usually specifies the name of the module. This means that when this module is being used in the main.yml file in the tasks directory, it should be referenced by the name of the file in the .py format.

The security standards related modules of the compliance verification script in some cases make use of the library directory for custom module implementation. This is done because if the strictly defined functionality that the custom module is supposed to handle was to be achieved by the use of standard Ansible modules, it would create large implementation difficulties. In the context of the script the role implementing the security requirements of the FIPS 140-2 standard implements a custom Ansible module. The details of the implementation shall be explained in the

respective subsection about the FIPS 140-2 role.

When it comes to further customization options in terms of Ansible roles, another important part can be used. This part is called filter plugins and is also instantiated as a directory. This directory is expected to hold files in the .py format that implement custom filters for the use in the role itself. Filters in Ansible are generally used to parse data and are part of the Jinja templating engine. Custom filters are used when a very specific type of nested data has to be parsed. The filter_plugins directory can be created on the same level as the root directory of the project or in the specific role where the plugin should be used. The following is an example of the placement of the filter_plugins directory.

```
/ ..... root directory of the role
├── filter_plugins .....
└── roles .....
    └── role_1 .....
        └── filter_plugins .....
```

All of the modules that the script uses implement their custom filter plugins. The implementation details of these filter plugins shall be presented in their respective subsections.

3.3 Implementation of the compliance verification script

Implementation of the security standards related roles

The implementation of the security standards related modules builds up on the presented architecture of the module. For the use of the script three security related modules were developed. These modules correspond to the definitions of compliance criteria set by the particular security standards and are applicable to the object of compliance verification. The object of verification is the oVirt/RHV virtualization platform and the related security standards are the already mentioned FIPS 140-2, DISA STIG and Common Criteria.

It is important to state that not all of the compliance criteria is tested against the oVirt/RHV platform, since some of them are either not applicable or cannot be automated by a script. This is the case of some of the security requirements from DISA STIG and Common Criteria. How and why some of those security requirements are not covered, shall be explained in their respective subsections.

Implementation of the FIPS 140-2 role

In the context of the architecture the FIPS 140-2 module implements four directories from the presented directory structure of a Ansible role. These are the tasks, vars, library and templates directories. In the following text all implementation aspects

shall be explained and applied to the high level overview and high level architecture already presented.

The high level purpose of this role is to verify compliance with the FIPS 140-2 standard of the deployed oVirt/RHV infrastructure. This purpose is achieved by validating a checklist. If this checklist is validated, meaning every rule of this checklist passes, the deployed oVirt/RHV is compliant with the standard. The checklist can be viewed as the database with the formal definitions. It therefore also resides in the vars directory. All of the rules in the checklist contain three parts - name of the rule, an information indicating whether the rule passed and a hint specifying how to remediate the state of the infrastructure in a case that the rule fails. The checklist of the FIPS 140-2 role is made of seven rules, all of them implementing a different aspect of the compliance criteria.

The purpose of first rule in the checklist is determine, whether the product that runs the deployed infrastructure is Red Hat Virtualization specifically. This requirement is crucial because it is strictly specified in the security policy of the given cryptographic modules that the operational environment in which the cryptographic module operates must be RHEL. Since Red Hat Virtualization is a layered product based on the RHEL operating system, it fulfils this criterion. It is specified in the *Applicability* section in the security policy. In terms of implementation, this is accomplished by the use of a role The Listing 3.5 is the representation of this rule in the YAML format that resides in the *main.yml* file in the vars directory.

Listing 3.5: Product type rule in the compliance checklist

```
product_type:                                                    1
  name: Product type must be Red Hat Virtualization              2
  pass: false                                                    3
  hint: According to the security policy,                        4
        the product applicability allows only                    5
        Red Hat Virtualization (RHV)                             6
```

The second rule also validates an aspect of the operation environment in which the cryptographic modules are running. The security policy strictly limits the type of the operating system only to RHEL. This means that both the hosts and the hypervisor must run the RHEL operating system. In terms of implementation, this is done through the use of Ansible facts. Before Ansible executes the contents of the playbook, it tries to gather information about the managed host. This information includes things such as storage devices available on the system, network interfaces and other. Most importantly it contains information about the type of the operating system that is installed on the managed host, the version of it and the distribution. This information is then compared to the desired output of this action. The desired

output resides in the vars directory and is fetched by the *main.yml* file in the tasks directory for comparison and evaluation. The Listing 3.6 is the representation of this rule in the YAML format.

Listing 3.6: OS type rule in the compliance checklist

```
os_type:                                                          1
  name: Operating system must be RHEL                             2
  pass: false                                                     3
  hint: According to the security policy,                         4
        the operational environment must be RHEL                  5
```

The third rule of the compliance checklist incorporates the interface for working with the FIPS mode of the cryptographic modules. This part of the module is crucial because in order to fetch any information relevant to the FIPS 140-2 security requirements it needs to have such an interface incorporated. The developed FIPS 140-2 role makes use of the *fips-mode-setup* binary program. This binary is a supported interface for working with the FIPS system-wide cryptographic policy. In the case of this role, the only use of the fips-mode-setup binary program is to confirm that the given cryptographic modules operate in FIPS mode. When FIPS mode is enabled, it means that the given cryptographic modules operate in a defined mode of operation specified by the FIPS 140-2 standard.

For every cryptographic module within the RHEL operating system a security policy exists. A security policy is a documentation for the particular cryptographic module. It follows the domains in which the cryptographic module is tested for FIPS 140-2 certification and also specifies how to work with the module in accordance with the standard. This information is crucial for the implementation of this role. It resides in the *Guidance* section of all the security policies and it recommends the use of the already mentioned fips-mode-setup binary program. The Listing 3.7 is the recommended method of enabling FIPS mode for all of the cryptographic modules.

Listing 3.7: Recommended method of FIPS mode installation

```
1. To switch the system to FIPS enablement in RHEL 8:
# fips-mode-setup --enable
Setting system policy to FIPS
FIPS mode will be enabled.
Please reboot the system for the setting to take effect.
2. Restart your system:
# reboot
3. After the restart, you can check the current state:
# fips-mode-setup --check
FIPS mode is enabled
```

The purpose of the third rule is then to validate whether the cryptographic module are running in FIPS mode. This is achieved by the use of the fips-mode-setup binary and the Shell Ansible module which execute arbitrary Shell code on the managed hosts. Since this rule only checks for the enablement of FIPS mode, it makes use of the *–is-enabled* parameter of the binary. The following *task* then parses the output from the check and compares it with the desired output from the vars directory. The –is-enabled parameter returns 0 if FIPS mode is enabled, otherwise 2. The Listing 3.8 is the representation of this rule.

Listing 3.8: FIPS mode enforcement rule in the compliance check

```
  fips_mode_enabled:
    name: FIPS mode must be enabled on the system
    pass: false
    hint: FIPS mode can be enabled
          via fips-mode-setup --enable
```

The fourth and fifth rules are there to ensure that the FIPS mode is truly enabled on the system. Since RHEL handles the settings of cryptographic libraries via its system-wide cryptographic policies, these rules check 2 aspects of them. The fourth rules checks whether the system-wide cryptographic policy currently set on the system is FIPS. The fifth rule check whether the FIPS system-wide cryptographic policy is applied. Both of these checks are implement via the use of the *update-crypto-polices* binary. The fourth rule uses the –show parameter which outputs the currently set cryptographic policy. The fifth rule uses the –is-applied parameter that outputs whether the currently set cryptographic policy is applied or not. Both of these rules then check the returned output with the desired output from the vars directory. The implementation is done through the use of the Shell module provided by Ansible. The update-crypto-policies binary is then executed by the module on

the managed hosts with the corresponding parameters. These rules are represented once again in the YAML format which can be seen from the Listing 3.9.

Listing 3.9: FIPS cryptographic policy enforcement rules in the compliance check

```
fips_crypto_policy_enabled:
  name: FIPS system-wide cryptographic policy
        must be enabled
  pass: false
  hint: FIPS cryptographic policy can be enabled
        via update-crypto-policies --set FIPS
fips_crypto_policy_applied:
  name: FIPS system-wide cryptographic policy
        must be applied
  pass: false
  hint: FIPS cryptographic policy can be applied
        via update-crypto-policies --set FIPS
```

The sixth rule of the compliance checklist makes sure that the specific kernel parameter that corresponds to the FIPS mode enablement is also enabled on the system. In case of inconsistencies on the system, it might happen that the fips-mode-setup binary would output that the modules operate in FIPS mode but the specific kernel parameter would not be enabled. The specific kernel parameter is the *crypto.fips_enabled* and is checked via the *sysctl* command with the use of the -n parameter. The Ansible Shell module is also used. The output of this command is then compared with the desired output from the vars directory. The Listing 3.10 is the representation of this rule.

Listing 3.10: Kernel parameter enforcement rule in the compliance check

```
fips_kernel_param:
  name: Kernel parameter crypto.fips_enabled
        must be set to 1
  pass: false
  hint: crypto.fips_enabled can be set
        to 1 via sysctl -w crypto.fips_enabled=1
```

The last rule from the compliance check ensures that only the versions of the cryptographic libraries that the cryptographic modules represent are certified and their certificates are active. These are the cryptographic modules that this role checks:

- OpenSSL,
- Libgcrypt,
- Kernel Cryptographic API,

- GnuTLS,
- NSS

The versions of the active certificates of the cryptographic modules can be gathered from the NIST website. Since Red Hat hosts a website that holds this information in a machine readable format, this website is scraped and parsed by the role. This functionality is achieved by the use of the custom Ansible module which fetches the contents of the website in raw HTML format and parses out the related information about the cryptographic modules. This module resides in the library directory. The information that this custom module retrieves is then considered to be the desired output which is loaded into memory. The actual output, meaning the current versions of the binaries on the system, is fetched by using the Shell Ansible module and the *rpm* command together with the -qa parameters. The actual output and the desired output is then compared. The comparative action is executed by a custom filter plugin that resides in the root directory of the project. A custom filter plugin was developed for this purpose because the structure of the data is quite complex and makes it easier to create a custom parser for the job rather than use the standard ones provided by the templating engine. A specific condition also applies to this rule. Since a state in which the versions of the cryptographic modules might not be certified for a given version of RHEL is expected to occur, this state is automatically marked as defective. Therefore this rule fails when such a state occurs. The Listing 3.11 is the representation of this rule.

Listing 3.11: Active cryptographic module versions enforcement rule in the compliance checklist

```
fips_binaries_versions:                                          1
  name: "Binaries of the cryptographic modules must be of        2
         the versions specified by the security policy           3
         for RHEL {{ fips_rhel_version }}"                       4
  pass: false                                                    5
  hint: "Access this website to check which versions of          6
         the binaries are compliant:                             7
         {{ redhat_gov_standards_url }}"                         8
```

The purpose of this role is then to go through the compliance checklist and execute their respective verification commands. If all of the rules from the checklist pass, the infrastructure is compliant with the standard. In the last stage of the role a scan report gets generated. In the top level of the report a text indicating whether the infrastructure is compliant or not gets printed. Most importantly this report contains all the rules with the information whether the rule passed or not. If the rule did not pass, a hint is generated for a possible remediation of the defect. The Jinja

templating engine is used for generating this report and the format of it is HTML. An example report is attached to this thesis in the example_reports directory in the root of the project.

Implementation of the DISA STIG role

The DISA STIG module implements three directories from the presented directory structure of a Ansible role. These are the tasks, vars and templates directories. In the following text all implementation aspects shall be explained and applied to the high level overview and high level architecture already presented.

The high level purpose of this role is to verify compliance with the DISA STIG standard of the deployed oVirt/RHV infrastructure. This purpose is achieved by validating a checklist. If this checklist is validated, meaning every rule of this checklist passes, the deployed oVirt/RHV is compliant with the standard. The checklist can be vied as the database with the formal definitions. The checklist differs from the implementation of the FIPS 140-2 role. Since the DISA STIG module makes use of the oscap scanner tool from the OpenSCAP project, it utilizes the checklist provided by the OpenSCAP project. The checklist provided by the OpenSCAP projects implements the STIG published on DISA's website. The checklist formally includes all the security requirements imposed by DISA and is produced by Red Hat through the OpenSCAP project. The Listing 3.12 is an example from the RHEL DISA STIG that is used to validate compliance in this role.

Listing 3.12: Example of a STIG rule

```
Rule: Configure SSH Server to Use FIPS~140-2 Validated MACs:
opensshserver.config

Crypto Policies provide a centralized control over
crypto algorithms usage of many packages...

Warning: The system needs to be rebooted for these changes
to take effect.

Warning: System Crypto Modules must be provided
by a vendor that undergoes FIPS-140 certifications.
FIPS-140 is applicable to all Federal agencies ...

Rationale: Overriding the system crypto policy makes
the behavior of the OpenSSH server violate expectations,
and makes system configuration more fragmented.

Severity: medium

Rule ID: xccdf_org.ssgproject.content_rule_harden_sshd_macs_
opensshserver_conf_crypto_policy

Identifiers and References:
Identifiers: CCE-85899-3
References: CCI-001453, AC-17(2), SRG-OS-000250-GPOS-00093,
RHEL-08-010290, SV-230251r743937_rule

Remediation Shell script
Remediation Ansible snippet
```

This example rule, just as every rule in the checklist, contains these main parts: rule, rationale, severity, rule id, identifiers and references and lastly remediations. If needed, warnings can be added to the rule explaining an important aspect of this rule. The rule part provides a name for this rule in a human readable format. Rationale provides an explanation for the rule. Severity is there to indicate how severe is passing (not passing) this specific rule. Rule id serves the purpose of having an unique identifier. The identifiers and references link the rule to the sources that this checklist was built upon. Last but not least is the remediation part. Here two possible methods of remediation can be chosen, either Ansible code or Shell script. The purpose of the remediation part is to generate executable code that fixes the

non-compliant state of the machine to the correct state.

This particular role of the compliance verification script uses the oscap scanner tool to determine compliance of the particular oVir/RHV infrastructure. The workflow for this scanning tool was presented in the section about oVirt and DISA STIG. The management of the scanning tool is handled in the tasks directory. The workflow implemented there is the following:

- determine product type,
- determine operating system type,
- execute scan,
- fetch scan results from the machines,
- evaluate scan results

The presented workflow is formalized in the vars directory and is made out of three rules in a checklist format. All of the rules in the checklist contain three parts - name of the rule, an information indicating whether the rule passed and a hint specifying how to remediate the state of the infrastructure in a case that the rule fails.

The first rule corresponds to the product type condition, that is that the product the deployed infrastructure runs on must be Red Hat Virtualization. This condition is there because RHV is a layered product based on RHEL. The compliance with the RHEL STIG is maintained only for this specific operating system, meaning other operating systems like CentOS stream are inherently not compliant. The sole implementation of this rule is the same as in the FIPS 140-2 role. In terms of implementation the information about the product gathered from the base info role is used. This information is then compared to the desired output obtained from the vars directory.

Listing 3.13: Product type enforcement rule

```
product_type:                                                    1
  name: Product type must be Red Hat Virtualization              2
  pass: false                                                    3
  hint: Since RHV is a layered product based on RHEL,            4
  only this type of product is sufficient                        5
```

The second rule is a sub-rule of the first rule, meaning it demands the RHEL operating system to be installed on the machines. Ansible facts are used to gather information about the operating system running on the managed hosts. This information is then compared to the desired output residing in the vars directory. The Listing 3.14 is the representation of this rule.

Listing 3.14: Operating system type enforcement rule

```
os_type:
  name: Operating system must be RHEL
  pass: false
  hint: Since the STIG is developed for RHEL,
        this operating system must be used
```

The last rule incorporates the DISA STIG checklist for RHEL from the OpenSCAP project. The condition is fairly primitive. If the scan executed by the oscap scanning tool passes, this rule is marked as passed as well. Although the workflow for this particular rule seems to be fairly straight forward, it is not. Some security requirements that are defined for the STIG are not covered by the checklist provided by the OpenSCAP project. These security requirements are related to physical security and since such properties cannot be determined by the use of an automation tool, they simply cannot be covered. The tool itself will mark the rules that are not related as *notapplicable*. The Listing 3.15 is the representation of this rule in the *main.yml* file residing in the vars directory.

Listing 3.15: Succesfull oscap scan enforcement rule

```
  name: Scan from the oscap tool must pass
  pass: false
  hint: Use the oscap tool with the generate fix argument to
        generate a remediation script, for example
        oscap xccdf generate fix
        --profile xccdf_org.ssgproject.content_profile_stig
        --fix-type ansible --output rem.yml
```

The purpose of this role is then to go through the compliance checklist and execute its respective verification commands. If all of the rules from the checklist pass, the infrastructure is compliant with the standard. In the last stage of the role two scan reports get generated. The first one being the scan report generated by the oscap scanning tool that contains all the security requirements from the RHEL STIG indicating whether the particular rule passed or did not pass. The second one being the report containing the three rules presented in this subsection and their results. The third rule of the second report then include a relative link to the scan generated by the oscap tool. The Jinja templating engine is used for generating the second report and the format of it is HTML. The report generated by the oscap tools is also in HTML format. An example report is attached to this thesis in the example_reports directory in the root of the project.

Implementation of the Common Criteria role

In the context of the architecture the Common Criteria module also implements four directories from the presented directory structure. These are the tasks, vars, files and templates directories. In the following text all implementation aspects shall be explained and applied to the high level overview and high level architecture already presented.

The high level purpose of this role is to verify compliance with the Commmon Criteria standard of the deployed oVirt/RHV infrastructure. This purpose is achieved by validating a checklist. Since no formal automated way of verifying compliance with a security target exists, there are two possible methods of achieving this goal:

- implement the security requirements from the security target of the product itself,
- use the already created testing suite that an accredited laboratory uses to determine compliance

For the purpose of this thesis the second method was chosen. The testing suite had been provided to the author of this thesis by Red Hat and is distributed under the General Public License (GNU) Version 2. It primarily provides functional testing of the SFRs in the RHV Security Target. It is divided into domains of tests corresponding to the security functional classes of the SFRs of the TOE. All of the SFCs contain one to n number of SFRs. This is also reflected in the testing suite where the SFCs are represented as directories and each of the directories contain a different directory called *tests*. In the tests directory a test for each of the SFR exists in the form of a Bash script.

The test suite for verifying compliance to the RHV Security Target resides in the files directory. This directory was chosen for this use-case because its primary function is to store external files which are not templates but are intended for transfer to the remote host. These files will get executed on the remote host after the transfer.

A custom checklist that resides in the vars directory and its primary function is to represent the database with the formal definitions was also created for the purpose of this role. This custom checklist incorporates the testing suite and also other requirements that are there to ensure compliance. These other requirements are primarily derived from the Security Target itself. The testing suite is also a part of the Security Target which only tests SFRs. This checklist is of the same format as the checklists presented before for the FIPS 140-2 and DISA STIG roles. It once again contains the three parts - name of the rule, an information indicating whether the rule passed and a hint specifying how to remediate the state of the infrastructure in the case the rule fails.

The first rule is directly pulled out from the Security Target for RHV. It is

specified in the TOE identification that the TOE is Red Hat Virtualization, meaning no other product is sufficient e.g. oVirt[27]. The implementation of this rule is then the comparison between the information retrieved about the product type from the base info role and the desired output residing in the vars directory. The Listing 3.16 is the representation of this rule.

Listing 3.16: Product type enforcement rule

```
product_type:
  name: Product type must be Red Hat Virtualization
  pass: false
  hint: The Security Target specifies that the TOE of
        this product is Red Hat Virtualization,
        therefore only this product is sufficient
```

The second requirement from the custom checklist specifies that only the version 4.3.17 is compliant for the particular deployment of RHV. This requirement is also pulled out from the Security Target where the TOE identification specifies that Red Hat Virtualization *4.3.17* is the TOE[27]. In terms of implementation it checks the obtained version of the product provided by the base info role and compares it with the desired output from the vars directory. The Listing 3.17 is the representation of this rule.

Listing 3.17: Product version enforcement rule

```
product_version:
  name: Product version must be 4.3.17
  pass: false
  hint: The Security Target identifies the version
        of RHV to be 4.3.17
```

The third rule corresponds to the description of the TOE in the Security Target. The Security Target specifies that Red Hat Enterprise Linux is the provider of *virtualization primitives* for RHV and the components where RHEL is used i.e. hosts and the hypervisor are also part of the TOE[27]. This rule then compares the information gathered from Ansible facts about the operating system running and compares it with the desired output from the vars directory. The Listing 3.18 is the representation of this rule.

Listing 3.18: Operating system type enforcement rule

```
os_type:
  name: Operating system must be RHEL
  pass: false
  hint: The Security Target identifies the RHEL
        operating system as the provider of
        virtualization primitives for RHV
```

The fourth rule from the custom checklist specifies that only the version 7.9 is compliant for the particular installation of RHEL on the hosts and the hypervisor in a particular deployment of RHV. This requirement is also pulled out from the Security Target where the TOE description specifies that RHEL *7.9* is also part of the TOE[27]. In terms of implementation it checks the obtained version of the operating system provided by Ansible facts and compares it with the desired output from the vars directory. The Listing 3.19 is the representation of this rule.

Listing 3.19: Operating system version enforcement rule

```
os_version:
  name: Operating system version must be 7.9
  pass: false
  hint: The Security Target identifies the version of
        RHEL to be 7.9
```

The last rule from the custom checklist specifies that all tests from the test suite made for testing the SFRs from the TOE must pass. In terms of implementation it utilizes the shell module provided by Ansible to execute all of the tests from their respective security functional classes that then test the single SFRs. Each test from the test suite implements four functions which are run_test, show_test, startup_hook and cleanup. These functions are sequentially ran in a block of tasks in the *main.yml* file residing in the tasks directory. If the exit status of a given test is 0, it means that the test had passed. If the exit status of a given test is 1, it means that the test had not passed. Any other exit status indicates that an error had occurred during the execution of the test. The last task in the block of tasks evaluates the exit status for the given task and assigns whether the test had passed.

Listing 3.20: Successful test run enforcement rule

```
test_results:
  name: All tests from the test suite must pass
  pass: false
  hint: Unfortunately there is no formal way of remediating
  this issue
```

The purpose of this role is then to go through the compliance checklist and execute its respective tests. If all of the rules from the checklist pass, the infrastructure is compliant with the standard. In the last stage of the role a scan report get generated. The generated report includes all of the rules from the compliance checklist indicating whether it passed or not. In the top level of the report a statement indicating whether the deployed infrastructure is compliant is printed. The Jinja templating engine is used for generating the report and the format of it is HTML. An example report is attached to this thesis in the example_reports directory in the root of the project.

Implementation of the not security standards related roles

The compliance verification script also utilizes roles that are not related to the functionality that manages the compliance verification aspect of the given security standards. These roles are there to utilize a rather supporting function which is initial information gathering about the deployed infrastructure and dependency checking and resolvement.

Implementation of the base info role

The base info role implements only two directories from the presented directory structure. The implemented directories are tasks and vars. In the tasks directory the executable part of the role resides. The purpose of the vars directory is to store relevant data for the use in the role itself. This relevant data is for instance the FQDN of the oVirt/RVH engine or the username and password used to authenticate against the oVirt API.

The primary use of this role is to implement an interface for communicating with the oVirt API. This is achieved through the use of Ansible modules provided by the oVirt Ansible Collection. The modules in this collection implement every functionality needed to manage oVirt/RHV infrastructure. This role follows this particular workflow:

- extract authentication url from the vars_files directory,
- extract authentication credentials from the vars_files directory,
- authenticate against the oVirt/RHV API with the extracted variables,
- retrieve information about the oVirt/RHV hosts,
- dynamically register the hosts into the inventory,
- retrieve general information about the deployed oVirt/RHV infrastructure (product type for instance),
- parse and save this information for later use,
- revoke authentication session with the oVirt/RHV API

The first two items from the workflow are utilized by pure functionality provided by Ansible. Ansible automatically registers variables in the namespace of the role or in the whole playbook. These variables are then used in the next stages of this workflow.

In order to authenticate against the oVirt/RHV API the extracted variables are needed. These variables are then given to the module managing authentication to the oVirt/RHV API. For the purpose of authentication against the oVirt/RHV API, the *ovirt_auth* module provided by the oVirt Ansible Collection is used. This module establishes an authentication session by creating a SSO token. This SSO token is then revoked for the purpose of unauthentication[50].

The particular module that this role utilizes for API data retrieval is the *ovirt_api_info*[58]. In order to fetch information related only to the hosts that are registered in the oVirt/RHV manager, the *ovirt_host_info* module is used[51]. Since the script does not know the hosts beforehand, it needs to register them into the inventory. This is done through the use of the *add_host* Ansible module.

For parsing and saving information for later use, the *set_fact* Ansible module is made use of. This module saves and exports variables for later use in the playbook or role[57]. The information gathered by this role is then used in the roles related to the security standards.

The last stage of the workflow is achieved also with the ovirt_auth module with a slight change in the configuration. Here the state parameter is used. This parameter indicates to the module that the already established authentication session is ought to be revoked.

Implementation of the dependency checking and resolvement role

The dependency checking and resolvement role implements only two directories from the presented directory structure. The implemented directories are tasks and vars. In the tasks directory the executable part of the role resides. The purpose of the vars directory is to store relevant data for the use in the role itself. This relevant data are for instance the dependencies needed to run the Ansible modules used in the base info role or the dependencies needed to run the oscap scanning tool in the DISA STIG role.

This role primarily uses the yum Ansible module. The function of this module is to manage packages on RHEL-based systems[59]. This functionality is implemented in the executable part of the role i.e. the tasks directory. The related dependencies are stored in the vars directory. The workflow for this role is the following:

- determine for which role dependencies should be met,
- fetch dependencies from the vars directory,

- install them on the managed host via the yum module,
- remove the dependencies if needed

The first stage of the presented workflow is done through the dep_checker_deps variable. This variable is required because without this variable the role would not be able to determine, which dependencies should be installed on the particular managed host. The role stores the dependencies in the vars directory for each of the role used by the compliance verification script.

Once the role recognizes which role's dependencies should be managed, it fetches them from the vars directory and continues to the next stage of the workflow.

The role then installs the particular dependencies on the remote host using the yum Ansible module. A list of the dependencies that are to be installed on the managed host is given to the yum module. The yum module has to be told what it should do with those dependencies. For this the state parameter is used. Since it is desired to install the dependencies, the parameter has to be set to *present*[59].

It is desirable to remove the already installed dependencies from the system once the compliance evaluation process finishes. This is also done with use of the yum module together with the state parameter. In this specific case the state parameter has to be set to *absent*[59].

Implementation of the execution and orchestration layer

The execution and orchestration layer of the compliance verification script is the mid-level layer that sits above the implemented roles but is managed by the user interface layer. This mid-level layer utilizes the Ansible playbook as the primary functionality. The Ansible playbook is a file in the YAML format that is expected to contain reserved words. These reserved words are then parsed by the engine running underneath Ansible. The engine assigns a corresponding function to manage the reserved word. The reserved words that are used by the compliance verification script are the following:

- name,
- hosts,
- vars_files,
- roles

Since the playbook utilizes *plays* as stages for its execution, name, as one of the reserved words, is there to assign a name for the particular play. The hosts reserved word is there for Ansible to determine onto which managed hosts from the *inventory*[11] the particular play will get executed. The remaining roles reserved

[11] Ansible inventory specifies managed hosts and is represented as a text file. It can classify them into groups. It most importantly assigns an IP or a FQDN to the particular host, so Ansible knows

word is there to indicate where Ansible roles will be located. It is also important to mention that the roles that this layer manages also use another reserved word called tasks. This reserved word is used for indicating where the executable part of the playbook starts. Ansible tasks are put under these reserved words

In terms of usage for the script, the playbook provides an option to execute the developed roles in an orchestrated manner. This is achieved by utilizing plays. For each of the developed Ansible role a new play is reserved. This is done for better management of Ansible facts and overall better control of the execution of the particular role. The orchestration order of the plays is managed by conditional logic. This feature is provided by Ansible in the form of a when statement. The when statement works similarly to an if statement in ordinary programming languages and can be enriched with additional logical operators like AND, OR and so on[55].

The execution and orchestration layer utilizes a playbook. This playbook resides in the root directory of the whole project. It manages the execution and orchestration of the developed roles. The playbook is in the YAML format [63] and the presented reserved words are used. For orchestration purposes a play for each of the roles is reserved. The playbook starts with a first play that is reserved for the base info role. The start of the play is indicated by the *name* reserved word. All of the plays utilize the variables stored in the vars_files directory. These variables include engine's FQDN, credentials of an user that is able to authenticate against the oVirt/RHV API of the tested infrastructure and credentials of an user with root privileges. The user with root privileges is needed for package management and other elevated processes. The same credentials of the user with root privileges is also given to the user interface layer. The Listing 3.21 is the start of the play reserved for the base info role.

Listing 3.21: Start of the play reserved for the base info role

```
---                                                                    1
- name: Compliance Verification Script v1.0 | Base info                2
  hosts: localhost                                                     3
  vars_files:                                                          4
    - vars_files/engine.yml                                            5
    - vars_files/hosts.yml                                             6
```

The last part of the reserved play for the base info role is the part marked with the roles reserved word. Under this reserved word the base info role is used together with the dependency checker role. The roles are put there in a sequential order,

to which machine it should try connecting to[56]. In terms of the compliance verification script, the inventory is a file containing localhost as the managed host. It is one of the functions of the base info role to dynamically register oVirt/RHV hosts into the inventory.

meaning that at first the dependency checker will check whether the dependencies needed to run the base info role are met. This is ensured by the *dep_checker_deps* variable which is assigned to the dependency checker role. After dependencies are resolved, the base info role gets executed. When the execution of the base info role is finished, the dependencies are not needed anymore. They are then removed with help of the dependency checker role. The Listing 3.22 is the second part of the play reserved for the base info role.

Listing 3.22: End of the play reserved for the base info role

```
---
  roles:
    - role: dependency_checker
      dep_checker_os_flavor: |
        ansible_facts["ansible_distribution_file_variety"]
      dep_checker_deps: base_info
      when: check_base_info|default(false)
    - role: base_info
      base_info_ovirt_url: "{{ engine_url }}/ovirt-engine/api"
      base_info_ovirt_username: "{{ engine_username }}"
      base_info_ovirt_password: "{{ engine_password }}"
      when: check_base_info|default(false)
    - role: dependency_checker
      dep_checker_os_flavor: |
        ansible_facts["ansible_distribution_file_variety"]
      dep_checker_deps: base_info
      dep_checker_remove_deps: true
      when: check_base_info|default(false)
```

The remaining plays of the playbook are all reserved for the roles managing the compliance evaluation process of the discussed security standards. The structure of the play is very similar to the one presented for the base info role. The only difference is onto which hosts the play is going to be applied to. In the case of the base info role, the role is applied to localhost. This is done because the script does not know which are the virtualization hosts running in the oVirt/RHV infrastructure beforehand. During the execution of the base info role, the virtualization hosts get dynamically registered for later use. In other words in the case of the plays where the security standards related roles are used, the dynamically registered virtualization hosts are considered the managed hosts and therefore are put as the value for the hosts key. The dynamically registered managed hosts are grouped into *ovirt_hosts*. The first part of the play then looks very similar to the one already presented for the base info role. It makes use of the name, hosts and vars_files reserved words.

The Listing 3.23 is the start of the play reserved for the FIPS 140-2 role.

Listing 3.23: Start of the play reserved for the FIPS 140-2 role

```
- name: Compliance Verification Script v1.0 | FIPS~140-2
  hosts: ovirt_hosts
  vars_files:
    - vars_files/engine.yml
    - vars_files/hosts.yml
```

The remaining part of the play follows the same structure as presented for the base info role. It first checks if dependencies are met for the particular role and it resolves them. After that the functional aspects of the specific role get executed. When the execution of the role gets finished, the dependencies that were installed on the managed host get removed with the help of the dependency checker role. The Listing 3.24 representation of the remaining part of the role is the same for every security related role used by this script and therefore will not be covered in the coming text.

Listing 3.24: End of the play reserved for the FIPS 140-2 role

```
  roles:
    - role: dependency_checker
      dep_checker_os_flavor: |
        ansible_facts["ansible_distribution_file_variety"]
      dep_checker_deps: fips
      when: check_fips|default(false)
    - role: fips_140_2
      fips_rhel_version: "{{ ansible_distribution_version }}"
      fips_os_distribution: "{{ ansible_distribution }}"
      fips_prod_type: "{{
        hostvars['localhost']['prod_type']
      }}"
      fips_groups: "{{
        hostvars['localhost']['groups']['hosts']
      }}"
      fips_host_username: "{{ host_username }}"
      when: check_fips|default(false)
    - role: dependency_checker
      dep_checker_os_flavor: |
        ansible_facts["ansible_distribution_file_variety"]
      dep_checker_deps: fips
      dep_checker_remove_deps: true
      when: check_fips|default(false)
```

Implementation of the user interface layer

The primary role of the user interface layer is to parse a given input by the user and mediate the user input to the orchestration and execution layer. The reason for this layer is to give the user an interface for a bounded customization. The customization is bounded because only expected states can be chosen by the user. The states of the bounded customization were already presented in the section explaining the high level architecture of the script.

This layer is represented as a *Makefile*. This Makefile is located in the root directory of the project. It recognizes four commands which correspond to the states the user can choose from. These states include:

- test-fips,
- test-stig,
- test-cc,
- test-all

The Makefile also includes variables needed for successful execution of the script. They have to be added by the user that wants to use this script. These variables are the credentials of an user with root privileges that exists on the machines registered as hosts in the oVirt/RVH manager. The condition on elevated privileges is crucial, since some operations made by the script require such privileges (installing dependencies, running oscap scanning tool).

All of the commands implemented by this Makefile internally use the *ansible-playbook* binary program. This program executes Ansible playbooks with specified parameters[63]. In order to hand over the required variables to the ansible-playbook binary program for proper execution of the playbook, the *–extra-vars* parameter is made use of. This specific parameter allows passing extra variables to the binary program at the command line[69]. The Listing 3.25 is the representation of the Makefile containing the test-fips command and variables containing privileged user's credentials.

Listing 3.25: Makefile as the user interface layer

```
SECRET = "this is a secret"                                          1
USERNAME = "priv_user"                                               2
                                                                     3
test-fips:                                                           4
  ansible-playbook -i inventory -u $USERNAME main.yml \              5
  --extra-vars "check_fips=true check_base_info=true" \              6
  "ansible_password=${SECRET} ansible_user=${USERNAME}"               7
```

In order to execute the script with the test-fips command, it is required to be located in the root directory of the project and then run the following command in

the command line represented in the Listing 3.26.

Listing 3.26: Example of executing the script with the test-fips command

```
make test-fips
```

3.4 Testing scenario for the compliance verification script

The testing scenario for the compliance verification script includes a tester's laptop with the script installed on its operating system and put into operation and then most importantly the deployed oVirt/RHV infrastructure made out of three virtualization hosts and a hypervisor with the oVirt/RHV manager installed. The operating system running on the tester's laptop shall be a RHEL-based one (CentOS Stream 8). The figure 3.4 depicts this scenario. 3.4

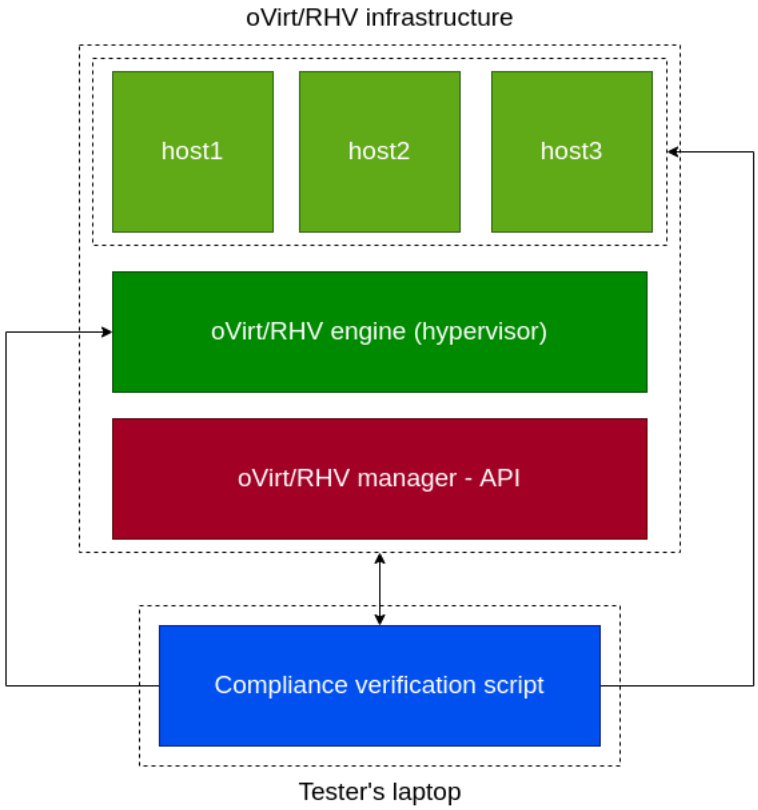


Fig. 3.4: Testing scenario for the compliance verification script

The specific system configurations of all components of the scenario are the following:

- Red Hat Virtualization; version 4.4,

- hypervisor and hosts; RHEL 8.5,
- tester's laptop; CentOS Stream 8

In order to run the script a few things have to be done beforehand. This includes installing Ansible on the operating system of the tester's laptop. Since the operating system running on the laptop is RHEL-based, the following commands can be run in order to install Ansible as according to the documentation[48].

Listing 3.27: Commands to install Ansible on RHEL-based systems

```
$ sudo yum install epel-release                                    1
$ sudo yum install ansible                                         2
```

After Ansible is installed, the git repository containing the compliance verification script must be cloned. In order to clone a git repository, git itself has to be present on the system. This can be achieved by running the following command.

Listing 3.28: Command to install git on RHEL-based systems

```
$ sudo yum install git                                             1
```

Once git is installed on the system, the git repository can finally be cloned. The git repository containing the compliance verification script is hosted on GitHub. The link[12] obtained from GitHub is then used together with the git clone command. This can be done by the following command.

Listing 3.29: Command to clone the specific repository

```
$ git clone url_to_the_repository                                  1
```

Depending on where the repository was cloned, the user has to change into the newly cloned directory. In order to change to the newly created repository, the *cd* command together with specified *relative* path has to be used.

Listing 3.30: Command to change to the specific directory

```
$ cd ovirt_security_compliance_verification_script                 1
```

A few configuration setting has to be done before using the script. Firstly it requires putting the username and password of a privilege user into the Makefile and secondly the FQDN, the username and password of a user that is able to authenticate against the oVirt/RHV manager. The second variables need to be put into the *engine.yml* YAML file residing in vars_files directory.

[12]https://github.com/vojtfa135/ovirt_security_compliance_verification_script.git

Listing 3.31: Configuring variables needed to run the script

```
$ vim Makefile
# needs to be filled in by the user
SECRET = "this is a secret"
# needs to be filled in by the user
USERNAME = "priv_user"
:wq
$ vim vars_files/engine.yml
---
engine_profile: internal
# needs to be filled in by the user
engine_username: admin
engine_username_full: "{{
    engine_username }}@{{ engine_profile }}"
# needs to be filled in by the user
engine_password: "123456"
# needs to be filled in by the user
engine_url: "https://hosted-engine.lab.testing.com"
:wq
```

In order to be able to use the Makefile provided by the script as the user interface layer, other dependencies need to be installed on the system. For being able to run the make command, the Development Tools need to be installed. The following command in the Listing 3.32 takes care of this problem.

Listing 3.32: Command to install Development Tools

```
$ sudo yum groupinstall "Development Tools"
```

The last dependencies that need to be met in order to run the compliance verification script, are included in the *install_local.sh* script located in the root directory of the project. Just running the script is sufficient.

The steps leading to this point were there just to make sure that all prerequisites for running the compliance verification script are met. After this the commands defined in the Makefile can be used. In the most basic scenario the *make test-all* command would used for verifying compliance to all of the security standards.

Listing 3.33: Command to run all tests for compliance verification

```
make test-all
```

After this command the sequence of base info, FIPS 140-2, DISA STIG and lastly Common Criteria roles will get executed. The first role to be executed will

be the base info which will first resolve dependencies and then start gathering relevant information about the infrastructure. After it gathers relevant information, it removes the dependencies installed beforehand. The dots at the end of the Listing 3.34 symbolize that the execution continues with other tasks.

Listing 3.34: Execution of the base info role

```
PLAY [Compliance Verification Script v1.0 | Base info] ******

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

TASK [dependency_checker : Install oVirt repository]*********
skipping: [localhost]

TASK [dependency_checker : Ensure that all dependencies
are met in order to install ovirt-engine-sdk-python] ********
skipping: [localhost] => (item=sshpass)
skipping: [localhost] => (item=libcurl-devel)
skipping: [localhost] => (item=python38-devel)
skipping: [localhost] => (item=openssl-devel)
skipping: [localhost] => (item=libxslt-devel)
skipping: [localhost] => (item=libxml2-devel)

TASK [dependency_checker : Intall oVirt SDK] ****************
skipping: [localhost]

TASK [base_info : Authenticate against oVirt engine] ********
ok: [localhost]

TASK [base_info : Extract relevant information about hosts] *
ok: [localhost]

TASK [base_info : Retrieve oVirt API data] ******************
ok: [localhost]
...
...
...
```

The next role in the sequence is FIPS 140-2. This role shall start evaluating compliance to the corresponding standard. The process of evaluation corresponds to the compliance checklist presented in the subsection explaining the implementation of this role. Next role that shall be executed is DISA STIG and once the execution

of this role gets finished, the remaining Common Criteria role also gets executed. Since the execution of the other remaining roles related to the security standards is very similar to following one represented by the Listing 3.35, it will not be covered.

Listing 3.35: Execution of the FIPS 140-2 role

```
PLAY [Compliance Verification Script v1.0 | FIPS~140-2] *****
TASK [Gathering Facts] ***************************************
ok: [host1]
ok: [host2]
ok: [host3]
TASK [fips_140_2 : Evaluate product type] ******************
ok: [host1]
ok: [host2]
ok: [host3]
TASK [fips_140_2 : Evaluate OS type on the hosts] ***********
ok: [host1]
ok: [host2]
ok: [host3]
...
...
...
```

For each role a report gets generated. The contents of this report is the presented checklist implemented for each role and a total evaluation stating whether the deployed infrastructure is or is not compliant with the particular standard. This report is to be found in the root directory of the project. 3.5

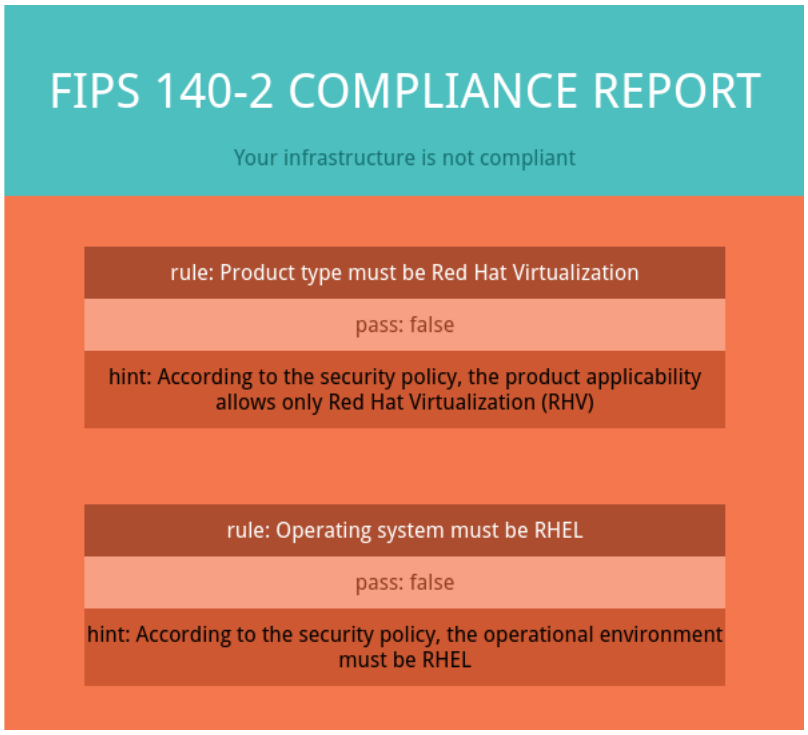


Fig. 3.5: A part of the generated report for the FIPS 140-2 role

In this specific testing scenario, the FIPS 140-2 evaluation report ended up with the following results:

- rule: Product type must be Red Hat Virtualization; pass,
- rule: Operating system must be RHEL; pass,
- rule: FIPS mode must be enabled on the system; fail,
- rule: FIPS system-wide cryptographic policy must be enabled; fail,
- rule: FIPS system-wide cryptographic policy must be applied; fail,
- rule: Kernel parameter crypto.fips_enabled must be set to 1; fail,
- rule: Binaries of the cryptographic modules must be of the versions specified by the security policy for RHEL 8.5; fail

The final evaluation of such deployed RHV infrastructure with the corresponding results of the FIPS 140-2 compliance evaluation then states that such an infrastructure with such configuration is not FIPS 140-2 compliant.

The third, fourth, fifth and sixth rules are marked as failed because according to those rules none of the presented mechanisms designated to handle properties of the FIPS 140-2 standard provided by the RHEL operating system were employed. The last rule was marked as failed because as of now[13] there are no versions of the binaries of the cryptographic modules for RHEL of the version 8.5 that are in *active* status.

[13]May 29 2022

In this specific testing scenario, the DISA STIG evaluation report ended up with the following results:

- rule: Product type must be Red Hat Virtualization; pass,
- rule: Operating system must be RHEL; pass,
- rule: Scan from the oscap tool must pass; fail

The final evaluation of such deployed RHV infrastructure with the corresponding results of the DISA STIG compliance evaluation then states that such an infrastructure with such configuration is not DISA STIG compliant.

The last rule was marked as failed because the scan results from the oscap tool were also marked as failed. The reason for that was that not all of the requirements specified in the RHEL STIG were employed on the hosts of this particular deployment of the virtualization platform. There exists a link to the oscap generated report in the final report generated by the DISA STIG role under the section where the last rule resides. This report can be found in the *example_reports* directory that resides in the root of the project.

In this specific testing scenario, the Common Criteria evaluation report ended up with the following results:

- rule: Product type must be Red Hat Virtualization; pass,
- rule: Product version must be 4.3.17; fail,
- rule: Operating system must be RHEL; pass,
- rule: Operating system version must be 7.9; fail,
- rule: All tests from the test suite must pass; fail

The final evaluation of such deployed RHV infrastructure with the corresponding results of the Common Criteria compliance evaluation then states that such an infrastructure with such configuration is not Common Criteria compliant.

Since the testing configuration employs RHV of the version 4.4 and RHEL of the version 8.5, it can by no means pass the second and the fourth rule. The last rule failed because the tests were primarily designed to test the functional aspects of RHV of the version *4.3.17*. Since the functional aspects of RHV of the version *4.4* might have changed, it did not pass.

Achieved goals of the thesis

The thesis managed to lay out the theoretical grounds of the security standards that are to be evaluated. It also provided an explanation of oVirt architecture and how the implementation of those standards is done in oVirt. It then managed to sketch out compliance verification checklists for FIPS 140-2, DISA STIG and Common Criteria. These compliance verification checklists represented theoretical ground for further implementation in the compliance verification script. They are also the formal way of assessing compliance of oVirt or RHV configurations. All of the compliance verification checklists are based on the texts of the related standards and also other findings that the author was able to derive from the study of those standards and their representation in the virtualization platform. In the context of the security standards presented, security improvements were proposed by the thesis for the oVirt platform. These security improvements were rather procedural.

An automated way of deploying oVirt was also established. This was a crucial step in testing and verifying correct functionality of the scanning tool made for the purpose of assessing compliance to the given security standards. The automated deployment of oVirt enabled the author of the thesis to dynamically provision different configurations of the oVirt platform in very little time when compared to manual deployment. These configurations could be then analyzed and according to the analysis a right implementation for the script could then be chosen.

The thesis managed to show an architecture of the scanning tool which turned out to be layered and modular. This architecture was then implemented in the form of Ansible playbook together with Ansible roles. The Ansible roles represent modules dedicated for assessing compliance to each of the given standards. The modules and the compliance verification script was developed in accordance with the theoretical grounds laid out by the thesis.

The implemented compliance verification script representing the scanning tool capable of assessing compliance was properly tested and its functionality was verified. It is therefore not defective and can be used in real environments where Red Hat Virtualization is used as a virtualization platform and compliance to the FIPS 140-2, DISA STIG and Common Criteria security standards is of high concern. This is applicable to governmental institutions such as federal agencies in the US.

Conclusion

This thesis established an overview of the particular standards and strived to point out important and relevant information for a whole understanding of the presented issue. It explained the architecture and functioning of FIPS 140 standards (the 140-2 in particular). It laid out an explanation of DISA STIGs and their link to SRGs and vice versa. And lastly it created a general view of the Common Criteria standard.

Furthermore, it declared what full virtualization technology means in the general context and in the context of oVirt and its respective downstream product Red Hat Virtualization. It then applied the presented security standards to the oVirt platform. There a number of issues occurred regarding the difference between compliance and compatibility, for which a satisfying definition was presented. For each standard, the implementation of it in the oVirt/RHV platform was showed and explained. It also presented an important link between oVirt/RHV and RHEL, when it comes to the implementation of the security standards.

A compliance verification checklist was then established for FIPS 140-2, DISA STIG and Common Criteria. In the case of FIPS 140-2, it used RHEL's system-wide cryptographic policies and criteria from the Security Policy. In the case of DISA STIG, it used tools provided by the OpenSCAP project. Lastly it explained that in order to verify compliance with the Common Criteria standard the functional testing suite provided by Red Hat shall be used.

The thesis also presented practical security enhancements for the oVirt/RHV platform. These security enhancements were more of a procedural concern in some cases. The security enhancement for oVirt in the context of FIPS 140-2 was to prepare and start a certification process of its respective cryptographic modules for FIPS 140-3. In terms of DISA STIG, the author recommends to participate with DISA on an official RHV STIG. Lastly in the case of oVirt/RHV and Common Criteria, the author recommended to set a more frequent process of undergoing certification. This was recommended because the actual versions of RHV and RHEL in the RHV ST are not the latest.

In the context of the practical implementation of this thesis, it first demonstrated the importance of an automated setup of oVirt deployment and testing. This was done through the means of CI/CD. All the functional pieces of the automated setup were explained. It then laid out the high level overview and high level architecture of the compliance verification script and how it is constructed. It also provided a general architecture of all roles used in the script itself. The practical implementation of the high level architectural aspects of both the script and roles was introduced and properly described. Since the architecture was presented as modular, for each of

the security standard a dedicated module was implemented. As for the technology chosen for the script and modules managing the compliance evaluation process to the particular standards, Ansible was the best fit. The compliance verification script makes use of the Ansible playbook properties and the corresponding security standards related modules make use of the Ansible role properties.

In terms of the role, the thesis used the compliance verification checklists it sketched out in the theoretical part explaining the link between the individual security standards and oVirt/RHV itself. The compliance verification checklists were transformed in a formalized form that could be used in the implementation of the script's security standards related roles.

Since there were other functionalities that had to be implemented as well, non-security related roles were also created. These were shown to have a rather supporting functionality such as initial information gathering about the deployed oVirt/RHV infrastructure and the resolvement of dependencies.

The thesis was able to implement a piece of software that is capable of assessing compliance of a given deployed oVirt/RHV infrastructure. In the last section of the thesis, a testing scenario of the whole compliance verification script was presented.

The author therefore claims to have successfully satisfied the goals of the thesis.

Bibliography

[1] Red Hat: *Government Standards* [online]. [cit. 2. 11. 2021]. Available at: <https://access.redhat.com/articles/2918071>.

[2] NIST: *Current Approved and Draft FIPS* [online]. [cit. 23. 10. 2021]. Available at: <https://csrc.nist.gov/publications/fips>.

[3] NIST: *FIPS 140-2* [online]. [cit. 23. 10. 2021]. Available at: <https://csrc.nist.gov/publications/detail/fips/140/2/final>.

[4] NIST: *About NVLAP* [online]. [cit. 23. 10. 2021]. Available at: <https://www.nist.gov/nvlap/about-nvlap>.

[5] NIST: *FIPS General Information* [online]. [cit. 23. 10. 2021]. Available at: <https://www.nist.gov/itl/fips-general-information>.

[6] NIST: *FIPS 140-3 Transition Effort* [online]. [cit. 5. 12. 2021]. Available at: <https://csrc.nist.gov/projects/fips-140-3-transition-effort>.

[7] NIST: *Cryptographic Module Validation Program* [online]. [cit. 12. 11. 2021]. Available at: <https://csrc.nist.gov/projects/cryptographic-module-validation-program>.

[8] NIST: *Federal Information Processing Standards Publication 140-2: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC* [online]. Change Notice 2, 12/3/2002. USA: NIST, 2002. Available at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf>.

[9] NIST: *Annex B:Approved Protection Profiles for FIPS PUB 140-2, Security Requirements for Cryptographic Modules* [online]. USA: NIST, 2019. Available at: <https://csrc.nist.gov/CSRC/media/Publications/fips/140/2/final/documents/fips1402annexb.pdf>.

[10] NIST: *Federal Information Processing Standards Publication 140-3: SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC* [online]. USA: NIST, 2002. Available at: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-3.pdf>.

[11] Cryptographic Module Validation Program: *CMVP FIPS 140-2 Standards and Documents* [online]. Available at: <https://csrc.nist.gov/Projects/cryptographic-module-validation-program/fips-140-2>.

[12] NIST: *Implementation Guidance for FIPS 140-2 and the Cryptographic Module Validation Program* [online]. [cit. 7. 11. 2021]. Available at: <https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Module-Validation-Program/documents/fips140-2/FIPS1402IG.pdf>.

[13] NIAP: *Approved Protection Profiles* [online]. [cit. 23. 5. 2022]. Available at: <https://www.niap-ccevs.org/profile/pp.cfm>.

[14] DISA: *Security Technical Implementation Guides (STIGs)* [online]. [cit. 23. 5. 2022]. Available at: <https://public.cyber.mil/stigs/>.

[15] Red Hat: *Chapter 1. Introduction to Red Hat Virtualization* [online]. [cit. 7. 11. 2021]. Available at: <https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.4/html/product_guide/introduction#RHV_Architecture>.

[16] oVirt Community: *oVirt User Documentation* [online]. [cit. 7. 11. 2021]. Available at: <https://ovirt.org/documentation/>.

[17] *Installing oVirt as a standalone Engine with local databases* [online]. [cit. 7. 11. 2021]. Available at: <https://www.ovirt.org/documentation/installing_ovirt_as_a_standalone_manager_with_local_databases/index.html#Red_Hat_Virtualization_Hosts_SM_localDB_deploy>.

[18] Brian Proffitt: *Moving Focus to the Upstream* [online]. [cit. 7. 11. 2021]. Available at: <https://www.redhat.com/en/blog/moving-focus-upstream>.

[19] SolarWinds: *Understanding DISA STIG Compliance Requirements* [online]. [cit. 7. 11. 2021]. Available at: <https://www.solarwinds.com/federal-government/solution/disa-stig-compliance>.

[20] SolarWinds: *General Purpose Operating System SRG* [online]. [cit. 7. 11. 2021]. Available at:
<https://www.stigviewer.com/stig/general_purpose_operating_system_srg/2019-07-01/finding/V-56761>.

[21] *SRG/STIG Tools* [online]. [cit. 1. 12. 2021]. Available at:
<https://public.cyber.mil/stigs/srg-stig-tools/>.

[22] *POWERFUL OPEN SOURCE VIRTUALIZATION* [online]. [cit. 7. 11. 2021]. Available at:
<https://www.ovirt.org/>.

[23] Red Hat: *Chapter 2. Red Hat Virtualization Components* [online]. [cit. 18. 11. 2021]. Available at:
<https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.4/html/product_guide/rhv-components>.

[24] Red Hat: *What is CentOS Stream?* [online]. [cit. 7. 11. 2021]. Available at:
<https://www.redhat.com/en/topics/linux/what-is-centos-stream>.

[25] Common Criteria for Information Technology Security Evaluation: *Part 1: Introduction and general model* [online]. [cit. 4. 12. 2021]. Available at:
<https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf>.

[26] Common Criteria for Information Technology Security Evaluation: *Part 2: Security functional components* [online]. [cit. 4. 12. 2021]. Available at:
<https://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R5.pdf>.

[27] Red Hat: *Red Hat Virtualization Security Target* [online]. [cit. 22. 5. 2022]. Available at:
<https://ocsi.isticom.it/documenti/certificazioni/redhat/rhv/st_red_hat_virtualization_43_v2.3.pdf>.

[28] Common Criteria for Information Technology Security Evaluation: *Part 3: Security assurance components* [online]. [cit. 4. 12. 2021]. Available at:
<https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf>.

[29] *About The Common Criteria* [online]. [cit. 4. 12. 2021]. Available at:
<https://www.commoncriteriaportal.org/ccra/index.cfm>.

[30] Organismo di Certificazione della Sicurezza Informatica: *Elenchi certificazioni: In corso di valutazione* [online]. [cit. 4. 12. 2021]. Available at: <https://ocsi.isticom.it/index.php/elenchi-certificazioni/in-corso-di-valutazione>.

[31] National Information Assurance Partnership: *Common Criteria Evaluation and Validation Scheme: Validation Report for the Red Hat Enterprise Linux Version 8.1, Version 1.0* [online]. [cit. 4. 12. 2021]. Available at: <https://www.niap-ccevs.org/MMO/Product/st_vid11107-vr.pdf>.

[32] National Information Assurance Partnership: *Common Criteria Evaluation and Validation Scheme: Validation Report for the Red Hat Enterprise Linux Version 7.6, Version 1.0* [online]. [cit. 4. 12. 2021]. Available at: <https://www.niap-ccevs.org/MMO/Product/st_vid11039-vr.pdf>.

[33] Acumen Security, LLC: *Red Hat Enterprise Linux 8.1 Security Target* [online]. [cit. 4. 12. 2021]. Available at: <https://www.niap-ccevs.org/MMO/Product/st_vid11107-st.pdf>.

[34] Red Hat: *Strong crypto defaults in RHEL 8 and deprecation of weak crypto algorithms* [online]. [cit. 29. 11. 2021]. Available at: <https://access.redhat.com/articles/3642912>.

[35] NIST: *Compliance FAQs: Federal Information Processing Standards (FIPS)* [online]. [cit. 29. 11. 2021]. Available at: <https://www.nist.gov/standardsgov/compliance-faqs-federal-information-proce

[36] Red Hat: *Chapter 5. Using system-wide cryptographic policies* [online]. [cit. 29. 11. 2021]. Available at: <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardening/using-the-system-wide-cryptographic-policies_security-hardening>.

[37] Red Hat: *Chapter 9. Scanning the system for configuration compliance and vulnerabilities* [online]. [cit. 29. 11. 2021]. Available at: <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/security_hardening/scanning-the-system-for-configuration-compliance-and-vulnerabilities_security-hardening>.

[38] Red Hat: *Appendix F. Securing Red Hat Virtualization* [online]. [cit. 29. 11. 2021]. Available at: <https://access.redhat.com/documentation/en-us/red_hat_virtualization/4.4/html/installing_red_hat_virtualization_as_a_self-hosted_engine_using_the_command_line/security>.

[39] Red Hat: *The value of a Red Hat subscription* [online]. [cit. 29. 11. 2021]. Available at: <https://www.redhat.com/en/about/value-of-subscription>.

[40] Red Hat: *How RHEL 8 is designed for FIPS 140-2 requirements* [online]. [cit. 29. 11. 2021]. Available at: <https://www.redhat.com/en/blog/how-rhel-8-designed-fips-140-2-requirements>

[41] atsec information security corporation: *Red Hat Enterprise Linux 8 libgcrypt Cryptographic Module: FIPS 140-2 Non-Proprietary Security Policy* [online]. [cit. 29. 11. 2021]. Available at: <https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3784.pdf>.

[42] Department of Defense: *Control Systems Security Requirements* [online]. [cit. 29. 11. 2021]. Available at: <https://dl.dod.cyber.mil/wp-content/uploads/external/pdf/071421_Control_Systems_SRG.pdf>.

[43] NIST: *Guide to Security for Full Virtualization Technologies* [online]. [cit. 29. 11. 2021]. Available at: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-125.pdf>.

[44] NIST: *NIST National Checklist for Red Hat Enterprise Linux 8.x content v0.1.50 Checklist* [online]. [cit. 29. 11. 2021]. Available at: <https://ncp.nist.gov/checklist/909>.

[45] *OpenSCAP User Manual* [online]. [cit. 29. 11. 2021]. Available at: <static.open-scap.org/openscap-1.3/oscap_user_manual.html>.

[46] *Best Practices* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html>.

[47] *Configuring Ansible* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/installation_guide/intro_configuration.html>.

[48] *Installing Ansible* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html>.

[49] *oVirt REST API documentation* [online]. [cit. 29. 11. 2021]. Available at: <http://ovirt.github.io/ovirt-engine-api-model/>.

[50] *Module to manage authentication to oVirt/RHV* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_auth_module.html>.

[51] *Retrieve information about one or more oVirt/RHV hosts* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_host_info_module.html>.

[52] *Module to manage users in oVirt/RHV* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_user_module.html>.

[53] *Add a host (and alternatively a group) to the ansible-playbook in-memory inventory* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/collections/ansible/builtin/add_host_module.html>.

[54] *Print statements during execution* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/collections/ansible/builtin/debug_module.html>.

[55] *Conditional* [online]. [cit. 23. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html>.

[56] *How to build your inventory* [online]. [cit. 23. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html>.

[57] *Set host variable(s) and fact(s)* [online]. [cit. 22. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/collections/ansible/builtin/set_fact_module.html>.

[58] *Retrieve information about the oVirt/RHV API* [online]. [cit. 22. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_api_info_module.html#ansible-collections-ovirt-ovirt-ovirt-api-info-module>.

[59] *Manages packages with the yum package manager* [online]. [cit. 22. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/collections/ansible/builtin/yum_module.html>.

[60] *Understanding DevOps* [online]. [cit. 29. 11. 2021]. Available at: <https://www.redhat.com/en/topics/devops>.

[61] *Git* [online]. [cit. 29. 11. 2021]. Available at: <https://git-scm.com/>.

[62] *Introduction to modules* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/user_guide/modules_intro.html>.

[63] *Intro to playbooks* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html>.

[64] *Roles* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html>.

[65] *Using collections* [online]. [cit. 29. 11. 2021]. Available at: <https://docs.ansible.com/ansible/latest/user_guide/collections_using.html>.

[66] *Gerrit Code Review Product Overview* [online]. [cit. 29. 11. 2021]. Available at: <https://gerrit-documentation.storage.googleapis.com/Documentation/3.5.0.1/intro-quick.html>.

[67] *YAML: YAML Ain't Markup Language™* [online]. [cit. 29. 11. 2021]. Available at: <https://yaml.org/>.

[68] *OVERVIEW: How Ansible Works* [online]. [cit. 29. 11. 2021]. Available at: <https://www.ansible.com/overview/how-ansible-works>.

[69] *Using Variables* [online]. [cit. 14. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html>.

[70] *Developing Ansible modules* [online]. [cit. 15. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_general.html>.

[71] *Developing plugins* [online]. [cit. 15. 5. 2022]. Available at: <https://docs.ansible.com/ansible/latest/dev_guide/developing_plugins.html>.

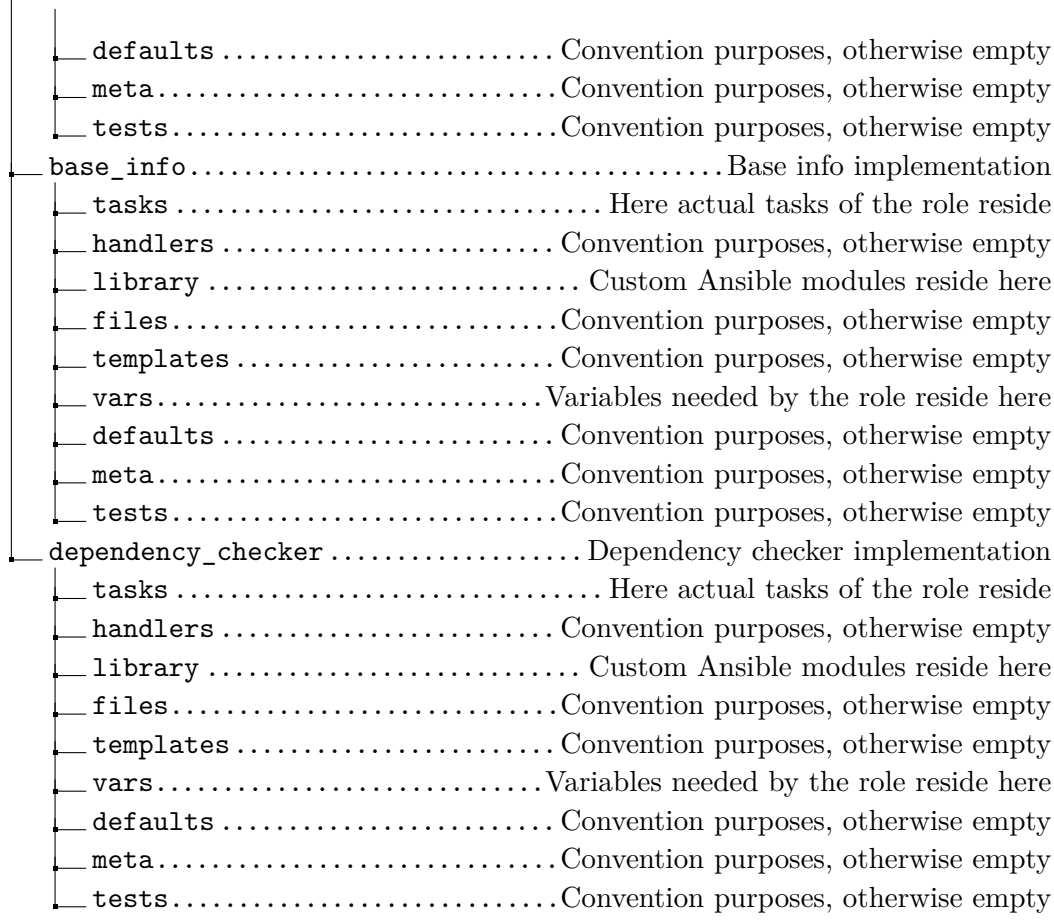
[72] Jenkins: *Pipeline* [online]. [cit. 29. 11. 2021]. Available at: <https://www.jenkins.io/doc/book/pipeline/#overview>.

[73] Jenkins: *What is Jenkins?* [online]. [cit. 29. 11. 2021]. Available at: <https://www.jenkins.io/doc/#what-is-jenkins>.

A Content of electronic attachment

The graph on this page explains the directory structure of the attachment to this thesis.

- `/` — root directory of the attachement
  - `main.yml` — Executable part of the Playbook
  - `ansible.cfg` — Configuration file for Ansible
  - `inventory` — FQDN of the engine resides here
  - `group_vars` — Generated according to the convention, resides empty in this case
  - `host_vars` — Generated according to the convention, resides empty in this case
  - `vars_files` — Here engine and host related configuration files reside
    - `engine.yml` — Engine related variables
    - `hosts.yml` — General hosts related variables
  - `example_reports` — Examples of scan reports for each role reside here
  - `install_local.sh` — Script for installing dependencies needed by the script
  - `README` — Readme file for setting up the script
  - `requirements.yml` — Ansible collection requirements
  - `roles` — Here implementations of the Roles shall reside
    - `fips_140_2` — FIPS evaluation implementation
      - `tasks` — Here actual tasks of the role reside
      - `handlers` — Convention purposes, otherwise empty
      - `library` — Custom Ansible modules reside here
      - `files` — Convention purposes, otherwise empty
      - `templates` — Scan report template resides here
      - `vars` — Variables needed by the role reside here
      - `defaults` — Convention purposes, otherwise empty
      - `meta` — Convention purposes, otherwise empty
      - `tests` — Convention purposes, otherwise empty
    - `disa_stig` — DISA STIG evaluation implementation
      - `tasks` — Here actual tasks of the role reside
      - `handlers` — Convention purposes, otherwise empty
      - `library` — Convention purposes, otherwise empty
      - `files` — Convention purposes, otherwise empty
      - `templates` — Scan report template resides here
      - `vars` — Variables needed by the role reside here
      - `defaults` — Convention purposes, otherwise empty
      - `meta` — Convention purposes, otherwise empty
      - `tests` — Convention purposes, otherwise empty
    - `common_criteria` — Common Criteria evaluation implementation
      - `tasks` — Here actual tasks of the role reside
      - `handlers` — Convention purposes, otherwise empty
      - `library` — Convention purposes, otherwise empty
      - `files` — Functional tests for Common Criteria reside here
      - `templates` — Scan report template resides here
      - `vars` — Variables needed by the role reside here


defaults ..... Convention purposes, otherwise empty
meta ..... Convention purposes, otherwise empty
tests ..... Convention purposes, otherwise empty
base_info ..... Base info implementation
tasks ..... Here actual tasks of the role reside
handlers ..... Convention purposes, otherwise empty
library ..... Custom Ansible modules reside here
files ..... Convention purposes, otherwise empty
templates ..... Convention purposes, otherwise empty
vars ..... Variables needed by the role reside here
defaults ..... Convention purposes, otherwise empty
meta ..... Convention purposes, otherwise empty
tests ..... Convention purposes, otherwise empty
dependency_checker ..... Dependency checker implementation
tasks ..... Here actual tasks of the role reside
handlers ..... Convention purposes, otherwise empty
library ..... Custom Ansible modules reside here
files ..... Convention purposes, otherwise empty
templates ..... Convention purposes, otherwise empty
vars ..... Variables needed by the role reside here
defaults ..... Convention purposes, otherwise empty
meta ..... Convention purposes, otherwise empty
tests ..... Convention purposes, otherwise empty