APPLICATION OF STANDARDS FOR VULNERABILITY PREVENTION IN VIRTUALIZATION Sys
Mysleidy D. M. D'Oliveira
1 2 9 0 UNIVERSIDADE D COIMBRA

1 2  9 0

UNIVERSIDADE D
COIMBRA

Mysleidy Duturna Mendonça D'Oliveira

Application of standards for vulnerability prevention in virtualization systems

Dissertation in the context of the Master of Informatics Security advised by Prof. Dr. Nuno Antunes and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Application of standards for vulnerability prevention in virtualization systems

Mysleidy Duturna Mendonça D'Oliveira
mdmdo@student.dei.uc.pt

Dissertation in the context of the Master of Informatics Security advised by Prof. Dr. Nuno Antunes and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2021

1 2 9 0

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Acknowledgements

First of all, thank God for life, blessings, for the opportunity to achieve another goal and continue to fulfill my dreams.

Grateful for the unconditional love, the support of my parents, my family, my boyfriend and my friends, because without them nothing would be possible.

Special thank you to Felipe Carneiro, a person of great importance to the family, a brother to my father, an uncle to me, who was always available to help and give advice. He was present in one of my special moment and directly and indirectly contributed to my formation.

To the University of Coimbra, in particular the Department of Informatics, for the welcome and knowledge acquired, to my teachers for their teachings and guidance.

I would like to thank Professor Nuno Antunes for his patience as teacher and coordinator, for his knowledge, appreciate his guidance in this process.

"... I am the master of my fate: I am the captain of my soul." - William Ernest Henley

This page is intentionally left blank.

Abstract

In recent years, virtualization has becoming popular and increasing interest from personal and enterprise level, providing many benefits for letting servers run multiple OS at the same time, reducing costs and augment the flexibility of physical infrastructures. As the popularity increased new security challenges emerged and as result attackers exploring vulnerabilities that affects confidentiality, integrity and availability of resources and services. In C and C++ programming languages, the probability of writing insecure code is very high, due to lack of notion on the part of developers about safe coding and their own programming style. As time goes by, more and more use of static code analyzers to detect defects in early stages of testing codes, leaving aside the necessary standards to prevent defects from being introduced early on.

The goal of this work is to understand the extent to which different types of standards and best practices could have prevented the introduction of vulnerabilities in hypervisors. A detailed analysis and systematizing of the standards that exist and that can be applied to prevent vulnerabilities was carried, not limited to coding standards, but also including development and information security standards. Then, the application of standards in vulnerabilities collected and storage in a database provide from previous works and in the end a discussion of results was performed showing the differences between standards and applicability. The overall result is a body of knowledge about the different areas in which standards can help, their advantages and their gaps .

Keywords

Virtualization, Standards, vulnerabilities, secure coding

This page is intentionally left blank.

Resumo

Nos últimos anos, a virtualização tornou-se popular e aumentou o interesse ao nível pessoal e empresarial, fornecendo muitos benefícios por permitir que servidores executassem vários sistemas operacionais ao mesmo tempo, reduzindo custos e aumentando a flexibilidade de infraestruturas físicas. Com a ampla expansão, novos desafios de segurança surgiram e como resultado, cada vez mais a exploração de vulnerabilidades que afetam a confidencialidade, integridade e disponibilidade de recursos e serviços. Em C e C++ a probabilidade de escrever códigos com defeitos é muito elevado, por causa da falta de noção dos programadores em relação a codificação segura e seu próprio estilo de programação. Com o passar do tempo, muitos analisadores de código estático estão sendo utilizados para detectar defeitos nos estágios iniciais dos códigos de teste, deixando de margem os standards que são igualmente necessários para evitar que vulnerabilidades sejam introduzidas logo no início do desenvolvimento do código.

O objetivo deste trabalho é compreender até que ponto diferentes tipos de standards e melhores práticas poderiam ter evitado a introdução de vulnerabilidades em hypervisors. Para o efeito, procedeu-se a uma análise detalhada e sistematica dos standards existentes que podem ser aplicadas para prevenir vulnerabilidades, não se limitando a standards de codificação, mas incluindo também standards de desenvolvimento e segurança da informação. Em seguida, foi efetuada a aplicação dos standards nas vulnerabilidades provinientes de trabalhos anteriores, já coletadas e armazenadas em banco de dados e por fim foi realizada uma discussão dos resultados mostrando as diferenças entre os diferentes standards e aplicabilidade. O resultado geral é um corpo de conhecimento sobre as diferentes áreas nas quais os standards podem ajudar e sobre suas lacunas e vantagens.

Palavras-Chave

Virtualização, Standards, vulnerabilidades, codificação segura

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

ANSI American National Standard Institute. 14

CEN European Committee for Standardization. 13

CNA CVE Numbering Authority. 21

CVE Common Vulnerabilities and Exposure. 9, 11, 21, 22

CWE Common Weakness Enumeration. 11, 15, 21, 22

DoS Denial of Service. 2, 8, 9, 11, 30, 36

DT Directory Traversal. 11

EC Execute Code. 11

GP Gain Privileges. 11

HIC++ High Integrity C++. 21, 24, 26, 30

IEC International Electrotechnical Commission. 13, 15–18, 24, 26, 31, 34, 46

InP Infrastructure providers. 7

ISMS Information Security Management System. 16, 17

ISO International Standard Organization. 13, 15–18, 24, 26, 31, 34, 46

ITU-T International Telecommunication Union (standardization section). 14

KVM Kernel-based Virtual Machine. 6

MISRA Motor Industry Software Reliability Association. 19–21, 24, 26, 30

NIST National Institute of Standards and Technology. 13

NVD National Vulnerability Database. 2, 11, 21, 22

OASIS Organization for the Advancement of Structured Information Standards. 14

OI Obtain Information. 11

OS Operating systems. 2, 5–7

OWASP Open Web Application Security Project. 21, 22, 26–30, 36

PCI Payment Card Industry. 14

PDCA Plan-Do-Check-Act. 17

SC Subcommittee. 15

SEI CERT C++ The Software Engineering Institute Computer Emergency Response Team. 24

SP Service Providers. 7

VM Virtual Machine. 6–8

VMM Virtual Machine Monitor. 6

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

Chapter 1

Introduction

"If builders built buildings the way programmers write programs then the first woodpecker that come along would destroy civilization"- Gerald Weinberg[6].

1.1 Motivation

The chosen theme ***"Application of standards for vulnerability prevention in virtualization systems"*** aims to be objective and concise to demonstrate the vulnerability of virtualization systems and how to prevented the onset of the vulnerabilities by application of standards. A theme that shows concern in this era of technology, with the growth and use of virtualization systems. Many researches[7][8][9][2] available on vulnerability prevention focuses more on static code analysers[10] and not on standards and their skills for vulnerability prevention, which woke up the curiosity to investigate and questioning why aren't there many related works on vulnerability prevention by standards applications. Through vulnerabilities storage in databases collected from previous works[2][3], there are a significant amount of vulnerabilities that could be prevented by just applying a standard in the beginning of development avoiding seriously damage in entire systems, data compromise and attacks.

For better understanding, questions have been posed so that it can create promising and helpful results and they will be answer in the Discussion section:

1. Does the implementation of standards really reduce the appearance of vulnerabilities in the software of virtualization systems?
2. All the standards can be applied or does it need a very specific group of standard to apply for vulnerabilities prevention?
3. For standards implementation, do developers need more hours of classes on secure coding or will they be able to deal with the extensive rules of standards?

Virtualization a core feature of Cloud Computing[11] with is vast benefits, enable server run multiple Operating systems (OS) at same time, is popularity increased dramatically in past decade at personal and enterprise level. Adherence to this technology includes range of benefits related to processing, storage, testing and development.

With the tremendous increase of use, the implementation of virtualization introduce many security challenges, because of the access to hardware resources [11]. With successful exploitation, attackers can gain access to the base system, multiple virtual machines and access to sensitive data [12]. For instance, vulnerability (CVE-2014-9357) found in Docker container [2], allows remote attackers to execute arbitrary code with root privileges via a crafted (1) image or (2) build in a Dockerfile in an LZMA (.xz) archive, gain privileges. An recent example report by National Vulnerability Database (NVD) related to hypervisors is a critical vulnerability (CVE-2020-15564) in Xen hypervisor, allowing Arm guest OS users to cause a hypervisor crash because of a missing alignment check in VCPUOP register vcpuinfo, resulting in a crash of hypervisor and Denial of Service (DoS)[2].

Therefore is necessary a different and essential approach to ensure security and prevention of vulnerabilities in virtualization systems. One of the approach is the application of standards and best practices to reduce and mitigate vulnerabilities. [13]

Standards are complex, large and incorporate high level objectives for compliance. By the level of complexity, they are necessary and one of the best methods for managing security in terms of information or software development, thus, it is important to understand the advantages of applying the standards correctly in each area and complying with all the requirements.

In this work, an extensive investigation into all the different types of standards was carried out, looking for more of 200 standards, categorizing in groups and explore how their approaches can be applied to preventing vulnerabilities. Then analysis of hypervisors and Docker's vulnerabilities, a targeted attacks due to vulnerabilities and for each one of them, understand how the different standards would have prevented if they had been followed during the software development.

1.2 Objective

The main goal of this work is to understand to what extent different types of standards and development processes could have prevented the introduction of vulnerabilities in hypervisors. A necessary analyses of existing standards was performed, encompassing many groups that can be essential.

To fulfill the main objective, specific objectives were determined, first a construction of a state of the art on the different types of standards that is interconnected. With knowledge of the standards, analysis of the hypervisors and Dockers vulnerabilities databases provided, and find the best standards that can applied to the prevention of vulnerabilities.

The second specific objective is a demonstration of how standards can be applied on vulnerabilities, document if they can prevented some vulnerabilities, advantages, challenges. The vulnerabilities datasets used for the work were provided by Ana Duarte [2] and Xavier F[3].

1.3 Approach

To fulfill the research objectives of this work, it was necessary to group the thesis in into three main phases, each designated with its objectives to accomplish.

Standards investigation (state-of-art)- The first phase focus on research and extraction of relevant material of the different existing standards, understanding the concepts and their applications, verifying whether or not it can be applied to virtualization systems. The results are presented in Chapter 2.

Vulnerability analysis - For the second phase, this dissertation uses the vulnerabilities databases collected by Duarte, F. A [2] and Mendes, X. report[3]. The first database is vulnerabilities collected from Docker by Duarte,A., where the purpose of the work was to presents how static code analyzers could be applied for vulnerability prevention in Decker's.

The second database of vulnerabilities was collected from Mendes, X., an internship report[3] focused in the two open-source hypervisors, KVM and Xen. The purpose of the work was to study hypervisor's vulnerability and their environment, gather information's and create a vulnerability database. With all the information gathered, graphs were created with main consequences and prevention's.

263 of the 344 reported vulnerabilities were analyzed, 25 vulnerabilities of which are the main ones in the Duarte, F. A.[2] database and 238 vulnerabilities in the Mendes, X.[3] database. The vulnerabilities were systematize by type, CWE, causes, effects, consequences and patches. The objective is to understand the frequency of vulnerabilities, the main cause of vulnerabilities, how they affect the systems, how they were fixed, if they were fixed correctly, also examined the code provide from patches, presented in Chapter 2.

Application of Standards In third phase, once the relevant data from database is reviewed and gathered fundamental information's (**State-of-art**), an application of standard is done individually for each vulnerability of the databases presented and added fundamental discussions.

1.4 Thesis Structure

The document is divided into the following chapters described:

Chapter 1 A introduction about the motivation, general context in which this document is placed, the main objectives, approach and thesis structure.

Chapter 2 presents the state of the art. Introduces the relevant concepts used in this work. In the beginning, an introduction of the concepts of virtualization, followed by history, types and categories. Next, vulnerability definition, impact and description of vulnerabilities databases reported by Duarte. F. A[2] from Docker and vulnerabilities by Mendes, X. [3] from KVM and XEN. Lastly, concept of standards, the categorized groups and their respective features and examples.

Chapter 3 demonstrates the standards chosen for application in the vulnerabilities.

Chapter 4 presents an overview of the results and discussion of results.

Chapter 5 systematizes the lessons learned and concludes the dissertation.

This page is intentionally left blank.

Chapter 2

State-of-the-art

This chapter introduces the relevant concepts used in this work. In the beginning, an introduction of the concepts of virtualization, followed by history, types and categories. Next, vulnerability definition, impact and description of vulnerabilities databases reported by Duarte. F. A[2] from Docker and vulnerabilities by Mendes, X. [3] from KVM and XEN. Lastly, concept of standards, the categorized groups and their respective features and examples.

2.1 Virtualization

Virtualization can be define as a technology that enable multiple operating systems to run on a single host at the same time [14]. Investing in virtualization can save money, maximize uptime, flexibility, resource optimization an protecting applications from failures of the server.

The concept of virtualization was originally introduced by Christopher Strachey, an professor of computation at Oxford University [15] in the 1960s. With the new concept that made a historical mark in history of computer, the goal was to created a time-sharing model, to increase efficiency for both users and computers by sharing computers resources, enabling users and organizations to use a computer without owning one.

With the ideas brought to life, a series of supercomputers - Atlas and IBM M44/44X - was followed, which thankfully to experiments made with them, we can have access now to virtualization [15] [14][16].To maintain the dominance in the computer field, IBM build the IBM M44/44X a supercomputer who first introduced the concept of virtual machines, making a great contribution to the area of time-sharing concepts [15] in that time. Further ahead in late 1960s and early 1970 [14] introduced an modern virtualization by then name of IBM's VM/370 that could allow multiple Operating systems (OS) to run at the same time in a single computer. [15].

Virtualization can be classified in two (2) major categories: **hypervisor-based virtualization** and **container-based virtualization** [1].

2.1.1 Hypervisors

Hypervisors also called **Virtual Machine Monitor (VMM)**[17] are considered the main component in the virtualization, designated as a virtual platform, allowing that multiple guest operating systems are executed and monitored[14][17] as showed in figure 2.1. Hypervisors controls the flow of instructions between the Virtual Machine (VM) operating system and the physical hardware. By providing a virtualization layer, if compromised, could result in loss or damage to critical assets owned by Cloud Service Providers and customers.

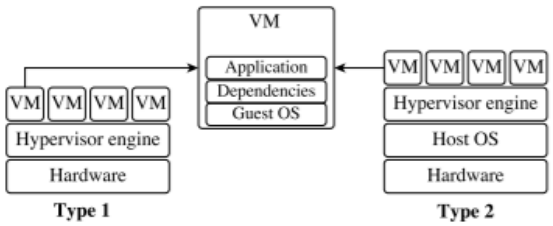


Figure 2.1: Hypervisor based virtualization[1]

Hypervisors are classified in two (2) different types: ***native or bare metal and hosted*** . **The native or bare metal** are software systems that run underlying the host hardware to control the hardware monitor of the guest OS [14] and intercepting communications between multiples VMs and physical hardware[17]. Examples Microsoft Hyper-V, VMware ESXi, XenServer [11].

Hosted hypervisors run in a traditional OS, adding a distinct software layer on the top of the host OS and the guest OS becomes a third software level[14], proving communication between hypervisor and hardware. Example: Oracle VirtualBox, VMWare Workstation, Redhat Kernel-based VM (KVM) [11].

2.1.2 Types of Virtualization

There are numerous types of virtualization available such as Server Virtualization, Desktop virtualization, Client Virtualization, Network virtualization, Data virtualization, Storage virtualization, Application virtualization [18][19]. The four major areas of virtualization and most common are Server virtualization, Network virtualization and Desktop Virtualization.

Server Virtualization

The most common and successful form of virtualization, when a single server performs a high volume of tasks of multiples servers, by portioning the resources of an server across multi-environment. This type of virtualization is used in cloud, having many advantages as efficient use of resource, application uptime, save power, high availability and cost savings [15]. The server virtualization is divided into three (3) following types: full virtualization, para-virtualization and partial virtualization [18].

Full virtualization: the complete simulation of hardware is done to allow software to run an unmodified guest OS, example *Kernel-based Virtual Machine (KVM)* [18][19].

Para-virtualization: in this type, partial simulation of the host machine hardware are simulated and the software unmodified runs in modified OS as separate system [18][19][15].

Partial virtualization: simulation of the physical hardware of a system and the software need some modifications to run [18][19].

Desktop Virtualization

Authorize to deploy multiple operating systems in a single machine, can be accessed remotely, providing security, flexibility and confidentiality of data. It also allow administrators to make configurations, security checks and updates on virtual desktops [18].

Network Virtualization

Allows coexistence of multiple virtual networks on the same physical server or device in real-time [20] [12]. Disassociates functionality in a network environment and separates the role of Internet service providers into two parts: Infrastructure providers (InP) and Service Providers (SP) [20].

2.1.3 Containers

Containers are a lightweight solution for virtualization using the host kernel to run multiple environments [1] as show in figure 2.2. Isolate software from the environment, allowing multiple applications to run [1] separating applications from infrastructures and providing isolated environments with resources, whose that can be shared with the host or installed separately, to execute applications.

Comparing to hypervisor-based virtualization, the performance of an container are by far much better, for not including an entire OS, the size and resources run in applications are less, more VM are deployed in the same host which provide an higher density and better performance[1]. Unfortunately with all the advantages, container-based virtualization is unable to support a variety of environments such as hypervisors virtualization[1].

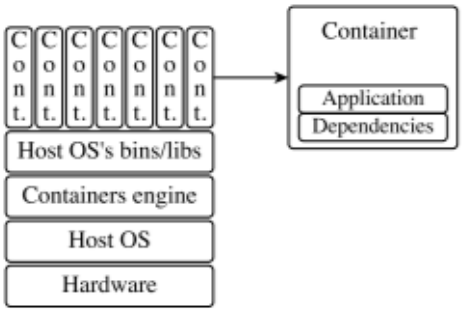


Figure 2.2: Container-based virtualization[1]

Docker

Docker is an open-source application container launched in 2013 developed using Golang-Go programming language - becoming the most popular container in cloud environment [2], due to is portability ease configuration and deployment is a lightweight alternative to VM [21]. Provides interfaces to create safely and control containers, more virtual environments, good cooperation with third-party tools, and also allows developers to pack application and operate anywhere without modifications compared to others platforms [1].

Integrates into the platform four (4) major components:

Docker Engine is a open-source, an client-server application composed by Docker Client, use a method of communication RESTful APIs that enable to run on the same host or in different hosts, also provides user interface for interactions between users, [1] receive commands and send to **Docker Daemon** which is responsible of container management [2]. **Docker Registry** stores and allow users to distribute Docker images. ***Docker Hub*** is a Software-as-a-Service platform that stores and shares Docker images [1].

2.2 Vulnerabilities

Protecting resources will always be important to the good performance of any functionality. The objective is to guarantee the confidentiality, integrity and availability of information and services **(CIA Triangle)** [22]:

- **confidentiality** keeps the data secure, preventing that unauthorized entities have access to sensible data.
- **integrity** prevent from unauthorized changes of information, keeping the data clean and untouchable.
- **availability** keeps the data accessible and protected from malicious attacks.

To protect systems is necessary to implement mechanisms, security measures to mitigate vulnerabilities in the software, also before protecting a system is necessary is the knowledge of vulnerabilities, risks and what harm they can bring to systems in general and users if they are available.

A **Vulnerability** is a weakness in the system, that can be explored to perform an **attack** which causes serious consequences[2]. When a vulnerability appears, it is necessary to report, which allows the responsible company verify and correct the vulnerability(ies). When verification is in progress, a patch is developed to maintain the software safe, fixing the vulnerabilities and released via updates to customers. **Risks** can be defined as factors that put a particular sector vulnerable, if the system is vulnerable occurs an **attack** by malicious attackers[23].

2.2.1 Impact of a vulnerability

Unfortunately, vulnerabilities arise due to poor development or design. With enormous pressure and competition between companies, functionality and speed are often decided on top of security and the emergence of vulnerabilities causes huge losses for companies and users. And when they are exploited, they allow attacks executed such as Denial of Service (DoS) attacks, information gathering, code execution and more attacks to be applied.

2.3 Vulnerabilities Databases

2.3.1 Database Docker Vulnerabilities

The study performed by Duarte, F. A., consisted of collecting Docker vulnerabilities by searching on Common Vulnerabilities and Exposure (CVE) report databases. Analyzed the vulnerabilities in order to understanding and verify if static code analyzers could prevent vulnerabilities. The vulnerabilities have been listed in a database accordingly to their description, causes, effects, consequences and corresponding patches[2].

Figure 2.5 lists the 25 vulnerabilities that affected Docker Security analyzed by Duarte, F. A.. Accordingly, to understand the vulnerabilities it was necessary to systematize them according to causes, consequences and effects as show in the figure 2.3 and figure 2.5, on what[2]:

- **Causes** is determined by the changes in the code to correct it, and the reason for the vulnerabilities existence, for example: incorrect configuration, improper security validation, unprotected resources.
- **Consequences** are the result of the attack when exploiting the vulnerability, example DoS.
- **Effects** is the impact in the system which leads to the consequence of the attack, example resource exhaustion and exposed system.

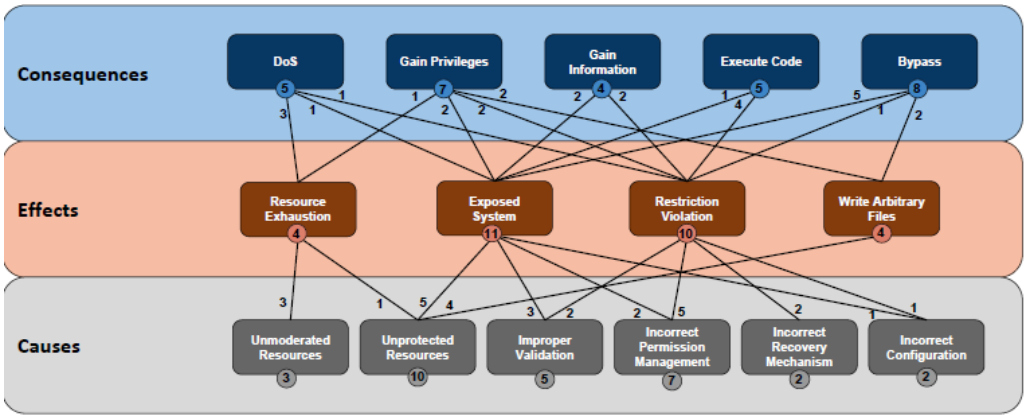


Figure 2.3: Classification of Docker Vulnerabilities (from [2])

In some cases, some vulnerabilities can have two or three consequences as showed in figure 2.3. Because of the smaller amount of stored vulnerabilities, the consequences of vulnerability exploitation are few in relative to the next database (KVM/Xen) as show in the figure 2.4.

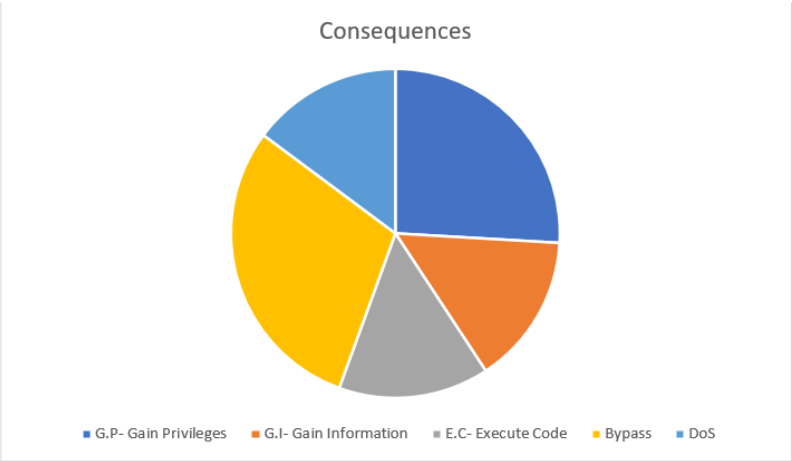


Figure 2.4: Vulnerabilities consequences (from [2]).

CVE	Cause	Effect	Consequence
2014-0047	Unprotected Resources	Exposed System	Gain Information
2014-3499	Incorrect Permission Management	Exposed System	Gain Privileges
2014-5277	Incorrect Recovery Mechanism	Restriction Violation	Gain Information
2014-5279	Unprotected Resources	Exposed System	Gain Privileges
2014-5280	Incorrect Permission Management	Restriction Violation	Execute code
2014-5282	Improper Validation	Exposed System	Bypass
2014-6407	Incorrect Permission Management	Restriction Violation	Execute Code
2014-6408	Incorrect Permission Management	Restriction Violation	Bypass
2014-8178	Incorrect Configuration	Exposed System	Execute code
2014-8179	Improper Validation	Restriction Violation	Execute code
2014-9356	Unprotected Resources	Write Arbitrary Files	Gain Privileges
2014-9357	Incorrect Permission Management	Restriction Violation	Execute Code
2014-9358	Improper Validation	Exposed System	Bypass
2015-1843	Incorrect Recovery Mechanism	Restriction Violation	Gain Information
2015-3627	Unprotected Resources	Write Arbitrary Files	Gain Privileges
2015-3629	Unprotected Resources	Write Arbitrary Files	Bypass
2015-3630	Incorrect Permission Management	Exposed System	Bypass
2015-3631	Unprotected Resources	Write Arbitrary Files	Bypass
2015-9258	Improper Validation	Exposed System	Bypass
2015-9259	Incorrect Permission Management	Restriction Violation	DoS
2016-3697	Improper Validation	Restriction Violation	Gain Privileges
2016-6595	Unmoderated Resources	Resource Exhaustion	DoS
2016-8867	Incorrect Configuration	Restriction Violation	Gain Privileges
2016-9962	Unprotected Resources	Exposed System	Gain Information
2017-6074	Unmoderated Resources	Resource Exhaustion	Gain Privileges
2017-6507	Unprotected Resources	Exposed System	Bypass
2017-11468	Unmoderated Resources	Resource Exhaustion	DoS
2017-14992	Unprotected Resources	Resource Exhaustion	DoS
2017-16539	Unprotected Resources	Exposed System	DoS

Figure 2.5: Vulnerabilities characterization by Duarte, A. (from [2]).

2.3.2 Database KVM and Xen vulnerabilities

The second database of vulnerabilities comes from an report by *Mendes, X.*[3], focused in two open-source hypervisors, **KVM/ Xen**. The database contains 344 vulnerabilities related to KVM and Xen found in National Vulnerability Database (NVD), as described in figure 2.8. The vulnerabilities were classified and systematized according to their CVE, Common Weakness Enumeration (CWE), causes, consequences and effects as described in figure 2.6.

In the work it is described the vulnerabilities, explained how they use a reverse engineering on the patch to fix vulnerabilities, identify different attack targets, attack vector and trigger source. With all information gathered is possible to understand different paths of an attack enabling better security enforcement.

Of the classification it is verified that there is six major consequences represented in figure 2.7, Denial of Service DoS, followed by Execute Code (EC), Obtain Information (OI), Gain Privileges (GP), Bypass and Directory Traversal (DT). Also, figure 2.7 shows the more types of consequences found in the database.

To fulfill the objective 263 of the 344 vulnerabilities, were utilized for the purpose of standards applications.

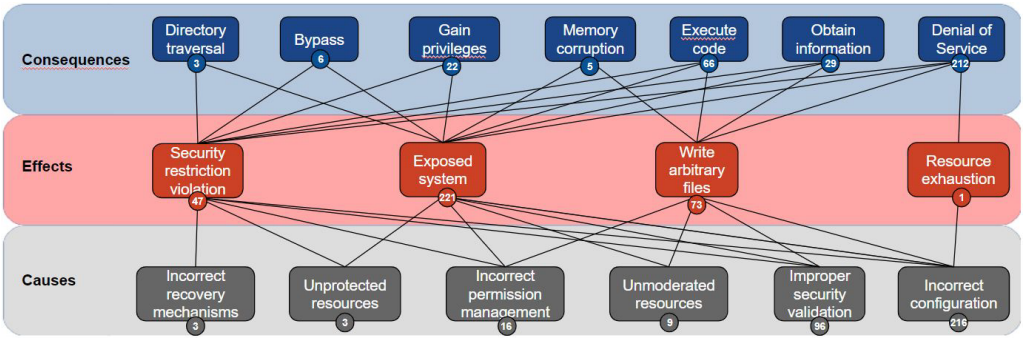


Figure 2.6: Classification of KVM/XEN vulnerabilities[3]

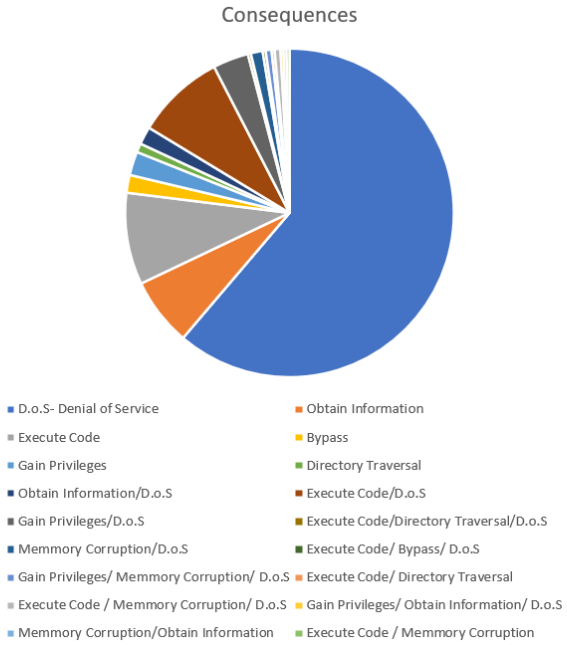


Figure 2.7: KVM/Xen vulnerabilities consequences from [3]

Database	Type of attack	Vulnerabilities stored by Mendes, X.
KVM and XEN	D.o.S	CVE-2015-5307, CVE-2020-8834, CVE-2019-7221, CVE-2019-6974, CVE- 2019-3887, CVE-2019-19332, CVE-2019-14821, CVE-2018-19407, CVE-2018-19406, CVE-2018-18021, CVE-2018-12904, CVE-2018-18021, CVE-2018-12904, CVE-2017-8106, CVE-2017-2596, CVE-2017-15306, CVE-2017-12168, CVE-2017-1000252, CVE-2016-9588, CVE-2016-8630, CVE-2016-5412, CVE-2015-8104, CVE-2015-7513, CVE-2015-4692, CVE-2014-8481, CVE-2014-8480, CVE- 2014-8369, CVE-2014-7842, CVE-2014-3690, CVE-2014-3647, CVE-2014-3646, CVE-2014-3645, CVE- 2014-3411, CVE-2014-3610, CVE-2014-0155, CVE-2013-6376, CVE-2013-6367, CVE-2013-4592, CVE-2013-4129, CVE-2013-1766, CVE-2012-4461, CVE-2012-2121, CVE-2012-2119, CVE-2012-1601, CVE-2012-1179, CVE-2012-0045, CVE-2011-4622, CVE-2011-4347, CVE-2010-5313, CVE-2010-3698, CVE-2010-0309, CVE-2009-4031, CVE-2009-3722, CVE-2009-3290, CVE-2009-2287, CVE-2009-1242, CVE-2020-8608, CVE-2020-13800, CVE-2020-13791, CVE-2020-13754, CVE-2020-13659, CVE-2020-13361, CVE-2020-13253, CVE-2020-12430, CVE-2020-11869, CVE-2020-11102, CVE-2020-10761, CVE-2020-10717, CVE-2019-6778, CVE-2019-5008, CVE-2019-3840, CVE-2019-20485, CVE-2019-20382, CVE-2019-15890, CVE-2019-15034, CVE-2019-14284, CVE-2019-14283, CVE-2019-12155, CVE-2018-5748, CVE-2018-5683, CVE-2018-20216, CVE-2018-20191, CVE-2018-20126, CVE-2018-20125, CVE-2018-20124, CVE-2018-20123, CVE-2018-19489, CVE-2018-19407, CVE-2018-19364, CVE-2018-18849, CVE-2018-18438, CVE-2018-17963, CVE-2018-17962, CVE-2018-17958, CVE-2018-15746, CVE-2018-12617, CVE-2018-10908, CVE-2018-1064, CVE-2017-9060, CVE-2017-8380, CVE-2017-8379, CVE-2017-8309, CVE-2017-8112, CVE-2017-8086, CVE-2017-7718, CVE-2017-7539, CVE-2017-7377, CVE-2017-6505, CVE-2017-6058, CVE-2017-5987, CVE-2017-5973, CVE-2007-4997, CVE-2016-5011, CVE-2017-1000407, CVE-2020-12888, CVE-2007-1366, CVE-2008-0928, CVE-2008-4553, CVE-2008-5714, CVE-2012-2652, CVE-2013-2230, CVE-2013-4153, CVE-2013-4154, CVE-2013-4377, CVE-2013-6458, CVE-2013-7336, CVE-2017-5857, CVE-2017-5579, CVE-2017-5578, CVE-2017-5552, CVE-2017-5526, CVE-2017-5525, CVE-2017-2633, CVE-2017-18043, CVE-2017-18030, CVE-2017-17381, CVE-2017-16845, CVE-2017-15289, CVE-2017-15119, CVE-2017-15118, CVE- 2017-13711, CVE-2017-11434, CVE-2017-11334, CVE-2017-10806, CVE-2017-10664, CVE-2016-9923, CVE-2016-9916, CVE-2016-9915, CVE-2016-9914, CVE-2016-9913, CVE-2016-9106, CVE-2016-9105, CVE-2016-9104, CVE-2016-9102, CVE-2016-8910, CVE-2016-8909, CVE-2016-8669, CVE-2016-8668, CVE-2016-8667, CVE-2016-8578, CVE-2016-8577, CVE-2016-8576, CVE-2016-8575, CVE-2016-8574, CVE-2016-7909, CVE-2016-7908, CVE-2016-7907, CVE-2016-7466, CVE-2016-7422, CVE-2016-7421, CVE-2016-7170, CVE-2016-7157, CVE-2016-7156, CVE-2016-7155, CVE-2016-6888, CVE-2016-6835, CVE-2016-6834, CVE-2016-6833, CVE-2016-6490, CVE-2016-5403, CVE-2016-5238, CVE-2016-4964, CVE-2016-4453, CVE-2016-4441, CVE-2016-4037, CVE-2016-4001, CVE-2016-3712, CVE-2016-2857, CVE-2016-10155, CVE-2015-8818, CVE-2015-8817, CVE-2015-8819, CVE-2015-8567, CVE-2015-8558, CVE-2015-7549, CVE-2015-7295, CVE-2015-6815, CVE-2015-5745, CVE-2015-5162, CVE-2015-4037, CVE-2015-1779, CVE-2014-9718, CVE-2014-8136, CVE-2014-8131, CVE-2014-7815, CVE-2014-3640, CVE-2014-3633, CVE-2014-3471, CVE-2014-0222, CVE- 2014-0146, CVE-2014-0142
	EC	CVE-2014-0049, CVE-2019-3812, CVE-2019-12068, CVE-2018-7550, CVE-2018-20815, CVE-2018-11806, CVE-2017-2630, CVE-2017-14167, CVE-2016-7161, CVE-2016-3710, CVE-2015-7512, CVE-2015-3214, CVE-2015-3209, CVE-2014-7840, CVE-2014-3461, CVE-2014-0182, CVE-2014-0150, CVE-2013-6399, CVE-2013-4542, CVE-2013-4541, CVE-2013-4540, CVE-2013-4539, CVE-2013-4537, CVE-2013-4535, CVE-2013-4532, CVE-2013-4527, CVE-2013-4151, CVE-2013-4149, CVE-2013-4148, CVE-2007-5729.
	OI	CVE-2020-2732, CVE-2019-7222, CVE-2017-17741, CVE-2017-12154, CVE-2016-9756, CVE-2016-4020, CVE-2013-7130, CVE-2010-4525, CVE-2010-3881, CVE-2020-13765, CVE-2020-13362, CVE-2019-9824, CVE-2019-8934, CVE-2018-18954, CVE-2018-16872, CVE-2017-15268, CVE-2017-15038, CVE-2016-9103, CVE-2016-6836, CVE-2016-5337, CVE-2014-3615, CVE-2008-2004, CVE- 2008-1945
	GP	CVE-2018-16882, CVE-2018-10901, CVE-2018-10853, CVE-2017-7518, CVE-2013-4587, CVE-2013-0311, CVE-2013-2016, CVE-2019-14835
	Bypass	CVE-2014-8134, CVE-2011-0011, CVE-2020-10702, CVE-2019-13164, CVE-2017-1000256, CVE-2011-2527
	DT	CVE-2018-12473, CVE-2020-7211, CVE-2016-7116

Figure 2.8: Vulnerabilities recorded in database[3]

2.4 Standards

Being applied since the beginning of industrial revolution, standards are very important to maintain a high level compliance for organizations, stakeholders and users. The concept of *"standards"* encompasses different notions and definitions, ranging from a specific group of products, services to complex definitions from standardization organizations.

Define by European Committee for Standardization (CEN) as ***"a technical document designed to be used as a rule, guideline or definition. It is a consensus-built, repeatable way of doing something."*** [24]. When applied they allow the various stakeholders, companies and organizations to communicate data, interconnection, portability reuse work and compliance [25].

Standards have different designated parts[26]:

- *De facto standards* created through informal adoption of norms or practices and used by users of a function.
- *Voluntary standards* are developed through voluntary consensus process, intended for optional use, which a regulating agency or company could adopt or mandate their use, in this case also stakeholders agree and participate in the process.
- *Mandatory standards* implement laws and regulations, his use is prescribed by implementing organizations or regulatory agencies.
- *Proprietary or company standards* developed by companies without external participation of third parties, with a specific practices for the company.
- *International Standards* are adopted by international organizations, designated as International Standards Development Organization (SDO) and make standards available for public.
- *Regional standards* are standards adopted by nations in different geographic region e.g. Europe, Asia.
- *National Standards* are standards developed for use by particular countries, which may contain rules different from those stipulated by international organizations.
- *Industry Standards* are standards directed to industrial entities.
- *Prescriptive Standards* specifies how requirements can be achieved, determined the design and construction.

Currently there are around 4000 different types of standards, each one with a defined area of application. From hundreds of organizations, only 30 international organizations with published standards are well-known and most common to be designated, in which each has an important and different role for the development and security of information, assets, software and resources.[27].

Below are named some of the most well-known and common international organizations[28]:

- National Institute of Standards and Technology (NIST)
- International Standard Organization (ISO)
- International Electrotechnical Commission (IEC)

- American National Standard Institute (ANSI)
- Organization for the Advancement of Structured Information Standards (OASIS)
- International Telecommunication Union (standardization section) (ITU-T)
- Payment Card Industry (PCI)

They develop and implement standards based on the consensus of users, organizations, governments and interest groups to help maximize security, quality, compatibility between the standards[26].

As the dissertation is entirely focused on vulnerabilities prevention in virtualization systems, it was researched around or more than 200 standards that are or can be directed towards the application and prevention of vulnerabilities, therefore four (4) major groups were created with their respective standards and examples. In the Venn's diagram are represented all the standards groups that will be presented and discussed (figure 2.9). Those in the center and in bold, we suppose are the right ones, capable of being applied for the prevention of vulnerabilities.

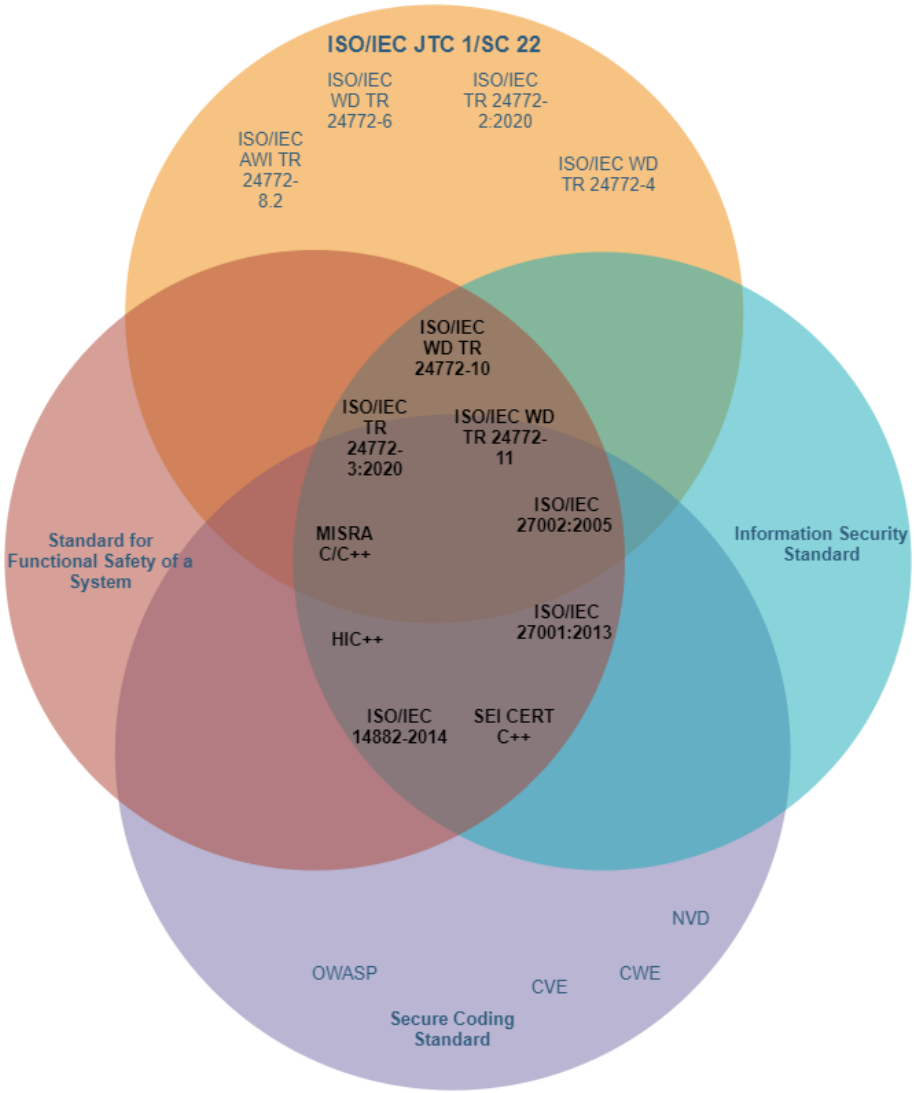


Figure 2.9: Venn diagram with all groups of standards categorized

2.4.1 ISO/IEC/JTC 1/SC 22/ Group

The first group is a specific set of standards grouped by ISO, called ISO/IEC JTC 1/SC 22/ in which it is responsible for standards related to programming languages, their environments and system software interfaces. In the set there are 139 standards and projects with the objective of providing support, preventing vulnerabilities in programming languages, providing library extensions for various languages namely, C, ADA, Java, C++, Eiffel, Fortran, Spark, Python and Forth.

To begin with, it was necessary from the 139 standards and projects that belong to this group, separate only the standards that are related to programming languages and prevention of vulnerabilities as shown in the figure 2.10. Of the first selection, we separated only the standards that are directly related to vulnerabilities prevention in programming languages will be explained, ***ISO/IEC TR 24772-3:2020 (C language), ISO/IEC WD TR 24772-10(C++ language) ISO/IEC WD TR 24772-11 (Java)***. Unfortunately there is no standard from this specific group directed towards the Golang- Go programming language.

ISO/IEC JTC 1/SC 22 standards aimed at preventing vulnerabilities in programming languages	Description
ISO/IEC WD TR 24772-11	Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 11: Guidance for programming language Java
ISO/IEC WD TR 24772-10	Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 10: Guidance for programming language C++
ISO/IEC AWI TR 24772-8.2	Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 8: Fortran
ISO/IEC WD TR 24772-6	Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 6: Spark
ISO/IEC WD TR 24772-4	Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 4: Python
ISO/IEC TR 24772-3:2020	Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 3: C
ISO/IEC TR 24772-2:2020	Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 2: Ada
ISO/IEC TR 24772-1:2019	Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 1: Language-independent guidance

Figure 2.10: ISO/IEC JTC 1/SC 22 standards aimed at preventing vulnerabilities in programming languages

The ISO/IEC/JTC 1/Subcommittee (SC) 22/ is a working group that analyze and identify vulnerabilities in programming language since 2006. The group documents the findings in ISO IEC TR 24772:2019 *"Programming languages — Guidance to avoiding vulnerabilities in programming languages* "[29] which described guidance for all types of vulnerabilities in different environments. The first edition was published in 2010 relied in CWE and MITRE, describing in portions vulnerabilities reported which gives developers an specific guidance of how to avoid vulnerabilities, but in that time was a lot of vulnerabilities not identified and documented[30].

Released the second edition in 2013 a document with new vulnerabilities and added language-specific annexes for different languages to help avoid or mitigate vulnerabilities identified. There was annexes for C, PHP, ADA, Python, Spark and Ruby[30].

In 2019, was released the third edition, which was subdivided into parts, each part contain different programming languages. TR 24772-1- Part 1: Language-independent guidance was released in 2019, in 2020 was released the other parts for each language[30]:

1. **ISO/IEC TR 24772-2:2020** - Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 2: Ada;
2. **ISO/IEC TR 24772-3:2020** - Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 3: C;
3. **ISO/IEC WD TR 24772-4** - Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 4: Python;
4. **ISO/IEC WD TR 24772-6**- Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 6: Spark;
5. **ISO/IEC AWI TR 24772-8.2**- Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 8: Fortran;
6. **ISO/IEC WD TR 24772-10** - Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 10: Guidance for programming language C++;
7. **ISO/IEC WD TR 24772-11**- Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages — Part 11: Guidance for programming language Java.

Unfortunately, as well as ISO/IEC14882-2014 from the **secure coding standard** group, it was not possible to get more information and the fundamentals documents of the standards to analyze, which made difficult the application process in vulnerabilities, but according to initial descriptions by ISO[29], they are guides to avoid vulnerabilities in various programming languages and we assume they are suitable into vulnerabilities prevention also.

2.4.2 Information Security Standard

Information being one of the greatest resources of a company, it is necessary to ensure that sensitive information does not become public knowledge. Companies must actively manage information security, hence the need to implement an information security management system takes a systematic approach to protecting and managing company's information. It has a set of rules necessary to be established, rules that are written in the form of security policies, procedures or other documents.

The **Information Security Management System (ISMS)** is done according to the ISO/IEC 27001:2013 standard. To implement a system, it is necessary to comply with all the requirements proposed by ISO/IEC 27001:2013, which range from defining the scope of the ISMS to implementing controls that allow rapid responses to security incidents. An active security system allows the reduction of information loss, fraud risks and shows how companies are committed to the security of their information, covering all aspects of CIA triangle.

After complying with all the requirements of ISO/IEC 27001:2013, the company obtains a security certificate that makes customers, shareholders and suppliers feel secure regarding the availability, integrity and confidentiality of their data. Numerous companies, stakeholders believe that it only takes ISO/IEC 27001:2013 to obtain a certificate and keep the information and system safe, but also is necessary the intervention of ISO/IEC 27002:2005. The standard ISO/IEC 27001:2013 aims to define the requirements for ISMS and ISO/IEC 27002: 2005 provides detailed guidance to organizations on how to implement the controls of ISO/IEC 27001:2013 present in ANNEX A (5).

ISO/IEC 27000 FAMILY

Currently, there are 50 standards in the ISO/IEC 27000 series. ISO/IEC 27000 is considered as *vocabulary standard* [31], provides an overview of the set of standards developed to work with information security management systems, also provides terms and definitions that are used in the information security management system. Due to importance and content, the chosen ones were ISO/IEC 27001:2013 and ISO/IEC 27002:2005.

ISO/IEC 27001:2013

Main standard of the ISO 27000 family, revised in 2013, internationally recognized and being the most applied, focus on Information Security Management Systems (ISMS). ISO/IEC 27001:2013 published by ISO in partnership with the IEC, provides a list of best practices for information security and lists all certification requirements for implementing and maintaining an organization's information security systems[32].

ISO / IEC 27001:2013 provides a solid framework for companies to decide on the best suitable protection for their information systems and assets. The only disadvantage of this pattern is that it defines what is needed, but it doesn't specify how it should be done, so it needs other information security standard. Has the aim of to protect the confidentiality, integrity and availability of data through security controls and procedures. The standard is not only applied in the areas of information technology, it can also be applied in the health[33], telecommunications[34], software development[35] and other areas[32].

Operation of ISO/IEC 27001:2013

A complex process ranging from control selection, risk assessment, risk treatment to legal security compliance, looking for potential threats that can jeopardize the confidentiality, integrity and availability of information. It's process is designed to make the organization select the different technological and management controls that are adequate to protect information assets.

The standard is divided in two parts: the main section and appendix A 5. The main section specifies a general structure of information security, using a cycle model called **Plan-Do-Check-Act (PDCA)** (also known as "Deming Wheel")[4] in order to implement, establish, monitor and evaluate the effectiveness and quality of an organization. Appendix A 5 lists the information security controls to be implemented, total of 114 controls.

ISO/IEC 27002:2005 helps to configure the controls of appendix A from ISO/IEC 27001:2013. It should also be noted that controls are not mandatory for organizations, each chooses the controls needed to be implemented.

Cycle PDCA

The PDCA cycle, also known as "Deming Wheel" and "Shewhart Cycle" is a method used to organize processes, interactive. It is divided into 4 phases for control and improvement of processes and products as shown in the figure 2.11[4]:

• Plan phase (clauses 4 to 7) - Identify information security risks, define security policies according to the organization's context, define scopes, risk treatment plan.

• Phase "Do" (clause 8) - Implementation of the risk treatment plan, processes and risks and also risk assessment.

• Phase "Check" (clause 9) – Monitoring, management review, external audit.

• Phase "Act" (clause 10) - Implementation of corrective actions.

Figure 2.11: PDCA Cycle[4][5]

Anyone who thinks that ISO/IEC 27001:2013 is not suitable for the software area is wrong. ISO/IEC 27001:2013 is very versatile and some of the controls in Annex A are relevant for testers and developers. The loss of sensitive data due to bad configurations and bad code writing, leads international organizations specialized in standards to ensure that important specifications are met so as not to lose important data and suffer attacks. Therefore, it is necessary for each organization to ensure that projects store data on external host systems, that there is a secure development environment for the good fulfillment of the development cycle, which in terms of contracting has to be taken into account the standard is to hire high quality developers and testers, who have knowledge in the area of secure coding and basic programming fundamentals.

ISO / IEC 27002: 2005 Information Technology - Code of Practice for Information Security Control

A guideline - code of practice for information security, an updated version of the former ISO / IEC 17799:2005 standard revised in 2005, which aims to guide organizations to implement ISO / IEC 27001: 2013 security controls through best practice recommendations, providing a more detailed explanation of each recommendations[36].

It is based on risk management, with the purpose of generating adequate security policies, procedures for the control of risks in information systems. The weaknesses of this standard is that it does not contain product-oriented measures (in this point it is weaker than COBIT) and cannot be used to acquire certifications, being the work of ISO / IEC 27001: 2013[36].

2.4.3 Standard for Functional Safety of a System

This second category groups functional safety standards. The functional safety standards are designed to ensure that users are protected from injury due to software failures, human errors, environmental factors, with the aim of mitigating every risk and failure. For this group, fundamental and extremely important standards were selected such as MISRA C/C++ and HIC++ for software development.

Motor Industry Software Reliability Association (MISRA) C/C++

MISRA in 1990 developed the coding guidelines called MISRA-C[37][38], an older standard specific to the C programming language, and over the years the C++ programming standard was added. A development standard in C and C++ language, whose main objective is to offer a set of best practices for safe and secure software development formed by manufacturers and engineering consultants and is considered safe and reliable as it allows it to be used by the automotive industry[39] and but over the years also to other areas such as aerospace, communications, medical devices and railways[38].

Types of MISRA C/C++

MISRA C/C++ is divided in two main guidelines, **directive** and **rules**, which all together are 173 guidelines, 4 directives.

1. **Directive** is a guideline where the information that concerns compliance is not fully contained in the source code (external source) or requirements or specifications[38].
2. **Rules** is a guideline where information is fully contained in the source code[38].

MISRA C/C++ released editions

There are three editions of MISRA C/C++ published: 1998, 2004, 2012[38]. Each edition contains a number of coding rules with respectively amendments and additions.

The first edition of guidelines was MISRA C: 1998, published in 1998 contained 127 guidelines. Used for software development in critical systems for quality requirements, where it is necessary to specify guidelines to applied a disciplined software that makes the development process possible[38].

The second edition published in 2004, an improved version of C guidelines, focused on critical systems, with purpose of include all industries that developed C software [38][37]. In 2008 with the implementation of C++ in critical systems a new version was published especially for MISRA C++ [38][40].

The third edition and last version MISRA C/C++:2012 released in 2012, providing a clarification of the rules of other editions proving more examples, covers more languages problems, more detailed explanations of the existing rules, added additional safety guidelines called Amendments (1 and 2) [38][40].

MISRA C/C++ - Rules, amendments

All the rules from the MISRA guidelines follows a certain format:

Rules <**number**>(<**category**>)**:** <**application text**> where:

• The <number> is the unique number of each rule;

• The <category> is classification of the rule, which can be classified as *"required"*- the C code that complies to MISRA C shall comply with every required guideline[38]; or *"advisory"* - recommendations that should be followed; non-compliance's should be documented[38]; It has a third non-optional called *"mandatory"*- all the C code that complies to MISRA C shall comply with every mandatory guideline[38];

The following lines shows one example of directive and rule structure.

Example of a Directive structure

An example of how a **Directive** from MISRAC/C++:2012 is structured[41]:

Dir 2.1 All source files shall compile without any compilation errors

Category Required

Applies to C90, C99

Rationale

A conforming compiler is permitted to produce an object module despite the presence of compilation errors. However, execution of the resulting program may produce unexpected behaviour.

Example of Rules structure[41]

Rule 1.2 Language extensions should not be used

Category Advisory

Analysis Undecidable, Single Translation Unit

Applies to C90, C99

Rationale

A program that relies on language extensions may be less portable than one that does not. Although The Standard requires that a conforming implementation document any extensions that it provides to the language, there is a risk that this documentation might not provide a full description of the behaviour in all circumstances.

If this rule is not applied, the decision to use each language extension should be justified in the project's design documentation. The methods by which valid use of each extension will be assured, for example checking the compiler and its diagnostics, should also be documented.

It is recognized that it is necessary to use language extensions in embedded systems. The Standard requires that an extension does not alter the behaviour of any strictly conforming program. For example, a compiler might implement, as an extension, full evaluation of binary logical operators even though The Standard specifies that evaluation stops as soon as the result can be determined. Such an extension does not conform to The Standard because side effects in the right-hand operand of a logical AND operator would always occur, giving rise to a different behaviour.

Amendments

The amendments are additional, also designed safety guidelines, was introduced in the new version of MISRA C/C++:2012. In amendment 1 there are 14 rules to be complied with, whereas in amendment 2 there are changes in the structure of the rules, corrections, removal and addition of new syntax and sections.

Compliance and implementation of MISRA C/C++

To comply with MISRA C/C++ it is mandatory to know how to apply all rules and directives, check codes frequently, set baselines, prioritize violations based on risk, document

deviations, verify that the code continues to comply with MISRA C/C++, hence the need to choose a correct code analyzer so that accurate, automatically correct analysis can be done prioritizing vulnerabilities based on risk. To implement MISRA C/C++ it is necessary to follow specific steps such as first writing the code, performing a static analysis and if errors are found, the code must be corrected, passing again the static analysis and when it is without errors (called "clean code") compiles and the software is ready to use.

High Integrity C++ (HIC++)

Published in 2003, it is considered one of the most respected longest established adopted C++ coding standards with initially 202 semantic and syntactic rules and guidelines, designed HIC++ V3.3, helping developers write high quality code in C++, with best practices for software development, specifying coding rules provides examples of compliant and non-compliant code, with goal of defending against dangerous, confusing language constructs. Now with new updates contain only 155 rules and it is designed as HIC++ V4.0[42]. With rules and diretives has mane **MISRA C and HIC++ comparison**

The construction of the HIC++ standard is done in a similar way as MISRA C/C++, same safety approach, rules and directives rules written in similar ways. There's a slightly difference in terms of target domain, on what MISRA C/C++:2012 targets software for use in critical embedded systems that can result in long term failure[42] and HIC++ is designed to operate in a less constrained software environment where system resources can reasonably expect to be acquired, external access and exception handling techniques that can accommodate a wide range of failures[42]. Also MISRA C/C++ have distinction to HIC++ in terms of source code analysis, MISRA C/C++ goes through analysis of source code which ends up dividing them into rules and diretives and its documented outside of the source code while HIC++ have a set of rules that doesn't need to split the analysis like MISRA C/C++ does and allows an analysis to be poured into the code[42].

2.4.4 Secure Coding Standard

Coding standards provide discipline, guidelines and rules for a better code quality. The main goal is to create layers of protection for software code, allowing coding errors to be identified and corrected as quickly as possible contributing to a safe software[43].

In this group will be presented the public database repositories (CVE, CWE, NVD), together with the best practices Open Web Application Security Project (OWASP) and the SEI CERT C++, MISRA C/C++, HIC++ and ISO / IEC 14882-2014 standards.

Public Database Repositories

Common Vulnerabilities and Exposures (CVE)

CVE provides a list of vulnerabilities found in specific software with identification numbers, description and public reference. Responsible for categorizing vulnerabilities into classes, it provides a list of software vulnerabilities in order to help organizations understand the types of vulnerabilities that can be found in their software. It has an unique Identifier by a CVE Numbering Authority (CNA) which makes it easy to identify vulnerabilities and cross-reference them with other repositories for investigation and research[44][45].

Common Weakness Enumeration (CWE)

Operated by MITRE CORPORATION, CWE provides a list of the most common and critical weaknesses that can lead to serious software vulnerabilities grouped into categories, with the aim of helping to understand, discuss and deal with the vulnerabilities found and which tools can be used to prevent, mitigate or identify future vulnerabilities. The list is compiled according to community scores and feedback[46][47].

National Vulnerability Database (NVD)

A US repository of standards-based vulnerability management data is connected to the CVE list and provides additional content including how to fix vulnerabilities, severity scores and impact ratings. It have a relationship with CVE where the NVD is based on CVE records with detailed information for each record of vulnerabilities.[48].

BEST PRACTICES

OWASP

An entity called OWASP "Open Web Application Security Project", whose objective is to strengthen software security through tools and knowledge of various experts [23]. Define a set of application coding security best practices that can be integrated throughout software development.

It is a project made up of specialists in the security area, who carry out a risk assessment and ways to combat these risks. They implement, develop and test different web applications in order to guarantee a safe final product. A free and open security community, it have a list of the 10 most persistent and highest risk security risks for web applications and effective methods to prevent and combat. It should be noted that OWASP is not a standard, but a benchmarking of best pratices project to deal with software security[49].

OWASP released in 2017, the list of the 10 most persistent risks for web application: [50][49][23]:

- **Injection** Occur when untrusted data is used as an injector vector, the attacker's send malicious code to interpreter which can trick to execute unintended commands, result in data loss, denial of service. Injection flaws, such as SQL, NoSQL, OS, and LDAP injection.
- **Broken Authentication** allow attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to gain privileges to control the systems, users' identities temporarily or permanently.
- **Sensitive Data Exposure and Session Management** unprotected data and sensitive data, attackers modify and compromise data to steal and execute attacks which allows to gain information.
- **XML External Entities (XXE)** external entities with poorly configured XML processors can be used to disclose internal files using the file URI handler, internal file shares, remote code execution, and denial of service attacks.

- **Broken Access Control** security restrictions not suitable for users, allowing an attacker to access unauthorized data, gain privileges and access to control the system, change access rights and modify sensitive data.
- **Security Misconfiguration** lack of upgrade and poor security misconfigurations allows attackers to gain unauthorized access and privileges.
- **Cross-Site Scripting XSS** inclusion of untrusted data in a web page without proper validation, allowing attacker's to execute scripts in victim's browser which can hijack user sessions or redirect the user to malicious sites.
- **Insecure Deserialization** Insecure deserialization leads to remote code execution. replay attacks, and privilege escalation attacks.
- **Using Components with Known Vulnerabilities** the use of components known vulnerabilities facilitates the exploitation and enable attacks that lead to data loss or server takeover.
- **Insufficient Logging and Monitoring** insufficient logging leading to breach allowing attackers to extract and destroy data

For the analyses and application of standards, OWASP Secure Coding Practices - Quick Reference Guide [51] was used has one of to prevent vulnerabilities, which was developed for the benefit of the community of programmers and security of developed systems, aims to define a set of good security practices in application development. All recommendations are presented in a checklist format, which can be integrated into the application development cycle. Using these security measures can reduce the most common vulnerabilities in web applications.

In the OWASP Secure coding practices guide there are 14 checklists [51]:

- **Input data validation**.
- **Output data encoding**.
- **Authentication and credential management**.
- **Session management**.
- **Access control**.
- **Encryption Practices**.
- **Error Handling and Logging**.
- **Data protection**.
- **Communications security**.
- **System setup**.
- **Database security**.
- **File management**.
- **Memory management**.
- **General Programming Practices**.

STANDARDS

SEI CERT C++

The Software Engineering Institute Computer Emergency Response Team (SEI CERT C++) is a secure coding standard that supports commonly used programming languages such as C, C ++ and Java. Has a set of rules similar to MISRA C++, focus on security of applications written by applying a set of guidelines and recommendations, performing a risk assessment to determine the possible consequences of the violation or recommendation to avoid certain risks and improvements to be made in the code[52]. Developed by the Software Engineering Institute at Carnegie Mellon University, it guarantees the protection of software against security vulnerabilities.

MISRA C/C++ and HIC++

MISRA C/C++ and HIC++ are also part of this group for secure standard. MISRA C/C++ is widely-used and commonly accepted industry standard, is also presented in the standard group for the functional safety of a system. Reliable and secure with its rules for C and C++ language for software development, it is use in telecommunications, railways, automotive, industrial areas. As explained in the previous section, it has in its constitution 173 rules, 4 diretives with amendments 1 and 2 and addenda. **HIC++** is a coding standard that helps write high quality, portable and robust code, has in its constitution 155 coding rules for better quality software development.

ISO/ IEC 14882-2014

A **information technology- programming languages C++ standard**, that specifies a set of requirements for implementations of the C++ programming language, providing additional templates, data types, classes, exceptions, functions names, declarations, standard conventions, expressions and library[53]. The C++ language in this standard is based on C programming language rules which were described in ISO/IEC 9899:1999 Programming language - C[53].

Unfortunately it was not possible to obtain more information and necessary documents about the standard, as it is a standard that needs to be paid to be available. As it is not possible to analyze in full, is not be possible to align it with other standards for vulnerability prevention.

This page is intentionally left blank.

Chapter 3

Standards Applications

This chapter focuses on standards application verifying if standards could avoided vulnerabilities in hypervisors and containers.

3.1 Selection of standards

For Docker vulnerability database, the standards ISO/IEC 27001:2013 and best practices OWASP were those that complement with the programming language -**Golang**. Due to language differences, unfortunately was not possible to apply Motor Industry Software Reliability Association (MISRA) C/C++, High Integrity C++ (HIC++), SEI CERT C++ standards rules on Golang.

For KVM/Xen vulnerability database, the selected standard MISRA C/C++, HIC++, ISO/IEC 27001:2013, SEI CERT C++ and best practice Open Web Application Security Project (OWASP) complement with present programming language vulnerabilities. Unfortunately due to restrictions of access to documents with the rules, it was not possible to apply the International Standard Organization (ISO)/International Electrotechnical Commission (IEC) 14882-2014 standards, ISO/IEC TR 24772-3:2020 (C language), ISO/IEC WD TR 24772-10(C++ language) and ISO/IEC WD TR 24772-11 (Java).

3.2 Analysis of vulnerabilities patches and Application of Standards

Hypervisors virtualization are claimed to be more secure than container-based virtualization, due to a extra layer of isolation between applications. Containers connected directly with the host, allowing an easy attack in to the system.[1].

On the next pages, a set of example of patches with the vulnerabilities will be demonstrated and how the bugs that gave rise to the vulnerabilities could be avoided by standards. It was analyzed 263 vulnerabilities, which 25 vulnerabilities are from Docker database systematized by Duarte, F. A. [2] and the other's 238 vulnerabilities are from KVM/Xen database systematized by Mendes, X.[3].

The approach for analyzing vulnerabilities and application of standards consisted of the following steps:

1. Individual analysis of each vulnerability patches found in the databases (Docker [2] and KVM/Xen) described as example in figure 3.1 and coming figures.

2. When analyzing vulnerabilities, assign rules or best practices to vulnerabilities that could prevent the appearance of vulnerabilities.

Docker vulnerabilities analysis

This subsection is dedicated to demonstration of few examples of patch and which standard could be applied to prevent vulnerabilities that appear in the patches.

CVE-2014-6407

```
@@ -292,11 +293,23 @@ func createTarFile(path, extractDir string, hdr *tar.Header, reader io.Reader, L
292 293                 }
293 294
294 295         case tar.TypeLink:
295     -               if err := os.Link(filepath.Join(extractDir, hdr.Linkname), path); err != nil {
    296 +               targetPath := filepath.Join(extractDir, hdr.Linkname)
    297 +               // check for hardlink breakout
    298 +               if !strings.HasPrefix(targetPath, extractDir) {
    299 +                       return breakoutError(fmt.Errorf("invalid hardlink %q -> %q", targetPath, hdr.Linkname))
    300 +               }
    301 +               if err := os.Link(targetPath, path); err != nil {
296 302                         return err
297 303                 }
298 304
299 305         case tar.TypeSymlink:
    306 +               // check for symlink breakout
    307 +               if _, err := symlink.FollowSymlinkInScope(filepath.Join(filepath.Dir(path), hdr.Linkname), extractDir); err != nil {
    308 +                       if _, ok := err.(symlink.ErrBreakout); ok {
    309 +                               return breakoutError(fmt.Errorf("invalid symlink %q -> %q", path, hdr.Linkname))
    310 +                       }
    311 +                       return err
    312 +               }
300 313                 if err := os.Symlink(hdr.Linkname, path); err != nil {
301 314                         return err
302 315                 }
```

Source: `https://github.com/moby/moby/commit/1852cc38415c3d63d18c2938af9c112fbc4dfc10#diff-53174c8e1f0ff67b3e3cb9149146a37d`

Figure 3.1: Code snippet from patch 2014-6407.

The problem present in figure 3.1 was the Docker engine was vulnerable to extracting files to arbitrary paths on the host during 'docker pull' and 'docker load' operations. This was caused by *symbolic link* and *hardlink* traversals present in the Docker image extract. This vulnerability could be leveraged to perform remote code execution and privilege escalation. With incorrect permission management, non authorized users could get access modifying and disclosure information.

Code fixes were done correctly because no related vulnerabilities appeared in later patches.

For prevention accordingly to OWASP it would be necessary to use one of the checklists- **Access Control**[23]- *to restrict access to files and other resources, including those outside the application's direct control, to authorized users only.*

For ISO/IEC 27001:2013 there is the need to establishes a requirement to control and restrict privileged access rights, protecting against misuse or deliberate access - **Privileged access rights management - A.9.2.3[32]**.

CVE-2014-5277

The problem present in figure 3.2 is HTTPS with registration failure allowing main in the middle with downgrade attacks and getting authentication and image data. With incorrect recovery mechanism, the docker client could, under certain circumstances, erroneously fall back to HTTP an insecure connection when an HTTPS connection registration fails.

For prevention accordingly to OWASP it would be necessary to use one of the checklists- **Input data validation**[23]- *to validate all data from clients prior to processing, including all parameters, form fields, URL contents and HTTP headers, such as Cookie names and values. Also be sure to include postback mechanisms.*

For ISO/IEC 27001:2013 there is the need to establishes that users should only have access to the network and network services they need to use; Authorization procedures to show who (based on role) is allowed to access what and when. **-Access to networks and network services - A.9.1.2 [32]**

```
43  43          endpoint.URL.Scheme = "https"
44  44          if _, err := endpoint.Ping(); err != nil {
45      -               log.Debugf("Registry %s does not work (%s), falling back to http", endpoint, err)
46      -               // TODO: Check if http fallback is enabled
        45  +
        46  +               //TODO: triggering highland build can be done there without "failing"
        47  +
        48  +               if secure {
        49  +                       // If registry is secure and HTTPS failed, show user the error and tell them about `--insecure-registry`
        50  +                       // in case that's what they need. DO NOT accept unknown CA certificates, and DO NOT fallback to HTTP.
        51  +                       return nil, fmt.Errorf("Invalid registry endpoint %s: %v. If this private registry supports only HTTP or HTTPS with an unknown CA certificate,
        52  +               }
        53  +
        54  +               // If registry is insecure and HTTPS failed, fallback to HTTP.
        55  +               log.Debugf("Error from registry %q marked as insecure: %v. Insecurely falling back to HTTP", endpoint, err)
47  56              endpoint.URL.Scheme = "http"
48      -               if _, err = endpoint.Ping(); err != nil {
49      -                       return nil, errors.New("Invalid Registry endpoint: " + err.Error())
        57  +               _, err2 := endpoint.Ping()
        58  +               if err2 == nil {
        59  +                       return endpoint, nil
50  60              }
        61  +
        62  +               return nil, fmt.Errorf("Invalid registry endpoint %q. HTTPS attempt: %v. HTTP attempt: %v", endpoint, err, err2)
51  63          }
52  64
53  65          return endpoint, nil
54  66      }
55      - func newEndpoint(hostname string) (*Endpoint, error) {
        67  + func newEndpoint(hostname string, secure bool) (*Endpoint, error) {
56  68          var (
57      -               endpoint        Endpoint
        69  +               endpoint        = Endpoint{secure: secure}
```

Source:
`https://github.com/moby/moby/commit/6a1ff022b0744213ed588d9c16dbb13ce055eda6`

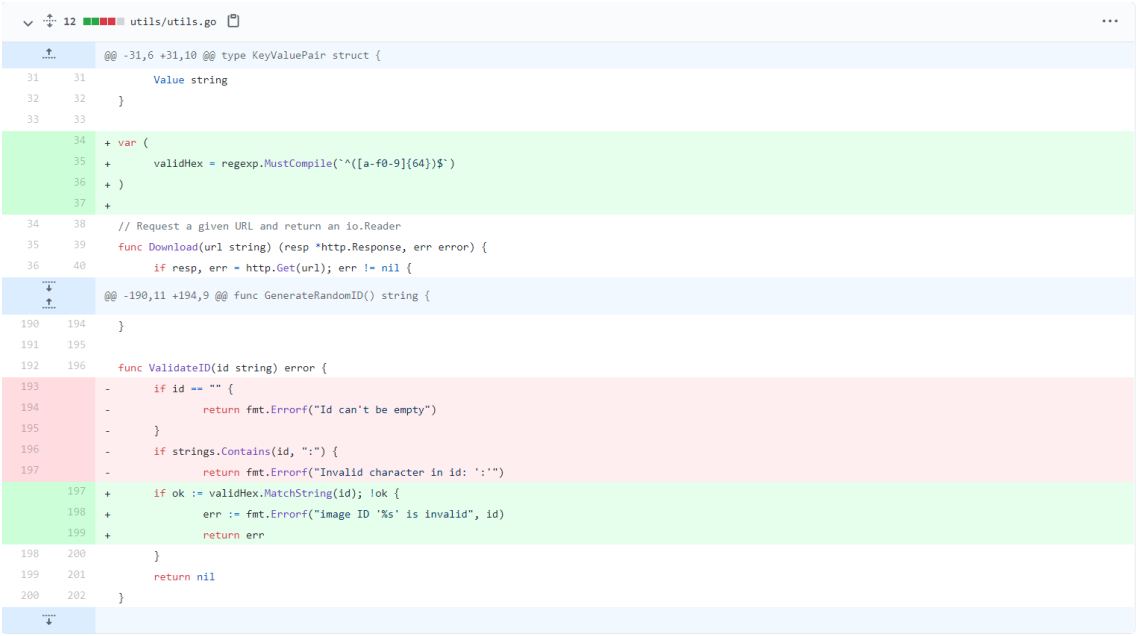
Figure 3.2: Code snippet from patch 2014-5277.

CVE-2014-9358

The problem presented in figure 3.3 the Docker does not sufficiently validate (improper validation) the image IDs as provided through 'docker load' or through registration communications. This allows for *"path traversal"* attacks, causing graphical corruption and malicious image manipulation, as well as repository spoofing attacks.

For prevention accordingly to OWASP it would be necessary to use one of the checklists- **Input data validation**[23]- *validate data size.*

Unfortunately we didn't find anything in the ISO/IEC 27001:2013 that could be related to data size validation and applied to this type of vulnerability.



Source:
`https://github.com/moby/moby/commit/bff1d9dbce76bed1e267a067eb4a1a74ef4da312`

Figure 3.3: Code snippet from patch 2014-9358

KVM/XEN vulnerabilities analysis

This subsection is dedicated to demonstration of few examples of patch and which standard could be applied to prevent vulnerabilities that appear in the patches. For this specific database was selected standard MISRA C/C++, HIC++, ISO/IEC 27001:2013, SEI CERT C++ and best practice OWASP for application in vulnerabilities.

CVE-2020-8834

The problem present in figure 3.4, was the implementation of KVM in the Linux kernel for PowerPC which processors did not properly keep guest state separate from host state. A local attacker on a guest KVM can use this to cause a Denial of Service (DoS).

For prevention it is necessary to establish a separation of privileges and a reduction in the number of resource sections. With **OWASP** best practices- **Access Control[23]**- it would be necessary to restrict access to files and other resources, including those outside the application's direct control, to authorized users only.

Source: `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=7b0e827c6970e8ca77c60ae87592204c39e41245`

Figure 3.4: Code snippet from patch 2020-8834

OWASP Best Practices for prevention

Access Control[23]

- Restrict access to files and other resources, including those outside the application's direct control, to authorized users only.
- Restrict access to protected URLs to authorized users only.
- Restrict access to protected functions to authorized users only.
- Restrict access to direct object references to authorized users only.
- Restrict access to application data to authorized users only.
- Restrict access to user attributes and data, as well as policy information used by access control mechanisms.

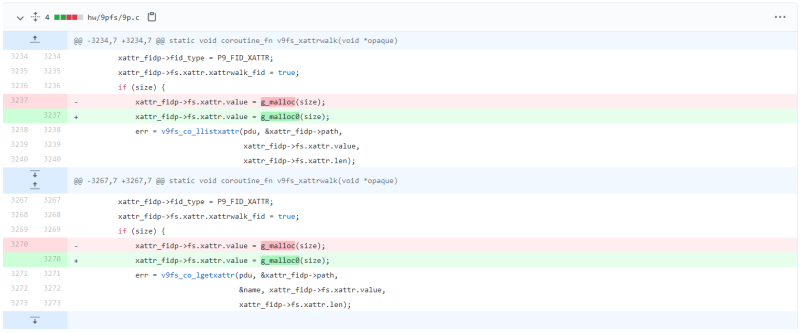
- Restrict access to relevant security settings to authorized users only.

ISO / IEC 27001:2013

For this context the annex control used for application is **annex 9 - access control**. Was applied A.9.2.3 - management of Privileged Access Rights, aims to manage privilege levels of access[32].

CVE-2017-15038

The vulnerability: *9p back-end first queries the size of an extended attribute, allocates space for it via g malloc() and then retrieves its value into allocated buffer. Race between querying attribute size and retrieving its could lead to memory bytes disclosure. Use g malloc0() to avoid it.*



Source: `https://github.com/qemu/qemu/commit/7bd92756303f2158a68d5166264dc30139b813b6`

Figure 3.5: Code snippet from patch CVE-2017-15038

For the prevention: rules from MISRA C/C++ and SEI CERT++.

MISRA C/C++

The C language, being a language that promotes performance rather than code security, doesn't make it easy for programmers to write secure code. For MISRA C/C++, writing more secure codes is necessary to replace some functions by equivalent ones that are more secure and not allow the placement of unknown variables or functions.

There are in MISRA C/C++ rules and directive that prohibit the use of flexible array members: Dir. 4.1 and Rule 21.3[41]

Dir 4.12 Dynamic memory allocation sh all not be used

Category- Required

This rule applies to all dynamic memory allocation packages including:

- Those provided by The Standard Library;
- Third-party packages.

The Standard Library's dynamic memory allocation and deallocation routines can lead to undefined behaviour as described in Rule 21.3. Any other dynamic memory allocation system is likely to exhibit undefined behaviours that are similar to those of The Standard Library.

The specification of third-party routines shall be checked to ensure that dynamic memory allocation is not being used inadvertently. If a decision is made to use dynamic memory,

care shall be taken to ensure that the software behaves in a predictable manner. For example, there is a risk that:

- Insufficient memory may be available to satisfy a request — care must be taken to ensure that there is a safe and appropriate response to an allocation failure;
- There is a high variance in the execution time required to perform allocation or deallocation depending on the pattern of usage and resulting degree of fragmentation.

Rule 21.3 The memory allocation and deallocation functions of <stdlib.h> shall not be used

Category Required Analysis Decidable, Single Translation Unit

Amplification The identifiers *calloc, malloc, realloc* and free shall not be used and no macro with one of these names shall be expanded.

Use of dynamic memory allocation and deallocation routines provided by The Standard Library can lead to undefined behaviour, for example:

- Memory that was not dynamically allocated is subsequently freed;
- A pointer to freed memory is used in any way;
- Accessing allocated memory before storing a value into it

Note: this rule is a specific instance of Dir 4.12.

SEI CERT C++

For SEI CERT C++, there two fundamental rules to aplly in this context. First Rule: ***7.2 MEM51-CPP. Properly deallocate dynamically allocated resources[54]***

There several ways to allocate memory, for example such as std::malloc(), provide by C programming language which can be used by a C++ program.

Second rule:***7.4 MEM53-CPP. Explicitly construct and destruct objects when manually managing object lifetime[54]***

Chapter 4

Discussion

As new threats emerge, regulations and standards continue to increase in number and complexity. Now many organizations are forced to comply with various industry regulations and mandates, failure to comply with laws, caring penalties for data breaches and for not complying with notification. In this dissertation is presented an overview of relevant standards concerning application in vulnerabilities in virtualization systems. Such standards encompass a large number of complex and extensive guidelines and rules that provides.

4.1 Observations from Docker and KVM/Xen vulnerabilities analysis and standards applications

Docker's vulnerabilities database

With the analysis of patches and application of standards we verified that:

1. For the type of programming language it was only possible to work with one standard, ISO/IEC 27001:2013 and a best practice OWASP, which can make the results very widespread.

2. First, the three detailed vulnerability analyzes mentioned above show a recurring pattern of Docker vulnerabilities. The main problem found in majority of vulnerabilities is the mismanagement of access privileges, that is why the recommendations most of the time focused on the same types of checklists and standard control.

3. For some of the cases, OWASP and standard ISO/IEC 27001:2013 may have very generic solutions, not demonstrating exactly what could be changed.

4. The OWASP checklist - *Access control and input data validation[23]* and ISO/IEC 270011:13 control - *A.9 Access control* were the dominant factors in the prevention criterion.

5. There were cases of return of the same vulnerability in the following months or the following year, which means that the error correction was not done correctly, for example the vulnerability CVE 2014-5277, the problem was HTTPS with registration failure which allows main in the middle with downgrade attacks and getting authentication and image data, failure to correct it correctly led to the re-emergence of the vulnerability that is designated as CVE 2015-1843. CVE 2015-184 only exists because of a CVE-2014-5277 regression.

6. It is urgent to create a standard that addresses the Go programming language, to allow rules and directives to be applied to prevent vulnerabilities.

Within the OWASP access control checklist there is 14 types of recommendations that can be applied, but within these recommendations, the chosen ones were, according to the vulnerabilities, two (2) different types:

OWASP Best Practices

Access Control[23]:

- Restrict access to files and other resources, including those outside the application's direct control, to authorized users only.
- Restrict access to protected URLs to authorized users only.
- Restrict access to protected functions to authorized users only.
- Restrict access to direct object references to authorized users only.
- Restrict access to application data to authorized users only.
- Restrict access to user attributes and data, as well as policy information used by access control mechanisms.
- Restrict access to relevant security settings to authorized users only.

Input Data Validation[23]:

- Validate all data coming from customers prior to processing, including all parameters, form fields, URL contents and HTTP headers, such as Cookie names and values. Also be sure to include postback mechanisms.
- Validate data range.
- Validate data length.
- Validate expected data types.

ISO / IEC 27001:2013

For this context the annex control used for application is **annex 9 - access control**[32]. This control divided into four subsections(A.9.1, A.9.2, A.9.3, A.9.4), aims to establish access control requirements, access control policy implementation, user access management (authorization for certain systems, services and applications) and user responsibilities. Was applied A.9.2.3 - management of Privileged Access Rights[32], aims to manage privilege levels of access.

KVM/Xen vulnerabilities database

1. Rules in MISRA C/C++ and SEI Cert C/C++ are complex and difficult to put in practice and even understand.
2. In this database, OWASP followed with the same types of checklist as in the Docker vulnerability database: Input Data Validation and Access Control.

3. The application of standards in vulnerabilities was much more complex, as they have extremely extensive rules.

As previously written in the introduction, questions are designed to help you really understand how vulnerability prevention standards can be applied.

For this the need to understand, questions were made that can be answered during the development of the investigation:

1. **Does the implementation of standards really reduce the appearance of vulnerabilities in the software of virtualization systems?**

   Yes, with all the research and documents coming from international organizations, standards can really help to prevent vulnerabilities, because in each document there is a specific set of rules to build secure and objective codes. Each programming language has a specific set of rules for writing the code. So just follow the rules. The problem is not with the rules but with the developers. Based on vulnerabilities analysis and standards application, there is a behaviour that needs to be changed. To improve the security and quality of software development, developers need to learn secure coding, for example in the Docker Database, many vulnerabilities reappeared due to incorrect correction of a certain code. Developers are aware that standards are necessary to have a quality product, but at the same time they think that standards can restrict or slow down the development, which can make an average product.

2. **All the standards can be applied or does it need a very specific group of standard to apply for vulnerabilities prevention?**

   No, there are standards for all types of services and resources and each standard is specific a determined area, for example there is a group of standards for metal fabrication, for aircraft construction, medicines and several other extreme important areas, also there is a book with denominated KWIC Index with approximately 4000 standards for all areas to be applied. For vulnerabilities prevention there is specific group determined by ISO and with more investigations can be found more that can be added for more protection.

3. **For standards implementation, do developers need more hours of classes on secure coding or will they be able to deal with the extensive rules of standards?**

   Due to stress, pressure and development time, many developers fall back on habits, and with the lack of good software practices, they write poor quality code which consequently makes the software vulnerable and full of bugs. There's a awareness that rules are extensive and complex, but there is no way possible to summarize the rules and recommendations. Majority of times standards are ignored by disenchanted programmers who dislike or disagree with it's poorer and extend guidelines.

Advantages of standards application

- Reduce number of attacks, because of application of rules and best practices on vulnerabilities.
- Provide low cost of repairs, because it was fixed in advance.
- Provide speedy recovery, if have any type of system down.
- Provide low costs in terms of communication between stakeholders and authorities.

4.2 Security concerns in Hypervisors

Analyzing Docker and KVM databases the most common vulnerabilities are buffer overflows in KVM database and Privilege escalation vulnerability in Docker.

Buffer Overflow - Buffer overflow vulnerabilities are intertwined with buffers or memory allocations in languages that provide low-level direct access to read and write memory. In C languages, reading or writing to allocations does not entail any automatic bounds checking. This results in data being written beyond its end and overwriting the contents of subsequent addresses on the stack or heap, or extra data being read[55][56] the non-correction allows attacks like DoS, execution of arbitrary code .

How to Detect Buffer Overflow in Source Code

1. First, paying attention to where buffers are used, modified and accessed. Note the functions that handle input provided by a user or other external source.
2. Look for external inputs and buffer manipulations.
3. Know which functions are susceptible to buffer overflow vulnerability. The ***gets, strcopy, strcat, printf / sprint*** functions are functions that write beyond the limits of the buffer provided to them[40].

Mitigation's against buffer overflow vulnerabilities

Would like to recommend the set of best practices from OWASP that can help developers ensure prevention of buffer-overflow vulnerabilities. Buffer overflow prevention through OWASP comes from the best practices in the memory management section[23]:

- When using functions that accept a certain number of bytes to perform copies, such as strncpy(), be aware that if the destination buffer size is equal to the source buffer size, it cannot terminate the null-valued string (null).
- Check buffer limits if function calls are made in cycles and check that there is no risk of overwriting data beyond the reserved space.
- Avoid using known vulnerable functions like printf(), strcat(), strcpy() etc.

Also, researches[55][57] mention that one of the easiest ways to prevent buffer overflow vulnerabilities is simply to use a language that does not allow these types of vulnerabilities to occur. Unfortunately, the C language allows direct access to memory, which facilitated the appearance of the vulnerability. Languages like Java, Python, ADA, and more languages are virtually immune to buffer overflow vulnerabilities Languages that do not share these aspects are typically immune. Use of safe practices and rules that deal with buffers.

Privilege escalation vulnerability - The Privilege Escalation vulnerability occur when the operating system or application becomes vulnerable, by allowing a user to use another user's privileges to access a certain system[58].

Prevention against Privilege escalation vulnerability

1. Separation of privilege in the begin of the development
2. Minimize Privileges in order to limit the access as an intruder may able to get some information misusing the privileges

Privilege escalation prevention through OWASP comes from the Access Control section which[23]:

- Restrict access to files and other resources, including those outside the application'sdirect control, to authorized users only.
- Restrict access to protected URLs to authorized users only.
- Restrict access to protected functions to authorized users only.
- Restrict access to direct object references to authorized users only.
- Restrict access to application data to authorized users only.
- Restrict access to user attributes and data, as well as policy information used by access control mechanisms.
- Restrict access to relevant security settings to authorized users only.

Chapter 5

Conclusions

Due to the large increase in the number of vulnerabilities reported by CERT in recent years[59] (in 2016 alone, 2,300 vulnerabilities were reported) there is a concern on the part of large companies to build secure software, preventing bugs from being implemented, not allowing malicious attackers to exploit systems failing through vulnerabilities. Due to the alarming numbers of vulnerabilities, there is an increasing concern in have knowledge in the common threats, adopting a set of best practices to solve the problem.

In this dissertation it was researched and developed the state-of-the-art of the various and different standards existing, with elaboration of groups with standards that have potential for vulnerabilities prevention, also the analysis of the vulnerabilities to understand how they were discovered, how can they be exploited, and what caused the vulnerability in the first place and the end, the code examination and specific standards applications and discussions.

It took a significant amount of time to full understand the different types, because of the complexity and accessibility, majority required a payment to purchase. Although the difficulty, the extensive analysis from standards and vulnerabilities provided enough information to make concise and relevant observations, we verified that there are numerous standards capable of vulnerability prevention, depending of the vulnerability some can give generic solutions, other's can be more specific. Another observation the main reason of vulnerabilities to appear is because developers make always the same errors and try to fix the bugs but end up keeping the same bugs or they can see the real problem and try to soften the problem, but unfortunately gets back months or years later.

Majority of vulnerabilities from databases, did not have patches, which makes it difficult to really see where the correction was made and if the correct correction was made. Unfortunately, organizations often find standards redundant, a generic solution to an extremely individualized problem set and don't have influence in the problem, sometimes can be too expensive or impossible to keep up, particularly those that operate on a global perspective. While the rules are very complex and sometimes difficult to understand, organizations, developers will continue to use tools such as static code analyzers for testing procedures and vulnerability detection.

Sometimes standards are not apply in isolation, there can combination of multiple standards that which imposes specific requirements, that enhance security and do not conflict with other standards and supportive and complementary.

Finding vulnerabilities in patches and applying standards seems to be a viable remediation of preventing and exploiting vulnerabilities but given the complex nature of hypervisor

code, it is unlikely that all vulnerabilities will be found and all standards can be applied to vulnerabilities.

We believed that this is good a contribute for improvement of the security of the hypervisors and containers by providing the developers other perspective of vulnerability prevention. Some recommendations for security improvement, minimize privileges in order to limit the access as an attacker may able to get some information misusing the privileges, validate all input values and verify the size of the inputs, avoid or not user functions like gets; strcpy; strcat; sprintf; vsprintf, they should be replaced by functions like fgets; strncpy; strncat; snprintf, all string results need to be null terminated, it is necessary to provide basic secure coding training to all developers because they think that other teams will be checking the code for errors and flaws and so they don't need to worry about developing the code safely, not being forced to follow a set of rules or guidelines.

References

[1] Thanh Bui. Analysis of docker security. *arXiv preprint arXiv:1501.02967*, 2015.

[2] Ana Filipa Sêco Duarte. *Security Assessment and Analysis in Docker Environments*. PhD thesis, Universidade de Coimbra, 2018.

[3] Xavier Mendes. Hypervisor's vulnerabilities discovering process, 2020.

[4] Ronald Moen and Clifford Norman. Evolution of the pdca cycle, 2006.

[5] Michal Pietrzak and Joanna Paliszkiewicz. Framework of strategic learning: The pdca cycle. *Management (18544223)*, 10(2), 2015.

[6] Herb Sutter and Andrei Alexandrescu. *C++ coding standards: 101 rules, guidelines, and best practices*. Pearson Education, 2004.

[7] Katerina Goseva-Popstojanova and Andrei Perhinschi. On the capability of static code analysis to detect security vulnerabilities. *Information and Software Technology*, 68:18–33, 2015.

[8] Melina Kulenovic and Dzenana Donko. A survey of static code analysis methods for security vulnerabilities detection. In *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1381–1386. IEEE, 2014.

[9] Dejan Baca, Kai Petersen, Bengt Carlsson, and Lars Lundberg. Static code analysis to detect software security vulnerabilities-does experience matter? In *2009 International Conference on Availability, Reliability and Security*, pages 804–810. IEEE, 2009.

[10] Alexandru G Bardas et al. Static code analysis. *Journal of Information Systems / Operations Management*, 4(2):99–107, 2010.

[11] Alan Litchfield and Abid Shahzad. A systematic review of vulnerabilities in hypervisors and their detection. In *Twenty-third Americas Conference on Information Systems*. Association for Information Systems (AIS), 2017.

[12] Edward Ray and Eugene Schultz. Virtualization security. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, pages 1–5, 2009.

[13] Steven J Vaughan-Nichols. Virtualization sparks security concerns. *Computer*, 41(8):13–15, 2008.

[14] Oracle. Brief history of virtualization.

[15] John Hoopes. *Virtualization for security: including sandboxing, disaster recovery, high availability, forensic analysis, and honeypotting*. Syngress, 2009.

[16] Massimo Cafaro and Giovanni Aloisio. Grids, clouds, and virtualization. In *Grids, Clouds and Virtualization*, pages 1–21. Springer, 2011.

[17] Fatma Bazargan, Chan Yeob Yeun, and Mohamed Jamal Zemerly. State-of-the-art of virtualization, its security threats and deployment models. *International Journal for Information Security Research (IJISR)*, 2(3/4):335–343, 2012.

[18] Aaqib Rashid and Amit Chaturvedi. Virtualization and its role in cloud computing environment. *International Journal of Computer Sciences and Engineering*, 7(4), 2019.

[19] Gabriel Cephas Obasuyi, Arif Sari, et al. Security challenges of virtualization hypervisors in virtualized hardware environment. *International Journal of Communications, Network and System Sciences*, 8(07):260, 2015.

[20] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[21] Charles Anderson. Docker [software engineering]. *Ieee Software*, 32(3):102–c3, 2015.

[22] Lorenzo D Martino and Elisa Bertino. Security for web services: Standards and research issues. *International Journal of Web Services Research (IJWSR)*, 6(4):48–74, 2009.

[23] Dave Wichers. Owasp top-10 2017. *OWASP Foundation, February*, 2017.

[24] European Committee for Standardization. What is Standard. 2021.

[25] CEN Compass and CEN Compass. The world of european standards. *The CEN management center*, 2008.

[26] Karen Scarfone, Dan Benigni, Tim Grance, et al. Cyber security standards. *National Institute of Standards and Technology (NIST), Gaithersburg, Maryland*, 2009.

[27] Sophie J Chumas. *Index of international standards*, volume 13. National Bureau of Standards, 1974.

[28] Eduardo Fernández-Medina and Mariemma I Yagüe. State of standards in the information systems security area, 2008.

[29] ISO. Standards by iso/iec jtc 1/sc 22 programming languages, their environments and system software interfaces.

[30] Stephen Michell. Time issues in programs vulnerabilities for programming languages or systems. *ACM SIGAda Ada Letters*, 36(1):77–82, 2016.

[31] Klaus Haller. what developers and testers need to know about the iso 27001 information security standard. *Texting experience/The magazine for professional tester*, 2014.

[32] SNV Schweizerische. Information technology-security techniques-information security management systems-requirements. *ISO/IEC International Standards Organization*, 2013.

[33] S Tyali and D Pottas. Information security management systems in the healthcare context. In *Proceedings of the South African Information Security Multi-Conference: Port Elizabeth, South Africa, 17-18 May 2010*, page 177. Lulu. com, 2011.

[34] NK Sharma, Prabir Kumar Dash, et al. Effectiveness of iso 27001, as an information security management system: an analytical study of financial aspects. *Far East Journal of Psychology and Business*, 9(3):42–55, 2012.

[35] Mirabela Luciana GAŞPAR and Sorin Gabriel Popescu. Integration of the gdpr requirements into the requirements of the sr en iso/iec 27001: 2018 standard, integration security management system in a software development company. *Acta technica napocensis-series: Applied Mathematics, Mechanics, and Engineering*, 61(3/*Spe*), 2018.

[36] Shamsul Sahibudin, Mohammad Sharifi, and Masarat Ayat. Combining itil, cobit and iso/iec 27002 in order to design a comprehensive it framework in organizations. In *2008 Second Asia International Conference on Modelling / Simulation (AMS)*, pages 749–753. IEEE, 2008.

[37] Roberto Bagnara, Abramo Bagnara, and Patricia M Hill. The misra c coding standard and its role in the development and analysis of safety-and security-critical embedded software. In *International Static Analysis Symposium*, pages 5–23. Springer, 2018.

[38] Roberto Bagnara, Michael Barr, and Patricia M Hill. Barr-c: 2018 and misra c: 2012: Synergy between the two most widely used c coding standards. *arXiv preprint arXiv:2003.06893*, 2020.

[39] Miroslaw Staron. Detailed design of automotive software. In *Automotive Software Architectures*, pages 117–149. Springer, 2017.

[40] C Misra. Guidelines for the use of the c language in critical systems. *MIRA Limited. Warwickshire, UK*, 2004.

[41] C Misra. Guidelines for the use of the c language in critical systems. *MIRA Limited. Warwickshire, UK*, 2012.

[42] Wojciech Basalaj and Richard Corden. High integrity c++ coding standard v4. 0-an overview. 2013.

[43] Gary McGraw. Software security. *IEEE Security & Privacy*, 2(2):80–83, 2004.

[44] Steve Christey and Robert A Martin. Vulnerability type distributions in cve. *Mitre report, May*, 2007.

[45] Peter Mell and Tim Grance. Use of the common vulnerabilities and exposures (cve) vulnerability naming scheme. Technical report, National Inst of Standards and Technology Gaithersburg md Computer Security Div, 2002.

[46] Steve Christey, J Kenderdine, J Mazella, and B Miles. Common weakness enumeration. *Mitre Corporation*, 2013.

[47] Robert A Martin and Sean Barnum. Common weakness enumeration (cwe) status update. *ACM SIGAda Ada Letters*, 28(1):88–91, 2008.

[48] Ju An Wang and Minzhe Guo. Ovm: an ontology for vulnerability management. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, pages 1–4, 2009.

[49] Matthew Bach-Nutman. Understanding the top 10 owasp vulnerabilities. *arXiv preprint arXiv:2012.09960*, 2020.

[50] Jinfeng Li. Vulnerabilities mapping based on owasp-sans: a survey for static application security testing (sast). 2020.

[51] K Turpin. Owasp secure coding practices-quick reference guide, 2010.

[52] Aaron Ballman and David Svoboda. Avoiding insecure c++—how to avoid common c++ security vulnerabilities. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 65–65. IEEE, 2016.

[53] C++ Standards Committee et al. iso/iec 14882: 2011, standard for programming language c++. Technical report, Technical report, 2011. http://www. open-std. org/jtc1/sc22/wg21, 2011.

[54] CERT SEI CERT C Coding Standard. Rules for developing safe, reliable, and secure systems. *Software Engineering Institute—Carnegie Mellon University*, 2016.

[55] David Larochelle and David Evans. Statically detecting likely buffer overflow vulnerabilities. In *10th USENIX Security Symposium*, 2001.

[56] Kyung-Suk Lhee and Steve J Chapin. Buffer overflow and format string overflow vulnerabilities. *Software: practice and experience*, 33(5):423–460, 2003.

[57] Crispin Cowan, F Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. In *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, volume 2, pages 119–129. IEEE, 2000.

[58] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. In *USENIX Security Symposium*, 2003.

[59] CERT RO et al. Report on cyber security alerts processed by cert-ro in 2016. *International Journal of Information Security and Cybercrime (IJISC)*, 6(1):83–92, 2017.

Appendices

This page is intentionally left blank.

Appendix A

Annex A of ISO / IEC 27001:2013- CATALOG OF SAFETY CONTROLS APPLICABLE FOR ORGANIZATIONS

There are 114 controls listed in ISO / IEC 27001:2013 and they are structured into 14 sections (A.5 to A.18) and each section contains a number of controls[32].

• A.5 Information security policies – control over policies, the objective is to support the level of information security and comply with the requirements of organizations (regulations, laws). Contains 2 controls A.5.1.1, A.5.1.2.

• A.6 Information security organization – Internal organization, establishes a structure of controls in order to control implementation, ensure that responsibilities are correctly assigned, one of the important controls for obtaining ISO 27001 certification. A.6.1.1, A.6.1.2, A.6.1.3, A.6.1.4, A.6.1.5, A.6.2.1, A.6.2.2.

• A.7 Human resource security – ensuring that all hiring are done in the right way, contractors must act in accordance with their responsibilities and roles and that the security responsibilities implemented by the company are met. It is a control used for before, during and after hiring. Contains controls A.7.1.1, A.7.1.2, A.7.2.1, A.7.2.2, A.7.2.3, A.7.3.1.

• A.8 Asset management – responsible controls to identify, classify information assets and define the protection responsibilities for those assets and handling of "media". Contains controls A.8.1.1, A.8.1.2, A.8.1.3, A.8.1.4, A.8.2.1, A.8.2.2, A.8.2.3, A.8.3.1, A.8.3.2, A.8.3.3.

• A.9 Access control – access control requirements, implementation of access control policy, user access management (authorization to certain systems, services and applications), and user responsibilities. Contains controls A.9.1.1, A.9.1.2, A.9.2.1, A.9.2.2, A.9.2.3, A.9.2.4, A.9.2.5, A.9.2.6, A.9.3.1, A.9.4.1, A.9.4.2, A.9.4.3, A.9.4.4, A.9.4.5.

• A.10 Encryption – controls for the use of cryptography to ensure the 3 pillars of security (confidentiality, integrity and availability) through usage policies and key management. Contains controls A.10.1.1, A.10.1.2.

• A.11 Physical and environmental security – Define safe areas and security boundaries, protection against possible threats, define areas of authorized and unauthorized physical access, protection against external and environmental threats, equipment security. Contains controls A.11.1.1, A.11.1.2, A.11.1.3, A.11.1.4, A.11.1.5, A.11.1.6, A.11.2.1, A.11.2.2, A.11.2.3, A.11.2.4, A.11.2.5, A.11.2.6, A.11.2.7, A.11.2.8, A.11.2.9.

• A.12 Operations security – ensuring the security of information processing resources, IT production management, backups, resource monitoring, event logging. Contains controls A.12.1.1, A.12.1.2, A.12.2.1, A.12.3.1, A.12.4.1, A.12.4.2, A.12.4.3, A.12.4.4, A.12.5.1, A.12.6.1, A.12.6.2, A.12.7.1.

• A.13 Communications security – network security, protection of information on the network, transfer of information. Contains controls A.13.1.1, A.13.2.1, A.13.1.3, A.13.2.1, A.13.2.2, A.13.2.3, A.13.2.4.

• A.14 Acquisition, development and maintenance of systems – definition of security requirements for public and private networks, data protection, security in development and support processes. Contains controls A.14.1.1, A.14.1.2, A.14.1.3, A.14.2.1, A.14.2.2, A.14.2.3, A.14.2.4, A.14.2.5, A.14.2.6, A.14.2.7, A.14.2.8, A.14.2.9, A.14.3.1.

• A.15 Relationship with suppliers – protection of information assets of organizations in relation to suppliers, development of security policies, security in supplier contracts. Contains controls A.15.1.1, A.15.1.2, A.15.1.3, A.15.2.1, A.15.2.2.

• A.16 Information security incident management – defining responsibilities, reporting incidents, events and security weaknesses, responsibilities and incident response and response procedures. Contains controls A.16.1.1, A.16.1.2, A.16.1.3, A.16.1.4, A.16.1.5, A.16.1.6.

• A.17 Information security aspects in business continuity management – continuity of information security, requesting procedures, verification, business continuity and IT review, implementation of information process resources. Contains controls A.17.1.1, A.17.1.2, A.17.1.3, A.17.2.1.

• A.18 Compliance – compliance with laws and regulations, identification of laws, protection of personal data, intellectual property rights and information security reviews. Contains controls A.18.1.1, A.18.1.2, A.18.1.3, A.18.1.4, A.18.1.5, A.18.2.1, A.18.2.2, A.18.2.3

This page is intentionally left blank.