INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Intelligent System for Automation of Security Audits (SIAAS)

João Pedro Delgado Seara

Masters in Telecommunications and Computer Engineering
Dissertation

Supervision:
PhD Carlos José Corredoura Serrão, Associate Professor,
ISCTE – Instituto Universitário de Lisboa

November, 2023

TECNOLOGIAS
E ARQUITETURA

Department of Information Science and Technology (DCTI-ISTA)

Intelligent System for Automation of Security Audits (SIAAS)

João Pedro Delgado Seara

Masters in Telecommunications and Computer Engineering
Dissertation

Supervision:
PhD Carlos José Corredoura Serrão, Associate Professor,
ISCTE – Instituto Universitário de Lisboa

November, 2023

Aos meus pais

Acknowledgments

First of all, I want to express my gratitude to Prof. Carlos Serrão for guiding me through the development and writing of this dissertation.

Also, to Prof. Nuno Souto, for accepting my re-enrollment request in this master's course, and to Prof. Isabel Alexandre for providing me valuable advice in the earlier steps of this work.

To all the artifact testers: Matt Golden from Trilio Data (thank you also for reviewing this dissertation), Ricardo Ramalho from Altice Portugal, and Jorge Teixeira from VTXRM.

To my company, for supporting me in my academic efforts. More exactly, my current manager, Shin Horiuchi. This acknowledgment extends to my former companies and managers.

To the FOSS community.

And finally – above all – to my close family for always being there for me.

Thank you all for your support and input. None of this would be possible without you.

Resumo

Eventos relacionados com falhas de cibersegurança têm um elevado e crescente impacto financeiro, operacional, e reputacional, em organizações por todo o mundo. Ao mesmo tempo, há uma escassez de profissionais de cibersegurança. Além disso, a especialização de profissionais com as competências necessárias na área da cibersegurança é dispendiosa e morosa. Esta investigação adereça estes problemas ao desenvolver um rastreador de vulnerabilidades que é gratuito para a comunidade, e não requer pré-especialização por parte do operador. O artefacto foi testado por empresas de TI, por engenheiros de sistemas, na sua maioria sem especialização específica na área de cibersegurança. Os resultados demonstram que o artefacto foi fácil de instalar e reportou resultados que podem ser usados pelo operador no contexto de uma securitização automática e proativa dos sistemas envolvidos.

Palavras-chave: cibersegurança, auditoria de segurança, rastreador de vulnerabilidades, software de código livre

Abstract

Events related to cybersecurity failures have a high and growing financial, operational, and reputational impact, on organizations around the world. At the same time, there is a shortage of cybersecurity professionals. In addition, the specialization of professionals with the necessary skills in the area of cybersecurity is expensive and time-consuming. Taking these facts into consideration, this research has focused on the automation of cybersecurity processes, specifically those related to continuous vulnerability detection. To address this problem, a cybersecurity vulnerability scanner that is free to the community and requires no pre-expertise on the part of the operator, was developed. The artifact was tested by companies in the IT business, by systems engineers, most without cybersecurity background. The results demonstrated that the artifact was easy to install and that the reported results can be used by the operator in the context of an automatic and proactive securitization of the systems involved.

Keywords: cybersecurity, security auditing, vulnerability scanner, free open-source software (FOSS)

Contents

List of Figures, Tables, Equations, and Technical Notes

Figures

Tables

Equations

Technical Notes

Glossary

AAA - Authentication, Authorization, and Accounting
AI - Artificial Intelligence
AIO - All-In-One
AMQP - Advanced Message Queueing Protocol
API - Application Programming Interface
ARM - Advanced RISC Machine
ARP - Address Resolution Protocol
BSON - Binary JavaScript Object Notation
CA - Certificate Authority
CCE - Common Configuration Enumeration
CISA - Cybersecurity and Infrastructure Security Agency
CLI - Command Line Interface
CPE - Common Platform Enumeration
CPU - Central Processing Unit
CRUD - Create, Read, Update, and Delete
CSV - Comma-Separated Values
CVE - Common Vulnerabilities and Exposures
CVSS - Common Vulnerability Scoring System
CWE - Common Weakness Enumeration
DB - Database
DevSecOps - Development, Security, and Operations
DNS - Domain Name System
DR - Disaster Recovery
DSRM - Design Science Research Methodology
FOSS - Free and Open-Source Software
FS - File System
FTP - File Transfer Protocol
GB - Gigabyte
GHz - Gigahertz
GNU - GNU’s Not Unix
GPL - GNU’s General Public License
GPT - Generative Pre-Trained Transformer
HA - High Availability
HTTP - Hyper Text Transfer Protocol
HTTPS - Hyper Text Transfer Protocol Secure
I/O - Input/Output
ICMP - Internet Control Message Protocol
ICT - Information and Communications Technology (interchangeable with IT)

IDE - Integrated Development Environment
IEEE - Institute of Electrical and Electronics Engineers
IP - Internet Protocol
IPv4 - IP version 4
IPv6 - IP version 6
ISCTE-IUL - ISCTE – Instituto Universitário de Lisboa (Portuguese translation of “University Institute of Lisbon”)
ISO - International Organization for Standardization
IT - Information and Technology (interchangeable with ICT)
JSON - JavaScript Object Notation
KB - Kilobyte
KVM - Kernel-based Virtual Machine
LAN - Local Area Network
LXC - Linux Containers
MAC - Media Access Control
MB - Megabyte
MBSA - Microsoft Baseline Security Analyzer
MHz - Megahertz
MITRE - Massachusetts Institute of Technology Research and Engineering
ML - Machine Learning
NASL - Nessus Attacking Scripting Language
NDP - Neighborhood Discovery Protocol
NFS - Network File System
NIC - Network Interface Card
NIST - National Institute of Standards and Technology
NSE - Nmap Scripting Engine
NVD - National Vulnerability Database
NVT - Network Vulnerability Test
OOM - Out Of Memory
OS - Operating System
OVAL - Open Vulnerability and Assessment Language
OWASP - Open Worldwide Application Security Project
PenTesting - Penetration Testing
PEP - Python Enhancement Proposals
PnP - Plug and Play
PRISMA - Preferred Reporting Items for Systematic Reviews and Meta-Analysis
PTES - Penetration Testing Execution Standard
R&D - Research and Development
RAM - Random Access Memory
REST - Representational State Transfer
RFC - Request For Comments
RISC - Reduced Instruction Set Computer
SaaS - Software as a Service
SBC - Single-Board Computer
SCAP - Security Content Automation Protocol
SIAAS - Sistema Inteligente para Automação de Auditorias de Segurança (Portuguese translation of “Intelligent System for Automation of Security Audits”)
SIEM - Security Information and Event Management

SMB - Small and Medium Business
SOAR - Security Orchestration, Automation and Response
SOC - Security Operations Center
SQL - Structured Query Language
SSH - Secure Shell
SSL - Secure Sockets Layer (interchangeable with TLS)
TCP - Transmission Control Protocol
TLS - Transport Layer Security (interchangeable with SSL)
UDP - User Datagram Protocol
UI - User Interface
UID - Unique Identifier
URI - Uniform Resource Identifier (interchangeable with URL)
URL - Uniform Resource Location (interchangeable with URI)
US - United States (of America)
USD - US Dollar
UTC - Universal Coordinated Time
UUID - Universally Unique Identifier
UX - User Experience
VM - Virtual Machine
Wi-Fi - Wireless Fidelity
WLAN - Wireless Local Area Network
x86 - 8086 microprocessor family
XCCDF - Extensible Configuration Checklist Description Format
XML - Extensible Markup Language
XSS - Cross-Site Scripting
XXE - XML External Entity

CHAPTER 1

Introduction

An introductory contextualization of the presented work. The context and motivation for this work will be explained. This motivation emerges from a specific problem needing to be solved, and specific research question(s) will be formulated taking into consideration if and how this problem can be solved. Finally, the methodology for research and the planning of all the phases of work – from problem formulation to development of an artifact and its results/documentation/communication – will be detailed.

1.1. Context and Motivation

Cybersecurity-related occurrences are often in the news. The number of global attacks keeps increasing in recent years [1]. Two of the most impactful recent occurrences that raised awareness for the importance of proper security auditing were the *WannaCry* and *Petya* global ransomware outbreaks of 2017 [2].

Back in 2020, a study by the US Cybersecurity and Infrastructure Security Agency (CISA) was made to “to understand the impacts, costs, and losses from cyber incidents, to enable cyber risk analysis and inform cybersecurity resource allocation decisions” [3]. This study compiled a list of other studies from organizations and private companies. It was found that, from 2013 onwards, the global average annual cost estimate of cybersecurity incidents ranged from approximately 1.75 trillion to a projection for 2021 of 6 trillion (6.000.000.000.000) USD.

Figure 1 shows a summary of the studies that were condensed in this global study.

Study	Data Subset	U.S. Annual Cost ($ billions)	Global Annual Cost ($ billions)		
			Lower	Best	Upper
Symantec (2017)	BEC-Only Subset	~$1	-	-	-
FBI (2016, 2017)	BEC-Only Subset	~$0.53	-	~$1.76	-
IC3 (2017)	BEC Subset	-	-	$0.676	-
	Cyber Subset	-	-	$0.980	-
	Total Dataset	-	-	$1.420	-
IC3 (2019)	BEC Subset	-	-	$1.298	-
	Cyber Subset	-	-	$1.885	-
	Total Dataset	-	-	$2.706	-
IC3 (2020)	BEC Subset	-	-	$1.7	-
	Cyber Subset	-	-	-	-
	Total Dataset	-	-	$3.5	-
Norton (2015)		$28.9	-	$150.0	-
Norton (2016)		$20.3	-	$125.9	-
Norton (2017)		$19.4	-	$172.0	-
McAfee (2013)		$24–$120	$300	-	$1,000
McAfee (2014)		~ $100	$375	$445	$575
McAfee (2018)		$134–170	$445	-	$600
Symantec (2016)			-	$575	-
CEA (2018)		$57–$109	-	-	-
RAND Corporation (Dreyer et al., 2018)	25th percentile	$27.8	-	-	-
	50th percentile	$241.9	-	-	-
	75th percentile	$665.0	-	-	-
	95th percentile	$7,710.0	-	-	-
WEF (2015)		$100	$100	-	$500
WEF (2018)		-	-	$1,600	-
BAML (2015)		-	-	$3,000	-
Cybersecurity Ventures (2017, 2019)		-	-	$6,000 (2021 projected)	-

Figure 1 - Summary of aggregate loss estimates due to cybersecurity incidents [3]

Putting these values into perspective: the global expenditures in R&D (for all areas; not only cybersecurity), for the year of 2020, were about 2.4 trillion (2.400.000.000.000) USD [4]. This means 2.5 (6 ÷ 2.4) times more is spent on security issues than with R&D efforts (this ratio might not be exact as the used values do not refer to the same year, but consecutive years (2020 vs 2021)).

Looking into the next years, the total amount of global costs with cybersecurity-related events is expected to grow considerably, with a worldwide predicted value of 10.5 trillion (10.500.000.000.000) USD, annually, by 2025 [5]. This will mean, roughly, 28.8 billion (28.800.000.000) USD per day, or 333.000 USD per second!

There is a similar study that is important to refer (year of the study is not disclosed by the authors, but it has data from at least 2015 onwards) which was made by Kaspersky – a private cybersecurity company – in which more than 5.500 companies from 26 countries around the world were surveyed [6]. This study contains more granular information, which provides insights at the organizational level. Managers and professionals were the focus of this survey, and they were asked about the cost of recovery when a security breach is experienced. This study has shown that almost half of businesses lost sensitive data due to a security threat. Financially, enterprises pay an average of 551.000 USD to recover from a single security breach; small and medium businesses (SMB) pay an average of 38.000 USD. The indirect costs add to this number: 69.000 USD for enterprises, 8.000 USD for SMBs. This study also noted that security breaches are an increasing concern for companies.

When it comes to equipping organizations with expertise in the area of cybersecurity, the difficulty lies not only in getting security professionals, but professionals with the right experience. This makes hiring a challenge. [7],[8]

The issues above are especially impactful in poorer countries, as they have “weak cybersecurity infrastructure, low inter-agency coordination and emergency responses as well as weak institutional capacity, limited ICT skills and awareness, and limited protection of critical national infrastructure” [9].

The conscience of the organizational need of getting the right expertise in cybersecurity is not recent. Back in 2013, it was predicted that it could take more than 20 years to fill the cybersecurity skills gap [10]. Shortage was mainly attributed to the rapid evolution of IT itself. The difficult legislative environment was also pointed out as a cause.

Automating security audits provides benefit in tackling the issues described before. Automated systems do not require a knowledge ramp-up and provide a systematic approach to these audits. The author of [11] starts by noting that not only the shortage of skills comes from the factors already described previously, but as well from the slow learning curve of professionals, as getting the necessary expertise to specific environments requires time, resources, and knowledge. Author predicts that these automated tools will not completely replace humans, but they will be the “cornerstones of cyber defense strategies”. Other authors, like [12], go farther and predict that automation tools are a single step in the direction of eventually achieving a state they call “cyber autonomy”, in which defensive systems will leverage AI to the point their defensive strategies can be abstracted into human language.

Still regarding AI, it should be mentioned that, at the moment this document is being written, AI tools like *ChatGPT* [13] are gaining rapid worldwide attention and adoption: there's already an intersection between automated cybersecurity mechanisms and AI, as outputs from automation can be fed into AI algorithms, which will cross check them against data sets to decide on the best course of remediation action [14].

It is also important to note that prioritizing what security flaws need to be addressed is an important aspect of the automation process. The survey in [15] (performed by Ponemon Institute, in 2022), notes that automation of security audits is an integral part of the *DevSecOps* paradigm. More than half of the survey respondents (634 IT organizations/practitioners) report that "automation significantly shortens the time to remediate vulnerabilities". However, almost half of them also note that "inability to prioritize what needs to be fixed is the primary reason vulnerability backlogs exist". This is an important aspect to note: information is important, but too much information without proper practical context/priority might overwhelm system administrators.

Finally, it is worth mentioning that a methodic cybersecurity auditing strategy is a core part of the compliance with current standards, policies, and guidelines; ISO 27001/27002 being such an example [16]. Even if an organization doesn't have a comprehensive approach to cybersecurity, it may conduct business with other organizations that might eventually require compliance with cybersecurity standards.

By condensing the information above, it can be concluded that the impact of security incidents affects organizations negatively in different ways. The costs are not only the productive impact, the media *buzz* and the reputational costs that come with it, but also the financial costs. Organizations might need to comply with cybersecurity norms to conduct their business, and this requires a systematic approach to cybersecurity auditing. On top of this, there's also a difficulty to cope with the increasing need for professionals with cybersecurity skills. Poorer countries are especially impacted by these problems, as they have less resources to prevent and respond to cybersecurity-related incidents. Automation plays a big part in helping conducting security audits, and it is important that the solutions provide outputs that properly prioritize what needs to be addressed.

The motivation of this work goes in the direction of helping to solve the problems and needs described above. To add value to the concerned efforts from academies and businesses to automate security audits, as well as making the results available to the community.

The major contributions of this work are:

- Offering a free vulnerability scanning solution, which runs on low-cost hardware, for anyone to use.

- Using a “plug and play” (PnP) philosophy, meaning that it is up and running right after the installation, providing outputs that are easily interpreted, making clear for the system administration what needs to be acted upon.

- Allowing technicians to be automatically reported about currently found vulnerabilities even when the tool is not being actively monitored, via e-mail.

- Facilitating integration with other systems, by offering an interface which shows results in a normalized format.

- Giving the flexibility to install the artifact in different hardware configurations: either in an agent-server or AIO (all-in-one) architecture, on real or virtualized hardware.

The work presented in this document differs from existing solutions (detailed in Section 2.3) by condensing all the characteristics above in a single open-source artifact – named *Intelligent System for Automation of Security Audits* (also referred to in its Portuguese acronym: SIAAS) – that requires little to no specialized staff to enable its installation and operation (thus contributing to solve the cybersecurity skills gap problem).

Finally, this work integrates in emergent DevSecOps paradigms. Such an example is an operational concept that defines a stack of security-related technologies, called *Security Orchestration, Automation, and Response* (SOAR) [17]. SOAR focuses on four main pillars: security orchestration, automation, security incident response, and threat intelligence platforms. The aim of SOAR is to help a *Security Operations Center* (SOC) in increasing speed in vulnerability detection and reaction times, lower costs in security-related tasks, simplify management efforts, and increase collaboration between teams by streamlining operations and shared reporting.

Figure 2 gives a graphical representation of SOAR.

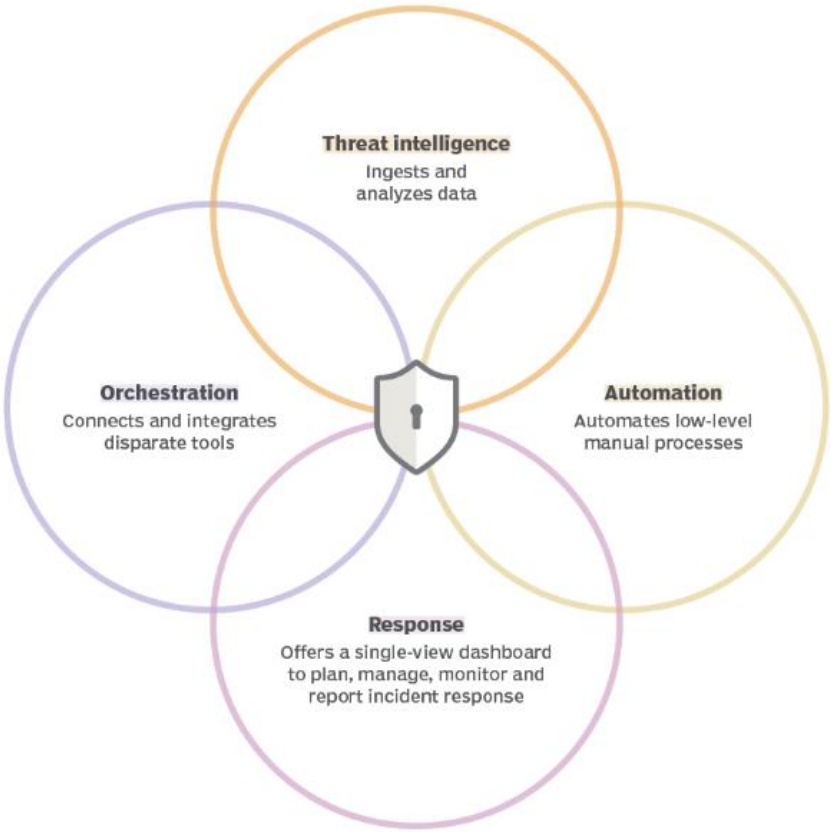

Figure 2 - Security Orchestration, Automation, and Response [17]

The concept of SOAR can be interpreted as an evolution of *Security Information and Event Management* (SIEM), as SIEM is focused only on collecting security logs and events from network elements. Therefore, in the SOAR context, SIEM is located under the “security orchestration” pillar. The design aspects of this work can be framed within the “security orchestration” and “automation” pillars of SOAR, and they provide an interface for both the other pillars to obtain their data from, as detailed in Section 2.3.

1.2. Research Questions

The context and motivations above have been condensed into a single research question:

“Is it possible to create and use a “plug and play / install and forget” system that enables the automation of continuous security auditing processes of organizations, using open-source software and low-cost hardware?”

1.3. Methodology

There are several documented approaches to describe a research framework for information systems. In this work, the framework *Design Science Research Methodology* (DSRM) is used [18].

It starts by defining a problem to be solved, and it eventually ends by creating an *artifact* (in this case, a software program) and then testing/documenting/communicating it. There is also an iterative aspect to it, as it allows the author to go back to the design phase, if needed, while developing the artifact. These aspects show that it is a model that adapts itself to the technical and iterative aspects of the development of this work.

Figure 3 describes the blocks that constitute the DSRM process model.

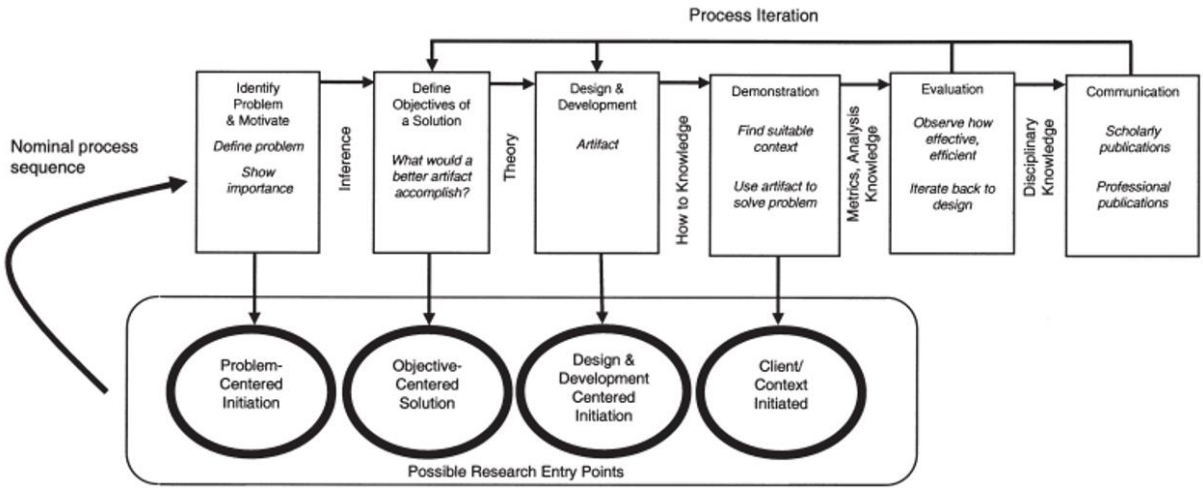

Figure 3 - The DSRM process model [18]

DSRM allows the entry point to be flexible. It can be the initial formulation of the problem to be solved, the definition of the required objectives to solve a problem already formulated, the design of the artifact given an existing objective, or to demonstrate the use of the artifact in a specific scenario. In this work, the starting point was identifying the problem to be solved and the respective research question (Chapter 1). So, the entry-point is “Problem-centered initiation”.

The next step in this model is to infer the objectives to be achieved, followed by designing the artifact and implementing it (Chapter 3). Even though the model does not consider it, there's a preliminary task before this, which is studying the current state-of-the-art attempting to solve the formulated problem (Chapter 2).

After this comes the "Demonstration" step, where the artifact must be running and showing results (this document doesn't have a specific chapter for this step. Showing that the artifact was correctly functioning was a continuous process done across Chapters 3 and 4).

Finally, the results must be quantifiable in a scientific way, and then critically reviewed to check if the initial problem can be solved using the developed artifact. In the context of software development, this might be measured in different ways. Some methods suggested by the authors of DSRM are user surveys, or measuring response times and availability. Regarding security auditing, some specific verifications can be added, as per example: checking if the artifact is correctly detecting hosts in the neighborhood, if it is correctly detecting the running services in those hosts, or if it is positively finding a vulnerability that a certain service version is known to have. This final step corresponds to the "Evaluation" phase (Chapters 4 and 5).

CHAPTER 2

State-of-the-Art

An analysis of the technological concepts that are used in this work, as well as a “state-of-the-art” overview of current approaches attempting to solve the formulated problem for this work – both from academic and commercial sources. A critical comparison between these software artifacts and the developed work is also done, to substantiate its pertinency.

2.1. Systematic Review

Regarding the review of the existing works and studies, there must be a different focus from existing commercial and academic solutions. In order to find commercial solutions and publicly available information on the Internet, the search engine *Google* [19] was used. The search for academic works was made both in the *B-On* portal [20] (a Portuguese online knowledge library which provides access to scientific releases), and in IEEE [21]. *Google Scholar* [22], which is a search engine focused on academic databases, was also used. A search in the Portuguese language was done as well, to eliminate any language bias, but no relevant documents in this language were found.

It was decided to use PRISMA (*Preferred Reporting Items for Systematic Reviews and Meta-Analysis*) [23] as guidance to systematize the research.

Figure 4 describes the information flow through the phases of a systematic review proposed by PRISMA.

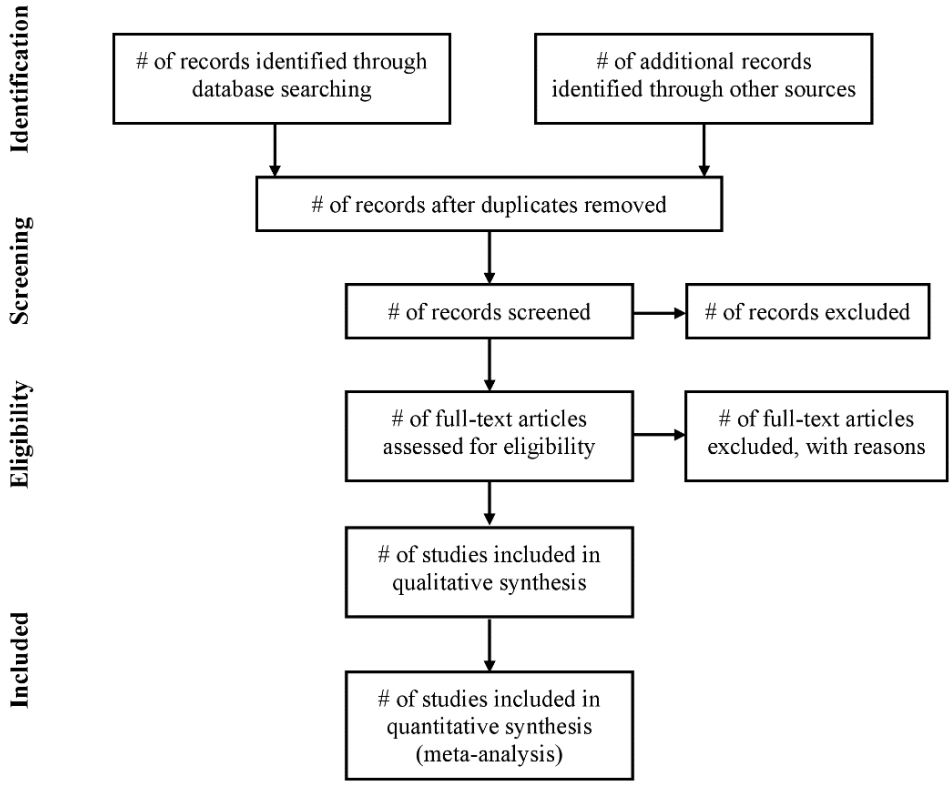

Figure 4 - PRISMA information flow [23]

For this work, the PRISMA methodology was boiled down to the following main steps:

1. Selection of databases (already described above);
2. Criteria and filters for searches;
3. Removal of duplicates;
4. Removal of non-related titles;
5. Removal of non-related abstracts.

It is important to note that, for commercial solutions, these steps could not be applied, so a simpler approach was taken. A web search, followed by analyzing the solution from the vendors website, and then excluding it if not exactly related to this work.

Table 1 summarizes the search criteria that was used per database.

B-On / IEEE	
Search in	Title
Area of study	“Computing” or “Engineering”
Date	2000 or newer
Resource type	Any
Sort	Relevance (more matches within the document) - 100 first entries
Queries	“(computer OR information OR it OR cyber) AND (security OR vulnerability OR vulnerabilities) AND (automate OR automated OR automation OR automatic OR scan OR scanner OR scanning OR audit) AND (system OR raspberry OR agent OR unit OR tool OR artifact)”
	“nmap AND (security OR vulnerability OR vulnerabilities OR audit OR intelligent OR scanner OR scanning OR vulners OR vulscan)”
	“(informática OR informação OR ti OR cyber) AND (segurança OR vulnerabilidade) AND (automatizar OR automático OR automaticamente OR automação OR rastreador OR inteligente OR auditoria) AND sistema”
	“nmap AND (segurança OR vulnerabilidade OR auditoria OR inteligente OR rastreador OR rastreio OR vulners OR vulscan)”
Google / Google Scholar	
Date	2000 or newer
Sort	Relevance (more matches within the document) - 100 first entries
Queries	“cost+cyber+incident+annual”
	“cost+cyber+incident+report”
	“cost+cyber+incident+review”
	“cyber+security+artificial+intelligence”
	“cyber+security+skills”
	“cyber+security+skills+automation”
	“cyber+security+skills+gap”
	“cyber+security+skills+need”
	“cyber+security+skills+workforce”
	“enterprise+security+audit”
	“enterprise+vulnerability+scan”
	“open+source+security+audit”
	“open+source+vulnerability+scan”
	“security+audit”
	“security+audit+automation”
	“security+audit+solutions”
	“vulnerability+scan”
	“vulnerability+scan+automation”
	“vulnerability+scan+solutions”

Table 1 - Search criteria per database

As stated above, from these results were then removed the duplicated and irrelevant entries (after analyzing titles and abstracts).

In the end, were considered a total of 47 documents, being 18 from IEEE, 3 from B-On, 10 from Google Scholar, and 16 from Google. It is important to note that these numbers serve as an approximation, and for the reader to have an idea how they were obtained. It might be possible that not all these references are used in this document. The opposite applies as well: this document might have some references that were not part of this specific systematic search effort. The writing of this dissertation was a dynamic effort in which all the searches made through the timespan of the research effort could not reasonably be recorded.

A final note to state that the searches involving the keyword “nmap” were done towards the end of the research after it was decided this would be the tool to use, as discussed in Section 3.1.

2.2. Background Concepts

Even though its capabilities go beyond that, the main goal of the artifact developed in this work is to perform security audits in order to find security *vulnerabilities* in hosts. This piece of software can therefore be categorized as a *vulnerability scanner*. In this section will be defined what a security vulnerability is, and then what a vulnerability scanner is and what it does.

2.2.1. Security Vulnerabilities

Authors in [24] define *vulnerability* as a weakness that is present in the network stack, in a network element, or in a host. A vulnerability is a potential threat, as it can be used by a person – either manually or using automatic tools – to gain access, steal information, or create an outage.

When a vulnerability can be exploited by an attacker, using a tool or code, to materialize its threat, it is said to be “exploitable” or, simply, an *exploit* [25]. It is important to note that not all vulnerabilities can be exploited. Some of these vulnerabilities are too complex to be exploited in a real-world scenario [26]. Therefore, in a production environment, it might be more efficient to just focus on exploits, instead of all vulnerabilities.

Another important concept to understand is the one of *attack vector*. An attack vector is a path that an attacker can use to penetrate a target system. An exploit is a type of attack vector. Other examples of attack vectors are phishing, malware, or password brute-forcing [27].

The next section describes how vulnerabilities are cataloged and normalized, and also the standards that exist to manage them.

2.2.2. Vulnerability Normalization and Standards

Vulnerabilities were collected and categorized over the years into databases. This type of database is called a *vulnerability database*. The sources of information of these databases include software vendors, researchers, and users. These databases organize vulnerabilities by assigning a unique ID to each, as well as a description of it. A commonly used vulnerability database is *MITRE Corporations' Common Vulnerabilities and Exposures*, and vulnerabilities in this database are commonly known as just CVEs [28],[29].

Figure 5 shows a snapshot with a frame from the MITRE's website referring to the *WannaCry* ransomware's CVE [1] which hit worldwide systems in 2017. In this snapshot, a unique vulnerability ID can be seen, as well as a description of the vulnerability.

CVE-ID	
CVE-2017-0144	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
The SMBv1 server in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; Windows 7 SP1; Windows 8.1; Windows Server 2012 Gold and R2; Windows RT 8.1; and Windows 10 Gold, 1511, and 1607; and Windows Server 2016 allows remote attackers to execute arbitrary code via crafted packets, aka "Windows SMB Remote Code Execution Vulnerability." This vulnerability is different from those described in CVE-2017-0143, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.	

Figure 5 - WannaCry ransomware's CVE page screenshot from the MITRE's website

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2017-0144

The figure also shows a clickable link to “learn more” at the *National Vulnerability Database* (NVD) [30]. NVD is a database which feeds itself from the CVE database. However, it enhances it, as it also includes information about patch availability, and severity scores according to a scoring system named *Common Vulnerability Scoring System* (CVSS) which measures the severity of a vulnerability from 0 to 10 [31]. Vulnerabilities that are easier to exploit, or whose exploits have a worse impact, are considered more severe. NVD is maintained by the US National Institute of Standards and Technology (NIST).

The NVD is not limited to being an extended version of the original CVE. It defines a whole framework of specifications responsible for managing vulnerability data. This standard is known as *Security Content Automation Protocol* (SCAP) [32].

The original 1.0 version of SCAP determines the backbone languages, scoring systems, and enumeration tools, that compromise it. CVE (enumeration tool) and CVSS (scoring system) are part of SCAP 1.0, and were already described above. The following components complete the SCAP 1.0 standard [33]:

- *Common Platform Enumeration* (CPE) [34]: serves to identify platforms or services, and respective versions, in a unique and standardized way. This format is used by vulnerability scanners to enumerate running services and versions in target hosts, and then crosscheck this normalized output against a vulnerability DB to find related vulnerabilities (as explained in the next section).

- *Common Configuration Enumeration* (CCE) [35]: it describes recommended service and software security-related configurations.

- *Open Vulnerability and Assessment Language* (OVAL) [36]: it is a declarative language used to describe a security vulnerability, system state, or the configuration of a system (example: the permissions that a certain configuration file for a certain service should have).

- *Extensible Configuration Checklist Description Format* language (XCCDF) [37]: it is another language, focused on generating security audit reports and benchmarking.

Common Weakness Enumeration (CWE) [38] is not part of SCAP, but is also worth mentioning. It might be confounded with CVE, as a vulnerability is equally a weakness. The difference is that a CVE describes a vulnerability, whereas a CWE defines a design aspect that can lead to a potential vulnerability. It is therefore a preventive indicator (examples: a service that uses an obsolete encryption algorithm, or a product that uses hard-coded credentials). It is relevant to note that, even though CWE is not part of SCAP, it is actually contemplated by *OpenSCAP* [39], which is the open-source implementation of the SCAP framework standard.

SCAP is currently at version 1.3 [32]. All the base components remain from version 1.0, but it has new additions. Some examples of these new additions are scoring systems for configurations, common enumerations for remediations, or new languages for generating standardized reports with systems information [33].

Vulnerability assessment and management is an essential part of the effort from organizations to perform security audits of their environment. This effort can be helped by an information security tool that scans target hosts and then searches vulnerability IDs from different databases that apply to them. This tool is called a *vulnerability scanner* [25], and the next section drills-down on this concept.

2.2.3. Vulnerability Scanner

As the name implies, a *vulnerability scanner* is a piece of software responsible for vulnerability scanning. Directly quoting the authors of [40], a vulnerability scanner is a “software application that assesses security vulnerabilities in networks or host systems and produces a set of scan results. Those vulnerabilities can be software bugs and backdoors, missing OS patches, insecure configurations and vulnerable ports and services”.

Authors in [25] discuss the types and advantages/drawbacks of vulnerability scanners. There are multiple types of vulnerability scanners according to the way they operate and according to their target. However, two main types of vulnerability scanners can be identified.

The first type is the *web vulnerability scanner*. This type of scanner aims to detect vulnerabilities in web-based applications by targeting the served website remotely.

The second type is usually called the *network vulnerability scanner*, which has a more generic scope. This type can be divided in two sub-types.

The first sub-type is the *network-based scanner*. It actively or passively tries to grab information and vulnerability information from the target host. This sub-type operates in 3 main steps [25],[41]:

1. *Reconnaissance* or *Enumeration:* In this phase, the target host is scanned for open ports, and services running on them. Closed and filtered ports are also checked. This process explores the characteristics and conventions of network protocols, like the TCP 3-way handshake [1], and their supported flags [2]. Port scans can be of different types: TCP connect scan (tests the full 3-way handshake process to identify open ports), SYN scan (only the initial part of the 3-way handshake protocol is used), FIN scan (probes for RST responses to FIN packets to identify closed ports), Null scan (similar to FIN, but checks for RST responses to TCP packets with any flag configuration, even if invalid [3]), Xmas scan (similar to FIN and Null, except in this case all flags are set in the sent packet), and UDP scan (checks for "ICMP Destination Unreachable" replies [4] to identify closed UDP ports) [24]. Diving into the implementation specifics of all of these methods is out of the scope of this document, so it'll focus on SYN scan, which is the default scan type for tools like *Nmap* [42], mainly because it is the most efficient method in terms of speed and obtrusiveness [43].

   It is also called "half-open scanning" because, as mentioned above, it doesn't need to perform the complete TCP 3-way handshake to identify the state of the remote port. This results in an increase in performance, compared to the complete TCP connect scan. Because it relies on the standard implementation of the TCP stack, it is supported by default on any TCP/IP network. Other methods, like FIN, Null, or Xmas scans, depend on the way the target replies to FIN or invalid packets, and this specific implementation aspect might not be as consistent across TCP stack implementations [43]. Per example, some systems send RST responses whether the port is open on not [5], meaning all ports will be incorrectly labeled as closed.

[1] https://learn.microsoft.com/en-us/troubleshoot/windows-server/networking/three-way-handshake-via-tcpip
[2] https://www.howtouselinux.com/post/tcp-flags
[3] https://www.rfc-editor.org/rfc/rfc793
[4] https://www.ccexpert.us/interface-ethernet-2/destination-unreachable-icmp-message.html
[5] https://medium.com/@niveet/how-to-hack-absolutely-anyone-part-1-r3c0n-2c691b2333f1

Figure 6 shows how a SYN scan works [44].

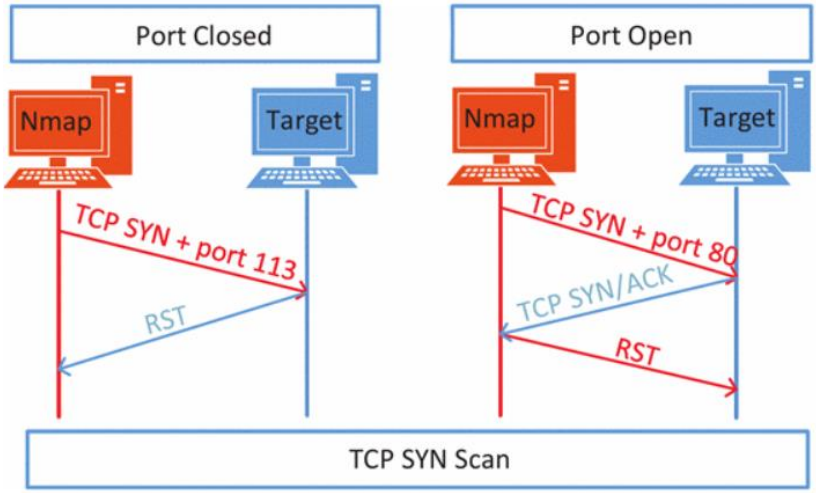

Figure 6 - TCP SYN Scan technique [44]

It can be observed that the 3-way handshake never needs to be completed in order for the scanner to define the port state of the target.

It is also worth noticing that the SYN scan will immediately understand if the port is open or closed right after the first reply. Other methods compare an RST response to an absence of response: this means that a port's state might be incorrectly identified if, for some reason, the RST packet is filtered along the way. Also, the origin must wait for a while before confirming that the RST reply has not arrived, which implies an impact in the time taken to perform the scan.

2. *OS Detection:* In this phase, the scanner tries to grab information about the OS that is running on the remote target. Different OSes use different Kernels – and this difference is important – as different methods are used to detect vulnerabilities in the next phase. The most popular method to do this operation, is called *fingerprinting.*

   The scanner uses a probing packet to get a specific reply from the host, and then analyses it to detect the OS. As each OS replies with a packet containing specific characteristics, these same unique characteristics can then be used to infer the remote OS running in the target host. These characteristics difference appears because the TCP/IP stack implementation also differs from operating systems implementations [44]. Per example, the default values for the TCP options "Window Size" and "Maximum

Segment Size" differ from OS to OS, and this can be used by the scanner to infer what the remote OS is [1].

Instead of sending a probing packet, another technique of fingerprinting is just listening to traffic from the target host passing by in the network, and then looking for characteristics in the packets similar to the ones described above. This method is called *passive fingerprinting*. When compared to the technique above (which can also be called *active fingerprinting*), passive fingerprinting is less invasive, but it might also be slower as it needs to wait for traffic to be captured. Also, active fingerprinting can craft special packages with specific TCP header options (like "No Operation" or "Window Scale") [2] to achieve more precise replies, when compared with passive fingerprinting. [45]

3. *Vulnerability Scanning:* The goal of the vulnerability scanning step is to obtain a list of known vulnerabilities from vulnerability databases, in the target host. Similar to what happens with OS fingerprinting, there are two main types of vulnerability scanning [46]. One is called *active scanning*, which involves sending packets or doing penetration tests against the hosts to emulate an attacker behavior. The other is called *passive scanning*, which implies sniffing for packets travelling within the network and analyze them. Active scanning is more invasive as it generates more noise in the network, and it might have an impact on the target host. However, it is way faster, and can detect some aspects of the target host that the passive scanning can't (e.g.: up/down status).

   Within the active scanning type, there are two sub-types of scans: credentialed and non-credentialed. In the credentialed sub-type, the scanner logs in in the remote system (usually via SSH) and performs commands remotely to grab more data. This is usually more invasive and more difficult to implement, but the results are mode detailed and comprehensive [47]. This work implements a scanner of the sub-type "non-credentialed".

   Vulnerability scanners usually implement the concept of *scripts* or *plugins*. These extensions contain the necessary logic to generate the probe packets and then interpret the results from the response to them. This also gives these tools the necessary flexibility for developers to push their capabilities beyond vulnerability

[1] http://www.misha.ro/2014/02/IP-MTU-vs-MSS.html
[2] https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml

scanning. These capabilities can therefore range from mere service version detection to actual penetration tests [40]. Some examples of penetration tests are URL crawling [1], password brute-forcing [2], service-specific attacks and checks to detect vulnerabilities like SQL injection [3] or XSS [4] on web servers, or perform *fuzzing* [5] tests against DNS servers by manipulating DNS packets.

When it comes specifically to vulnerability detection, the process starts – still taking Nmap as the example – by taking the information about the applications and respective versions running on a certain port (first step of this list), and then parsing it into a standardized format like NIST's CPE [48] (described in the previous section). Then, this string is crosschecked against online APIs of vulnerability databases, which by its turn returns a list of known vulnerabilities for the reported application version [49].

Because these vulnerability databases are so big and constantly being updated, not every script/plugin implementation downloads a local copy of them. When this is the case, it is necessary that the scanner has access to the Internet, so it can access the vulnerability DBs' APIs [49].

It must be added that, apart from these main steps, there are two extra steps that are optional for defining this concept but can add value to it. The first one comes before the 3 main steps, and it involves automatically recognizing hosts in the neighborhood instead of the human operator having to manually define them. The second one comes at the end, and it involves generating vulnerability reports and sending them to a list of recipients (probably system or security administrators) for further analysis.

[1] https://secwiki.org/w/Nmap/Spidering_Library
[2] https://linuxhint.com/passowrd_brute_force_nmap/
[3] https://www.hackingloops.com/sql-injection-penetration-testing-using-nmap/
[4] https://www.infosecmatter.com/nmap-nse-library/?nse=http-unsafe-output-escaping
[5] https://www.imperva.com/learn/application-security/fuzzing-fuzz-testing/

The second sub-type of the network vulnerability scanner is the *agent-based scanner*. This type of scanner implies that a piece of software – called the “agent” – is installed in the target-host. It then scans for security breaches within system components, such as installed packages, file permissions, firewall configuration, between others. An agent-based scanner is harder to be implemented. Firstly, it needs to be adapted to different operating systems. Secondly, it requires administrative level access to the system, which might be a complicated bureaucratic process in large organizations. The obvious advantage of agent-based scanners is that, having administrative access to the target hosts, they can fetch more and more precise information.

Regarding again the two main types of scanners, the main difference that can be observed is that a network vulnerability scanner has a more generic approach (it scans other services, like SSH), whereas a web vulnerability scanner is a more specifically contextualized scanner. Both these types require that network routes and inbound firewalling permissions are set in place, so the scanner can reach the hosts or network elements to be analyzed (except for the agent-based scanner sub-type).

As mentioned before, vulnerability scanners can perform penetration testing. The references above refer to penetration testing from a purely technical point of view, as to describe the capability of a vulnerability scanner to emulate an attack on the target host to discover specific vulnerabilities. But the term *penetration testing* or, more simply, *pentesting*, can be used to describe the complete theorical workflow of steps, from initial interactions with system administrators of the target hosts, till the final reporting of the vulnerabilities found. Vulnerability scanners and scanning are therefore a part of this workflow. The next section dives deeper on this concept.

2.2.4. Penetration Testing

Even though there is no such formal standard at present time, authors of [50] attempt to create a high-level standard to systemize the process of penetration testing, named *Penetration Testing Execution Standard* (PTES).

This standard is mostly framed in a business perspective, and uses the word “client” to refer to the organization that owns the target environment, and the word “tester” to the engineer or team performing the penetration tests. This same terminology shall be used now, to describe the main seven steps of penetration testing, proposed by this standard:

1. *Pre-engagement Interactions:* Mostly based on preparation tasks from a business and operational perspective. It implies understanding things like in which hours the tests can be performed in the client's environment, which addresses will be targeted, assessing any network elements that could affect the scanning (like firewalls), which tests should be done (example: brute-forcing, or only mere vulnerability assessment based on service banners).

2. *Intelligence Gathering:* It implies getting information from the list of addresses obtained in the previous step, in regard to installed OS, port list and state, and services/versions running. This step also refers to physical data, like physical places and people, but that is not the focus of this work.

3. *Threat Modeling:* Involves identifying the assets of an organization, and organizing them according to their importance. By assets it is meant, per example, financial reports, employee data, credit information, trade secrets, or R&D data. By prioritizing these items, it can be determined the type and order of importance of the tests to be performed.

4. *Vulnerability Analysis:* In other words, vulnerability scanning. In its simpler form, this means grabbing service banners and crosschecking them with vulnerability/exploit DBs, but it might also imply searching for other vulnerabilities related to specific services.

5. *Exploitation:* This step is meant to manually explore the potential exploits that were obtained in the last step, with the end goal of assessing a machine, and then determine which assets and information can be assessed in it (next step).

6. *Post-exploitation:* As said in the previous step, the goal of this step is to determine the value of each compromised machine, as per the sensitivity of the data it contains. This step potentially involves having direct access to client's data, so the PTES standard defines that "rules of engagement" must be previously agreed between the client and the tester, to determine what the tester can access and do inside the exploited machine.

7. *Reporting:* As the name implies, this step concerns about reporting the results to the client. PTES standard suggests a list of items that this report should contain. Some of these items are vulnerabilities and scores, types of vulnerabilities found, which machines were exploited, and potential impact on the organization.

A vulnerability scanner as defined in Section 2.2.3 is responsible for steps 2 (identifying OS and running services in the target hosts) and 4 (performing vulnerability scanning and identifying which of the results are known to be possibly exploited). The extended definition of the scanner, defined in that same section, also considers extending step 2 to add automatic host discovery and, by considering reporting/alarmistic, it can also be framed in step 7. There are exploitation-specialized tools which focus on steps 5 and 6. These are not the scope of this work, but such an add-on is suggested as future work in Section 5.3. Steps 1 and 3 focus mostly on client-specific details like timelines, scope of operation, operational impact, and risk-assessment, so they are not the focus of this work.

Existing vulnerability scanning solutions range from simple portable open-source tools to academic/commercial solutions that help organizations automate the process of finding hosts, scanning vulnerabilities, and performing reporting. These solutions might use and bundle the previously mentioned open-source tools into the comprehensive solution that the final product attempts to be. In the next sections, the current state-of-the-art tools and projects will be analyzed.

2.3. Related Works and Projects

In this section, the current state-of-the-art scenario will be studied. This involves existing scanning tools, and academic/commercial projects where these tools are condensed into a comprehensive security auditing solution.

2.3.1. Community and Commercial Tools

There is a wide variety of vulnerability scanning tools that can be found through online searches which are ready to be used. In the research made in this work, it was not realistically possible to go into detail of them all.

Based on the amount of hits returned from online searches, five were picked for an analysis in more detail: Nmap [42], *OpenVAS* [51], *Nessus* [52], *Nexpose* [53], *Faraday* [54], and *Vuls* [55].

Nmap is short for "network mapper". It is a port and vulnerability scanner that is free, open-source, and can be used in security auditing. It comes pre-installed in most of the *Linux* [56] distributions and, when not pre-installed, usually available in the distribution's repositories. Its flexibility allows the operator to run parallel scans for multiple target hosts, by launching multiple instantiations. It is also known for being very simple and portable, as it is just a binary that is called per scan and not a service that needs to be constantly running. It uses IP packets to determine the OS running in a host, and the state of the target ports and services running in them (Nmap's implementation is explored in Section 2.2.3). [57],[58]

Its capabilities do not end here. Nmap supports NSE scripts, using the *Lua* language [1], allowing Nmap to perform enumeration, vulnerability scanning, and penetration testing [2] [40].

Nmap's standard library of NSE scripts [59] comes bundled out-of-the-box when Nmap is installed, and they are organized under "categories" [60]. These categories allow the operator to easily select a set of scripts based on the description of the category. Per example, the category called "intrusive" lets the operator know it will run brute-force and penetration related scripts. The category "discovery" lets the operator know it will do tasks like crawling a website. And the category "vulns" includes – among others – the *vulners* script [61], which returns a list of CVEs, their CVSS score, link for more details, and whether if they're an exploit or not. External NSE scripts can also be downloaded and used. Another famous script is *vulscan* [62], which also performs vulnerability scanning, using several of the most famous vulnerability DBs. It also supports offline operation.

It is also worth mentioning that Nmap has solid library support for *Python* [63],[64].

Nmap still does not have the same number of scripts of Nessus or OpenVAS, as pointed out in [40]. Despite that, these same authors found that Nmap performed significantly faster and, in terms of vulnerability scanning, "the results showed that the scanners are not that far in comparison". Nessus was slightly better by detecting an extra vulnerability, while Nmap and OpenVAS got the same scanning results.

[1] https://www.lua.org/about.html
[2] https://medium.com/@sushantkamble/how-to-use-nmap-effectively-for-web-application-penetration-testing-cdf85f9222ea

Nmap is natively a command-line tool, but there are projects that implement a dashboard interface. The official project for this purpose is called *Zenmap* [65].

OpenVAS (*Open Vulnerability Assessment System*) is a free open-source vulnerability scanner, which was originally forked from the previously-free version of Nessus (which went proprietary in 2005). It also provides a Web-based dashboard for administration and reporting. It is supported in multiple OSes. Like Nmap, it supports scripted plugins for vulnerability assessment. These are called *Network Vulnerability Tests* (NVT), and are written in the NASL language [1]. [40],[58]

It is highly customizable and powerful, but this makes it also a relatively complex to install and use [40]. Unlike Nmap, it requires a service to be running in order to perform vulnerability assessment.

Nessus provides a comprehensive tool for discovery and vulnerability scanning of network devices and hosts, OS detection, supporting different types of applications and IP versions [58]. Like, OpenVAS, it supports scripts using the NASL language. As mentioned above, OpenVAS is a forked version from Nessus (Nessus is currently owned by Tenable), so OpenVAS can do most of the things Nessus can, with the difference being a smaller number of plugins available [40].

Nessus has the concept of *Nessus Agent* available as part of their commercial offer [66]. Their agent is a software module that can be installed in a hardware host. This host performs the scanning itself, and then send the metrics to the central server. The operator can then access this server and review the scanned metrics. This allows bypassing many of the administration efforts necessary to allow a centralized scanner to have access to all the target hosts. This architecture, shown in Figure 7, has similarities to what is done in this work. (Important note: this architectural concept must not be confused with the one similarly named “agent-based scanner” from Section 2.2.3).

[1] http://student.ing-steen.se/java/javacoding/toys/more_toys/nessus/txtfilez/nasl.html

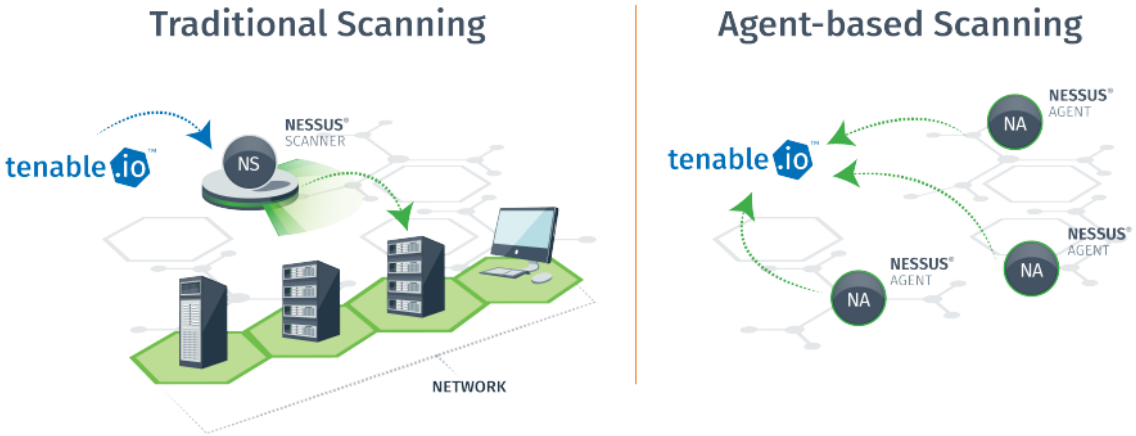

Figure 7 - Architecture of Nessus agent-based scanning versus traditional scanning [66]

Nexpose is a multi-OS network vulnerability scanner. It is currently owned by Rapid7, but it still has a free community edition. The community edition has limitations, one of them having a maximum limit of 32 target IP addresses [58]; these limitations also mean its interest in a FOSS project is limited.

Faraday is a security auditing framework which has both commercial and community editions. Faraday allows the deployment of an agent-based architecture, like described in Figure 7. Faraday does not implement its own vulnerability scanner, but it allows the operator to use or integrate with a comprehensive set of existing tools (like the already reviewed Nmap or Nessus), via plugins. Faraday supports network discovery and a dashboard to trigger and analyze vulnerability scanning results. Faraday does not support automatic reporting and continuous audit in the free edition.

Vuls is a vulnerability scanner developed for Linux, completely free. It analyses the installed packages and libraries existing in a system, matches them with vulnerability DBs, and sends the results via e-mail. It can even integrate with existing instant messaging platforms, for notification purposes. In order to perform remote scans, it requires SSH access to each of the target hosts.

Besides these tools, special note for a study from Russian origin [25] which compared a set of famous Russian and international vulnerability scanners: *Scanner-VS* [67], *Cybot* [68], *XSpider* [69], Nessus [52], and *Qualys* [70]. This study attempted to find the differences between these scanners. An interesting point besides what was already written in the paragraphs above, is that this study has noted that all these scanners need network inbound permissions before scanning (towards the targets), and none of them has the ability to automatically find and explore the network environment. Finally, it was noted that all these tools involve an initial configuration process that requires a considerable amount of expertise.

Two other tools that also appeared in the conducted search were *Retina CS* and *Microsoft Baseline Security Analyzer* (MBSA). Retina CS is a vulnerability scanner that performs OS detection and port scanning, from a list of input target hosts. It also generates reports. MBSA does local or remote security vulnerability scanning on Windows-based OSes, including checking if latest patches are installed. It requires the software to be installed in the local or remote targets. These tools are currently discontinued. [71],[72]

Some other tools, even though being on the higher end of the online search result count, were not studied in detail due to having a too high-level scope (they are only focused on web applications), and therefore not fitting in this work (more information in Section 2.3.3). Examples of these are *HCL AppScan* [73] and *Acunetix* [74].

2.3.2. Literature Review

There were several academic efforts throughout the years to provide comprehensive security auditing solutions.

Authors of [41] developed a security auditing solution using the *libnet* and *libpcap* libraries. The first library allows to create and send network packets, and the second library performs packet listening and analysis. This solution detects open ports, OS details, and vulnerabilities (from the CVE DB).

In [75], authors have developed a vulnerability scanner in Python named “Net-Nirikshak”. It does target enumeration (finding open ports and running services), and interfaces with NVD to discover known vulnerabilities associated with them. Additionally, it has a SQL injection [1] exploitation module to detect this type of vulnerability in websites.

In [76] is described an implementation of a vulnerability scanner based on NVTs (the plugins supported by OpenVAS/Nessus). It performs a vulnerability scan and generates a report in the end, so the administrator can perform remediation activities.

Work in [77] describes a vulnerability scanner, focused on web servers and vulnerabilities, namely SQL injection and XSS. It performs vulnerability assessment by leveraging *pocsuite3* [2], which is an open-source vulnerability scanning framework for web services. It has a web interface for the end user.

Authors of [78] created a vulnerability scanner focused also on web servers. Contains a web crawler and tests SQL injection, XSS [3], and directory traversal [4] vulnerabilities. No relevant details are shared about the user interface. Supports reporting, but not continuous auditing.

Authors of [79] created a vulnerability scanner also focused on web servers, named “FalconEye”. This artifact has an interesting design aspect: the scanning process is distributed amongst servers that act as "workers". It leverages common messaging protocols like AMQP [5] to handle communication between the components. It focuses only on finding vulnerabilities related to web applications, including XSS and XXE injection [6]. No relevant details are shared about the user interface.

The work in [80] implements a vulnerability scanner based on Nmap that supports target enumeration, vulnerability scanning, and remote network mapping, focused on organizations and professionals that have little cybersecurity expertise (relevant to this work). It has a web-based interface.

[1] https://owasp.org/www-community/attacks/SQL_Injection
[2] https://github.com/knownsec/pocsuite3
[3] https://owasp.org/www-community/attacks/xss/
[4] https://owasp.org/www-community/attacks/Path_Traversal
[5] https://www.rabbitmq.com/tutorials/amqp-concepts.html
[6] https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing

Artifact developed in [81] is named “SecuBat”, and is a very similar scanner to [78]. Focused on web applications, has a website crawler, and tests the target against SQL injection and XSS vulnerabilities. Has a graphical user interface, and an API is provided that enables the users to launch customized attacks. It implements reporting (but no continuous auditing) and stores historical data.

Finally, the work in [82] is another high-level scanner focused on web vulnerabilities. It performs URL crawling and attacks the resulting URLs, to detect XSS, SQL injection, between other vulnerabilities. It has a web interface to launch scans, generating a report at the end.

None of these projects is prepared for automatic network discovery. In some of them, problems with memory exhaustion were reported.

2.3.3. Conclusions

From the analysis conducted in this chapter, the following was possible to conclude:

- Some of the solutions are too high-level. In order words, they don’t have a generic nature. They either focus exclusively on specific operating systems [72], or on specific services [73],[74],[77],[78],[79],[81],[82].

- Most of the solutions require inbound network permissions to access the target hosts, if run from outside of their network, to the exception of [54] and [66] which allow local agents to be installed. Some of them also require that target host credentials are known [55].

- Most of the solutions don’t automatically discover infrastructure information, except for [51],[52],[54]. This means that information about target hosts must be obtained by system administrators and manually configured in the tool, before scans are run.

- Some of these scanners require that a daemon is running [51],[52],[53],[54]. This implies potential issues if a disruptive situation appears, like a filled disk or memory exhaustion and, therefore, it requires the implementation of a watchdog to bring the service back up if needed.

- PnP philosophy is not adopted. No tool or framework that worked out of the box was found. All of them require previous configuration before running.

- Some of them are paid, or have limitations in their community editions [52],[53],[54],[70].

The fact that all the studied solutions have at least one of these shortcomings, means that no studied artifact above solves the problem this work attempts to tackle. In other words, none of them answers positively the research question (Section 1.2). The design and deployment aspects of this work – detailed in the next chapter – implicitly address these shortcomings.

CHAPTER 3

System Design and Deployment

A detailed explanation of deployment-related aspects and decisions, and artifact architecture. It starts by detailing the goals to be achieved, design aspects, architecture to be implemented, and tooling choices. Then it details the development process of each of the three main modules of the produced artifact (agent, server, CLI).

3.1. Goals and Design Aspects

Given the motivation behind this work, the formulated research question, and the findings from the study of the current state-of-the-art (Chapters 1 and 2), the main goal of this work is *to develop a vulnerability scanning and reporting system that is free to use and does not require any cybersecurity expertise to operate*.

To achieve it, a set of characteristics – or sub-goals – that the solution must implement, were defined:

- *Agent-server architecture:* This allows easy scalability (by adding or removing agents), load distribution, and the flexibility of placing agents directly inside multiple LANs – behind firewalls and proxies and therefore hitting target hosts directly – while the operator can access all the metrics from all agents in one single server. It is also possible to access metrics from agents independently from having a server, making it possible for the operator to have a portable vulnerability scanner. These aspects will be further detailed below.

- *Low-cost hardware:* The considered hardware platform, for the agents, is the *Raspberry Pi* [83], which is a single-board computer (SBC) that uses the ARM architecture. However, the solution should use an operating system flexible enough to also run on any regular system with a x86 architecture.

  Figure 8 shows the picture of a Raspberry Pi board.

Figure 8 - A Raspberry Pi 3 Model B+ board (2017) [83]

- *Free software:* Only open-source software (FOSS) and tools – preferably portable and low on resource requirements, due to the nature of the hardware used – must be used for the implementation. Some choices made: Linux (OS) [56], Python (programming language) [63], Nmap (open-source port scanning tool, already dissected in Section 2.2.3) [42], *MongoDB* (database) [84].

- *Scalability:* It should be easy to add resources and processing power to the solution, as well as remove them. The approach is to use a multi-agent solution, where agents can easily be added or removed from the environment.

- *Modularity:* Implementation details should be flexible enough to let users develop and customize on top of it, whenever possible. The API-centric approach allows users to interact directly with it by using the standard CLI, or even developing their own CLIs, web frontends, mobile applications, or AI/ML systems for data treatment and generation of remediation proposals. Agents should also be able to be used in isolation if needed, independently from the backend server.

- *Plug and play (PnP):* Should be up and running after being installed, but highly configurable at the same time, if needed. The operator can optionally define which scans to run and hosts to target, but otherwise the application should automatically enumerate hosts in its neighborhood and scan them without any operator's intervention. By "plug and play", it is also meant that the connectivity for agent-server communication should be established using as few ports as possible (only

HTTP/HTTPS), and only outbound (agent reaching out to the server, and not the other way around). This makes it easier for agents to work behind firewalls and in air-gapped environments where incoming connections are usually blocked, diminishing the chance of having to manually configure network permissions across the organization. Finally, the discovery and vulnerability scanning process should be a continuous process, running periodically in the background, indefinitely, with no human intervention.

Another decision to support this design choice is that the artifact should be a “network vulnerability scanner”, of the sub-type “network-based”. As explained in Section 2.2.3, this is the type of scanner that is more generic, less invasive, and easier to set up, as it scans a wide range of services and does not require software installation in the target hosts.

- *E-mail reporting:* Automatic e-mails with security vulnerability reports should be sent to configured recipients. The reporting granularity should also be configurable (as in, having the possibility of filtering only the vulnerabilities that can be exploited, hence needing to be fixed more urgently).

- *Security:* Communication between the solution’s elements and the final consumers, should be authenticated and run over HTTPS (which uses SSL/TLS).

- *Future-proofing:* Consider the evolution of the technological environment (IPv6 capability).

With this set of goals, this work attempts to favorably answer the research question formulated in Section 1.2, and solve the problems described in Section 1.1. Not only by building from the current state-of-the-art but, as mentioned before, also by solving some of its current shortcomings.

3.2. Architecture Overview

Figure 9 depicts a diagram of the SIAAS architecture, from a high-level perspective.

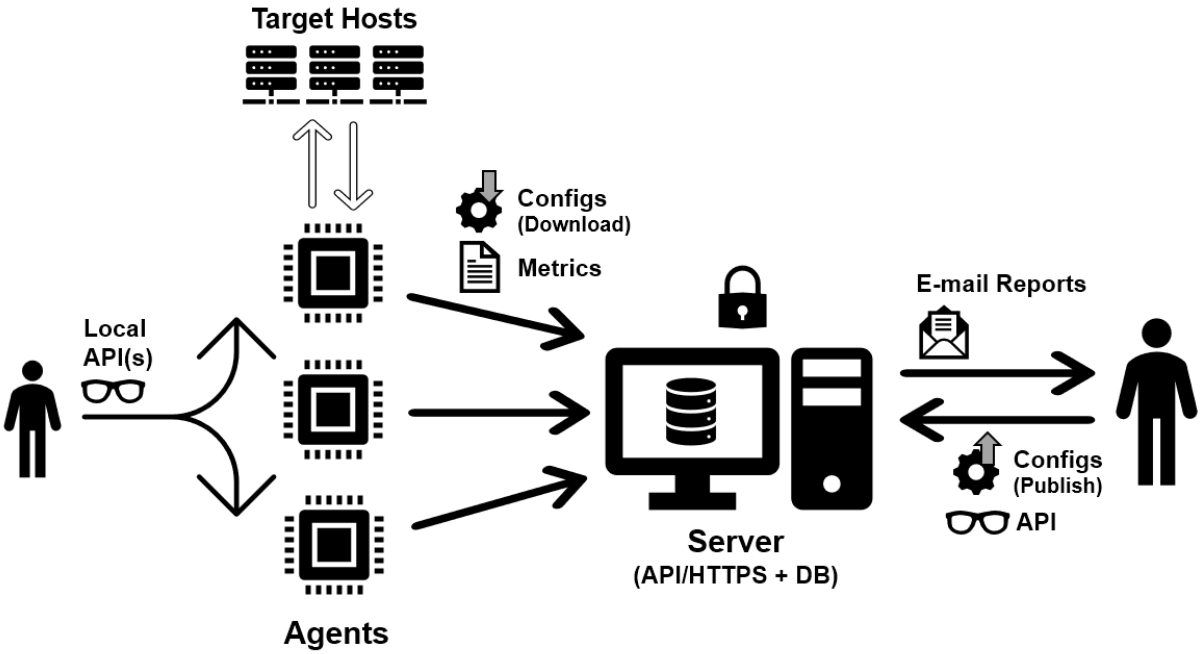

Figure 9 - SIAAS high-level architecture

The central part of the diagram shows a multiple number of agents – these can be any Linux machine or VM, but considering the paradigm proposed in this work it shall be assumed they're Raspberry Pi boards [83] – contacting the server. The connections between the agents and the server are always started from the agent, hence the arrows point from the agents to the server. All communications use HTTPS and HTTP password authentication, being therefore secured.

To obtain metrics, agents start by scanning their surroundings to check which hosts exist in the local network. The agents use these hosts, plus a list of manually configured hosts by the system administrator (if existing), and then run a Nmap vulnerability scan against the resulting list of hosts. All of this is configurable, giving the system administrator the option to scan only the manually configured hosts, or allow the agents to automatically discover and scan the surrounding hosts on their own (which is the default behavior). This last option allows the operator to just disconnect the agent from one network switch and connect it to a different network switch; the agent will scan the new network with no new configurations being needed.

The collected metrics are then, periodically, sent to the server. At the same time, the agents check if any new configurations are published to them. These consist of configurations published specifically for a certain agent UID (which is based on their product or serial number, or a randomly generated UUID if the former is not possible), and broadcasted configurations for all agents. Then, these configurations are downloaded and applied locally. Just a few possible examples of agent configurations might be the frequency of scans, Nmap scripts to be run, or the list of manually configured target hosts to scan.

As mentioned above, agents write metrics into, and obtain their configurations from, the server's API. This API allows the agents and any other clients to read and write data from and into the server's database (these clients might be the CLI developed in this work, a command-line HTTP client like cURL [85], a web browser like Firefox [86] (which has a JSON parser that cascades the API data in a human-readable way, as shown in Figure 16), or any frontend that can eventually be developed in the future for this matter; even an AI system that uses the API's outputs for further analysis). When it comes to the scanned metrics, it is the agents who write the data, and the consumer that reads it (like a human operator using one of the clients described before). When it comes to configurations, it is the operator who uploads them (again possibly by using one of the clients described above), and the agents who read them. The API also supports querying historical metric data from the agents. The maximum time range of this historical data is configurable; the server uses this value to regularly clean the older records from the DB.

The server also supports reporting via e-mail. The operator can configure the server to send e-mails with the most recently discovered vulnerabilities, containing reports in CSV format, to a list of recipients. The granularity of these reports can be configured. They can contain all the vulnerabilities, or only the exploits that need urgent care. This is very useful for system administrators to prioritize what needs to be actioned upon.

A convenient aspect of the designed architecture is that the agents' data can be accessed directly for read-only purposes, independently from the server. A local API that runs in the agents (which is disabled by default) can be activated. This allows an operator to carry the Raspberry Pi with oneself, and perform vulnerability scans in an isolated environment, even when the server is not accessible. In this case, configurations can be changed locally by editing the configuration file and then restarting the services. Obviously, this has the disadvantages of the data not being uploaded to the server (and therefore data will not be available centrally) and of e-mails not being sent.

There's even the possibility of both the agent and the server being installed in a single machine, or virtual machine, in an AIO setup. In case of the latter, the operator now has a fully-fledged vulnerability scanner in a single portable VM. However, in this case, the flexibility and scaling benefits from a distributed architecture with multiple agents is lost.

The next section will drill down on specific choices that were made regarding the software platforms and tools chosen for the deployment of the artifact.

3.3. Tooling Choices

Nmap [42] was chosen as the port and vulnerability scanner as it is a simple – yet powerful – tool when it comes to vulnerability scanning. One advantage of Nmap is that it does not require a daemon to operate. Furthermore, since it is portable, it can be installed and ready for use by running a single installation command. It can also be easily parallelized, allowing multiple instances to be run at the same time (useful to scan multiple targets simultaneously).

Although the focus of the presented work revolves around vulnerability scanning, there are several other features that had to be implemented. The tools to implement these features were not part of a systematic research; instead, the process involved searching online for the most famous tools to meet each requirement, briefly studying each of them, and then making a decision.

In terms of the language to be used, the decision was to use Python 3 [63]. Python is a free high-level programming language and has a widespread community of developers and maintainers around the world. It has a deep integration with the underlying OS (easily allowing file manipulation, grabbing platform information (as exampled in [87]), and running commands in the OS shell), and it has a vast collection of libraries and modules necessary to implement the required features. Some of these Python libraries and modules are now discussed below.

Scapy [88] is a packet manipulation Python library. It is useful to perform ARP-related operations, like scanning the neighborhood of a network adapter. This is especially useful to implement an automated scanning of neighborhood hosts. Project in [89] implements such a network scanner to find hosts in the same subnet.

In terms of the REST API implementation on the server side, there are several solutions based on Python. The solutions considered were *Django* [90], *Flask* [91], and *FastAPI* [92]. Django is the most versatile and complex. However, this makes it hard to learn. Flask and FastAPI were the two remaining valid options and had all the required functionalities to implement the API. Having FastAPI the smaller community and therefore less support, Flask was the choice.

The server API runs behind an *Apache* web server reverse proxy [93]. It is responsible to authenticate and encrypt HTTP connections between the agents/clients and the API, using its SSL and HTTP authentication modules. Nginx could be an acceptable choice as well [94]. Both these servers are available in the repositories of most Linux distributions. As the specific advantages/disadvantages of each of these web servers do not play a crucial role in this work, Apache was chosen due to the author's familiarity with it.

For the database technology running on the server side, the main contenders were *MySQL* [95] and MongoDB [84]. These solutions are fundamentally different, as MySQL is a structured DB, whereas MongoDB is a document-oriented DB (usually known as a *NoSQL* DB [1]). MySQL allows for greater data integrity, as it must obey the fixed structure of SQL tables. MongoDB, by its turn, is usually better suited for real-time analytics, and it has an easy integration with Python (it recognizes out-of-the-box objects created in Python; for example, a Python dictionary object can be uploaded to MongoDB directly as a DB document). Both have solid Python client support [96],[97]. The flexibility and ease of integration with Python data structures made MongoDB become the choice.

As mentioned in the previous section, a CLI (command-line interface; an accessory part of this work) that consumes the API was also developed, making it easier to perform tasks like consulting data, or adding or removing agent configurations. There's a specialized module for Python, for CLI implementation, called *Click* [98], which was used.

[1] https://www.ibm.com/topics/nosql-databases

All of the development was done in *Ubuntu* 20.04 “Focal” [99], but it was also installed and tested in Ubuntu 22.04 “Jammy” [99] (Server edition, or ARM edition in the agents), *Debian* 11 “Bullseye” [100] (main edition, or ARM edition in the agents), and *Raspberry Pi OS* (previously known as “Raspbian”) 11 “Bullseye” [101] (exclusively in the agents), due to being the most mature releases of these operating systems at current date, and all of the needed software packages being available in their native repositories or made available by product owners for them. All of them are Debian-based releases, so the installation scripts and source code are the same for all; Python 3 (3.9 is the default for upstream Debian, at current date) is also available on all [102].

SIAAS Agent and *SIAAS Server* are the two core software pieces of the developed work. Details about their architecture and internal logic will now be detailed. Finally, the proposed CLI that was developed as part of this work (*SIAAS CLI*) will also be presented.

3.4. Agent

The SIAAS Agent is divided in 4 main modules: the *Platform* module, which is responsible only for grabbing system-level metrics regarding the agent itself (CPU load, memory and disk usage, temperatures, general information like vendor and serial number, and others); the *Neighborhood* module, which is responsible for scanning and keeping a record of the available hosts in the agent’s surroundings (and manually configured target hosts, if any), as well as their state; the *Portscanner* module, which is responsible for enumerating the hosts that were found by the Neighborhood module (check which network ports are open) and then performing actual vulnerability scans against them; and, finally, the *Data Transfer* module, which periodically contacts the server’s API, in order to upload metrics and download configurations.

These modules do not directly talk with each other. They write into and read from the same local database. This local database consists in local files – one per module (except for the Data Transfer module), and another one for the applied configurations (local configuration file contents merged with configurations downloaded from the server) – in the JSON format.

Figure 10 depicts the architecture of the SIAAS Agent. The arrows represent the flows of data between the components. Orange arrows represent internal communication within the agent’s boundaries, whereas blue arrows represent external network communication.

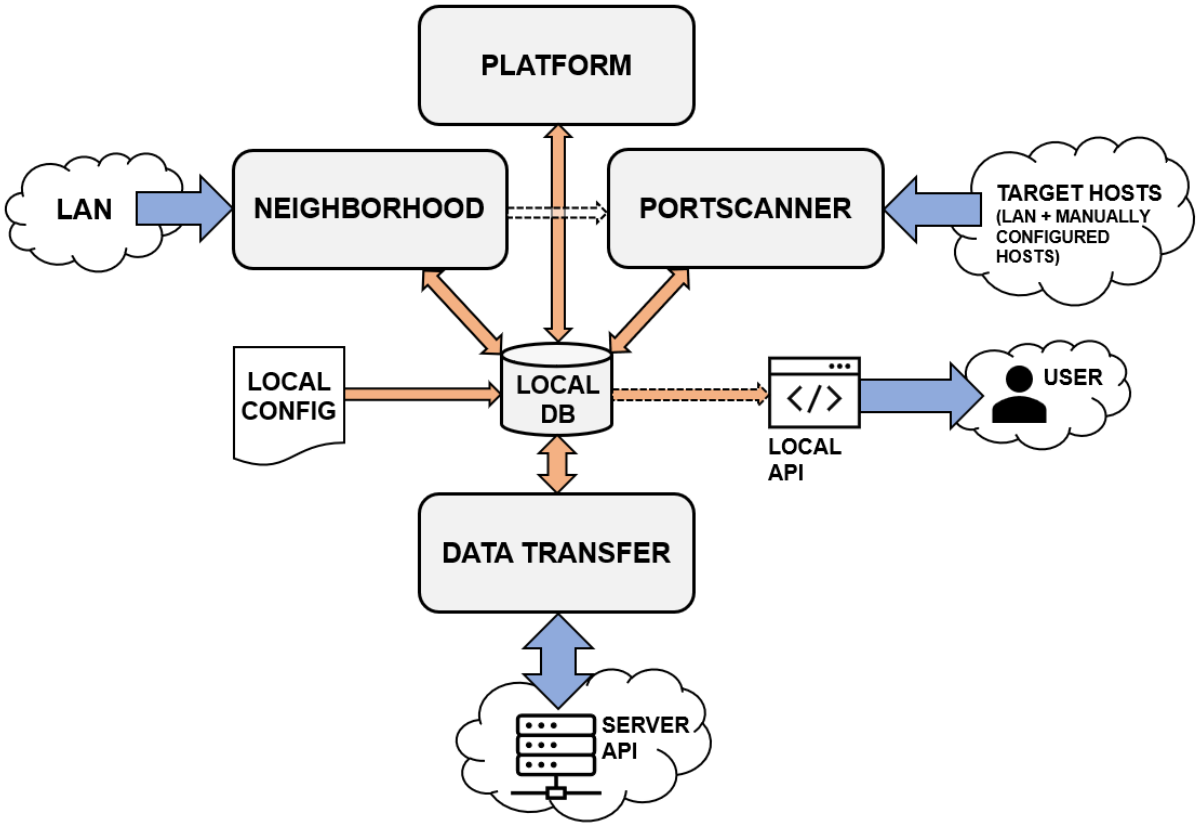

Figure 10 - SIAAS Agent architecture and data flow

Apart from the four modules, this diagram also shows a local configuration block. This block represents the local configuration file. This file has the configurations that the agent uses at boot time (or when the service is restarted). When the service is started, the contents of this file are immediately written in the configuration database. When there's a connection available to the server, any configurations downloaded from the server will overwrite the local configurations. It is important to note that the original configuration file is never changed – the overwriting happens only at the DB level.

The SIAAS Agent is completely stateless. When the service is restarted, all the local DBs (including the configurations DB) are cleared up, and then the process of populating them starts from the beginning. Also, while operating, the agent maintains no record of the results of previous scans. It is up to the server to save a historical record.

As mentioned previously, the local API is useful when using the SIAAS Agent independently from the SIAAS Server. It acts as a read-only interface to the database contents. It is disabled by default but can be activated by editing the local configuration file and restarting the service.

In the next sections, the internals of the agent's four main modules, as well of the local API will be detailed. Finally, a summary of the existing auxiliary system scripts (for installing and running the service), and logs, will be presented.

3.4.1. Platform Module

The Platform module has no active role in vulnerability scanning. The data it generates is system and hardware information from the platform underlying the service. It might be useful for the system administrator to monitor its state.

Figure 11 contains an architecture diagram of the Platform module.

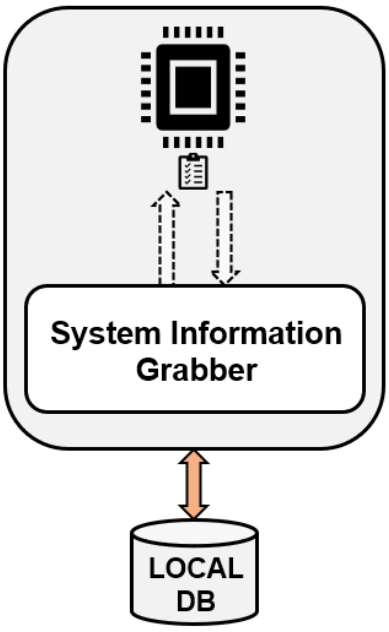

Figure 11 - SIAAS Agent's Platform module architecture and data flow

This module utilizes system-level tools such as *dmidecode* [103], and Python native libraries and modules including *psutil* [104] and *cpuinfo* [105], to gather information related to hardware, OS and architecture, CPU information and temperature, memory usage, I/O and disk statistics, network and NIC information, as well as boot time and uptime. The information is merged at the end of the iteration, and uploaded to the local Platform DB, being now ready to be consulted via the local API and uploaded to the server's API.

The Platform module reads configuration options from the configurations DB. The only configuration available is the interval time between iterations.

3.4.2. Neighborhood Module

The Neighborhood module is responsible for grabbing information from all the hosts already known by and discovered in the same network of the agent, plus information about manually configured hosts.

Figure 12 contains an architecture diagram of the Neighborhood module.

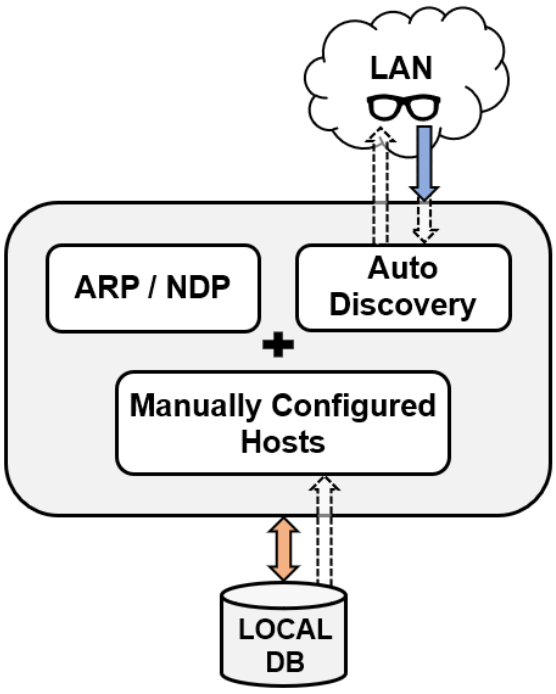

Figure 12 - SIAAS Agent's Neighborhood module architecture and data flow

As the diagram shows, the Neighborhood module implements three main functions. The algorithm of the module merges the outputs from these functions at the end of each iteration, and the local DB for the Neighborhood module is then updated. These functions are now described below:

1. This function will search the operating system's tables for ARP (IPv4) and NDP (IPv6) entries [1]. These tables contain Layer 2 hosts' information that the agent's underlying OS already knows about; more exactly, they contain IP/MAC address pairs. To read these tables, the agent leverages the OS-level command *ip neigh* [106]. Local link addresses or reserved ranges are ignored. Also, for IPv6, all addresses other than Global Unicast [2] are ignored, to avoid visual and processing pollution from ephemeral addresses existing in the network.

2. This function performs automatic discovery. It will send ARP requests to all the IPs that exist within all the local IPv4 subnets in all network interfaces, by leveraging the Scapy module for Python [88]. This function will take note of all the positive ARP replies, meaning they are available hosts in the surroundings. Only the IPv4 address spaces with a subnet mask greater than /16 are scanned, or else the total number of addresses becomes too big to scan in a reasonable amount of time (more than 50 thousand) [3]. Similarly, the IPv6 subnet spaces aren't probed due to their size. As an example, the standard /64 subnet space (which is the smallest IPv6 subnet that can be used locally) supports more than 18 quintillion addresses [4].

3. This function will parse and use the contents of the configuration "manual_hosts", which is a string of comma-separated IPs (IPv4 or IPv6) and/or domain names. In case the input is a domain name, the function will try to resolve it into IP address(es), using the system's configured DNS. If it can't be resolved, it is ignored. In case the target host happens to be in the local network, the MAC address is obtained using ARP and then added to the host's record. Unlike automatic discovery, this function adds the hosts to the list of neighbors whether they're online or offline.

[1] https://lemp.io/the-difference-between-arp-and-ndp/
[2] https://www.cidr-report.org/v6/as2.0/reserved-ipv6.html
[3] https://docs.netgate.com/pfsense/en/latest/network/cidr.html#understanding-cidr-subnet-mask-notation
[4] https://docs.netgate.com/pfsense/en/latest/network/ipv6/subnets.html#ipv6-subnetting

All the functions will try to ping the found hosts, and also try to find their DNS names by doing a reverse lookup. The complete merged output from all functions is finally uploaded to the local Neighborhood DB, ready to be consulted via the local API, uploaded to the server's API, and used by the Portscanning module.

Besides the list of manual hosts to be scanned, the Neighborhood module reads other configuration options that might exist in the configurations DB. These might include restricting the list of neighborhood hosts to only the manually configured hosts, disabling automatic discovery in the Wi-Fi ports (it was found that some network cards don't behave correctly when performing ARP discovery, as per the reliability test results described in Section 4.2.2 – it's important to note that none of the tested Raspberry Pi boards has shown this problem), or changing the default time interval between the Neighborhood module's runs.

3.4.3. Portscanner Module

Portscanner is the module responsible for performing the actual vulnerability scans against the target hosts which were previously discovered and inventoried by the Neighborhood module (Section 3.4.2). It will enumerate these hosts (grab OS and port/service information), then run Nmap scripts to perform vulnerability scanning against each of those ports and, finally, the resulting data is sorted and uploaded in the local Portscanner DB.

The enumeration and vulnerability steps are performed using Nmap. Nmap is available by default in the operating systems used in this work [107] and the developed artifact interacts with it through the *python3-nmap* module [64].

Figure 13 contains an architecture diagram of the Portscanner module.

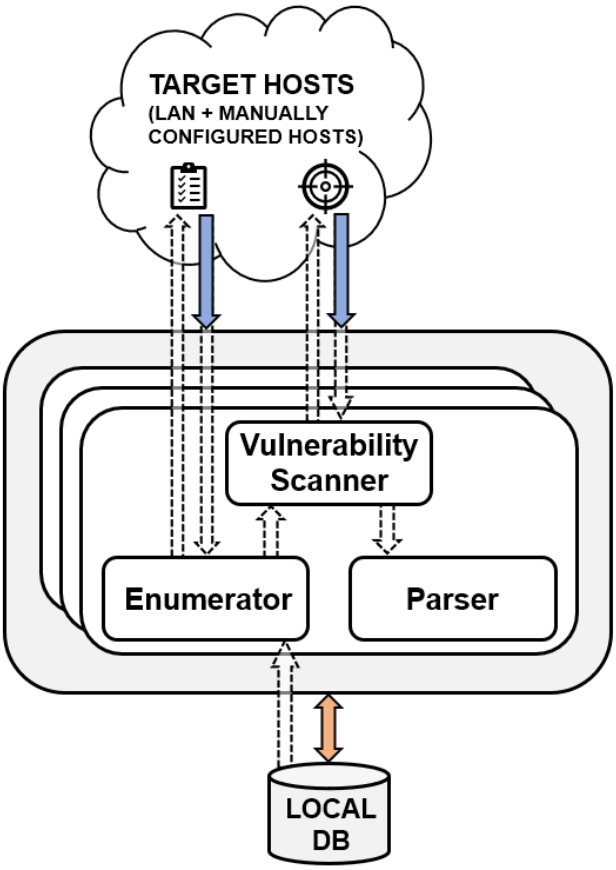

Figure 13 - SIAAS Agent's Portscanner module architecture and data flow

The Portscanner module iteration starts by grabbing the list of the hosts to target, from the Neighborhood DB. Then, this list used to launch multiple parallel workers – one per target host – and, inside each of these workers, enumeration, scanning, and output parsing, are performed. If the system administrator desires, the maximum number of parallel workers to launch can be manually defined in the agent's configuration. However, by default, the agent will let Python's native library *concurrent.futures* automatically handle this number, meaning in it will depend on the number of cores available in the agent's hardware [108]. It was opted to use thread parallelization instead of process parallelization, as it has shown shorter computing times for a single scan completion, as per the reliability test results described in Section 4.2.2.

Enumeration is performed by invoking the OS detection function from the python3-nmap module, together with an option to scan open ports (TCP and UDP). By its turn, this function leverages the operating system's "nmap" command with the flags that enable OS detection [109] and also detection of open ports and running services [110]. When this function is triggered, the most common 1000 ports are probed to check their state [111], and any eventual banners are used to determine which service is running. When it comes to the OS detection, agent sends a specially constructed packet towards the target hosts and analyses the reply, and then tries to guess the most probable OSes that the target might be using. The underlying process for grabbing this information was already detailed in the points 1 and 2 of the list presented in Section 2.2.3. (Note: for UDP, only the 10 most famous ports/services are scanned, as UDP port enumeration is especially slow [113]).

At the end of the enumeration function, the Portscanner module has now collected a list of hosts, their OSes, and open ports/services. The next step is doing a vulnerability scan against each of those ports.

A scan is triggered for all ports detected in the hosts, sequentially iterating all ports. If more than one script is configured in the agent's configuration, then multiple scans (one for each script) are sequentially triggered during each port's iteration. This vulnerability scan is done by leveraging the capability of Nmap to run CPE scripts using its scripting engine [112]. The command is called, once again, via the python3-nmap module. Vulnerability scanning and the capability of Nmap to leverage CPE scripts were already detailed in point 3 of the list presented in Section 2.2.3. The list of scripts to run can be configured by the operator, and these can be a single script name, or a category of scripts [60]. More details in Section 2.3.1, in the Nmap block. By default, the SIAAS Agent has the "vuln" category enabled [60].

At the end of the vulnerability scanning function, the Portscanner module has collected the same information it had at the end of the enumeration function, plus a list of vulnerabilities found per script, for each of the ports.

The next step is now parsing the vulnerability scanning outputs. The python3-module returns the outputs in the XML format, so they must be parsed. The parser will apply a generic parser (first getting multiple strings by slicing the output using newline special characters as a reference, then removing spaces in the beginning and end of the lines, then replacing tab special characters with vertical bars, and finally adding all the resulting strings in a single list), except when the script being run is vulners [61] or vulscan [62].

When the output from one of these two scripts is parsed, a dictionary is created using the CVE ID [29] of the vulnerabilities (or any other vulnerability ID if the DB is different) as keys. The value of these keys is a list of strings, containing the CVSS [31], description, and vulnerability link. Plus, if a vulnerability is exploitable, a tag named “siaas_exploit_tag” is also added to its list.

The parser detects whether a vulnerability is exploitable if at least one of these is positive: checking if the word “exploit” is part of the DB name (vulscan organizes the outputs per vulnerability DB, and some of these DBs are exclusive to exploits), checking if the word “exploit” is part of the vulnerability name or ID, and finally checking if this same word exists in the vulnerability description.

Finally, statistical information is added to the data being collected: number of ports scanned, number of scripts ran, number of vulnerabilities found, number of exploits found, and the total duration of the scan. All the data is merged in a single dictionary object for this target host, and this object is returned by the forked worker.

The last step of the main Portscanner module iteration is waiting that all forks end (additional note: the Nmap commands’ timeout is configurable via the agent’s configuration, so the module never hangs). Once all the forks end, all the outputs are merged into a single dictionary object, and this object is then uploaded to the local Portscanner DB. The data from this module is now ready to be uploaded to the server’s API or consulted via the local’s API.

Apart from the configuration options already described above, the Portscanner supports some more customizations: targeting specific ports (regardless of their state) instead of scanning for open ports, only scanning manually configured target hosts, configuring the time interval between iterations, and even disabling port scans altogether.

3.4.4. Data Transfer Module

The Data Transfer module is responsible for downloading published configurations from, and uploading agent data to, the SIAAS Server API.

Figure 14 contains an architecture diagram of the Data Transfer module.

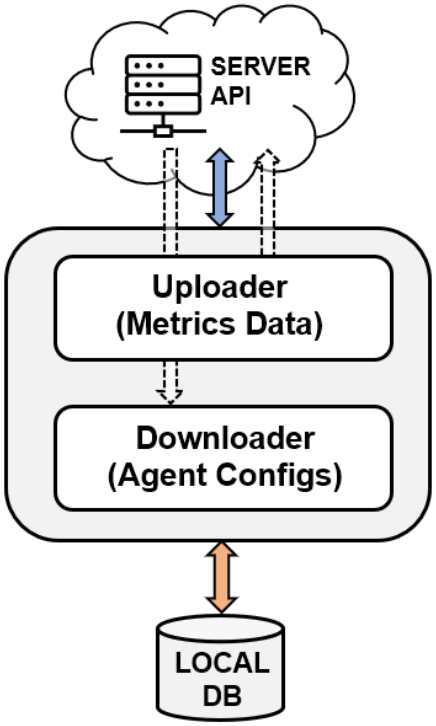

Figure 14 - SIAAS Agent's Data Transfer module architecture and data flow

The Data Transfer module has two main phases (download and upload):

1. The download phase starts by locally obtaining the agent's UID. This unique identifier is one of three possibilities, the first that is found: serial number, board serial, product UUID. If none is found, it creates a randomized UUID only for the current session, with the string "temp-" appended to the beginning. Then it sends a GET request to the server's API, asking for published configurations for this agent's UID. It adds the parameter "merge_broadcast=1" to this request. This parameter tells the server to merge the configurations that are destined for all agents (broadcast configurations) with the configurations for this specific agent's UID (additional note: if the same configuration key is used by both, the agent's specific UID configurations take precedence). The JSON object containing the configurations is received, and the local configurations DB is overwritten with it. Only the configuration keys that are received from the server are overwritten, the others will remain untouched (as in, they will remain as per the local configuration file).

   Equation 1 describes the logical overlap of the obtained agent configurations described above.

$$AgentConfig = LocalConfigFile \cup (PublishedBroadcastConfigs \cup PublishedConfigs)$$

Equation 1 - Logical overlap of local and published agent configurations

When reading this equation, the more to the right the variable is, the more precedence it takes. The green part corresponds to the dictionary merging done in the server even before the agent obtains it. Adding the red part, the complete equation – describing the merge of the local configuration with the downloaded configurations – is obtained.

2. The upload phase starts, again, by grabbing the agent's unique identifier (UID). Then, it merges the local DBs ("config" (the configurations DB), "platform", "neighborhood", "portscanner") in a single JSON object. This object is then uploaded as the body of a POST request against the entry point of the server's API for this specific agent's UID. Password entries are anonymized in the "config" DB before the data is sent.

The JSON object above mentioned in step 2, is schematized graphically in Figure 15.

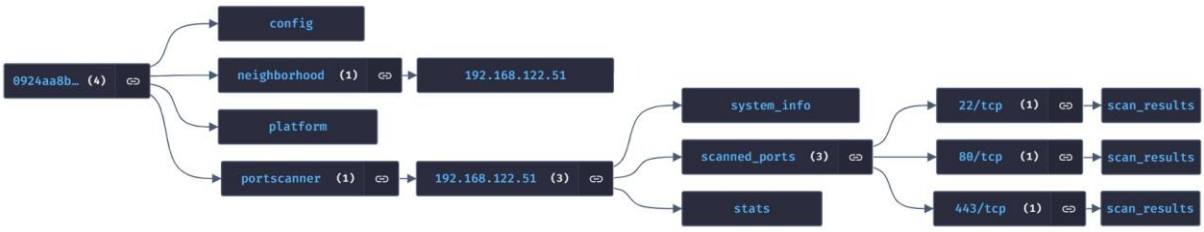

Figure 15 - Graphical representation of the JSON object's schema generated by the SIAAS Agent

In the figure above, it can be observed that the root key is the agent's UID. Then, there are 4 sub-keys, one for each module. In this figure, for example purposes, only one IP exists in the neighborhood: 192.168.122.51. Under the "portscanner" key, there are 3 sub-keys. One named "system_info" with operating system and network information about the target host, one named "scanned_ports", with the list of detected services and respective vulnerability scan results. Finally, there's a sub-key named "stats", with metadata information about the scan performed in this target host (more information in the section 3.4.3).

The Data Transfer module allows, via configuration, the following settings: enabling silent mode (only configuration and platform-related data is uploaded; data from the Neighborhood and Portscanner modules is not uploaded), offline mode (no uploads or downloads), and changing the time interval between iterations. Note that the offline mode is protected. This means it is not configurable via the server's API's published configurations, it can only be changed by editing the local configuration file and restarting the service. This mode is useful when an operator wants to use an agent independently from a server. The reason this setting is protected is to avoid enabling it by mistake, remotely, and then losing all contact with the agent.

The Data Transfer module also reads, from the configuration DB, the API connection details (URI, user, password, certificate path, SSL verification on/off). These are also protected configurations, for the same reasons as the offline mode.

It is important to remember that, if the service running on the agent is restarted, the configurations DB is cleared. Then, on boot, it is populated with the contents of the local configuration file. After this, the published configurations – if any – are merged, but only after the Data Transfer module runs once (which should happen right after the boot) and downloads them.

The HTTP calls towards the API are leveraged by using the *requests* [114] Python module.

3.4.5. Local API

The local API is implemented using the Python module Flask [91] in debug mode (this mode allows for human-readable "pretty" JSON output). When this API is accessed, it returns to the HTTP client a JSON output with all the local DBs merged: Platform, Neighborhood, Portscanner, and the configuration database. Passwords in the configuration output are anonymized.

As this API is disabled by default, it must be activated by editing the local configuration file (changing the "enable_internal_api" key to "true") and then performing a service restart. This setting is protected, so it can't be activated via published configurations in the server. This is done due to security reasons (not enabling the exposure of local agent data by mistake). After this, data can be consulted by performing a HTTP GET request against the following address: http://<SIAAS_Agent_IP>:5001/siaas-agent.

Figure 16 shows a clipped screenshot of this API's output, opened in Firefox.

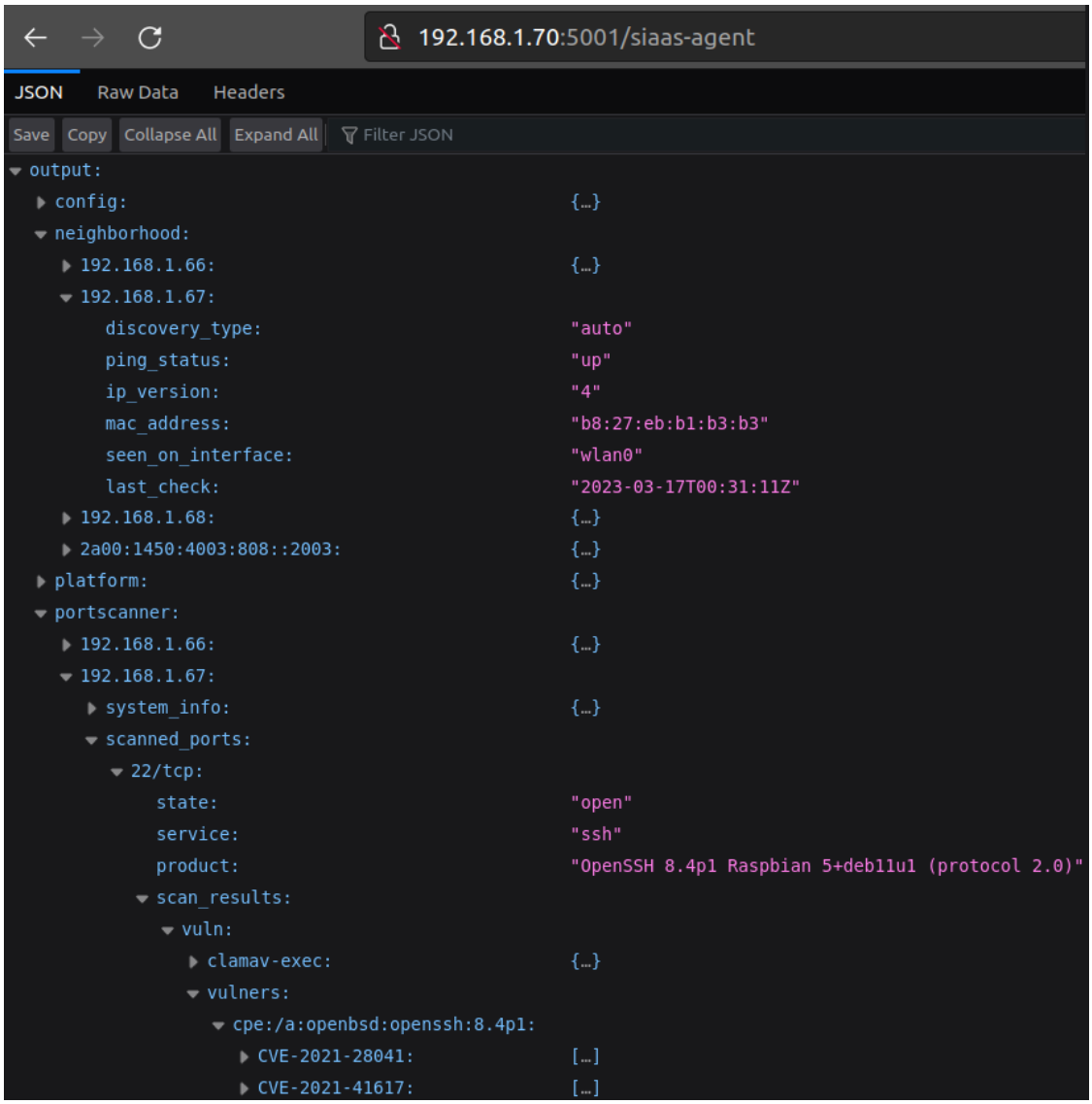

Figure 16 - Screenshot of the output of the SIAAS Agent local API in Firefox

In this screenshot, the list of hosts that the agent found in its surroundings can be seen. Also, it can be observed that this agent found at least two vulnerabilities in the target host with the IP address 192.168.1.67 (which happens to be a different agent in the same network). Platform and configuration database contents are not shown in this image, as they are collapsed. The structure of this output respects the JSON's schema presented in Figure 15 (section 3.4.4).

As a final note, it is worth mentioning that this API allows filtering the data from selected DBs, by supporting the use of the parameter “module”. This parameter has as input a comma-separated list of strings containing one or more of the keywords “platform”, “neighborhood”, “platform”, and “config”. As an example, if the user wants to show the list of hosts in the neighborhood and the port scanning results, and hide platform and configuration information, the following URL can be used: http://<SIAAS_Agent_IP>:5001/siaas-agent?module=neighborhood,portscanner.

A full web guide for the local API was written using a *Swagger* [115] implementation provided by the Python module *flask-swagger-ui* [116]. It is available in the URL http://<SIAAS_Agent_IP>:5001/docs. There’s also a simplified text formatted version of this guide, seen in Technical Note D1 (Appendix D).

3.4.6. System Scripts, Logs, and Configuration

Together with the developed artifacts, system shell scripts were developed to automatize service management.

An installation script is available. It installs the Python environment [102], and any other necessary packages that contain needed tools (like the Nmap package [107]). It also downloads the vulners [61] and vulscan [62] script repositories, and then creates a scheduled job so they are refreshed at least once a day (if the agent has Internet connectivity). It finally creates a system service (that calls the service run script) to start the service on boot and monitor its state. This also makes it easier to start/stop/restart the agent.

The created system service calls a script to run the SIAAS Agent. This script will check it the Python module environment is updated and update it, if necessary, then activate the Python’s virtual environment [117], and finally run the main agent’s code. It will also refresh the vulners and vulscan script repositories on each run.

Figure 17 shows the service for the SIAAS Agent being stopped, started, restarted, and then the status being checked, using *systemd* [118].

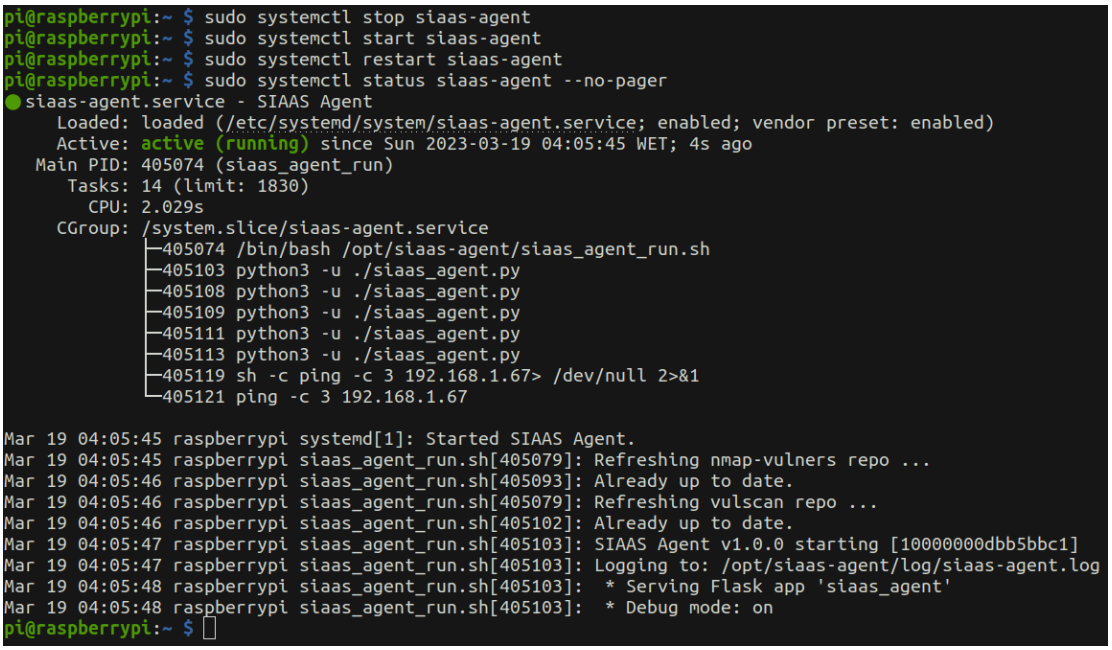

Figure 17 - SIAAS Agent service being controlled using command lines

It can also be observed that, after the service was started, the script repositories were refreshed, and the agent's UID was generated.

Logs are generated using Python's stock logging library [119], and a symbolic link for the log output directory is created in the system's default log location (/var/log/siaas-agent) when the installation is done. The log level can be configured to enable debug logs. The logging level configuration is protected. It cannot be changed via published configurations – only locally – by editing the configuration file and restarting the service. As the logs can only be consulted locally, and as the service must be restarted for the logging level to change, there is no point in changing log verbosity remotely.

Finally, there's a removal script removes the service from the operating system's list of services, and deletes the scheduled task that refreshes the Nmap scripts repositories.

There are two extra scripts: one for killing the service, and another one for archiving the SIAAS Agent directory. These are mostly development-related, and should not be needed in a normal day-to-day operation.

The configuration file for the SIAAS Agent is available in the directory "./config/siaas_agent.cnf". Full reference for this configuration file is available in Table A1 (Appendix A).

3.5. Server

The SIAAS Server is mostly responsible to implement the API (and the associated MongoDB database [84]) to where the agents report to and obtain their configurations from, and to where any clients – human or artificial – will obtain data from the agents, historical data, and also where configurations are published. It is also responsible for sending e-mails with vulnerability reports, if they're enabled.

Apart from the core architectural aspect of the SIAAS Server (the API plus the database), it has 3 main modules: like the SIAAS Agent, it has the *Platform* module, which is responsible only for grabbing system-level metrics regarding the agent itself (CPU load, memory and disk usage, temperatures, general information like vendor and serial number, and others); the *DB Maintenance* module, which is responsible for deleting historical data which is older than a defined threshold; and finally the *Mailer* module, which generates a CSV report with a list of vulnerabilities and then sends it to a configured list of recipients via a SMTP server. The depth of this report is configurable.

Figure 18 shows the architecture of the SIAAS Server. The arrows represent the flows of data between the components. Orange arrows represent internal communication within the server's boundaries, whereas blue arrows represent external network communication.

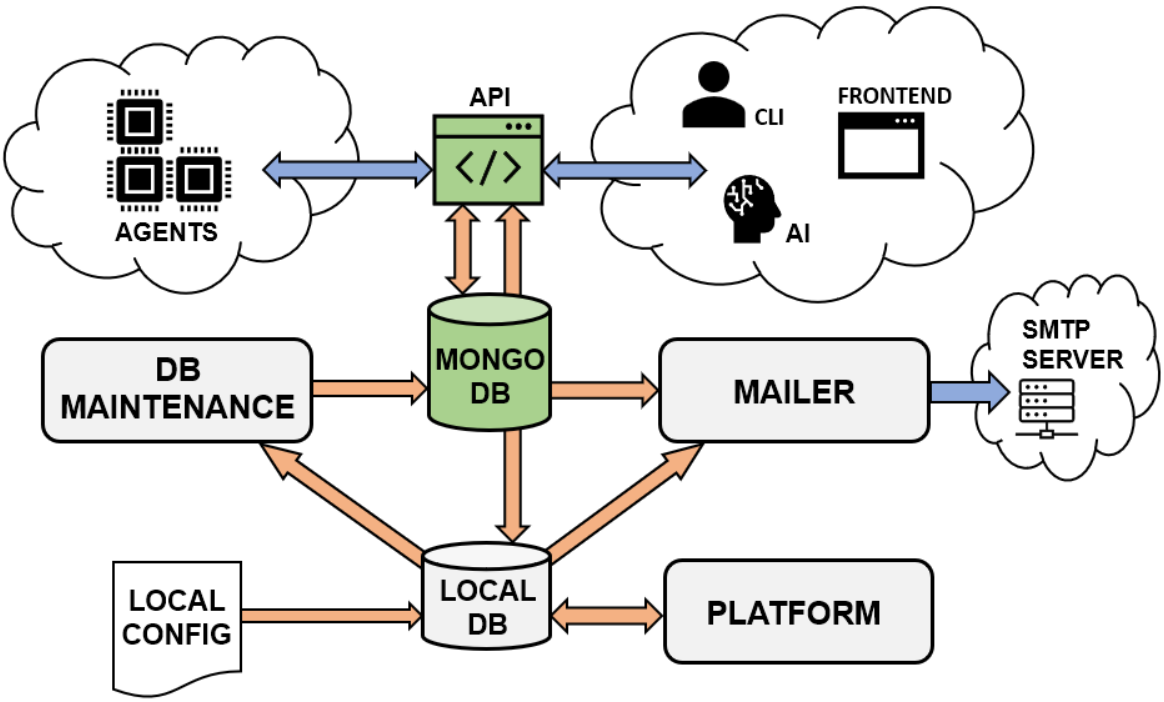

Figure 18 - SIAAS Server architecture and data flow

As with the agent's architecture, the server also uses a local database consisting in local files in the JSON format. One for the Platform module, and the other for the currently active configurations. These configurations have settings to control the DB Maintenance module, the Mailer module, and MongoDB connection settings for the API to use. This local DB works mostly the same way the agent's does. The original configuration file is never changed – the overwriting happens only at the DB level. The difference is that, unlike the agent which waits for the Data Transfer module to run once to download and merge upstream configurations, the server instantly grabs the published configurations from MongoDB and merges them with the local database.

In the next sections, the internals of the API and the database, as well as the prementioned modules, will be detailed. Finally, a summary of the existing auxiliary system scripts (for installing and running the service), and logs, will be presented.

3.5.1. API and Database

The API and the database where it stores data are the core part of the SIAAS Server architecture. The API follows the CRUD philosophy (creating, reading, updating, and deleting),

providing a set of entry points that are used by the agents and clients to upload and download data/configurations to/from the database.

The main API routes are described below. They are always prefixed with https://<SIAAS_Server_API>/api:

- /siaas-server: shows information from the Platform module running in the SIAAS Server, as well as the active configurations. Passwords in the configuration output are anonymized.

- /siaas-server/configs: shows information about the published configurations destined for the SIAAS Server. Also allows a client to post or delete configurations destined for the server.

- /siaas-server/agents: shows a list of known agents, their origin IP address, last ping, and any nicknames/descriptions that might be assigned (nickname and description strings for an agent are taken from the published configuration keys “nickname” and “description”, for its UID) (Figure 19).

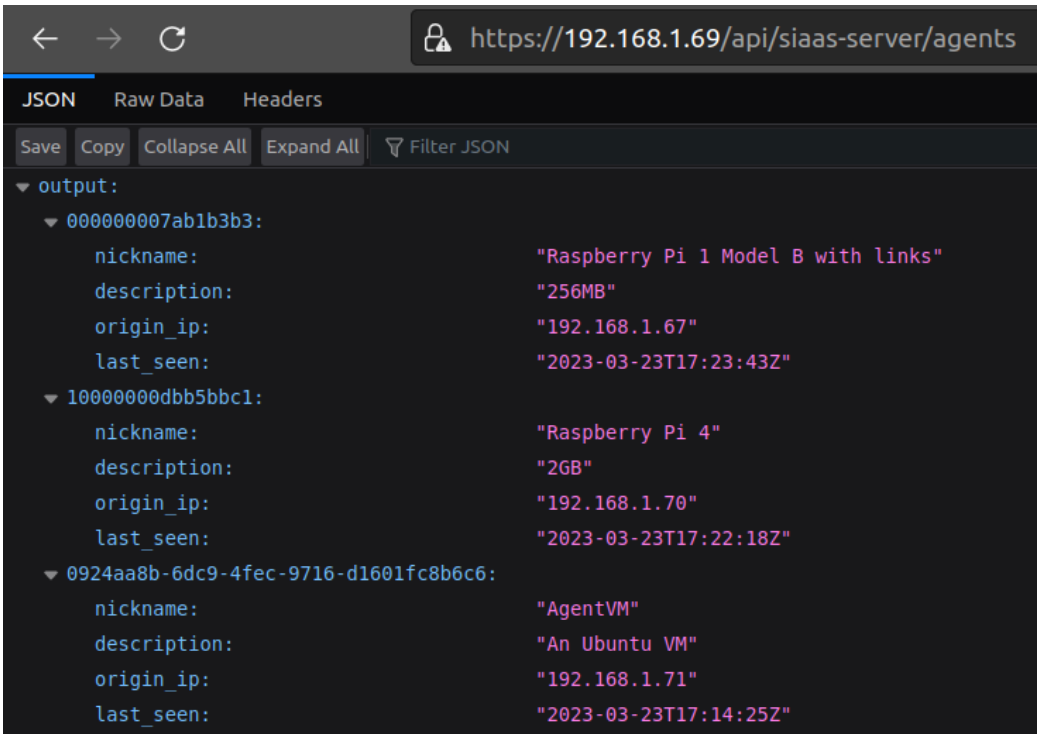

Figure 19 - Screenshot of the output of the SIAAS Server API in Firefox, showing active agents

- /siaas-server/agents/data: shows the most recent data collected from all agents (modules Platform, Neighborhood, Portscanner, and current configuration). The structure of this output respects the JSON's schema presented in Figure 15 (section 3.4.4).

- /siaas-server/agents/data/<SIAAS_Agent_UID>: shows the most recent data collected from an agent UID (or more than one, comma-separated). Also allows the agents to post new data for their specific UID. It also allows agent data older than a parameterized number of days in the request to be deleted.

- /siaas-server/agents/configs: shows the published configurations for all agents.

- /siaas-server/agents/configs/<SIAAS_Agent_UID>: shows the published configurations for an agent UID (or more than one, comma-separated). Also allows a client to post or delete configurations destined for this specific agent UID. Agent's access this endpoint to get published configurations destined to them merged with broadcasted configurations, as detailed in Equation 1.

- /siaas-server/agents/history: shows historical data collected from all agents (modules Platform, Neighborhood, Portscanner, and current configuration) (Figure 20). Allows parametrization like sorting by agent name or date, hiding empty results, filtering modules, showing older records first, limiting the number of days in the past to show, or limiting the number of outputs.

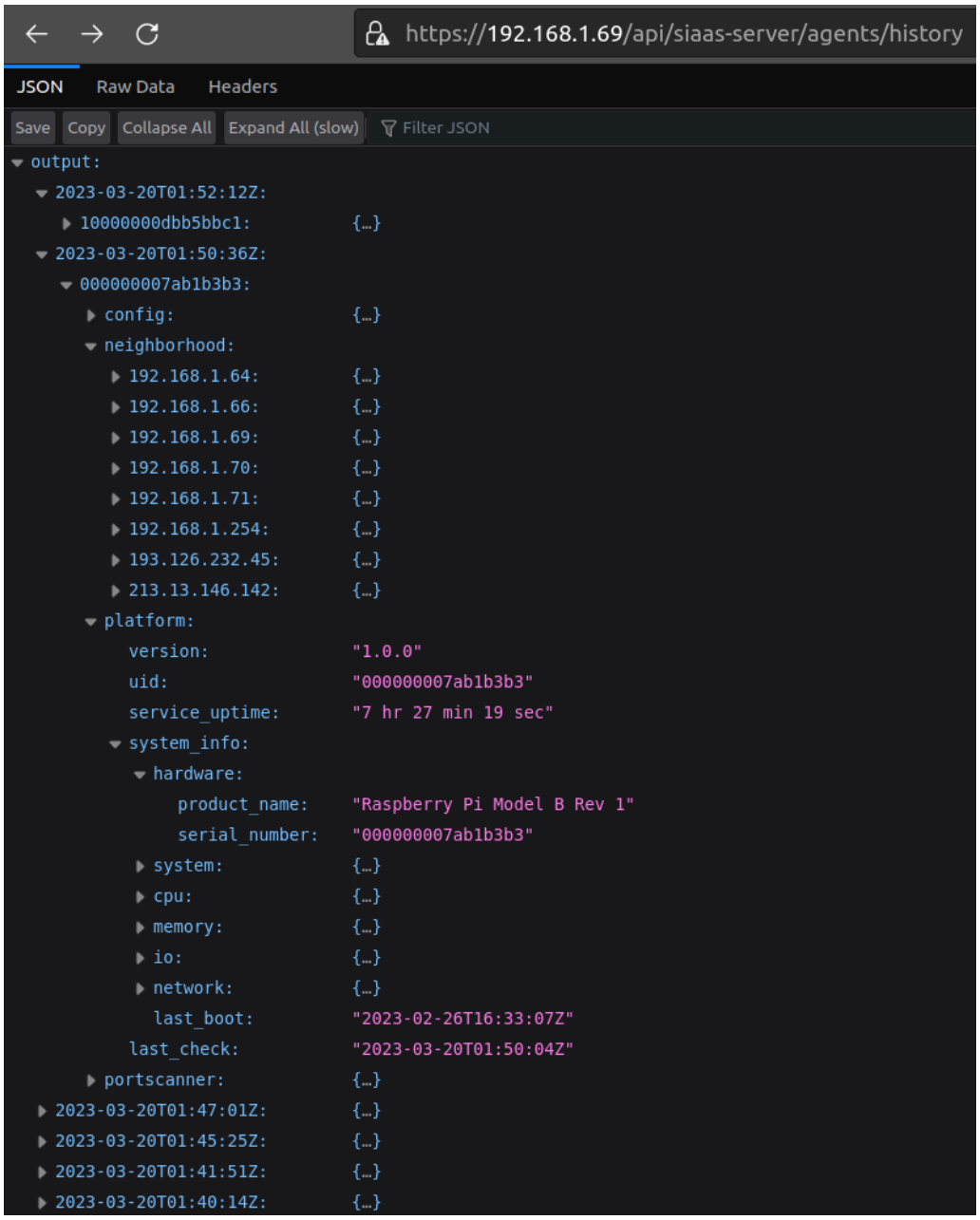

Figure 20 - Screenshot of the output of the SIAAS Server API in Firefox, showing historical data

- /siaas-server/agents/history/<SIAAS_Agent_UID>: shows historical data collected from an agent UID (or more than one, comma-separated).

The API is implemented using Python module Flask [91]. The underlying operating system supports MongoDB from its third-party repository [120]. The API interacts with it using the module *pymongo* [97].

An Apache web server with SSL support is responsible for exposing the API to the outside. It protects the incoming connections by using HTTPS and HTTP password authentication. Apache [121] and *OpenSSL* [122] are used for this purpose. Both technologies are natively supported by the operating system.

The server obtains, from the configuration DB, the MongoDB connection details (IP address, port, user, password, certificate path, DB name, and DB collection name). These are protected configurations so, in case of change, the local configuration file for the server must be edited and the service restarted. Restarting the service does not impact the MongoDB's status or contents.

A full web guide for the API was written using a *Swagger* [115] implementation provided by the Python module *flask-swagger-ui* [116]. It is available in the URL https://<SIAAS Server IP>/api/docs. There's also a simplified text formatted version of this guide, seen in Technical Note E1 (Appendix E).

3.5.2. Platform Module

The Platform module has the same underlying logic of the Platform module running in the agents, and it is configured the same way. Like in the SIAAS Agent case, it collects system and hardware information from the platform underlying the server service. Its details were already described in Section 3.4.1.

3.5.3. DB Maintenance Module

The DB Maintenance is responsible to delete the agent's metrics data records older than a certain time interval threshold. This is needed to avoid the database growing indefinitely. The default value is 2 weeks.

Figure 21 contains an architecture diagram of the DB Maintenance module.

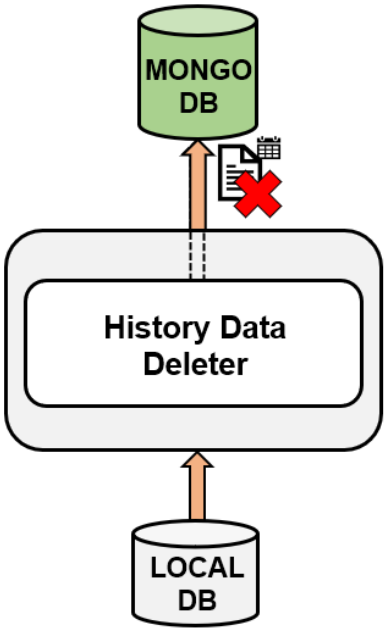

Figure 21 - SIAAS Server's DB Maintenance module architecture and data flow

The DB Maintenance module reads configuration options from the configurations DB. Apart from the number of days to keep in the database (already explained above), the only configuration available is the interval time between iterations.

It uses the pymongo [97] module to connect to the MongoDB, and grabs the connection-related options from the local configuration DB, just like detailed in Section 3.5.1.

3.5.4. Mailer Module

The SIAAS Server supports sending of vulnerability reports to a defined set of recipients. This is the task of the Mailer module.

This module starts by generating a JSON object with a tree of vulnerabilities found by all the agents known by the server. This JSON object might be more or less detailed, depending upon configuration. It may include all the data (including, per example, scanning statistics and OS detection information), it may include only the list of the vulnerabilities found per agent or, finally, it may include only the list of vulnerabilities that can be exploited. This last option allows the organization using this artifact to focus on only vulnerabilities that need an urgent fix.

Then, it compares this generated object to the last object that was sent. If no changes are observed, then no e-mail is sent. Otherwise, it will generate a CSV report with the grabbed information, and then e-mails it to the configured list of recipients.

Figure 22 contains an architecture diagram of the Mailer module.

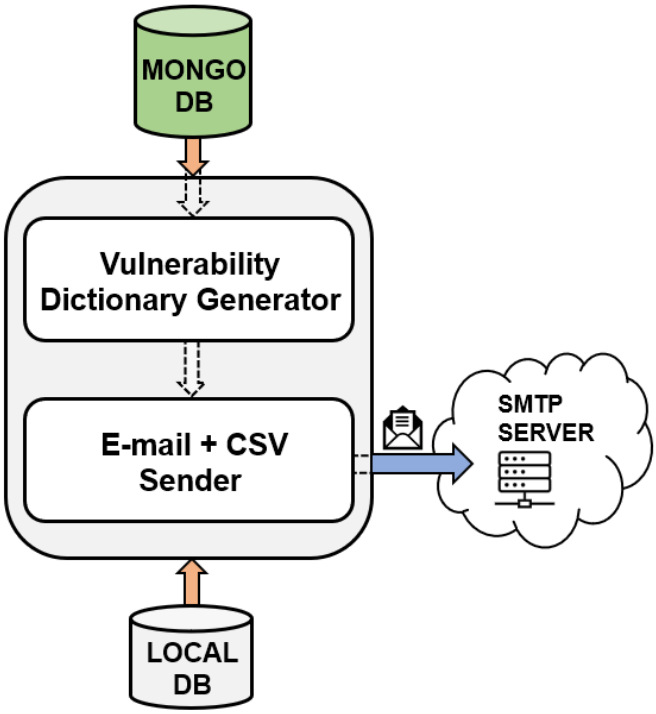

Figure 22 - SIAAS Server's Mailer module architecture and data flow

The generation of the CSV report file is done by leveraging the module *csv* [124], whereas the e-mail object is constructed using the module *email* [125]. Then this e-mail is sent using a secure SMTP connection to the configured SMTP server, using the module *smtplib* [126].

In terms of configuration, this module reads from the local configuration DB everything it needs to setup the SMTP connection: server address, port, account, password, a comma-separated list of report recipients, and the report type (from one of the described above). The time interval between the module's iterations is also obtained from the configuration.

It uses the pymongo [97] module to connect to the MongoDB, and grabs the connection-related options from the local configuration DB, just like detailed in Section 3.5.1.

The Mailer module is meant to be “vendor-agnostic”, and should interface correctly with any SMTP server. During the deployment of the artifact, both Google *Gmail* [127] and Microsoft *Outlook.com* [128] were used for sending hundreds of e-mails, and therefore are considered correctly validated. The official notes about SMTP integration for both these platforms [1] [2] were followed. There’s also a configuration example (for Gmail) in the “read me” notes of the SIAAS CLI project archive [153] (more information on SIAAS CLI in Section 3.6).

Figure 23 shows the contents of one of these e-mails, from the perspective of the recipient.

Figure 23 - Screenshot of an e-mail sent by the SIAAS Server being read in a Gmail client

To be noted that the e-mail’s subject shows the report type, the hostname of the server, and date/time; the body shows the server UID and IP; the report’s payload (CSV file) is attached.

3.5.5. System Scripts, Logs, and Configuration

As it happened with SIAAS Agent (Section 3.4.6), together with the developed artifacts for the server, system shell scripts were developed to automatize service management.

[1] https://support.google.com/mail/answer/7126229
[2] https://support.microsoft.com/en-us/office/pop-imap-and-smtp-settings-for-outlook-com-d088b986-291d-42b8-9564-9c414e2aa040

There's an installation script that installs the Python environment [102] and any other necessary packages, like the MongoDB database [120] (older CPUs may not support the most recent MongoDB versions due to missing mandatory instruction sets [1], so this script accepts the installation of older MongoDB versions by passing an older version – like "4.4" – as the argument), and other needed tools (like Apache [121] and OpenSSL [122]). It calls a secondary script to create a set of self-signed SSL certificates to be used with Apache by default (after the installation, these can be replaced by certificates signed by an external CA), and then it creates the API-related virtual hosts [123] in Apache. Then, it calls another secondary script to initialize MongoDB (creates the collection and the users used by the SIAAS Server), only if it wasn't already initialized before. It finally creates a system service (that calls the service run script) to start the service on boot and monitor its state. This also makes it easier to start/stop/restart the server.

The newly created system service calls a script to run the SIAAS Server. This script will check it the Python's module environment is updated and update it, if necessary, activate the Python's virtual environment [117], and finally run the main server's code.

Figure 24 shows the system service for the SIAAS Server being stopped, started, restarted, and then the status being checked, using *systemd* [118].

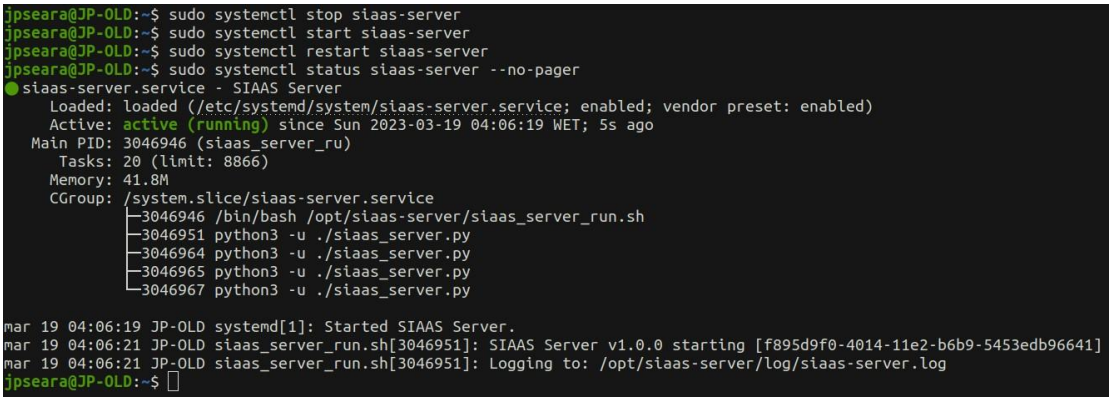

Figure 24 - SIAAS Server service being controlled using command lines

[1] https://www.mongodb.com/docs/manual/administration/production-notes/

Logs are generated using Python's stock logging library [119], and a symbolic link for the log directory is created in the system's default log location (/var/log/siaas-server). The log level can be configured to enable debug logs. The logging level configuration is protected. It cannot be changed via published configurations – only locally – by editing the configuration file and restarting the service.

There are two scripts related to backing up and restoring the database. Backing up database contents serves a dual purpose: it facilitates disaster recovery (DR), and enables the migration of the SIAAS Server from one host to another. The operator can create a backup of the DB on the old host, before installing SIAAS Server on the new host. Then, on the new host, the operator can use the backup file as an argument for the restore script (after copying it to the new host). The last step will be starting SIAAS Server service.

Finally, there's a removal script. It deletes the SSL certificates from the system and removes the SIAAS-related Apache virtual hosts. Finally, it removes the service from the operating system's list of services.

There are three extra scripts: one for killing the service, and another one for archiving the SIAAS Server directory. These are mostly development-related, and should not be needed in a normal day-to-day operation.

The configuration file for the SIAAS Server is available in the directory "./config/siaas_server.cnf". Full reference for this configuration file is available in Table B1 (Appendix B).

3.6. Command Line Interface (CLI)

The SIAAS CLI is a command line interface client with the purpose of facilitating human interaction with the SIAAS Server's API. It is important to note that the CLI is an auxiliary part of the developed artifact. In other words, its usage is not mandatory or necessary to use the API.

SIAAS CLI allows the operator to use all functions of the SIAAS Server, including viewing server statistics, managing published server and agent configurations (adding or removing configurations from a list of comma-separated key/value pairs, or simply clearing them), viewing the list of active agents, accessing agent data and historical data, and generating a report that may include all available data, current vulnerabilities, or current exploitable vulnerabilities (similar to the JSON object generated for sending e-mail reports, as explained in Section 3.5.4). It also enables the use of API parameterization by leveraging the available flags in requests.

It is relevant to note that, when it comes to the configuration management of the server/agents, the CLI adds a bit of functionality when compared to the API. Natively, the API does not support the concept of adding/updating or removing specific configuration keys: it only accepts a JSON dictionary of configuration keys for the server or an agent UID, and then replaces whatever exists. However, when adding or updating an individual configuration key, the CLI will download the current configuration dictionary, merge with the keys/values passed in the command, and finally upload it. The same applies for removing individual configuration keys: it will download the current configuration dictionary, remove only the requested keys, and upload it.

The CLI is implemented using the Python's Click [98] module, and requests towards the API leverage the module named "requests" [114]. The module *Pygments* [130] was used to colorize and highlight the output. Details about the API endpoint, debug logging activation, and the definition of the number of indentation spaces, can be set either using a flag or by sourcing an environment file (a "vanilla" environment file is provided in the project's archive [153]).

Figure 25 shows two examples of CLI outputs (the complete list of available commands, as well as a vulnerability report showing only exploits [26]).

```
jpseara@JP-OLD:~$ siaas-cli vuln-report --help
Usage: siaas-cli vuln-report [OPTIONS]

  Reports scanned vulnerabilities.

Options:
  -t, --report-type TEXT       Type of report to generate ('all', 'vuln_only',
                               'exploit_vuln_only'). (Default: 'vuln_only')
  -h, --target-host TEXT       Only shows results targeting these hosts
                               (comma-separated).
  -a, --agent TEXT             Only shows results scanned by these agents
                               (comma-separated).
  -S, --indent-spaces INTEGER  Number of indentation spaces per level in the
                               output. (Default: 4)
  -C, --colors                 Enable colors in the output. Don't use this
                               option if piping or redirecting the output!
  -D, --debug                  Enable debug logs.
  -T, --timeout INTEGER        SIAAS API timeout. (Default: 60)
  -I, --insecure               Don't verify SSL endpoint.
  -B, --ca-bundle TEXT         SIAAS SSL CA bundle path.
  -P, --password TEXT          SIAAS API password.
  -U, --user TEXT              SIAAS API user.
  -A, --api TEXT               SIAAS API URI. (Default: https://127.0.0.1/api)
  --help                       Show this message and exit.
jpseara@JP-OLD:~$
jpseara@JP-OLD:~$ siaas-cli vuln-report -t exploit_vuln_only
{
    "000000007ab1b3b3": {
        "portscanner": {
            "192.168.1.207": {
                "scanned_ports": {
                    "22/tcp": {
                        "scan_results": {
                            "vuln": {
                                "vulners": {
                                    "cpe:/a:openbsd:openssh:8.2p1": {
                                        "C94132FD-1FA5-5342-B6EE-0DAF45EEFFE3": [
                                            "6.8",
                                            "https://vulners.com/githubexploit/C94132FD-1FA5-5342-B6EE-0DAF45EEFFE3",
                                            "*EXPLOIT*",
                                            "siaas_exploit_tag"
                                        ],
                                        "10213DBE-F683-58BB-B6D3-353173626207": [
                                            "6.8",
                                            "https://vulners.com/githubexploit/10213DBE-F683-58BB-B6D3-353173626207",
                                            "*EXPLOIT*",
                                            "siaas_exploit_tag"
                                        ]
```

Figure 25 - SIAAS CLI showing help and vulnerability report outputs

As seen above, the output of the CLI commands is in JSON format, so it allows the operator to use a JSON parser like *jq* [129] to slice and filter the output. The CLI automatically disables colors when the output is redirected, as colorization adds special characters to the output which might not be recognized by JSON parsers.

The CLI was deployed in, and validated for, the same operating system environments used for the rest of this work (Debian-based [100]).

3.6.1. System Scripts and Environment Variables

The CLI is provided with an installation script. It installs the Python environment [102] and creates a system-wide alias so that the CLI can be run anywhere, by calling “siaas-cli”.

There’s a script to run the CLI, to where the system-wide alias links to. This script will activate the Python’s virtual environment [117] (installing it also, if missing), and run the main client’s code.

There’s also a removal script, which deletes the SIAAS CLI system alias.

Finally, there’s a script to archive the SIAAS CLI directory (which should not be needed in a normal daily operation).

As mentioned above, inputs for the CLI command can be passed using the available flags. Help menus that contain descriptions for the sub-commands, available flags, and their possible inputs, can be accessed by using the flag “--help". Some of the flags, namely related to the login information against the API, can also be set for an entire session using environment variables, so they don’t have to be explicitly called on every command. Full reference of the available environment variables for the CLI is available in Table C1 (Appendix C).

CHAPTER 4

Validation and Tests

The validation and testing of the artifact was performed both in a local laboratory and by external testers which tested the artifact in a real-world environment and then answered a survey. Local tests provide technical metrics related to the agent's reliability, accuracy, and security, whereas the surveys have the objective of testing usability and quantifying what is the added value of the artifact to an organization.

4.1. Methodology

As explained above, the tests of the developed solution were done both locally and by external testers (organizations and individuals).

Local tests – detailed in Section 4.2 – measured the solution's performance and security, and, as mentioned above, were focused more exactly on three aspects: reliability, accuracy, and security. When it comes to "reliability", the parameters taken into consideration were service stability over a prolonged uptime (days-long), and resource usage (CPU/memory) under stress. Some tests were also done regarding the impact of a more/less aggressive thread parallelization on scanning times and resource usage. "Accuracy" consisted in testing the scanner's ability to accurately detect the target hosts' OS and service information, and existing vulnerabilities. This is done by creating a test environment with known vulnerabilities and confirming that the scanner is reporting the correct results. Lastly, the "security" tests had the objective of validating that the API does not allow unauthenticated or insecure connections, at a network protocol level.

External tests – detailed in Section 4.3 – assessed the solution's usability and perceived added value to organizations. It was attempted to diversify the testers' nature, to have different perspectives based on different knowledge levels and organizational complexity. As such, the testers were: one organization in the IT and cloud business (Section 4.3.1), one organization in the telecommunications business with specialization in cybersecurity (Section 4.3.2), one organization in the intersection of the IT and financial businesses (Section 4.3.3), and an individual IT freelancer in the open-source field (Section 4.3.4).

The artifact was provided to an individual tester from each organization, and instructions were sent on how to set it up (these instructions already exist inside the project deliverables [151],[152],[153]). Then, a survey was produced, containing 10 statements related to user experience, 5 statements related to organizational impact, 2 statements related to overall experience, and 3 open-ended questions. The first 17 statements were to be answered using the Likert scale [131]: the tester had to select a value, from the range 1 to 5, to describe how strongly he or she agrees or disagrees with each of the statements. Here's the complete list of sentences/questions formulated for the surveys:

Technical aspects (10 sentences, participant rates from 1 (does not agree at all) to 5 (completely agrees)):

1. It is easy to install and configure the components of this vulnerability scanner.
2. The CLI usage and help menus are user-friendly.
3. The agent correctly discovers neighborhood hosts.
4. The vulnerability scan results are accurate.
5. The vulnerability scans do not impact network or targeted hosts negatively.
6. The e-mail reports are reliable and comprehensive.
7. The existing documentation is useful.
8. It is easy to customize the scanner for one's specific environment.
9. It is easy to integrate the API with existing tools and workflows.
10. You are satisfied with the performance and functionality of the vulnerability scanner.

Organizational aspects (5 sentences, participant rates from 1 (does not agree at all) to 5 (completely agrees)):

11. This vulnerability scanner is a valid approach to an organization which has no access to paid commercial solutions or cybersecurity human expertise.

12. When compared with other paid tools like Nessus, considering human expertise and configuration time, this scanner saves time/money/resources while keeping an acceptable performance.
13. The vulnerability reports generated by the scanner help with the process of addressing, prioritizing, and remediating vulnerabilities.
14. This vulnerability scanner helps an organization complying with relevant security standards and regulations.
15. This vulnerability scanner increases an organization's posture and ability to detect and mitigate vulnerabilities.

Overall (2 sentences, participant rates from 1 (does not agree at all) to 5 (completely agrees)):

16. This vulnerability scanner helps improving the security auditing process of an organization.
17. You would recommend this vulnerability scanner to other organizations.

Open-ended questions (3 questions, participant writes a text response):

18. What were the main technical or organizational shortcomings found (technical issues, false positives, or others)?
19. What features would you like to see added or improved in future versions of the scanner?
20. Any other comments?

There were two possible ways to approach these surveys: either anonymously or in-person. The author recognizes that there are pros and cons to both approaches: in one hand, an anonymous survey diminishes potential bias, in the other, having a personal survey allows the author to dive deeper into each question with the replier, understanding what the specific concerns of a company in a determined business field are, clarifying any answers that were not clear, or even getting a whole new perspective from the replier by allowing the conversation to divert into an unplanned scope. So, it was opted for an in-person approach. To diminish any potential bias, it was made clear to the participants that negative replies would not only be welcomed as means of suggestions for future improvements/corrections, but they would also have no negative impact in the outcome of this work. If accepted by the testers, a video call was set up using Microsoft *Teams* [132], and the conversation was recorded. If the testers were not available for a call, the survey was sent in a textual format, using the platform Microsoft *Forms* [133].

Local tests were run in a laboratory that was set up for the entire period of deployment and testing: real hardware (one laptop running as the server, two Raspberry Pi modules [83] running as agents), plus a virtualized environment that ran on top of KVM (Kernel-based Virtual Machine) [134] for Linux [56] which ran all the three SIAAS modules. Resource usage metrics were grabbed both manually and with the help of the monitoring tool *Munin* [135].

Testing the artifact was a continuous process that iterated with its deployment, following the DSRM [18] philosophy that was detailed in Section 1.3.

4.2. Performance and Security Tests

The following sections contain an analysis and review of the obtained results for the performance and security tests, done in a local environment.

4.2.1. Laboratory Specifications

The hardware used for the local tests used consisted of a Sony *Vaio* E11 laptop (2013) [1] with the hostname “JP-OLD” running as server and agent, and two Raspberry Pi running as agents

[1] https://www.sony.pt/electronics/support/laptop-pc-sve-series/sve1113m1e/specifications

("RPI4" being a 4th generation Model B (2019) [1], and "RPI1" being a 1st generation Model B (2012) [2]).

Figure 26 shows a photo of the setup of the local laboratory (except for the hypervisor host where the previously mentioned VM ran).

Figure 26 - Local laboratory

Specifications of this environment:

- JP-OLD (Server and agent, for testing and staging): 1.75 GHz dual-core processor, 8 GB RAM, connected via Wi-Fi.
- RPI4 (Agent for testing and staging): 1.5 GHz quad-core processor, 2 GB RAM, connected via Wi-Fi.
- RPI1 (Agent for testing and staging): 700 MHz single-core processor, 256 MB RAM, connected via Ethernet.

[1] https://elinux.org/RPi_HardwareHistory#Raspberry_Pi_4_Model_B
[2] https://elinux.org/RPi_HardwareHistory#Raspberry_Pi_Model_B_Full_Production_Board

- SIAAS (VM running in an external hypervisor; mostly for deployment): 1.8 GHz quad-core processor, 8 GB RAM, connected via Ethernet and Wi-Fi.

4.2.2. Reliability

As already explained, the reliability of the artifact was measured considering uptime and resource usage while the services are running under stress.

When it comes to the uptime, no special tests were performed, as the staging environment was always on during the entire deployment period, since the first working builds. This means at least 5 months of mostly continuous hardware uptime with no issues (from November 2022 until March 2023). Service restarts were performed from time to time to test new builds. System logs show at least 14 days of continuous uninterrupted service uptime in both the server (JP-OLD) and one of the agents (RPI4) between service restarts, as the raw data shows in Technical Note F1 (Appendix F). RPI1 logs were rotated recently, so no data could be retrieved for it regarding previous service uptime times.

When it comes to testing resource usage, the 3 staging agents (RPI1, RPI4, and JP-OLD) were left running on default configurations (Table) for 72 hours, and then data was grabbed for at least the last 24. Same was done for the server (Table) running in JP-OLD. This means they were finding and scanning all hosts in the neighborhood (the other agent, the server, an Internet gateway, and any other devices eventually connected to the Wi-Fi network), plus 2 manual hosts that were configured (“sapo.pt,google.pt”). 3 scripts were activated in the configuration (“vuln,vulscan,discovery”) (two of these are script categories [60], so the number of individual scripts was higher than 3). To put more stress in the agents, the local APIs and debug logs were activated, and the time interval between Neighborhood and Portscanner modules’ iterations was set to only 5 seconds. Wi-Fi host discovery was disabled in JP-OLD, as the wireless network card of this specific host was randomly disconnecting from the network due to unknown reasons, and this problem worsened under stress.

Even though JP-OLD and RPI4 seemed to behave properly during the tests, RPI1 was not reporting any data. Analyzing its logs, it was found out that the agent was facing constant OOM (Out of Memory) [1] kills of the Nmap [42] processes. To overcome this issue, the number of threads was manually decreased. This still didn't fix the problem: even when OOM kills were not being observed, the CPU load was still very high, the agent completely unreachable, and processes stuck with no progress. Eventually, the agent was brought to stability by using only the SIAAS Agent's default script ("vuln", which is also the one that uses less memory), with only 2 parallel workers or less. These tests are also documented in detail in Technical Note F1 (Appendix F).

Figure 27 shows the difference in CPU load as fewer parallel workers ran in RPI1.

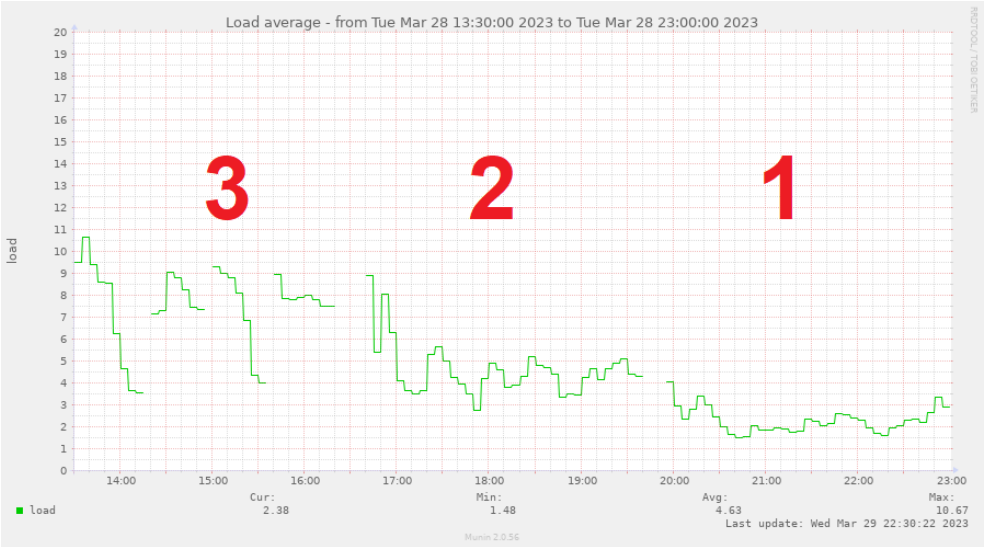

Figure 27 - CPU load in RPI1 versus number of workers

When 3 parallel workers are running, the load is so high that even the monitoring service sometimes fails to report (RPI1 has only one CPU, so a load of 1 is the maximum value before overloading). As the number of parallel workers is reduced, the agent stabilizes. RPI1 was then left running with just 1 worker for the 72-hour tests.

[1] https://www.kernel.org/doc/gorman/html/understand/understand016.html

It should be noted, as shown in Technical Note F1 (Appendix F), that RPI1 scans ended faster when using 2 parallel workers instead of 1 (which is expected), but the load was still too high for a day-to-day operation; even SSH connections to the agent were still extremely slow. So, it is preferable to use just 1 worker in older platforms like these with a slower single-core CPU, as the hardware is not so severely stressed to the point of almost creating an outage.

Figure 28 shows the CPU usage for all the agents during the last 24 hours (it is important to remember that JP-OLD is running an instance of the server as well). The light orange space in the graphs – labeled “idle” – represents free CPU. The rest, represent CPU usage. It can be observed that the CPU usage is stable in all agents, showing peaks as the scans run.

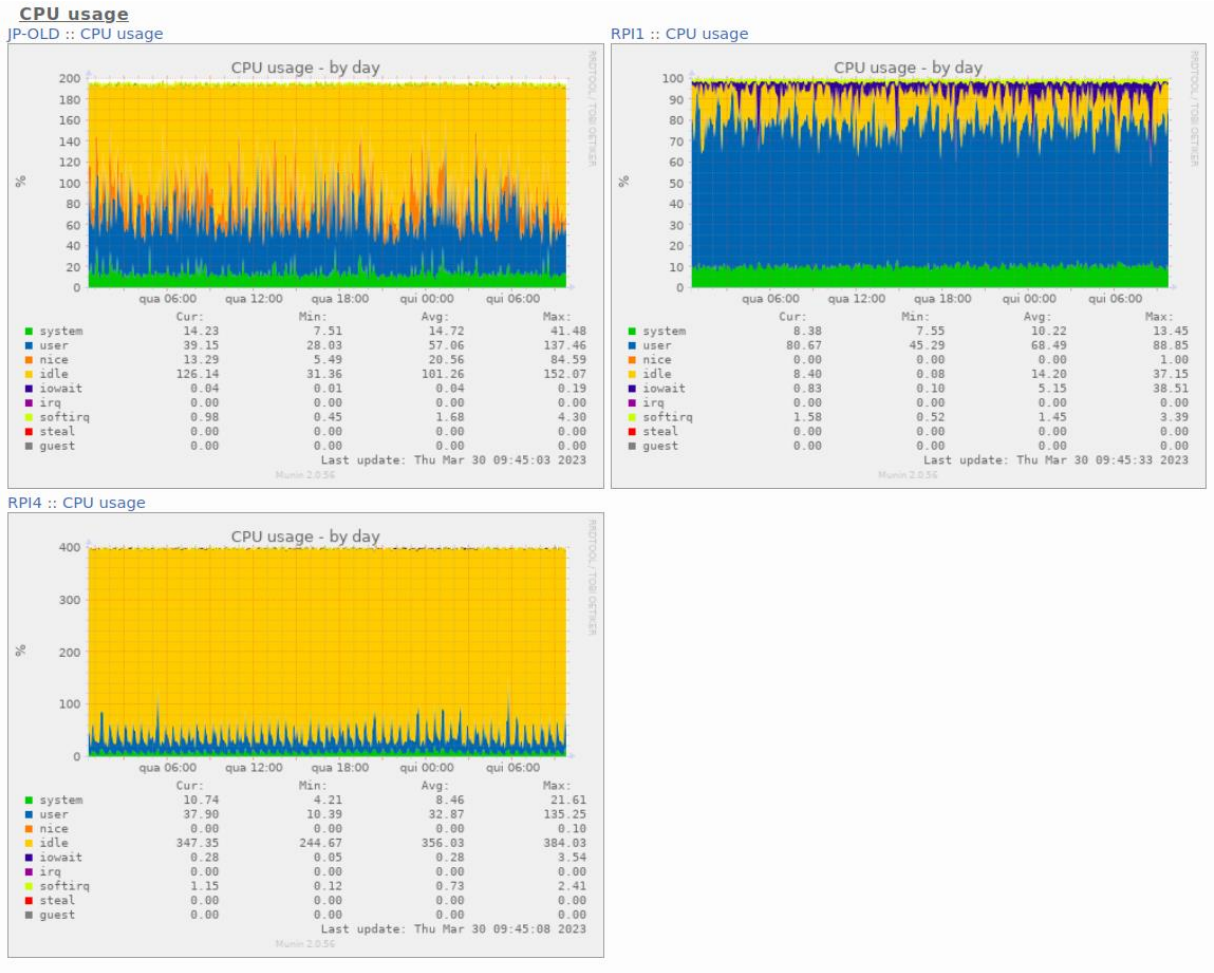

Figure 28 - CPU usage in all the agents during local tests

Figure 29 shows the memory usage for all the agents during the same period. The white space in the graphs represents free memory. The rest represent memory usage. It should be noted that the left axis of the RPI1's graph goes above the hardware's amount of physical memory (256 MB), as the system was using swap memory during the tests.

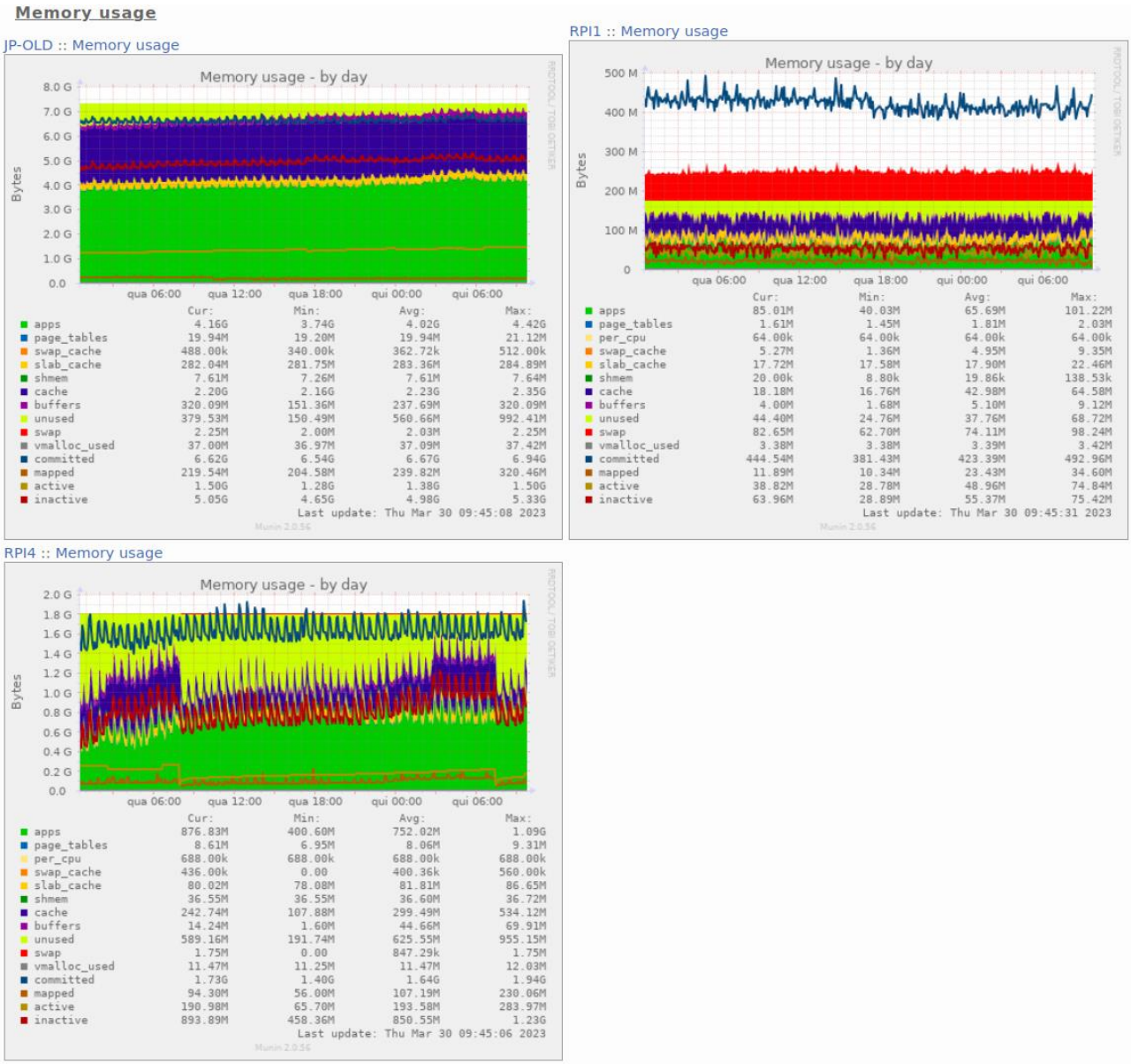

Figure 29 - Memory usage in all the agents during local tests

The memory usage was also quite constant for the entire period. In RPI4, some memory spikes can be observed. After monitoring the agent for one night, these spikes could not be attributed to any SIAAS related process. Also, in RPI1, swap memory is always being used. This can be easily explained, as this hardware configuration has very low memory available when compared to newer Raspberry Pi generations. As Technical Note F1 (Appendix F) shows, each Nmap script takes around 100 MB of memory. This is almost half of the total available memory of RPI1.

There was an extra test made – that was performed both on metal and on the VM – to test the performance of the Portscanner's task parallelization described in Section 3.4.3, and compare both Python subclasses *ThreadPoolExecutor* and *ProcessPoolExecutor* [108]. As the documentation in [108] explains, there are some differences: process parallelization fully dedicates a CPU to a certain process, instead of thread parallelization which only dedicates CPU time for a single thread at a time. In the other hand, thread parallelization runs a ratio of 5 workers per core, instead of 1 per core by its counterpart. Thread parallelization is usually more appropriate for tasks that depend mostly on I/O, and process parallelization more appropriate for CPU demanding tasks.

Nmap does not allow for a decisive choice between these two forementioned type of tasks (it uses both CPU to process requests and I/O to send them and wait for their responses), so tests were made to decide which would be the best. As shown in Technical Note F1 (Appendix F), it seems that both subclasses perform similarly per-target and in total time on an environment with 4 cores but, as the number of cores of the host decreases, thread parallelization starts performing better. This can be explained either by the differences in the ratios of workers per core (especially if I/O is the bottleneck, as more workers will transfer more data in parallel), or by the fact that process parallelization requires a dedicated CPU core, and therefore a worker might be stuck for longer periods of time waiting for the core to be available. Therefore, the decision was to stick with *ThreadPoolExecutor.*

It can be concluded that the SIAAS artifact operated continuously and uneventfully in terms of uptime and resource management during the entirety of the tests. In the agent running in older hardware, the configuration of which scripts to run and number of workers to launch had to be adapted, which is completely acceptable, as the first versions of Raspberry Pi are short on resources when compared to more recent hardware. But once the proper configuration was set, this agent also ran completely fine for hours in a row, till the end of the tests. During the whole time, the agents were successfully reporting data to the server.

4.2.3. Accuracy

Because it relies on Nmap, the developed work does not implement the logic responsible for querying the target hosts and finding related vulnerabilities. Still, some tests were done to validate that the artifact was correctly detecting the target's OS, services, and vulnerabilities.

To do this test, the following path was taken: a VM running Ubuntu 20.04 "Focal" [1] was created, and then a web server (Apache) [2] and SSH [3] services were installed in it. From the Ubuntu's security website, two vulnerabilities were picked: one for Apache [4] and other for SSH [5]. It was validated that the agent identified these vulnerabilities. Then both these services were upgraded to the next versions (this was done by upgrading the VM to Ubuntu 22.04 "Jammy" [6]). It was then validated that the agent reported that those vulnerabilities were cleared (it was previously verified online by the author that these two vulnerabilities were fixed on "Jammy"). At the same time, the accuracy of OS and service detection was also verified.

Figure 30 shows the port scanning results against the target VM, with the specific scanning results for port 80 (HTTP/web server) expanded. In this output, it can be seen that the vulnerability was correctly identified. The version of the web server was also correctly identified. The OS was correctly identified as "Linux"; the Kernel version, however, is not a match. This is not a problem as the vulnerabilities are searched per service name and version, and not by Kernel version. Even though the image does not show, it was found that the results were also correct for the SSH service (complete output in Technical Note F1 (Appendix F)).

[1] https://releases.ubuntu.com/focal/
[2] https://packages.ubuntu.com/search?keywords=apache2
[3] https://packages.ubuntu.com/search?keywords=openssh-server
[4] https://ubuntu.com/security/CVE-2020-9490
[5] https://ubuntu.com/security/CVE-2021-28041
[6] https://releases.ubuntu.com/jammy/

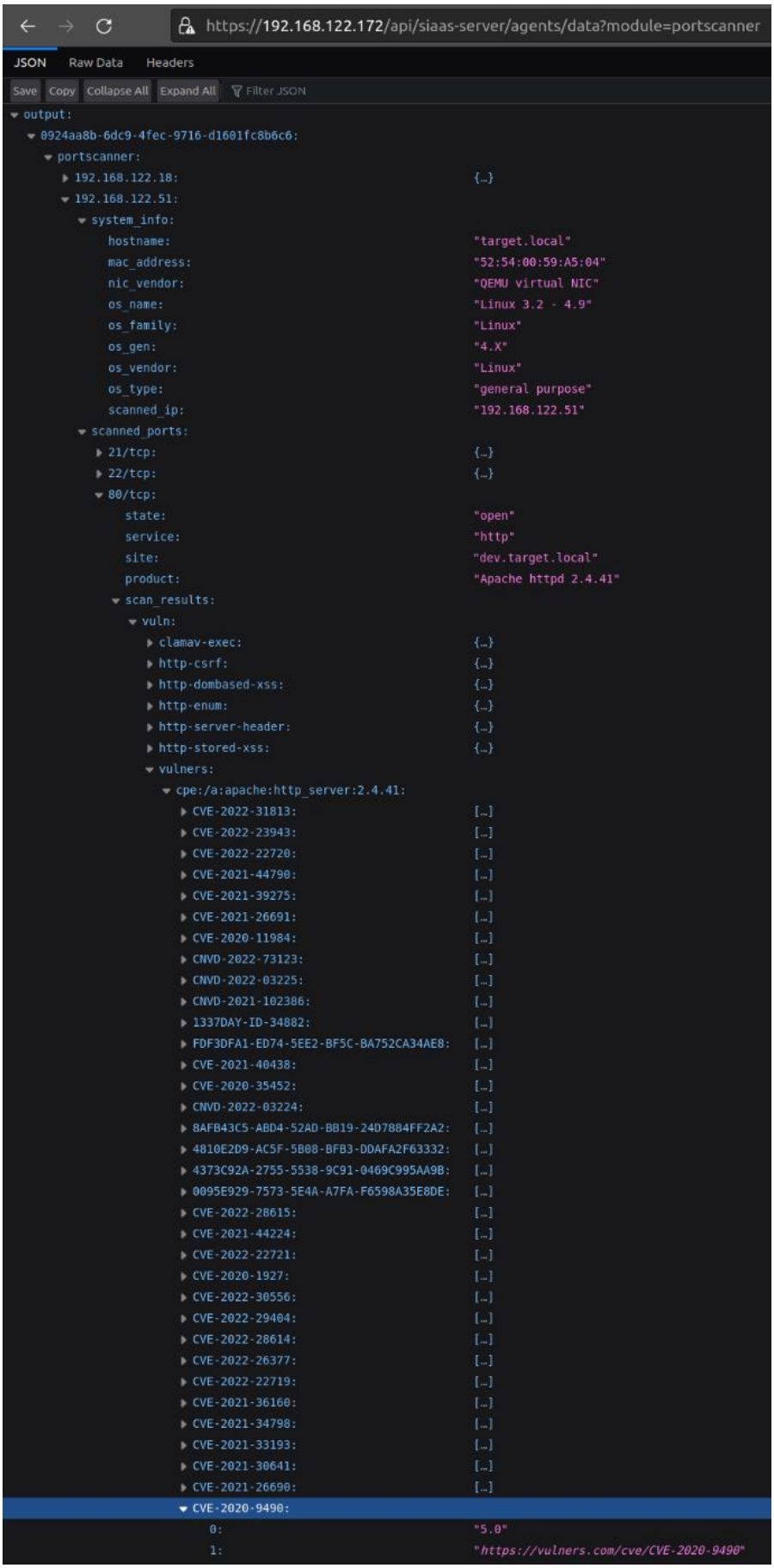

Figure 30 - Scanning results before the VM upgrade during local tests

As per the process described above, the VM was now upgraded, to upgrade the web server and the SSH service versions.

Figure 31 shows the port scanning results of the target VM after the upgrade, again with the specific scanning results of port 80 (HTTP/web server) expanded. In this output, the vulnerability disappeared, as it should. The version of the web server was also correctly updated. The OS was correctly identified as “Linux” (the Kernel version remained incorrect). The results were also correct for the SSH service.

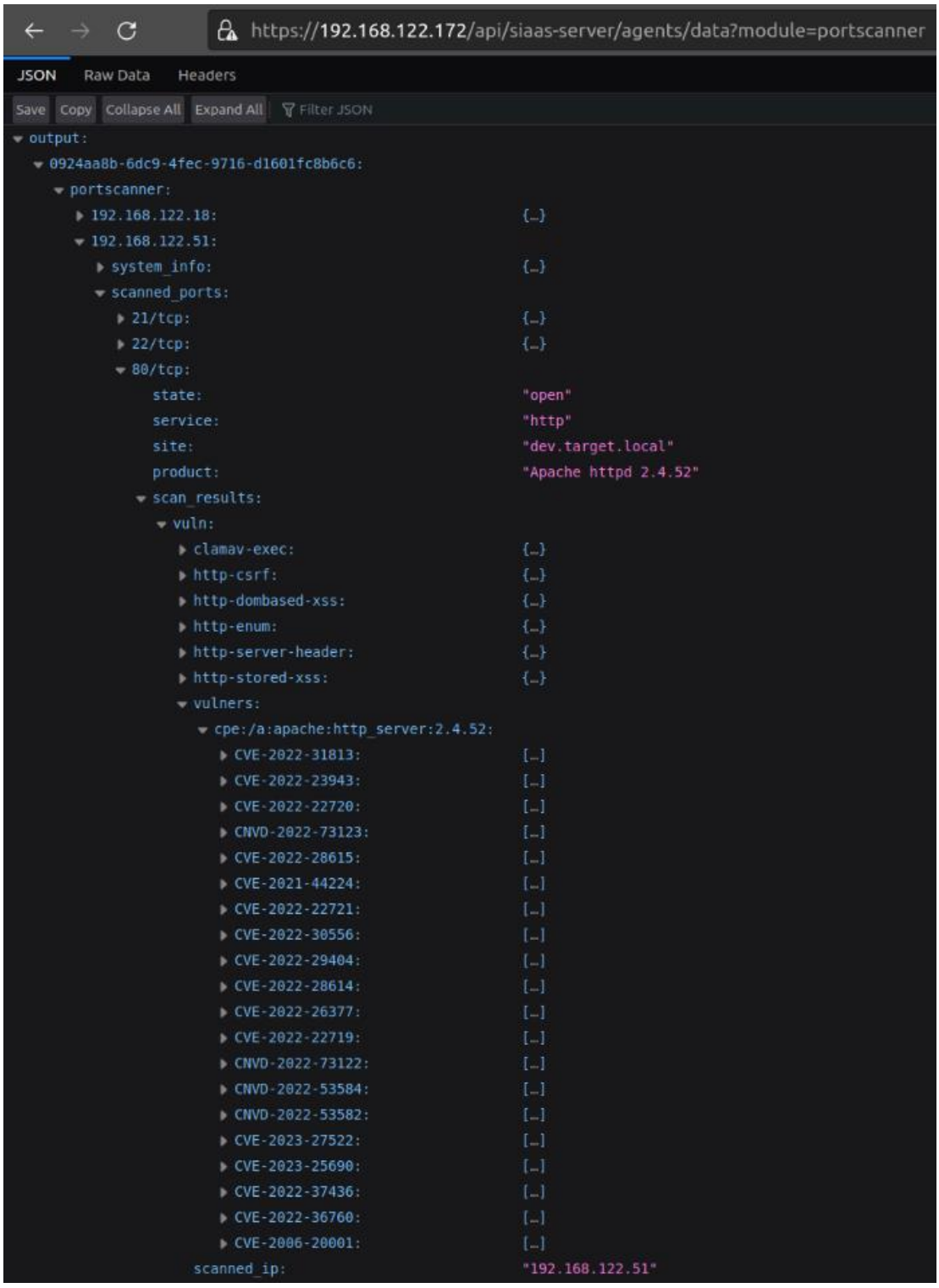

Figure 31 - Scanning results after the VM upgrade during local tests

The test was therefore considered passed. Complete outputs from this test are available in Technical Note F1 (Appendix F).

An experienced system administrator would note that, in a practical scenario, there would be no need to upgrade the OS and/or the service versions. This is because operating systems like Ubuntu backport upstream fixes (from new versions) to older service versions. This is done because, in a real-world scenario, organizations cannot simply move from one service version to the next, without risking disruption due to changes in functionality.

An example makes it clearer: considering that hypothetical service A has a known vulnerability in the upstream version 1.0, and that this vulnerability is fixed in upstream version 1.1. Considering also that service A version 1.0 is shipped with Ubuntu version X. If the vulnerability is serious enough, Ubuntu developers might decide to backport this fix into 1.0. Internally, in the OS, the installed version of this service will appear as something like “1.0 patch level 1”. However, to the client connecting to the service, the version appearing in the banner will still be 1.0.

This is important to understand because, as explained in Sections 2.2.3 and 3.1, the developed artifact is a “network-based” vulnerability scanner and, therefore, it has no access to the internal patch level of the services running on a host. It can only know what is the upstream version which is presented in the banner of the service running in a said port. In other words, it cannot know if a vulnerability is solved by internal patching of the service.

There is no way around this: the price of a vulnerability scanner being less invasive and simpler to configure is not being able to drill into details like the internal patching level of a service at the OS level, and determine if a said vulnerability is fixed or not. It can only report what are the known vulnerabilities for the upstream version of said service.

The author proposes two ways of approaching this, as suggestions for future work in Section 5.3 (either by allowing the operator to mark vulnerabilities as solved in a potential frontend, or by developing an optional software agent to be installed in a target host).

Finally, a test was done against a Microsoft *Windows Server* 2022 [136] VM, as shown in Figure 32. It can be observed that the OS is correctly detected, including its version (Windows Server 2022 is version 10 [1]). The web service is also correctly detected.

[1] https://support.microsoft.com/en-us/topic/windows-server-2022-update-history-e1caa597-00c5-4ab9-9f3e-8212fe80b2ee

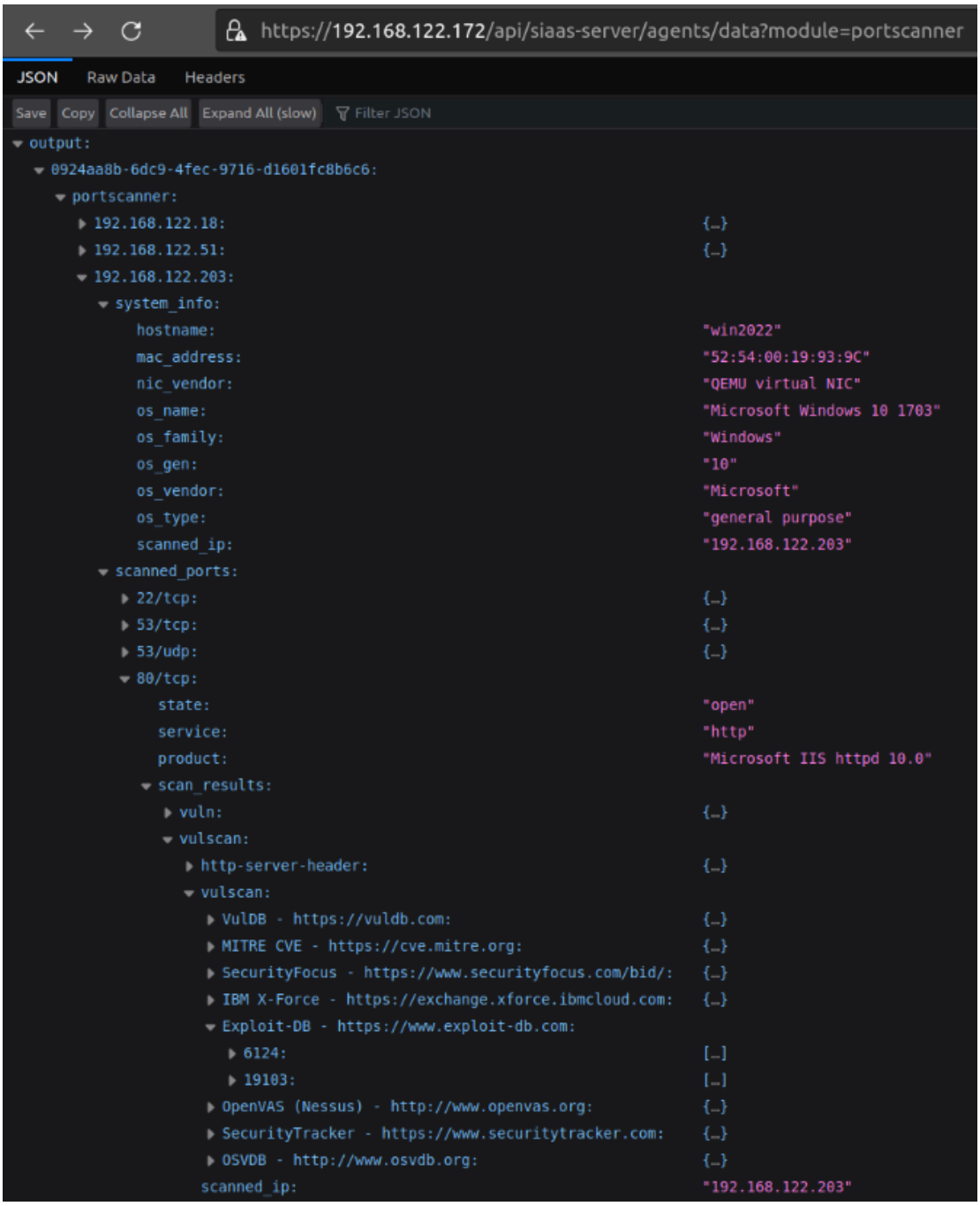

Figure 32 - Scanning results for a Windows Server VM during local tests

It was found out that the script “vulscan” [62] was more effective in detecting vulnerabilities in Windows than the pre-defined category “vuln” (which uses the script “vulners” [61] underneath), as the latter produced a significantly smaller output. Another important aspect to note is that vulscan works offline, while vulners does not. This is of special use when running the agent in the offline mode.

Apart from these cases, the “vuln” category worked perfectly well as the out-of-the-box configuration. The output is smaller and more organized, while detecting the most recent CVEs [29]. Also, it shows CVSS scores [31] and web links containing more information about the vulnerabilities, which is a useful addition.

Concluding, the developed artifact successfully detects the OS (even though sometimes the versions of the Kernel were not correct – merely a cosmetic issue, as this has no impact in service version detection), services and versions, and vulnerabilities, considering as far as it can go due to being a network-based scanner. The ability of using different scripts is also a plus, as vulscan proved to be a useful addition when scanning Windows hosts or when using the agent in the offline mode.

4.2.4. Security

To test and validate proper API responses to insecure requests, the cURL command line client [85] was used; from the SIAAS VM, against the SIAAS Server API running in JP-OLD.

As this work serves just as a proof of concept to assess the possibility of answering the research question (Section 1.2), the security configuration is minimal, implemented only at the protocol/service level. The API does not support multi-user or multi-tenancy. User authentication is implemented using simple HTTP authentication, so the login credentials are hard coded in the host (even though they can be changed by editing the SIAAS Server installation script, detailed in Section 3.5.5). Therefore, only minimalistic tests were done, to validate that the HTTPS protocol and simple HTTP authentication were correctly implemented.

A valid HTTPS (SSL) connection implies that, firstly, the client recognizes the CA that signs the endpoint certificate and, secondly, that the domain that the certificate is valid for matches the domain section of the requesting URL. SIAAS auto-generated SSL certificate is valid for the domains “siaas” and the server’s hostname, but not valid for the server’s IP address (as per commonly known recommendations, referenced in the related RFC [1]).

The following tests were done:

[1] https://www.rfc-editor.org/rfc/rfc6125#section-1.7.2

1. Request against the IP address (instead of the hostname), using the correct username/password, but without the endpoint's certificate to validate against (expected result: rejection);
2. Repeat, but explicitly using the client's flag to ignore certification validation (expected result: acceptance);
3. Request with proper username/password and CA bundle to validate against, but using IP address instead of hostname (expected result: rejection);
4. Repeat, but with correct hostname, and wrong username/password (expected result: rejection);
5. Repeat, but with correct user/name (expected result: acceptance);
6. Repeat, but using HTTP in the URL instead of HTTPS (expected result: client redirected from HTTP to HTTPS, and then accepted).

All the tests passed successfully, as shown in Technical Note F1 (Appendix F).

However, it is important to understand that, if this work is to be offered as a SaaS offering, a proper security implementation of the API at the application-level is necessary, with user/tenant separation, as well as proper security testing. Projects like *OWASP API Security Project* [137] attempt to compile strategies and solutions to implement API security in a systemized way. Adapting this work to be SaaS-ready (with multi-user and multi-tenancy), with a systemic approach to security compliance, is proposed in future work, in Section 5.3.

The complete raw and detailed outputs for all the local tests above can be consulted in Technical Note F1 (Appendix F).

4.3. User Tests

The following sections contain an analysis and review of the obtained results for the usability tests, done by external testers.

4.3.1. Trilio Data (US Company)

Trilio Data [138] is a company based in Massachusetts, United States, specializing in backing up popular cloud environments like *OpenStack* [139] and *Kubernetes* [140].

The artifact was tested, and the survey replied to, by Matt Golden [1], Senior Customer Success Engineer and Product Content Strategist, at the Customer Success Engineering team. The artifacts were provided to Matt on March 4, 2023, and the survey was replied to via a video call that took place on April 21, 2023, at 17:00 UTC. He opted for an AIO setup – installing both the agent and the server in a single VM – and performed tests in a local laboratory environment created by him for this purpose.

All of the replies were “agree” or “strongly agree”, to the exception of neutral replies to the statements regarding API integrability and network/host impact of the scanner, as these were not thoroughly tested by Matt. This denotes an overall satisfaction with the solution and a positive experience.

Regarding the open-ended questions, Matt pointed out that the only technical difficulty he experienced was crashing into the known issue detailed in Section 5.2 (BSON object size limitation). He got over this hurdle by configuring fewer scanning scripts to be run.

The improvement suggestions revolved around the creation of a GUI (more details in the “web frontend”-related future work suggestions in Section 5.3).

Finally, Matt pointed out that this project is interesting mainly because existing tools hide most of their features behind paid versions, whereas this project offers them for free.

Complete survey replies from Trilio Data can be consulted in Technical Note G1 (Appendix G).

Additional notes of interest: during the informal conversation after the survey, Matt referred that the possibility of having a portable scanner in a Raspberry Pi is also a great addition to a system administrator’s toolkit, so he considers installing an agent for his own use.

4.3.2. Altice Portugal

Altice Portugal [141] (formerly known as Portugal Telecom) is the oldest and largest telecommunications and cable provider in Portugal, providing a wide set of services for both

[1] https://www.linkedin.com/in/mattdgolden/

home and enterprise clients. It is mostly famous for its mobile and fixed telecommunications product named “MEO” [142].

The artifact was tested, and the survey replied to, by Ricardo Ramalho [1], Head of Cybersecurity Behaviour Analytics and Automation, at the Cybersecurity and Privacy Directorate. The artifacts were provided to Ricardo on March 30, 2023, and the survey was replied via a video call that took place on April 24, 2023, at 18:30 UTC. He opted for an AIO setup – installing both the agent and the server in a single VM – and performed tests in a local laboratory environment created by him for this purpose.

All of the replies were “agree” or “strongly agree”, to the exception of a neutral reply to the statement regarding e-mail reporting, as Ricardo did not test this. This denotes an overall satisfaction with the solution and a positive experience.

Ricardo has an extensive background in cybersecurity, so his inputs are of special value and based in large experience and knowledge. He noted that this project adds value to the cybersecurity auditing process of organizations, and even pointed out that this project has the potential to become a product. Ricardo also drilled down a bit on the construction of the survey’s organizational statements. Namely, he pointed out that prioritizing vulnerabilities is a much more complex matter than just discovering them. This observation matches with what was observed in Section 1.1.

The improvement suggestions were about delivering the SIAAS modules in *Docker* [143] containers (more details in the “containerization”-related future work suggestions in Section 5.3).

Complete survey replies from Altice Portugal can be consulted in Technical Note G1 (Appendix G).

Additional notes of interest: Ricardo noted that the usage of Swagger [115] is an important help in interacting with the API. He also noted that, when testing, the agent was not starting properly in an offline environment, right after installing. This bug had already been corrected by the time the survey was had, by forcing the SIAAS modules to install the virtual environment [117] in the installation script, as explained in Sections 3.4.6, 3.5.5, and 3.6.1 (at the time Ricardo was given the SIAAS Agent package, this correction had not been implemented yet).

[1] https://www.linkedin.com/in/ricardogramalho/

4.3.3. VTXRM (Portuguese Company)

VTXRM – Software Factory [144] is a Portuguese company focused on software development and expert consultancy, for the finance sector.

The artifact was tested, and the survey replied to, by Jorge Teixeira [1], Information Technology Team Lead. In the past, Jorge was also a Research Assistant in the Telecommunications Institute, at ISCTE-IUL [145]. The artifacts were provided to Jorge on March 30, 2023, and the survey was replied via a video call that took place on May 5, 2023, at 21:00 UTC. He opted for an AIO setup – installing both the agent and the server in a single VM – and performed tests in a local laboratory environment created by him for this purpose.

All of the replies were “agree” or “strongly agree”, to the exception of neutral replies to the statements regarding e-mail reporting, agent customization, API integration, and comparison with paid tools, as Jorge did not test them, or couldn’t assess the answer. This denotes an overall satisfaction with the solution and a positive experience.

In terms of limitations, Jorge pointed out that this tool – as it relies on ARP to discover hosts automatically, as explained in Section 3.4.2 – can be sensitive to ARP spoofing [2], by a malicious actor wanting to hide a machine in the network. Also, he noted that the deployment of the test scenario could be easier. For this end, Jorge suggested that the software could be provided in containers instead of installation scripts.

In terms of future improvements, Jorge suggested that a graphical interface could be deployed. In this interface, more comprehensive and less technical information about each vulnerability could be presented, compared to what is currently shown by the SIAAS Server API (detailed in Section 3.5.1).

Both the suggestions above (containerization and a graphical interface) are considered as future work in Section 5.3.

[1] https://www.linkedin.com/in/jorge-teixeira-8b364487/
[2] https://www.imperva.com/learn/application-security/arp-spoofing/

Finally, Jorge said that this tool got him more interested in cybersecurity and related concepts. He also said that this is a very interesting project, which he plans to use for personal projects.

Complete survey replies from VTXRM can be consulted in Technical Note G1 (Appendix G).

4.3.4. David Negreira (IT Freelancer, Ubuntu Community)

David Negreira [1], is an IT freelancing engineer with a strong background in Linux [56]. He's currently contracted by Canonical [146]. He is also an active member of the Ubuntu [99] community, co-authoring Podcast Ubuntu Portugal [2].

The artifacts were provided to David on May 6, 2023, and the survey was replied via a video call that took place on May 12, 2023, at 18:30 UTC. He opted for an AIO setup – installing both the agent and the server in a single Linux container (LXC) [147] – and performed tests in a local laboratory environment created by him for this purpose.

All of the replies were "agree" or "strongly agree", to the exception of neutral replies to the statements regarding the accuracy of the scanning results (more details below), and e-mail reporting and API integration (not tested). This denotes an overall satisfaction with the solution and a positive experience.

Regarding limitations, David pointed out that the results of the scanner are not completely accurate. This is caused by a known limitation of this type of scanner, already explained in Section 4.2.3: backported vulnerabilities via patching cannot be detected. Also, at the beginning of his tests, SIAAS Agent was not properly obtaining an ID. This happened because the agent was running in a container with limited privileges, which prevented it from obtaining a unique ID from the system. This was corrected by the author, and the problem fixed.

[1] https://www.linkedin.com/in/dnegreira/
[2] https://podcastubuntuportugal.org/

David's suggestions for improvements revolved around solving the forementioned limitation (detecting backported patches), as well as correctly detecting the OS distribution and version (example: Ubuntu 22.04), and even lifecycle status. These improvements are considered as future work ("agent-based software module"), in Section 5.3.

Finally, David mentioned that he sees potential for this solution to be widely adopted.

Complete survey replies from David Negreira can be consulted in Technical Note G1 (Appendix G).

4.3.5. Aggregated Results and Considerations

Figure 33 shows a graphical distribution of the agreeableness replies obtained.

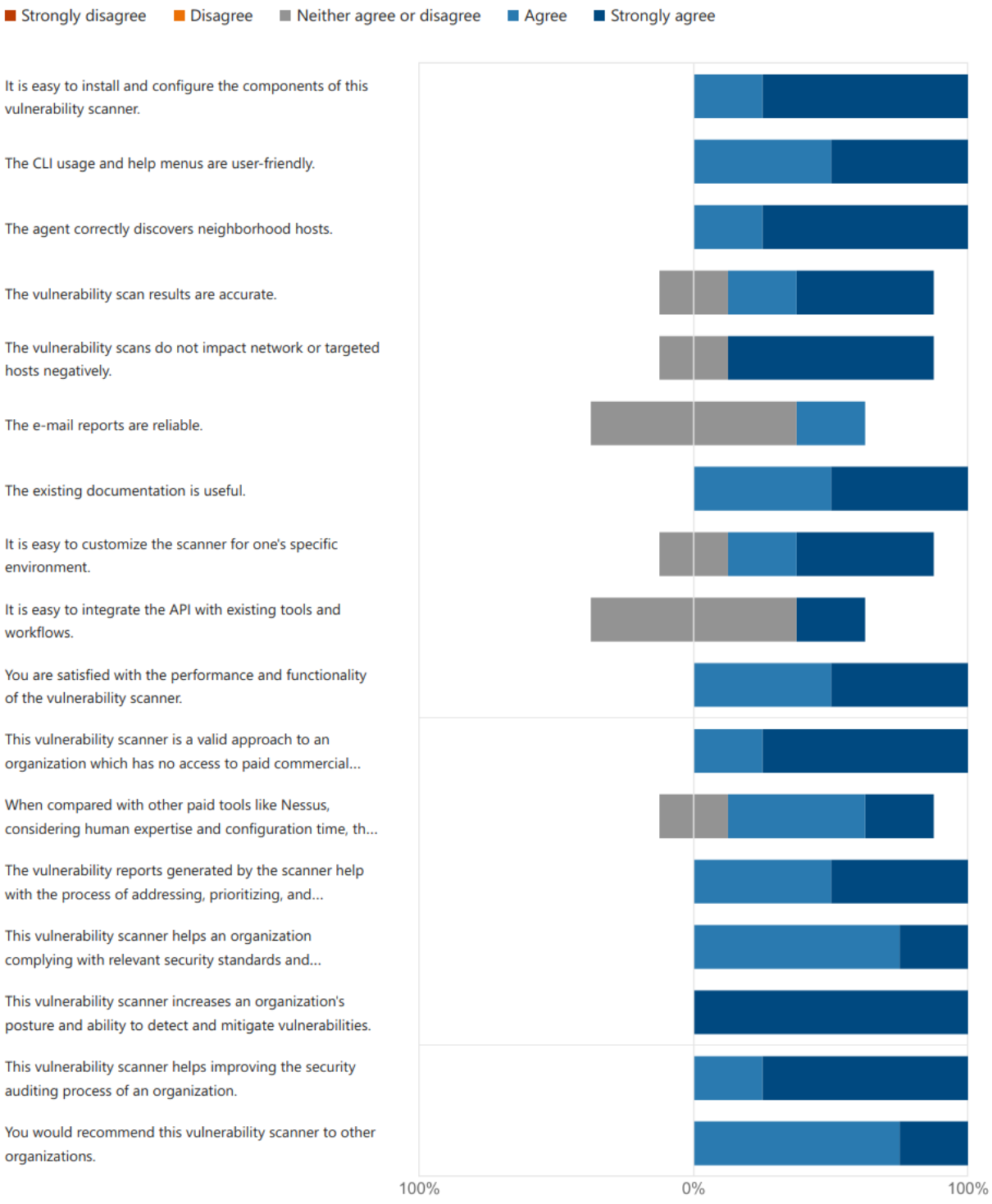

Figure 33 - Graphical distribution of the agreeableness replies given by the testers

As shown, the large majority of the replies are in the “agree” and “strongly agree” area, which denotes a strong satisfaction overall from all the testers.

In conclusion, external testers reported that the usage of the artifact is simple, effective, and would help an organization automate and improve their efforts towards having a better and more efficient cybersecurity posture. It helps them know which vulnerabilities need to be fixed on an urgent basis, by filtering exploits. It was also concluded that this tool provides special value for organizations that don’t have paid solutions or specialized cybersecurity specialists.

CHAPTER 5

Conclusions

Final considerations on the capabilities of the developed work compared with the current state-of-the-art, as well on the testing and validation results from the previous chapter. Formulation of an answer to the research question made in Section 1.2. Known issues are explained. Some ideas for future work are suggested. Project's availability to the community is detailed.

5.1. Final Considerations

This work attempted, first of all, to solve the research question presented in Section 1.2. Consequently, and as explained in Section 3.1, the goal was to – apart from other design aspects – design and develop a vulnerability scanner that was simple to use, configure, and scale, worked out of the box, could run on low-cost hardware, and available to the community.

Besides the port scanning performed by Nmap [42] – which was chosen for its simplicity, portability, efficiency, and easy integration with Python [63],[64] – this work implements an automatic discovery of hosts in the neighborhood. This list of hosts can then be fed to the port scanner, making it possible for the system to be working out of the box with minimal configuration, allowing a system administrator with no cybersecurity-related skills to operate it with minimal configuration effort. Still, its flexibility still allows the operator to configure a set of manual hosts, ports to scan, Nmap scripts to run, between other customizations.

The variety of Nmap script categories available [60] extends the scanner's capabilities beyond service/version crosschecking against vulnerability DBs. It also allows to perform more advanced penetration testing, sometimes even for specific vendors [1].

[1] https://packetstormsecurity.com/files/142066/ASUS-WRT-Cross-Site-Scripting-Nmap-NSE-Script.html

The presented work operates in an agent-server architecture. Not only this allows for load distribution, but the agents can also be connected directly to internal LANs, bypassing many of the bureaucratic and access granting needs of an external server scanner; especially in data centers which do not allow inbound connections.

This work is also fully modular. An agent can work independently from a server, if needed. The fully featured API can eventually be consumed by web interfaces, mobile clients, or even AI systems, implemented by the community (as suggested in Section 5.3). These web interfaces can potentially develop new features, like advanced graphing and reporting, as the API already provides all the necessary underlying data. As explained in Section 3.2, there's even the possibility of creating an AIO environment with both the agent and server installed in the same machine or VM (the testers of the artifact were offered and opted for this setup).

The developed solution also generates vulnerability reports and sends them via e-mail in a file using the well-known CSV format. The granularity of this report can be configured, allowing system administrators to focus on only exploitable vulnerabilities that need to be fixed on priority.

To the best extent of the author's knowledge – and considering the study of the current state-of-the-art solutions and their limitations (Section 2.3) – there is no solution at this time that covers all the aspects detailed above in a single open-source offering.

Testing the artifact (Chapter 4) revealed that it is stable and reliable, even in older hardware with fewer resources (given the proper configuration). Considering the limitations inherent to the type of scanner that is implemented in this artifact (more information in Section 4.2.3), it effectively and accurately detects hosts in the network, their OSes and services, and vulnerabilities. The possibility to select from multiple Nmap scripts allows the operator to use the best option for a given target host's OS, or for a specific mode of operation of the agent. The security-related tests, even though minimalistic, have shown that the API implementation does not allow unauthorized or untrusted connections.

Survey responses by external users – even those with no cybersecurity-specific background – have shown that the artifact is easy to install and operate, correctly assessing the characteristics of their environment, and providing valuable output, with no negative impact in the rest of the infrastructure. Additionally, all of the testers stated that the artifact has shown to be effective in providing added value in automating the security audit processes of organizations, even if no other software options are available. These two points are of particular interest, as they are essential to answer positively the research question of this work.

Considering all the paragraphs above, the author therefore considers that the research question defined in Section 1.2 was positively answered.

5.2. Known Issues

As with other technological products and projects, the developed artifact has issues that might be observed under specific circumstances. These might be addressed in the future, if needed. The following issues are observed:

- MongoDB has a 16 MB limitation when it comes to the BSON document being uploaded [148]. This might impact data uploading, especially in agents running a large number of scripts in a large number of hosts. In local tests (Technical Note F1 (Appendix F)), it was observed that an agent targeting a single host (running FTP, NFS, HTTP, and SSH), using the default “vuln” script, took around 26 KB of data. This gives a maximum value of around 615 hosts per agent. If 3 scripts are run (“vuln,vulscan,discovery”), it will take around 141 KB, lowering the maximum value to around 113 hosts. A potential solution to this problem is using *GridFS* [149].

5.3. Future Work

Considering the developed work and the testers’ feedback, the author suggests the following future work as means of improving the current artifact:

Related to UI/UX, a web frontend for the server's API. Some features that this frontend could implement: AAA/SaaS multi-tenant cloud offering with a systematic approach to API security (as described in Section 4.2.4), advanced graphical reporting, stateful metadata information (like marking vulnerabilities as resolved, remembering which vulnerabilities were already seen by the operator, or notifying the operator for new vulnerabilities found since last visit), improved e-mail reports (a more advanced implementation, directly in the frontend, bypassing the Mailer module from the backend described in Section 3.5.4), advanced scheduling of the agent's and server's modules (instead of the current method of having a fixed time interval between module loops). Ideally, this web frontend would be a completely independent module from the SIAAS Server, acting as just another client and consumer of the server's API.

This web frontend could also have a mobile application version;

Related to AI/ML, an AI system that feeds on the API's output and determines courses of remediation. One possible use case would be suggesting which possible patches could be applied in a system to solve the existing vulnerabilities, using OS information, services running and their version, and the list of found vulnerabilities for each of these services;

Related to vulnerability scanning, an optional agent-based software module that could be installed in target hosts and report extra information directly to the server (in order to drill down on information like backported vulnerability fixes for older versions of services as explained in Section 4.2.2, or obtain more detailed information about the OS distribution, version, and lifecycle, as suggested by the tester in Section 4.3.4);

The possibility of adding extra tools to the agent (suggestion: using specialized tools like *Metasploit* [150] to perform extensive pentesting on specific services, by probing found exploits and then running post-exploitable code [1] [2], like described in Section 2.2.4). This would imply changes in the agent's and server's code (and possibly the CLI's), as the agent would have to run the forementioned tools and support related configurations, and the operator would need to configure the agents remotely via the server's API;

[1] https://www.varonis.com/blog/what-is-metasploit
[2] https://www.cm-alliance.com/cybersecurity-blog/using-metasploit-and-nmap-to-scan-for-vulnerabilities

And, finally, generic improvements: code optimization and proper unit testing, packaging and/or containerization (containerization needs to be carefully evaluated in the agent's case, as the added complexity might negatively impact its performance), possibility of having a high-availability (HA) cluster running the server module, and accessibility and internationalization.

Authors of future improvements or changes should always bear in mind that the core objective underlying this work – as explained in Section 3.1 – is to offer to the community a tool that is simple enough to work out-of-the-box, with little to no extra configuration needed, that even operators with no specific cybersecurity expertise can operate. New features should not get in the way of this philosophical foundation.

5.4. Project Availability, Credits, and Contributions

The three artifacts produced by this work are available online in *GitHub* [151],[152],[153], and are licensed under GPL [154] (version 3). Code was written in the *vim* text editor [155] and in the *PyCharm* IDE [156], and automatically styled in accordance with PEP 8 (style guide for Python code) [157], with the help of *autopep8* [158].

The Platform module for both SIAAS Agent and SIAAS Server (Sections 3.4.1 and 3.5.2) contains pieces of code from a blog post from Abdou Rockikz, named "How to Get Hardware and System Information in Python" [87].

The Neighborhood module for SIAAS Agent (Section 3.4.2) contains pieces of code from the "Layer 2 network neighbourhood discovery tool" project, by Benedikt Waldvogel [89], which is licensed under GPL [154] (version 2).

A pull request [159] was also proposed by the author of this work in the python3-nmap project [64]. It was accepted by the owners of the project and incorporated in the module, meaning it is now available for the entire community to use.

Finally, an article based on the findings of this work was submitted and published in the peer-reviewed journal *European Alliance for Innovation: Endorsed Transactions on Scalable Information Systems* [160]. It is fully available online under open access [161].

References

[1] Check Point Blog, “Check Point Research: Third quarter of 2022 reveals increase in cyberattacks and unexpected developments in global trends”, checkpoint.com, https://blog.checkpoint.com/2022/10/26/third-quarter-of-2022-reveals-increase-in-cyberattacks/ (accessed: 2023/08/31)

[2] B. Connolly, “Is automation the future of cyber security?”, *CIO (13284045)*, IDG Communications, 2017/10/10, pp. 4. [Online] Available: https://www.cio.com/article/203489/is-automation-the-future-of-cyber-security.html (accessed: 2023/08/31)

[3] Cybersecurity and Infrastructure Security Agency (CISA), “Cost of a Cyber Security Incident: Systematic Review and Cross-Validation”, 2020

[4] Congressional Research Service, “Global Research and Development Expenditures: Fact Sheet”, 2022/09/14

[5] S. Morgan (Cybercrime Magazine), “Cybercrime To Cost The World $10.5 Trillion Annually By 2025”, cybersecurityventures.com, https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/ (accessed: 2023/08/31)

[6] Kaspersky Lab, “Damage Control: The Cost of Security Breaches; IT Security Risks Special Report Series”, 2015?

[7] S. Furnell, P. Fischer, and A. Finch, “Can't get the staff? The growing need for cyber-security skills”, *Computer Fraud & Security*, 2017, vol. 2017, i. 2, pp. 5-10, doi: 10.1016/S1361-3723(17)30013-1

[8] S. Furnell, “The cybersecurity workforce and skills”, *Computer Fraud & Security*, 2021, vol. 100, i. C, doi: 10.1016/j.cose.2020.102080

[9] C. Russu, “The impact of low cyber security on the development of poor nations”, developmentaid.org, https://www.developmentaid.org/news-stream/post/149553/low-cyber-security-and-development-of-poor-nations (accessed: 2023/08/31)

[10] T. Caldwell, “Plugging the cyber-security skills gap”, *Computer Fraud & Security*, 2013, vol. 2013, i. 7, pp. 5-10, doi: 10.1016/S1361-3723(13)70062-9

[11] G. Smith, “The intelligent solution: automation, the skills shortage and cyber-security”, *Computer Fraud & Security*, 2018, vol. 2018, i. 8, pp. 6-9, doi: 10.1016/S1361-3723(18)30073-3

[12] R. K. L. Ko, “Cyber Autonomy: Automating the Hacker – Self-healing, self-adaptive, automatic cyber defense systems and their impact to the industry, society and national security”, *arXiv*, 2020, doi: 10.48550/arXiv.2012.04405

[13] “Introducing ChatGPT”, openai.com, https://openai.com/blog/chatgpt (accessed: 2023/08/31)

[14] Deascona, “How ChatGPT will revolutionize the cyber security industry”, uxdesign.cc, https://bootcamp.uxdesign.cc/how-chat-gpt-will-revolutionize-the-cyber-security-industry-7847cc7fc24e (accessed: 2023/08/31)

[15] Ponemon Institute (sponsored by Rezilion), "The State of Vulnerability Management in DevSecOps", 2022

[16] Julia Anderson, “Updates to ISO 27001/27002 raise the bar on application security and vulnerability scanning”, invict.com, https://www.invicti.com/blog/web-security/iso-27001-27002-changes-in-2022-application-security-vulnerability-scanning/ (accessed: 2023/08/31)

[17] S. Shea, “SOAR (security orchestration, automation and response)”, techtarget.com, https://www.techtarget.com/searchsecurity/definition/SOAR (accessed: 2023/08/31)

[18] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, “A Design Science Research Methodology for Information Systems Research”, *Journal of Management Information Systems*, 2007, vol. 24, no. 3, pp. 45-77, doi: 10.2753/MIS0742-1222240302

[19] “Google”, google.com, https://www.google.com (accessed: 2023/08/31)

[20] “B-On”, b-on.pt, https://www.b-on.pt/ (accessed: 2023/08/31)

[21] “IEEE Xplore”, ieee.org, https://ieeexplore.ieee.org/ (accessed: 2023/08/31)

[22] “Google Scholar”, google.com, https://scholar.google.com (accessed: 2023/08/31)

[23] D. Moher, A. Liberati, J. Tetzlaff, D. G. Altman, and The PRISMA Group, “Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement”, *PLoS Medicine*, 2009, vol. 6, no. 7, doi: 10.1371/journal.pmed.1000097

[24] N. Mandal and S. Jadhav, "A survey on network security tools for open source”, *2016 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC)*, 2016, pp. 1-6, doi: 10.1109/ICCTAC.2016.7567330

[25] I. Zulkarneev and A. Kozlov, "New Approaches of Multi-agent Vulnerability Scanning Process", *2021 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, 2021, pp. 488-490, doi: 10.1109/USBEREIT51232.2021.9455061

[26] W. Haydock, "But is it exploitable?", deploy-securely.com, https://www.blog.deploy-securely.com/p/but-is-it-exploitable (accessed: 2023/08/31)

[27] Encyclopedia by Kaspersky, “What is an attack vector?”, kaspersky.com, https://encyclopedia.kaspersky.com/glossary/attack-vector/ (accessed: 2023/08/31)

[28] A. Granova and M. Slaviero, “Cyber Warfare*”, Computer and Information Security Handbook (Third Edition)*, Morgan Kaufmann, 2007, ch. 83, pp. 1085-1104, doi: 10.1016/B978-0-12-803843-7.00083-1

[29] “Common Vulnerabilities and Exposures (CVE)”, mitre.org, https://cve.mitre.org/ (accessed: 2023/08/31)

[30] “National Vulnerability Database (NVD)”, nist.gov, https://nvd.nist.gov/ (accessed: 2023/08/31)

[31] National Vulnerability Database (NVD), “Vulnerability Metrics”, nist.gov, https://nvd.nist.gov/vuln-metrics/cvss (accessed: 2023/08/31)

[32] “Security Content Automation Protocol (SCAP)”, nist.gov, https://csrc.nist.gov/projects/security-content-automation-protocol/ (accessed: 2023/08/31)

[33] A. D'Hondt and H. Bahmad, "Understanding SCAP Through a Simple Use Case", Hakin9 IT Security Magazine, 2016/03, vol. 11, no. 3, pp. 100-110. [Online] Available: https://dial.uclouvain.be/memoire/ucl/en/object/thesis:8128/datastream/PDF_05/view (accessed: 2023/08/31)

[34] “Common Platform Enumeration (CPE)”, nist.gov, https://nvd.nist.gov/products/cpe (accessed: 2023/08/31)

[35] “Common Configuration Enumeration (CCE)”, nist.gov, https://csrc.nist.gov/Projects/Security-Content-Automation-Protocol/Specifications/common-configuration-enumeration-cce (accessed: 2023/08/31)

[36] “Open Vulnerability and Assessment Language (OVAL)”, mitre.org, https://oval.mitre.org/ (accessed: 2023/08/31)

[37] “Extensible Configuration Checklist Description Format (XCCDF)”, nist.gov, https://csrc.nist.gov/projects/security-content-automation-protocol/specifications/xccdf (accessed: 2023/08/31)

[38] “Common Weakness Enumeration (CWE)”, mitre.org, https://cwe.mitre.org/ (accessed: 2023/08/31)

[39] “OpenSCAP”, open-scap.org, https://www.open-scap.org/ (accessed: 2023/08/31)

[40] I. Chalvatzis, D. A. Karras, and R. C. Papademetriou, "Evaluation of Security Vulnerability Scanners for Small and Medium Enterprises Business Networks Resilience towards Risk Assessment", *2019 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2019, pp. 52-58, doi: 10.1109/ICAICA.2019.8873438

[41] W. Liu, "Design and Implement of Common Network Security Scanning System", *2009 International Symposium on Intelligent Ubiquitous Computing and Education*, 2009, pp. 148-151, doi: 10.1109/IUCE.2009.24

[42] G. F. Lyon, *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/toc.html (accessed: 2023/08/31)

[43] G. F. Lyon, “Port Scanning Techniques and Algorithms – TCP SYN (Stealth) Scan (-sS)”, *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 5, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/synscan.html (accessed: 2023/08/31)

[44] S. Liao et al., "A Comprehensive Detection Approach of Nmap: Principles, Rules and Experiments", *2020 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC*), 2020, pp. 64-71, doi: 10.1109/CyberC49757.2020.00020

[45] E. Hjelmvik, “Passive OS Fingerprinting”, netresec.com, https://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting (accessed: 2023/08/31)

[46] E. Chikohora and L. Mogomeli, "A Study on the Impact of Network Vulnerability Scanners on Network Security”, *2021 3rd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, 2021, pp. 1-4, doi: 10.1109/IMITEC52926.2021.9714598

[47] A. Chen and Z. Zhang, "A Comparative Study of Credentialed Vulnerability Scanning and Non-credentialed Vulnerability Scanning", *2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2021, pp. 1613-1616, doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00215

[48] G. F. Lyon, "Nmap Output Formats – Common Platform Enumeration (CPE)", *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 13, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/output-formats-cpe.html (accessed: 2023/08/31)

[49] "Script vulners", nmap.org, https://nmap.org/nsedoc/scripts/vulners.html (accessed: 2023/08/31)

[50] "PTES – Penetration Testing Execution Standard", pentest-standard.org, http://www.pentest-standard.org/ (accessed: 2023/08/31)

[51] "OpenVAS", openvas.org, https://www.openvas.org/ (accessed: 2023/08/31)

[52] "Nessus", tenable.com, https://www.tenable.com/products/nessus (accessed: 2023/08/31)

[53] "Nexpose", rapid7.com, https://www.rapid7.com/products/nexpose/ (accessed: 2023/08/31)

[54] "Faraday", faradaysec.com, https://faradaysec.com/ (accessed: 2023/08/31)

[55] "Vuls", vuls.io, https://vuls.io/ (accessed: 2023/08/31)

[56] "What is Linux?", linux.com, https://www.linux.com/what-is-linux/ (accessed: 2023/08/31)

[57] A. Mohan, G. A. Swaminathan, and N. J. Shafana, "Automated Tools and Techniques in Vulnerability Assessment", *2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT),* 2022, pp. 533-540, doi: 10.1109/ICSSIT53264.2022.9716474

[58] Y. Wang and J. Yang, "Ethical Hacking and Network Defense: Choose Your Best Network Vulnerability Scanning Tool", *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 2017, pp. 110-113, doi: 10.1109/WAINA.2017.39

[59] "NSE Scripts", nmap.org, https://nmap.org/nsedoc/scripts/ (accessed: 2023/08/31)

[60] G. F. Lyon, "Nmap Scripting Engine – Usage Examples", *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 9, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/nse-usage.html (accessed: 2023/08/31)

[61] "Nmap plugin and http-vulners-regex", vulners.com, https://vulners.com/docs/plugins/nmap_vulners/ (accessed: 2023/08/31)

[62] "vulscan.nse", computec.ch, https://www.computec.ch/projekte/vulscan/ (accessed: 2023/08/31)

[63] "What is Python?", python.org, https://www.python.org/doc/essays/blurb/ (accessed: 2023/08/31)

[64] "python3-nmap", pypi.org, https://pypi.org/project/python3-nmap/ (accessed: 2023/08/31)

[65] "Zenmap", nmap.org, https://nmap.org/zenmap/ (accessed: 2023/08/31)

[66] Tenable Documentation, "Benefits and Limitations (Nessus Agents)", tenable.com, https://docs.tenable.com/nessusagent/Content/BenefitsAndLimitations.htm (accessed: 2023/08/31)

[67] "Scanner-VS", scanner-vs.ru, https://scaner-vs.ru/ (accessed: 2023/08/31)

[68] "CyBot", cronus-cyber.com, https://cronus-cyber.com/ (accessed: 2023/08/31)

[69] "XSpider", ptsecurity.com, https://www.ptsecurity.com/ww-en/products/xspider/ (accessed: 2023/08/31)

[70] "Qualys", qualys.com, https://www.qualys.com/ (accessed: 2023/08/31)

[71] "Retina CS", beyondtrust.com, https://www.beyondtrust.com/vulnerability-management (accessed: 2023/08/31)

[72] Microsoft Learn, "What is Microsoft Baseline Security Analyzer and its uses?", microsoft.com, https://learn.microsoft.com/en-us/windows/security/threat-protection/mbsa-removal-and-guidance (accessed: 2023/08/31)

[73] "HCL AppScan", hcltechsw.com, https://www.hcltechsw.com/appscan (accessed: 2023/08/31)

[74] Invicti, "Acutenix", acutenix.com, https://www.acunetix.com/ (accessed: 2023/08/31)

[75] S. Shah and B. M. Mehtre, "An automated approach to Vulnerability Assessment and Penetration Testing using Net-Nirikshak 1.0", *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, 2014, pp. 707-712, doi: 10.1109/ICACCCT.2014.7019182

[76] Y. Wang, Y. Bai, L. Li, X. Chen, and A. Chen, "Design of Network Vulnerability Scanning System Based on NVTs", *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 2020, pp. 1774-1777, doi: 10.1109/ITOEC49072.2020.9141812

[77] H. Chen, J. Chen, J. Chen, S. Yin, Y. Wu, and J. Xu, "An Automatic Vulnerability Scanner for Web Applications", *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2020, pp. 1519-1524, doi: 10.1109/TrustCom50675.2020.00207

[78] X. Zhang et al., "An Automated Composite Scanning Tool with Multiple Vulnerabilities", *2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2019, pp. 1060-1064, doi: 10.1109/IMCEC46724.2019.8983828

[79] C. Wang et al., "FalconEye: A High-Performance Distributed Security Scanning System", *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2019, pp. 282-288, doi: 10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00059

[80] P. Davies and T. Tryfonas, "A lightweight web-based vulnerability scanner for small-scale computer network security assessment", *Journal of Network and Computer Applications*, 2009, vol. 32, i. 1, pp. 78-95, doi: 10.1016/j.jnca.2008.04.007

[81] S. Kals, E. Kirda, C. Kruegel, and N. Jovanovic, "SecuBat: a web vulnerability scanner", *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, 2006, pp. 247-256, doi: 10.1145/1135777.1135817

[82] M. Noman, M. Iqbal, K. Rasheed, and M. Muneeb Abid, "Web Vulnerability Finder (WVF): Automated Black-Box Web Vulnerability Scanner", *International Journal of Information Technology and Computer Science*, 2020, vol. 12, pp. 38-46, doi: 10.5815/ijitcs.2020.04.05

[83] "What is a Raspberry Pi?", rasberrypi.org, https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/ (accessed: 2023/08/31)

[84] "What is MongoDB?", mongodb.com, https://www.mongodb.com/en-us/what-is-mongodb (accessed: 2023/08/31)

[85] "cURL", curl.se, https://curl.se/ (accessed: 2023/08/31)

[86] "Firefox", mozilla.org, https://www.mozilla.org/en-US/firefox/ (accessed: 2023/08/31)

[87] A. Rockikz, "How to Get Hardware and System Information in Python", thepythoncode.com, https://www.thepythoncode.com/article/get-hardware-system-information-python (accessed: 2023/08/31)

[88] "Scapy", scapy.net, https://scapy.net/ (accessed: 2023/08/31)

[89] B. Waldvogel, "Layer 2 network neighbourhood discovery tool", github.com, https://github.com/bwaldvogel/neighbourhood (accessed: 2023/08/31)

[90] "Django", djangoproject.com, https://www.djangoproject.com/ (accessed: 2023/08/31)

[91] "Flask", palletsprojects.com, https://flask.palletsprojects.com/ (accessed: 2023/08/31)

[92] "FastAPI", tiangolo.com, https://fastapi.tiangolo.com/ (accessed: 2023/08/31)

[93] "Apache HTTP Server", apache.com, https://httpd.apache.org/ (accessed: 2023/08/31)

[94] "Nginx", nginx.com, https://www.nginx.com/ (accessed: 2023/08/31)

[95] "What is MySQL?", oracle.com, https://www.oracle.com/mysql/what-is-mysql/ (accessed: 2023/08/31)

[96] "mysql", pypi.org, https://pypi.org/project/mysql/ (accessed: 2023/08/31)

[97] "pymongo", pypi.org, https://pypi.org/project/pymongo/ (accessed: 2023/08/31)

[98] "Click", palletsprojects.com, https://click.palletsprojects.com/ (accessed: 2023/08/31)

[99] "Ubuntu", ubuntu.com, https://ubuntu.com/ (accessed: 2023/08/31)

[100] "Debian", debian.org, https://www.debian.org/ (accessed: 2023/08/31)

[101] "Raspberry Pi OS", raspberrypi.com, https://www.raspberrypi.com/software/ (accessed: 2023/08/31)

[102] Debian Packages, "Details of package python3 in bullseye", debian.org, https://packages.debian.org/bullseye/python3 (accessed: 2023/08/31)

[103] Debian Manpages, "dmidecode(8) – dmidecode – Debian bullseye", debian.org, https://manpages.debian.org/bullseye/dmidecode/dmidecode.8.en.html (accessed: 2023/08/31)

[104] "psutil", pypi.org, https://pypi.org/project/psutil/ (accessed: 2023/08/31)

[105] "py-cpuinfo", pypi.org, https://pypi.org/project/py-cpuinfo/ (accessed: 2023/08/31)

[106] Debian Manpages, "ip-neighbour(8) – iproute2 – Debian bullseye", debian.org, https://manpages.debian.org/bullseye/iproute2/ip-neighbour.8.en.html (accessed: 2023/08/31)

[107] Debian Packages, "Details of package nmap in bullseye", debian.org, https://packages.debian.org/bullseye/nmap (accessed: 2023/08/31)

[108] Python Docs, “concurrent.futures – Launching parallel tasks”, python.org, https://docs.python.org/3/library/concurrent.futures.html (accessed: 2023/08/31)

[109] G. F. Lyon, “Nmap Network Scanning – Nmap Reference Guide – OS Detection”, *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 15, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/man-os-detection.html (accessed: 2023/08/31)

[110] G. F. Lyon, “Nmap Network Scanning – Nmap Reference Guide – Service and Version Detection”, *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 15, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/man-version-detection.html (accessed: 2023/08/31)

[111] G. F. Lyon, “Nmap Network Scanning – Nmap Reference Guide – Port Specification and Scan Order”, *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 15, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/man-port-specification.html (accessed: 2023/08/31)

[112] G. F. Lyon, “Nmap Network Scanning – Nmap Reference Guide – Nmap Scripting Engine (NSE)”, *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 15, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/man-nse.html (accessed: 2023/08/31)

[113] G. F. Lyon, “Nmap Network Scanning – Port Scanning Techniques and Algorithms – UDP Scan (-sU)”, *Nmap Network Scanning; The Official Nmap Project Guide to Network Discovery and Security Scanning*, Insecure Press, 2008, ch. 5, ISBN 978-0-9799587-1-7. [Online] Available: https://nmap.org/book/scan-methods-udp-scan.html (accessed: 2023/08/31)

[114] “requests", pypi.org, https://pypi.org/project/requests/ (accessed: 2023/08/31)

[115] “Swagger”, swagger.io, https://swagger.io/ (accessed: 2023/08/31)

[116] “flask-swagger-ui”, pypi.org, https://pypi.org/project/flask-swagger-ui/ (accessed: 2023/08/31)

[117] Python Docs, “venv – Creation of virtual environments”, python.org, https://docs.python.org/3/library/venv.html (accessed: 2023/08/31)

[118] “systemd”, systemd.io, https://systemd.io/ (accessed: 2023/08/31)

[119] Python Docs, “logging – Logging facility for Python”, python.org, https://docs.python.org/3/library/logging.html (accessed: 2023/08/31)

[120] MongoDB Docs, “Install MongoDB Community Edition on Debian”, mongodb.com, https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-debian/ (accessed: 2023/08/31)

[121] Debian Packages, “Details of package apache2 in bullseye”, debian.org, https://packages.debian.org/bullseye/apache2 (accessed: 2023/08/31)

[122] Debian Packages, “Details of package openssl in bullseye”, debian.org, https://packages.debian.org/bullseye/openssl (accessed: 2023/08/31)

[123] Apache Docs, “Apache Virtual Host documentation”, apache.org, https://httpd.apache.org/docs/2.4/vhosts/ (accessed: 2023/08/31)

[124] Python Docs, “csv – CSV File Reading and Writing”, python.org, https://docs.python.org/3/library/csv.html (accessed: 2023/08/31)

[125] Python Docs, “email – An email and MIME handling package”, python.org, https://docs.python.org/3/library/email.html (accessed: 2023/08/31)

[126] Python Docs, “smtplib – SMTP protocol client”, python.org, https://docs.python.org/3/library/smtplib.html (accessed: 2023/08/31)

[127] "Gmail", google.com, https://mail.google.com/ (accessed: 2023/08/31)

[128] Microsoft, "Outlook.com", live.com, https://outlook.live.com/ (accessed: 2023/08/31)

[129] “jq”, github.io, https://stedolan.github.io/jq/ (accessed: 2023/08/31)

[130] “Pygments”, pypi.org, https://pypi.org/project/Pygments/ (accessed: 2023/08/31)

[131] R. Likert, “A Technique for the Measurement of Attitudes”, Archives of Psychology, 1932, vol. 22, no. 140, pp. 5-55

[132] “Microsoft Teams”, microsoft.com, https://teams.microsoft.com/ (accessed: 2023/08/31)

[133] “Microsoft Forms”, microsoft.com, https://forms.microsoft.com (accessed: 2023/08/31)

[134] "KVM", linux-kvm.org, https://www.linux-kvm.org/page/Main_Page (accessed: 2023/08/31)

[135] "Munin", munin-monitoring.org, http://munin-monitoring.org/ (accessed: 2023/08/31)

[136] “Windows Server 2022, microsoft.com, https://www.microsoft.com/en-us/windows-server (accessed: 2023/08/31)

[137] “OWASP API Security Project”, owasp.org, https://owasp.org/www-project-api-security/ (accessed: 2023/08/31)

[138] “Trilio”, trilio.io, https://trilio.io/ (accessed: 2023/08/31)
[139] “OpenStack”, openstack.org, https://www.openstack.org/ (accessed: 2023/08/31)
[140] “Kubernetes”, kubernetes.io, https://kubernetes.io/ (accessed: 2023/08/31)
[141] “Altice Portugal”, altice.pt, https://www.altice.pt/pt (accessed: 2023/08/31)
[142] “MEO”, meo.pt, https://www.meo.pt/ (accessed: 2023/08/31)
[143] “Docker”, docker.com, https://www.docker.com/ (accessed: 2023/08/31)
[144] “VTXRM – Software Factory”, vtxrm.com, https://www.vtxrm.com/ (accessed: 2023/08/31)
[145] “ISCTE – Instituto Universitário de Lisboa”, iscte-iul.pt, https://www.iscte-iul.pt/ (accessed: 2023/08/31)
[146] “Canonical”, canonical.com, https://canonical.com/ (accessed: 2023/08/31)
[147] “Linux Containers”, linuxcontainers.org, https://linuxcontainers.org/ (accessed: 2023/08/31)
[148] MongoDB Docs, “MongoDB Limits and Thresholds – BSON Documents – BSON Document Size”, mongodb.com, https://www.mongodb.com/docs/manual/reference/limits/#mongodb-limit-BSON-Document-Size (accessed: 2023/08/31)
[149] MongoDB Docs, “GridFS”, mongodb.com, https://www.mongodb.com/docs/manual/core/gridfs/ (accessed: 2023/08/31)
[150] Rapid7, “Metasploit”, metasploit.com, https://www.metasploit.com/ (accessed: 2023/08/31)
[151] J. P. Seara, “Intelligent System for Automation of Security Audits (SIAAS) – Agent”, github.com, https://github.com/jpseara/siaas-agent (accessed: 2023/08/31)
[152] J. P. Seara, “Intelligent System for Automation of Security Audits (SIAAS) – Server”, github.com, https://github.com/jpseara/siaas-server (accessed: 2023/08/31)
[153] J. P. Seara, “Intelligent System for Automation of Security Audits (SIAAS) – Command Line Interface”, github.com, https://github.com/jpseara/siaas-cli (accessed: 2023/08/31)
[154] GNU Project, “Licenses”, gnu.org, https://www.gnu.org/licenses/licenses.html.en (accessed: 2023/08/31)
[155] “Vim”, vim.org, https://www.vim.org/ (accessed: 2023/08/31)
[156] JetBrains, "PyCharm", jetbrains.com, https://www.jetbrains.com/pycharm/ (accessed: 2023/08/31)
[157] Python Enhancement Proposals, "PEP 8 – Style Guide for Python", python.org, https://peps.python.org/pep-0008/ (accessed: 2023/08/31)
[158] “autopep8”, pypi.org, https://pypi.org/project/autopep8/ (accessed: 2023/08/31)
[159] J. P. Seara, "python3-nmap – Pull request – Added "task_results" to output", github.com, https://github.com/nmmapper/python3-nmap/pull/84 (accessed: 2023/08/31)
[160] “EAI Endorsed Transactions on Scalable Information Systems”, eai.eu, https://publications.eai.eu/index.php/sis (accessed: 2023/11/25)
[161] J. P. Seara and C. Serrão, “Intelligent System for Automation of Security Audits (SIAAS)”, *EAI Endorsed Transactions on Scalable Information Systems*, 2023, vol. 11, no. 1, doi: 10.4108/eetsis.3564

Appendix A – Agent Configuration Reference

Key	Protected (no runtime changes)	Default (by omission)	Extra details
api_pwd	X	(None)	SIAAS API user password
api_ssl_ca_bundle	X	(None)	SSL CA bundle of the SIAAS API endpoint's certificate
api_ssl_ignore_verify	X	(None)	Ignore SIAAS API endpoint's SSL CA verification
api_uri	X	(None)	SIAAS API URI
api_user	X	(None)	SIAAS API user name
datatransfer_loop_interval_sec		3600	Seconds between the Data Transfer module iterations
disable_neighborhood_discovery		false	Disables discovery of known hosts in the local networks
disable_portscanner		false	Disables port scanning (system info and vulnerabilities)
disable_wifi_auto_discovery		false	Disables discovering hosts on the Wi-Fi interfaces
enable_internal_api	X	false	Enables the agent's local API
log_level	X	info	Logging level. Possible values: debug, info, warn, error, critical
manual_hosts		(None)	Comma-separated list of manually configured hosts to scan. Example: 109.51.61.78,sapo.pt,google.com
neighborhood_loop_interval_sec		900	Seconds between the Neighborhood module iterations
nmap_portscan_timeout_sec		600	Nmap timeout value in seconds when scanning a target host's port for vulnerabilities
nmap_scripts		(None)	Comma-separated list of Nmap scripts/categories to run. When not defined, no Nmap scripts are run. Possible values: auth, broadcast, brute, default, discovery, dos, exploit, external, fuzzer, intrusive, malware, nmap-vulners, safe, version, vuln, vulscan
nmap_sysinfo_timeout_sec		600	Nmap timeout value in seconds when scanning a target host for system information
offline_mode	X	true	Agent will not communicate with the server at all
platform_loop_interval_sec		300	Seconds between the Platform module iterations
portscanner_loop_interval_sec		86400	Seconds between the Portscanner module iterations
portscanner_max_parallel_workers		(None)	Defines a fixed maximum number of parallel workers (threads) to run when port scanning the target hosts. If not defined, this number is automatically managed by Python
scan_only_manual_hosts		false	Port scans only manually configured hosts
silent_mode		false	Download published configurations, but don't upload any data
target_specific_ports		(None)	Restrict port scanning to a comma-separated list of ports/ranges. Example: 22,80,8080-8083

Table A1 - SIAAS Agent configuration reference

Important note: The sample configuration file shipped with the SIAAS Agent software package does not necessarily have the default values set.

Appendix B – Server Configuration Reference

Key	Protected (no runtime changes)	Default (by omission)	Extra details
dbmaintenance_history_days_to_keep		14	Number of days of historical agent data to keep in the database
dbmaintenance_loop_interval_sec		86400	Seconds between the DB Maintenance module iterations
log_level	X	info	Logging level. Possible values: debug, info, warn, error, critical
mailer_loop_interval_sec		(None)	Seconds between the Mailer module iterations
mailer_smtp_account		(None)	SMTP account name
mailer_smtp_pwd		(None)	SMTP account password
mailer_smtp_recipients		(None)	Comma-separated list of e-mail recipients of the reports
mailer_smtp_report_type		vuln_only	Granularity of the report to be sent. Possible values: all, vuln_only, exploit_vuln_only
mailer_smtp_server		(None)	SMTP server address
mailer_smtp_tls_port		(None)	SMTP TLS port
mongo_collection	X	(None)	MongoDB collection
mongo_db	X	(None)	MongoDB DB name
mongo_host	X	(None)	MongoDB host address
mongo_port	X	(None)	MongoDB port
mongo_pwd	X	(None)	MongoDB user password
mongo_user	X	(None)	MongoDB user name
platform_loop_interval_sec		300	Seconds between the Platform module iterations

Table B1 - SIAAS Server configuration reference

Important note: The sample configuration file shipped with the SIAAS Server software package does not necessarily have the default values set.

Appendix C – CLI Environment Variables Reference

Variable	Default (by omission)	Extra details
SIAAS_API_URI	https://127.0.0.1/api	SIAAS API URI
SIAAS_API_USER	siaas	SIAAS API user name
SIAAS_API_PWD	siaas	SIAAS API user password
SIAAS_API_SSL_IGNORE_VERIFY	false	Ignore SIAAS API endpoint's SSL CA verification
SIAAS_API_SSL_CA_BUNDLE	(None)	SSL CA bundle of the SIAAS API endpoint's certificate
SIAAS_API_TIMEOUT	60	Timeout of the SIAAS API calls
SIAAS_DEBUG_LOGS	false	Enabled debug logging
SIAAS_OUTPUT_COLORS	false	Colorful JSON output
SIAAS_OUTPUT_INDENT_SPACES	4	Number of indentation spaces in the JSON output

Table C1 - SIAAS CLI environment variables reference

Important note: The sample environment file shipped with the SIAAS CLI software package does not necessarily have the default values set.

Appendix D – Local API Guide (Agent)

SIAAS LOCAL API (AGENT) GUIDE

How to read this guide:

- **<method>** is a method for the route
-- **<key>** is an optional parameter accepted under the current method
() describes the body of a method, or the value of a parameter key
[] is the list of possible returns for a method
is a description of the method/parameter

Base URL: http://<SIAAS_Agent_IP>:5001
Template: http://<SIAAS_Agent_IP>:5001/<Local_API_Route>?key1=value1&key2=value2

List of Local API Routes:

/api
- GET [200] # shows API information

/api/siaas-agent
- GET [200, 500] # shows the data collected by the agent
-- module (string: platform, neighborhood, portscanner, config, *; default: *) # filters specific modules (accepts a comma-separated list)

cURL examples:

GET: curl -i -X GET http://192.168.1.11:5001/siaas-agent?module=platform,portscanner

Technical Note D1 - SIAAS Agent local API guide

Full web guide for the local API is available at: http://<SIAAS_Agent_IP>:5001/docs.

Appendix E – API Guide (Server)

SIAAS API (SERVER) GUIDE

How to read this guide:

- <method> is a method for the route
-- <key> is an optional parameter accepted under the current method
() describes the body of a method, or the value of a parameter key
[] is the list of possible returns for a method
is a description of the method/parameter

Base URL: https://<SIAAS_Server_IP>
Template: https://<SIAAS_Server_IP>/<API_Route>?key1=value1&key2=value2

List of API Routes:

/api
- GET [200] # shows API information

/api/siaas-server
- GET [200, 500] # shows the contents of the local server DBs
-- module (string: platform, config, *; default: *) # filters specific modules (accepts a comma-separated list)

/api/siaas-server/configs
- GET [200, 500] # shows the published configurations for the server
- POST [200, 500] (header: Content-Type: application/json; body: JSON dict of configs) # posts a dictionary of configurations for the server
- DELETE [200, 500] # deletes all server configurations

/api/siaas-server/agents
- GET [200, 500] # shows a summary of known agents (nickname, description, IP, last ping) (nickname and description strings for each agent are read from the published configuration keys “nickname” and “description”, for each agent’s uid)
-- sort (string: date, agent; default: date) # sorts the agents by last ping or uid

/api/siaas-server/agents/data

- GET [200, 500] # shows the data uploaded by all agents
-- module (string: platform, config, *; default: *) # filters specific modules (accepts a comma-separated list)

/api/siaas-server/agents/data/<siaas_agent_uid>
- GET [200, 500] # shows the data uploaded by a specific agent uid (accepts multiple uids, comma-separated)
-- module (string: platform, neighborhood, portscanner, config, *; default: *) # filters specific modules (accepts a comma-separated list)
- POST [200, 500] (header: Content-Type: application/json; body: JSON dict of agent-module data) # posts a dictionary of agent data from an agent's uid
- DELETE [200, 500] # deletes agent data from the server's DB from an agent's uid (accepts multiple uids, comma-separated)
-- days (int: 0-99999; default: 365) # number of days to keep

/api/siaas-server/agents/configs
- GET [200, 500] # shows the configurations published for all agents
-- merge_broadcast (int: 0, 1; default: 0) # merges the broadcasted configurations with the configuration of each agent's uid (broadcast address: ffffffff-ffff-ffff-ffff-ffffffffffff)

/api/siaas-server/agents/configs/<agent_uid>
- GET [200, 500] # shows the configurations published for a specific agent uid (accepts multiple uids, comma-separated)
-- merge_broadcast (int: 0, 1; default: 0) # merges the broadcasted configuration in the configuration of each agent's uid
- POST [200, 500] (header: Content-Type: application/json) (body: JSON dict of configs) # posts a dictionary of configurations for an agent's uid (accepts multiple uids, comma-separated)
- DELETE [200, 500] # deletes published configurations for an agent's uid (accepts multiple uids, comma-separated)

/api/siaas-server/agents/history
- GET [200, 500] # shows historical data for all agents
-- days (int: 0-99999; default: 15) # maximum number of days to show
-- hide (int: 0, 1; default: 0) # empty keys till module-key level
-- limit (int: 0-99999; default: 100) # maximum number of records to show
-- module (string: platform, neighborhood, portscanner, config, *; default: *) filters specific modules (accepts a comma-separated list)
-- older (int: 0, 1; default: 0) # shows older records first
-- sort (string: date, agent; default: date) # sort by most recent or agent uid

/api/siaas-server/agents/history/<agent_uid>
- GET [200, 500] # shows historical data for a specific agent uid (accepts multiple uids, comma-separated)
-- days (int: 0-99999; default: 15) # maximum number of days to show
-- hide (int: 0, 1; default: 0) # empty keys till module-key level
-- limit (int: 0-99999; default: 100) # maximum number of records to show
-- module (string: platform, neighborhood, portscanner, config, *; default: *) filters specific modules (accepts a comma-separated list)
-- older (int: 0, 1; default: 0) # shows older records first

```
-- sort (string: date, agent; default: date) # sort by most recent or agent uid
```

cURL examples:

GET: curl -i -X GET https://192.168.1.100/api/siaas-server/agents/history?sort=agent\&hide=1\&days=5 --user siaas:siaas --insecure

POST: curl -i -X POST https://192.168.1.100/api/siaas-server/agents/configs/10000000dbb5bbc1 -H "Content-Type: application/json" -d '{"disable_wifi_auto_discovery": "false","manual_hosts": "sapo.pt,google.pt"}' --user siaas:siaas --insecure

DELETE: curl -i -X DELETE https://192.168.1.100/api/siaas-server/agents/data/0924aa8b-6dc9-4fec-9716-d1601fc8b6c6?days=3 --user siaas:siaas --insecure

Technical Note E1 - SIAAS Server API guide

Full web guide for the API is available at: https://<SIAAS_Server_IP>/api/docs.

Appendix F – Local Tests

LOCAL TESTS

Lab:

JP-OLD (SERVER + AGENT) - 2013
Sony Vaio E11 (SVE1113M1EW)
Processor: 1.75GHz Dual Core
Memory: 8GB
WLAN

RPI4 (AGENT) - Q2 2019
Raspberry Pi 4 Model B Rev 1.2
Processor: 1.5GHz Quad Core
Memory: 2GB
Ethernet / WLAN

```
pi@raspberrypi:~ $ cat /proc/cpuinfo
(...)
Hardware        : BCM2835
Revision        : b03112
Serial          : 10000000dbb5bbc1
Model           : Raspberry Pi 4 Model B Rev 1.2
```

RPI1 (AGENT) - Q3 2012
Raspberry Pi Model B Rev 1
Processor: 700MHz Single Core
Memory: 256MB
Ethernet

```
pi@raspberrypi:~ $ cat /proc/cpuinfo
(...)
Hardware        : BCM2835
Revision        : 0003
Serial          : 000000007ab1b3b3
Model           : Raspberry Pi Model B Rev 1
```

SIAAS (VM) (SERVER + AGENT + CLI)
Processor: 1.8GHz Quad Core

Memory: 8GB
Ethernet / WLAN

Set up November 2022.

Nmap commands run by python3-nmap:

OS detection:

```
#TCP
nmap.nmap_os_detection(target, args="-%s -sV -Pn --host-timeout %s" % (ipv, timeout))
/usr/bin/nmap -v -oX - -O -4 -sV -Pn --host-timeout 600 sapo.pt

#UDP
nmap.scan_top_ports(scanned_ip, args="-%s -sU --top-ports 10 -Pn --host-timeout %s" % (ipv, timeout))
/usr/bin/nmap -v -oX - --top-ports 10 -4 -sU --top-ports 10 -Pn --host-timeout 600 sapo.pt
```

Vulnerability scan:

```
nmap.nmap_version_detection(target, args="-%s -p%s:%s --script %s -Pn --host-timeout %s" % (ipv,
prot_flag, port, nmap_script, timeout))
/usr/bin/nmap -v -oX - -sV -4 -pT:443 --script vuln -Pn --host-timeout 600 sapo.pt
/usr/bin/nmap -v -oX - -sV -4 -pU:445 --script vuln -Pn --host-timeout 600 sapo.pt

# -O: Enable OS detection
# --top-ports: most famous ports for this service (UDP is very slow when scanning ports, so we just
scan the 10 most famous ones)
# -sV: Probe open ports to determine service/version info
# -sU: UDP scan
# -Pn: Treat all hosts as online -- skip host discovery
```

Reliability tests:

Uptime agent @ RPI4:

```
pi@raspberrypi:~ $ journalctl -u siaas-agent | grep -Ei 'Started|Stopped'
(...)
Jan 26 15:42:29 raspberrypi systemd[1]: Started SIAAS Agent.
Feb 10 14:33:14 raspberrypi systemd[1]: Stopped SIAAS Agent. # 15 days
(...)
Feb 26 16:17:33 raspberrypi systemd[1]: Started SIAAS Agent.
Mar 15 14:22:31 raspberrypi systemd[1]: Stopped SIAAS Agent. # 17 days
```

Uptime server @ JP-OLD:

```
jpseara@JP-OLD:~$ journalctl -u siaas-server | grep -Ei 'Started|Stopped'
```

```
(...)
jan 24 19:30:56 JP-OLD systemd[1]: Started SIAAS Server.
fev 03 14:04:16 JP-OLD systemd[1]: Stopped SIAAS Server. # 10 days
(...)
fev 24 02:24:30 JP-OLD systemd[1]: Started SIAAS Server.
mar 10 00:05:17 JP-OLD systemd[1]: Stopped SIAAS Server. # 14 days

RPI4 and RPI1 customized configs for stress test:

enable_internal_api: true
log_level: debug
manual_hosts: google.com,sapo.pt
neighborhood_loop_interval_sec: 5
nmap_scripts: vuln,vulscan,discovery
platform_loop_interval_sec: 5
portscanner_loop_interval_sec: 5

RPI1 modified configs for stress test (check MULTIPROC/MULTITHREAD ANALYSIS):

enable_internal_api: true
log_level: debug
manual_hosts: google.com,sapo.pt
neighborhood_loop_interval_sec: 5
nmap_scripts: vuln
platform_loop_interval_sec: 5
portscanner_loop_interval_sec: 5
portscanner_max_parallel_workers: 1

JP-OLD customized configs for stress test:

disable_wifi_auto_discovery: true # problematic Sony Vaio WLAN card...
log_level: debug
manual_hosts: google.com,sapo.pt
neighborhood_loop_interval_sec: 5
nmap_scripts: vuln,vulscan,discovery
platform_loop_interval_sec: 5
portscanner_loop_interval_sec: 5
```

Multiproc/Multithread analysis:

```
Scripts: vuln,vulscan,discovery

$ cat /var/log/siaas-agent/siaas-agent.log | grep portscanner.py | grep -Ei 'running|sleeping'

SIAAS (VM)

4 cores:
```

```
(target - num_scanned_ports/num_valid_scripts/total_num_vulnerabilities/total_num_exploits/time_taken_sec_with_multiprocs | time_taken_sec_with_multithreads)

192.4.5.5 - 10/0/0/0/320 | 321
192.168.122.1 - 3/1/21/0/75 | 72
192.168.123.3 - 6/1/105/16/414 | 499
2a00:1450:4003:80f::500e - 0 | 1
aisense-qas.aitecservdevenv.local - 6/1/105/16/205 | 287
cgsa-dev.aitecservdevenv.local - 6/1/105/16/305 | 507
focal62 - 6/1/105/16/281 | 431
google.com - 12/1/0/0/828 | 834
sapo.pt - 12/1/0/0/269 | 277

# Multiproc

2023-03-26 20:28:49.610 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 20:42:38.092 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds before next loop ...
^ 14 min with multiprocs

2023-03-26 23:37:25.729 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 23:53:00.522 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds before next loop ...
^ 16 min with multiprocs

2023-03-27 00:30:38.661 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-27 00:44:21.775 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds before next loop ...
^ 14 min with multiprocs

# Multithread

2023-03-26 20:53:22.372 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 21:07:16.836 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds before next loop ...
^ 14 min with multithreads

2023-03-26 23:55:42.821 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-27 00:12:06.655 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds before next loop ...
^ 17 min with multithreads

2023-03-27 00:13:06.681 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-27 00:26:54.418 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds before next loop ...
^ 13 min with multithreads

1 core:
```

```
(target                                                                                                                -
num_scanned_ports/num_valid_scripts/total_num_vulnerabilities/total_num_exploits/time_taken_sec_with
_multiprocs | time_taken_sec_with_multithreads)

192.4.5.5 - 10/0/0/0/319 | 321
192.168.122.1 - 3/1/21/0/76 | 77
192.168.123.3 - 6/1/105/16/291 | 288
2a00:1450:4003:80f::500e - 0 | 0
aisense-qas.aitecservdevenv.local - 6/1/105/16/191 | 219
cgsa-dev.aitecservdevenv.local - 6/1/105/16/241 | 390
focal62 - 6/1/105/16/258 | 250
google.com - 12/1/0/0/837 | 822
sapo.pt - 12/1/0/0/403 | 333

# Multiproc

2023-03-26 21:16:35.895 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 22:00:15.958 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds
before next loop ...
^ 44 min with multiprocs

2023-03-26 22:25:07.402 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 23:07:10.023 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds
before next loop ...
^ 42 min with multiprocs

# Multithread

2023-03-26 22:07:57.615 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 22:21:40.491 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds
before next loop ...
^ 14 min with multithreads

2023-03-26 23:16:51.694 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 23:30:33.042 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 60 seconds
before next loop ...
^ 14 min with multithreads

***

JP-OLD (2 cores):

# Multiproc

2023-03-26 20:26:37.423 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 21:11:35.644 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds
before next loop ...
^ 45 min
```

```
2023-03-26 22:49:22.629 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 23:39:44.293 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 50 min

2023-03-27 09:09:23.603 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-27 09:52:51.857 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 43 min

# Multithread

2023-03-27 14:14:54.371 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-27 14:49:05.507 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 35 min

2023-03-27 23:39:46.666 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-28 00:12:47.279 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 33 min

2023-03-29 21:28:20.767 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-29 22:06:56.988 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 38 min

***

RPI4 (4 cores):

# Multiproc

2023-03-26 21:07:50.915 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 21:41:47.588 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 34 min

2023-03-26 22:52:43.420 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-26 23:32:21.365 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 40 min

2023-03-27 09:11:46.586 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-27 09:45:00.430 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 34 min
```

```
# Multithread

2023-03-27 15:44:48.612 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-27 16:18:19.489 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 34 min

2023-03-27 23:45:25.392 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-28 00:23:52.301 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 38 min

2023-03-29 22:03:27.798 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-29 22:39:06.373 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 36 min

***

RPI1 (1 core):

# Multiproc

2023-03-26 18:53:47.131 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
(Still stuck on discovery script at 10:23 next day (observed in two consecutive days))

# Multithread (auto=5)

2023-03-27 10:44:05.352 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
Mar 27 10:58:00 raspberrypi kernel: [251173.162310] Out of memory: Killed process 12525 (nmap) total-vm:49200kB, anon-rss:24476kB, file-rss:0kB, shmem-rss:0kB, UID:0 pgtables:56kB oom_score_adj:0

# Multithread (3)

2023-03-28 02:07:05.455 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
Mar 28 02:30:10 raspberrypi kernel: [307127.505793] Out of memory: Killed process 3296 (nmap) total-vm:49272kB, anon-rss:20656kB, file-rss:0kB, shmem-rss:0kB, UID:0 pgtables:56kB oom_score_adj:0

# Multithread (1)

2023-03-28 02:43:58.575 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
(Still stuck on discovery script at 11:15 next day)

# Multithread (3, just vuln+vulscan)

2023-03-28 11:19:17.734 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
Mar 28 12:28:56 raspberrypi kernel: [343049.420355] Out of memory: Killed process 21494 (nmap) total-vm:93968kB, anon-rss:38344kB, file-rss:0kB, shmem-rss:0kB, UID:0 pgtables:98kB oom_score_adj:0
```

```
# Multithread (3, just vuln)

2023-03-28 12:42:47.256 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-28 14:10:23.094 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 1 hr 28 min

2023-03-28 14:10:28.110 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-28 15:32:47.064 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 1 hr 22 min

(Works, but under very intense load (Munin fails, SSH fails):
top - 14:55:47 up 4 days,  1:44,  1 user,  load average: 10.47, 7.24, 7.68)

# Multithread (2, just vuln)

2023-03-28 17:00:39.340 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-28 17:56:55.046 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 56 min

2023-03-28 17:57:00.054 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-28 18:53:34.851 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 56 min

(Less load but still high:
top - 17:22:22 up 4 days,  4:11,  1 user,  load average: 5.36, 4.20, 4.50)

# Multithread (1, just vuln)

2023-03-28 19:55:54.036 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-28 21:27:22.010 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 1 hr 32 min

2023-03-29 14:25:23.225 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-29 15:47:50.464 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 1 hr 22 min

2023-03-29 21:16:40.642 DEBUG siaas_portscanner.py [Process-3|MainThread] Loop running ...
2023-03-29 22:31:19.090 DEBUG siaas_portscanner.py [Process-3|MainThread] Sleeping for 5 seconds before next loop ...
^ 1 hr 15 min

(Even less load (much more acceptable now):
top - 21:34:52 up 4 days,  8:23,  1 user,  load average: 2.08, 2.14, 2.03)
```

```
***

Peak memory usage observations (per script category):

(at RPI4)

$ watch sudo ps -C "nmap" -o vsz,rss,cmd

106548 98220 /usr/bin/nmap -v -oX - google.com -sV -4 -pT:443 --script vuln -Pn --host-timeout 600
^ 98MB

141912 133492 /usr/bin/nmap -v -oX - google.com -sV -4 -pT:443 --script discovery -Pn --host-timeout
600
^ 133MB

135132 127136 /usr/bin/nmap -v -oX - google.com -sV -4 -pT:443 --script vulscan -Pn --host-timeout 600
^ 127MB
```

Accuracy tests:

```
ubuntu@target:~$ uname -a
Linux target 5.4.0-131-generic #147-Ubuntu SMP Fri Oct 14 17:07:22 UTC 2022 x86_64 x86_64 x86_64
GNU/Linux

ubuntu@target:~$ sudo dpkg -l | grep -Eiw 'apache2|openssh'
ii  apache2                              2.4.41-4ubuntu3.12                                  amd64
Apache HTTP Server
ii  openssh-server                       1:8.2p1-4ubuntu0.3                                  amd64
secure shell (SSH) server, for secure access from remote machines

https://ubuntu.com/security/cves?q=&package=openssh
https://ubuntu.com/security/CVE-2021-28041 # Fixed in 20.04, no issue in 22.04

https://ubuntu.com/security/cves?q=&package=apache2
https://ubuntu.com/security/CVE-2020-9490 # Fixed in 20.04, no issue in 22.04

ubuntu@siaas:~$ siaas-cli vuln-report -h 192.168.122.51 -t vuln_only | grep -Ei 'CVE-2021-28041|CVE-
2020-9490'
'CVE-2021-28041': ['4.6', 'https://vulners.com/cve/CVE-2021-28041'],
'CVE-2020-9490': ['5.0', 'https://vulners.com/cve/CVE-2020-9490'],

^ OK!

ubuntu@siaas:~$ siaas-cli vuln-report -h 192.168.122.51 -t all | grep -i apache
'product': 'Apache httpd 2.4.41 (Ubuntu)',
'http-server-header': {'raw_lines': ['Apache/2.4.41 (Ubuntu)']},
```

```
^ OK!

ubuntu@siaas:~$ siaas-cli vuln-report -h 192.168.122.51 -t all | grep -i openssh
'product': 'OpenSSH 8.2p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)',

^ OK!

ubuntu@siaas:~$ siaas-cli vuln-report -h 192.168.122.51 -t all | grep -i linux
{'0924aa8b-6dc9-4fec-9716-d1601fc8b6c6':    {'portscanner':    {'192.168.122.51':    {'system_info':
{'hostname': 'target.local', 'mac_address': '52:54:00:59:A5:04', 'nic_vendor': 'QEMU virtual NIC',
'os_name': 'Linux 3.2 - 4.9', 'os_family': 'Linux', 'os_gen': '4.X', 'os_vendor': 'Linux', 'os_type':
'general purpose', 'scanned_ip': '192.168.122.51'},

^ Linux OK! Kernel info NOK...

ubuntu@target:~$ sudo apt update
ubuntu@target:~$ sudo apt-get dist-upgrade -y
ubuntu@target:~$ sudo do-release-upgrade

ubuntu@target:~$ uname -a
Linux target 5.15.0-40-generic #43-Ubuntu SMP Wed Jun 15 12:54:21 UTC 2022 x86_64 x86_64 x86_64
GNU/Linux

ubuntu@target:~$ sudo dpkg -l | grep -Eiw 'apache2|openssh'
ii  apache2                              2.4.52-1ubuntu4.4                       amd64        Apache
HTTP Server
ii  openssh-server                       1:8.9p1-3ubuntu0.1                      amd64        secure
shell (SSH) server, for secure access from remote machines

ubuntu@siaas:~$ siaas-cli vuln-report -h 192.168.122.51 -t vuln_only | grep -Ei 'CVE-2021-28041|CVE-
2020-9490'
(No output.)

^ OK!

ubuntu@siaas:~$ siaas-cli vuln-report -h 192.168.122.51 -t all | grep -i apache
'product': 'Apache httpd 2.4.52 (Ubuntu)',
'http-server-header': {'raw_lines': ['Apache/2.4.52 (Ubuntu)']},

^ OK!

ubuntu@siaas:~$ siaas-cli vuln-report -h 192.168.122.51 -t all | grep -i openssh
'product': 'OpenSSH 8.9p1 Ubuntu 3 (Ubuntu Linux; protocol 2.0)'

^ OK!

ubuntu@siaas:/opt/siaas-cli$ siaas-cli vuln-report -h 192.168.122.51 -t all | grep -i linux
{'0924aa8b-6dc9-4fec-9716-d1601fc8b6c6':    {'portscanner':    {'192.168.122.51':    {'system_info':
{'hostname': 'target.local', 'mac_address': '52:54:00:59:A5:04', 'nic_vendor': 'QEMU virtual NIC',
```

```
'os_name': 'Linux 3.2 - 4.9', 'os_family': 'Linux', 'os_gen': '4.X', 'os_vendor': 'Linux', 'os_type':
'general purpose', 'scanned_ip': '192.168.122.51'},

^ Linux OK! Kernel info NOK...
```

Security tests:

```
CLI (SIAAS (VM)) -> SERVER (JP-OLD)

-L: follow redirects
-X: method (GET)
--location-trusted: use the same authentication when redirected
--user: user/password auth
--insecure: don't check the CA validity
--cacert: local CA bundle to verify the endpoint against
-v: verbose (confirm redirection to https)

ubuntu@siaas:/opt/siaas-cli$ curl -LX GET https://192.168.1.69/api --user siaas:siaas
curl: (60) SSL certificate problem: self signed certificate
More details here: https://curl.haxx.se/docs/sslcerts.html
curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.

ubuntu@siaas:/opt/siaas-cli$ curl -LX GET https://192.168.1.69/api --user siaas:siaas --insecure
{"output":{"name":"Intelligent System for Automation of Security Audits
(SIAAS)","module":"Server","api":"v1","author":"João Pedro Seara","supervisor":"Carlos
Serrão"},"status":"success","total_entries":5,"time":"2023-04-04T18:43:25Z"}

ubuntu@siaas:/opt/siaas-cli$ curl -LX GET https://192.168.1.69/api --user siaas:siaas --cacert
./ssl/jp-old.crt
curl: (60) SSL: no alternative certificate subject name matches target host name '192.168.1.69'
More details here: https://curl.haxx.se/docs/sslcerts.html
curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it. To learn more about this situation and
how to fix it, please visit the web page mentioned above.

ubuntu@siaas:/opt/siaas-cli$ curl -LX GET https://jp-old/api --user baduser:badpassword --cacert
./ssl/jp-old.crt
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Unauthorized</title>
</head><body>
<h1>Unauthorized</h1>
<p>This server could not verify that you
are authorized to access the document
requested.  Either you supplied the wrong
credentials (e.g., bad password), or your
```

```
browser doesn't understand how to supply
the credentials required.</p>
<hr>
<address>Apache/2.4.41 (Ubuntu) Server at jp-old Port 443</address>
</body></html>

ubuntu@siaas:/opt/siaas-cli$ curl -L -X GET https://jp-old/api --user siaas:siaas --cacert ./ssl/jp-
old.crt
{"output":{"name":"Intelligent System for Automation of Security Audits
(SIAAS)","module":"Server","api":"v1","author":"João Pedro Seara","supervisor":"Carlos
Serrão"},"status":"success","total_entries":5,"time":"2023-04-04T18:45:37Z"}

ubuntu@siaas:/opt/siaas-cli$ curl -L -X GET http://jp-old/api --user siaas:siaas --location-trusted -
-cacert ./ssl/jp-old.crt -v
Note: Unnecessary use of -X or --request, GET is already inferred.
*   Trying 192.168.1.69:80...
* TCP_NODELAY set
* Connected to jp-old (192.168.1.69) port 80 (#0)
* Server auth using Basic with user 'siaas'
> GET /api HTTP/1.1
> Host: jp-old
> Authorization: Basic c2lhYXM6c2lhYXM=
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 301 Moved Permanently
< Date: Tue, 04 Apr 2023 18:45:45 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Location: https://jp-old/api
< Content-Length: 298
< Content-Type: text/html; charset=iso-8859-1
<
* Ignoring the response-body
* Connection #0 to host jp-old left intact
* Issue another request to this URL: 'https://jp-old/api'
*   Trying 192.168.1.69:443...
* TCP_NODELAY set
* Connected to jp-old (192.168.1.69) port 443 (#1)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: ./ssl/jp-old.crt
  CApath: /etc/ssl/certs
* TLSv1.3 (OUT), TLS handshake, Client hello (1):
* TLSv1.3 (IN), TLS handshake, Server hello (2):
* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):
* TLSv1.3 (IN), TLS handshake, Certificate (11):
* TLSv1.3 (IN), TLS handshake, CERT verify (15):
```

```
* TLSv1.3 (IN), TLS handshake, Finished (20):
* TLSv1.3 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.3 (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use http/1.1
* Server certificate:
*  subject: C=PT; ST=Lisbon; L=Lisbon; O=ISCTE; OU=METI; CN=jp-old
*  start date: Oct 30 23:52:16 2022 GMT
*  expire date: Oct 27 23:52:16 2032 GMT
*  subjectAltName: host "jp-old" matched cert's "jp-old"
*  issuer: C=PT; ST=Lisbon; L=Lisbon; O=ISCTE; OU=METI; CN=jp-old
*  SSL certificate verify ok.
* Server auth using Basic with user 'siaas'
> GET /api HTTP/1.1
> Host: jp-old
> Authorization: Basic c2lhYXM6c2lhYXM=
> User-Agent: curl/7.68.0
> Accept: */*
>
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):
* old SSL session ID is stale, removing
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Tue, 04 Apr 2023 18:45:45 GMT
< Server: waitress
< Content-Length: 238
< Content-Type: application/json
<
{"output":{"name":"Intelligent System for Automation of Security Audits
(SIAAS)","module":"Server","api":"v1","author":"João Pedro Seara","supervisor":"Carlos
Serrão"},"status":"success","total_entries":5,"time":"2023-04-04T18:45:45Z"}
* Connection #1 to host jp-old left intact
```

Max BSON size limitation:

```
2023-03-14 04:33:04.090 ERROR siaas_aux.py [MainProcess|waitress-3] Can't insert data in the DB server:
BSON document too large (71625954 bytes) - the connected server supports BSON document sizes up to
16793598 bytes.

Ports 21, 22, 80, 111, 2049, 8080

- FTP
- NFS
- HTTP
- SSH

discovery,vuln,vulscan:
```

```
~ 140.8 KB (~119 hosts)

ubuntu@siaas:/opt/siaas-agent/var$ du -b
140842  .
ubuntu@siaas:/opt/siaas-agent/var$ ls -lrta
total 160
-rw-r--r--  1 root   root      577 Mar 15 02:24 config_local.db
-rw-r--r--  1 root   root       36 Mar 15 02:24 uid
drwxr-xr-x  2 root   root     4096 Mar 15 02:24 .
drwxrwxr-x 11 ubuntu ubuntu   4096 Mar 15 02:25 ..
-rw-r--r--  1 root   root   133767 Mar 15 02:36 portscanner.db
-rw-r--r--  1 root   root      577 Mar 15 02:42 config.db
-rw-r--r--  1 root   root      265 Mar 15 02:43 neighborhood.db
-rw-r--r--  1 root   root     1524 Mar 15 02:44 platform.db

discovery,vuln:

~ 75 KB

ubuntu@siaas:/opt/siaas-agent/var$ du -b
75039   . (~223 hosts)
ubuntu@siaas:/opt/siaas-agent/var$ ls -lrta
total 96
drwxrwxr-x 11 ubuntu ubuntu  4096 Mar 15 01:55 ..
-rw-r--r--  1 root   root     569 Mar 15 02:10 config_local.db
-rw-r--r--  1 root   root      36 Mar 15 02:10 uid
drwxr-xr-x  2 root   root    4096 Mar 15 02:10 .
-rw-r--r--  1 root   root   67898 Mar 15 02:21 portscanner.db
-rw-r--r--  1 root   root     569 Mar 15 02:22 config.db
-rw-r--r--  1 root   root     265 Mar 15 02:23 neighborhood.db
-rw-r--r--  1 root   root    1606 Mar 15 02:23 platform.db

vuln:

~ 26.2 KB

ubuntu@siaas:/opt/siaas-agent/var$ du -b
26234   . (~640 hosts)
ubuntu@siaas:/opt/siaas-agent/var$ ls -lrta
total 48
drwxrwxr-x 11 ubuntu ubuntu  4096 Mar 15 01:55 ..
-rw-r--r--  1 root   root     559 Mar 15 02:02 config_local.db
-rw-r--r--  1 root   root      36 Mar 15 02:02 uid
drwxr-xr-x  2 root   root    4096 Mar 15 02:02 .
-rw-r--r--  1 root   root   19198 Mar 15 02:07 portscanner.db
-rw-r--r--  1 root   root     265 Mar 15 02:08 neighborhood.db
-rw-r--r--  1 root   root     559 Mar 15 02:08 config.db
-rw-r--r--  1 root   root    1521 Mar 15 02:09 platform.db
```

```
vulscan: 65803 (~65.8K) - 46.7%
discovery: 48805 (~48.8K) - 34.7%
vuln: 26234 (~26.2K) - 18.6%
```

Technical Note F1 - Raw notes from the local tests

Appendix G – Survey Responses from Testers

SIAAS SURVEY FOR TESTERS

Methodology:

A survey was replied via video call or text survey: https://forms.microsoft.com/e/t47we3fNCf

Participants:

Trilio Data: Video call (with Matt Golden, Customer Success Engineer), on April 21, 2023, at 17:00 UTC
Altice Portugal: Video call (with Ricardo Ramalho, Head of Cybersecurity Behaviour Analytics and Automation), on April 24, 2023, at 18:30 UTC
VTXRM – Software Factory: Video call (with Jorge Teixeira, IT Team Lead), on May 5, 2023, at 21:00 UTC
David Negreira: Video call, on May 12, 2023, at 18:30 UTC

Replies:

Technical aspects (10 sentences, rate 1-5):

1. It is easy to install and configure the components of this vulnerability scanner.
   Trilio -> 4
   Altice -> 5
   VTXRM -> 5
   David -> 5

2. The CLI usage and help menus are user-friendly.
   Trilio -> 5
   Altice -> 5
   VTXRM -> 4
   David -> 4

3. The agent correctly discovers neighborhood hosts.
   Trilio -> 5
   Altice -> 5
   VTXRM -> 4
   David -> 5

4. The vulnerability scan results are accurate.
   Trilio -> 5
   Altice -> 4
   VTXRM -> 5
   David -> 3

5. The vulnerability scans do not impact network or targeted hosts negatively.
   Trilio -> 3
   Altice -> 5
   VTXRM -> 5
   David -> 5

6. The e-mail reports are reliable.
   Trilio -> 4
   Altice -> 3
   VTXRM -> 3
   David -> 3

7. The existing documentation is useful.
   Trilio -> 4
   Altice -> 5
   VTXRM -> 4
   David -> 5

8. It is easy to customize the scanner for one's specific environment.
   Trilio -> 4
   Altice -> 5
   VTXRM -> 3
   David -> 5

9. It is easy to integrate the API with existing tools and workflows.
   Trilio -> 3
   Altice -> 5
   VTXRM -> 3
   David -> 3

10. You are satisfied with the performance and functionality of the vulnerability scanner.
    Trilio -> 4
    Altice -> 5
    VTXRM -> 5
    David -> 4

Organizational aspects (5 sentences, rate 1-5):

11. This vulnerability scanner is a valid approach to an organization which has no access to paid commercial solutions or cybersecurity human expertise.
    Trilio -> 5
    Altice -> 5
    VTXRM -> 5
    David -> 4

12. When compared with other paid tools like Nessus, considering human expertise and configuration time, this scanner saves time/money/resources while keeping an acceptable performance.
    Trilio -> 4
    Altice -> 4
    VTXRM -> 3
    David -> 5

13. The vulnerability reports generated by the scanner help with the process of addressing, prioritizing, and remediating vulnerabilities.
    Trilio -> 4
    Altice -> 4
    VTXRM -> 5
    David -> 5

14. This vulnerability scanner helps an organization complying with relevant security standards and regulations.
    Trilio -> 4
    Altice -> 4
    VTXRM -> 4
    David -> 5

15. This vulnerability scanner increases an organization's posture and ability to detect and mitigate vulnerabilities.
    Trilio -> 5
    Altice -> 5
    VTXRM -> 5
    David -> 5

Overall aspects (2 sentences, rate 1-5):

16. This vulnerability scanner helps improving the security auditing process of an organization.
    Trilio -> 4
    Altice -> 5
    VTXRM -> 5
    David -> 5

17. You would recommend this vulnerability scanner to other organizations.
    Trilio -> 4
    Altice -> 4
    VTXRM -> 5
    David -> 4

Open-ended questions (3 questions, text response):

18. What were the main technical or organizational-related shortcomings found (technical issues, false positives, or others)?
    Trilio -> Technical problems regarding the DB file limits (16MB BSON limit in MongoDB), which disappeared after disabling one of the scripts. This was the only technical problem observed.
    Altice -> (Empty.)
    VTXRM -> As it relies on ARP to discover hosts, the scanner might be sensitive to ARP spoofing from a malicious actor in the network. An easier installation process for the test scenario would be nice, as it requires some Linux expertise as it is (suggestion: containerization).

David -> The scanner does not detect vulnerabilities already fixed by server patching. Some confusion in understanding what the agent IDs meant (clarified with the author).

19. What features would you like to see added or improved in future versions of the scanner?
    Trilio -> GUI interface, with possibility of configuring the agents, and a way one can go through a list of targets and select them from a drop-down list to filter the scanning outputs. Also, this GUI should have the ability to mark false positives in order to remove them from the reports.
    Altice -> Docker release, especially for the server component. This would also be quite useful for testing.
    VTXRM -> Graphical interface. More details in the CVE descriptions (current information is too technical). These details could be presented in the graphical interface itself.
    David -> Detecting backported vulnerabilities by patching. Identifying the distribution of the neighborhood hosts (example: Ubuntu) and other aspects related to it, per example, if an OS version was already discontinued.

20. Any other comments?
    Trilio -> Popular tools have most of their features hidden behind their paid version, which makes this an interesting project overall. Regarding statement 5: no negative impact was observed, score of 3 just because the tests didn't drill down on this specific point; statement 9: score of 3 because it was not tested.
    Altice -> It is a very interesting and useful project. Has the potential to become a product. Regarding statement 6: score of 3 because it was not tested; statement 13: prioritizing is a different matter from addressing or remediating, and even the most expensive and complex solutions have difficulties in helping with this; statement 16: the tool helps but does not necessarily guarantee by itself that, for an audit, all requirements are passed; statement 17: the alternatives were not thoroughly explored.
    VTXRM -> Interesting project. Increased interest in cybersecurity and what CVEs are. Will use it for own projects. Regarding statements 6, 8, 9, and 12: score of 3 because those statements could not be tested or assessed.
    David -> It is a good product. Possibility of being widely adopted. It is easy to install, configure, and set up. Regarding statement 4: score of 3 because it does not detect vulnerabilities solved by patching; statement 6: score of 3 because it was not tested; statement 9: JSON output looks easy to parse and integrate, but score of 3 because this was not tested.

Technical Note G1 - Survey results from the testers of the artifact

Last page intentionally left blank