EAI Endorsed Transactions
on Scalable Information Systems Research Article **EAI.EU**

Intelligent System for Automation of Security Audits (SIAAS)

J. P. Seara [1,*], C. Serrão [1]

[1] DCTI, ISCTE – Instituto Universitário de Lisboa, 1649-026 Lisboa, Portugal

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)

Received on 12 July 2023, accepted on 19 October 2023, published on 20 October 2023

doi: 10.4108/eetsis.3564

1. Introduction

Cybersecurity-related occurrences are often in the news. The number of global attacks keeps increasing in recent years [1].

Back in 2020, a study by the US Cybersecurity and Infrastructure Security Agency (CISA) [2] 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 USD.

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 USD, annually, by 2025 [3]. This will mean, roughly, 28.8 billion USD per day, or 333 thousand USD per second!

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. [4][5]

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" [6].

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 [7] 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 [8], 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.

[*] Corresponding author. Email: joao_pedro_seara@iscte-iul.pt

Still regarding AI, it should be mentioned that, at the moment this document is being written, AI tools like *ChatGPT* 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 [9].

It's also important to note that prioritizing what security flaws need to be addressed is an important aspect of the automation process [10].

Finally, it's 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 [11].

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's important that the solutions provide outputs that properly prioritize what needs to be addressed.

Given the problems and needs described above, the following research question was formulated, as the starting point of this work: *"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?"*

The developed work answers this question by adding further value to the existing efforts from academies and businesses to automate security audits, as well as making the results available to the community. Its major contribution is offering a free, comprehensive, and "plug and play" (PnP) vulnerability scanning solution – from network discovery till e-mail reporting of the findings – which runs on low-cost hardware, for anyone to use, even with no previous cybersecurity expertise. Such a solution, as the next section will show, does not exist at current time. This solution is named *Intelligent System for Automation of Security Audits* (also referred to in its Portuguese acronym: SIAAS), and is divided in 3 main modules, which will be detailed during the next sections.

Finally, it should be mentioned that this work integrates in emergent *DevSecOps* paradigms, like the stack of security-related technologies called *Security Orchestration, Automation, and Response* (SOAR) [12].

This document started by providing a short introduction to the problem and will now present the methodology and results of the research made on related works, to justify the pertinency of the presented work. The following section then describes design choices and the implementation process of the resulting artifact. The next section will present the results of the validation tests, which were made both locally and with the help of external testers. The last section will detail the conclusions obtained from the conducted work.

2. Related Work

When it comes to the methodology used to search for related work, it was decided to use PRISMA (*Preferred Reporting Items for Systematic Reviews and Meta-Analysis*) [13] as guidance to systematize the research. This methodology was boiled down to the following main steps: selection of databases; criteria and filters for searches; removal of duplicates; removal of non-related titles; removal of non-related abstracts. Figure 1 describes this sequence of steps.

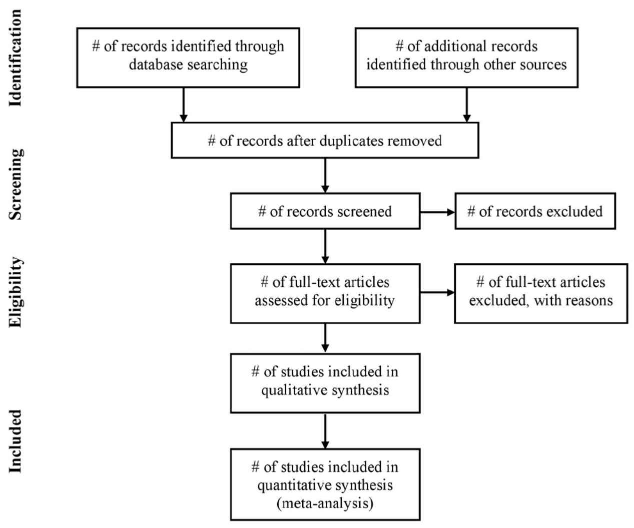

Identification
Screening
Eligibility
Included
# of records identified through database searching
# of additional records identified through other sources
# of records after duplicates removed
# of records screened
# of records excluded
# of full-text articles assessed for eligibility
# of full-text articles excluded, with reasons
# of studies included in qualitative synthesis
# of studies included in quantitative synthesis (meta-analysis)

Figure 1. The PRISMA flow [13]

The search for academic works was made in the *B-On* portal (a Portuguese online knowledge library which provides access to scientific releases), IEEE, *Google Scholar*, and *Google*. A search in the Portuguese language was done as well, to eliminate any language bias, but no relevant documents in this language were found. Some examples of the queries used for this research are "vulnerability+scan+solutions", and "security+audit+solutions".

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's important to note that these numbers serve as an approximation, and for the reader to have an idea how they were obtained. Not all these references are used in this document.

The following paragraphs describes the findings obtained.

Authors of [14] 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 [15], 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 exploitation module to detect this type of vulnerability in websites.

In [16] 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 [17] describes a vulnerability scanner, focused on web servers and vulnerabilities, namely SQL injection and XSS. It performs vulnerability assessment by leveraging *pocsuite3*, which is an open-source vulnerability scanning framework for web services. It has a web interface for the end user.

Authors of [18] created a vulnerability scanner focused also on web servers. Contains a web crawler and tests SQL injection, XSS, and directory traversal vulnerabilities. No relevant details are shared about the user interface. Supports reporting, but not continuous auditing.

Authors of [19] 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 to handle communication between the components. It focuses only on finding vulnerabilities related to web applications, including XSS and XXE injection. No relevant details are shared about the user interface.

The work in [20] 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.

Artifact developed in [21] is named “SecuBat”, and is a very similar scanner to [18]. 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 [22] 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 works is prepared for automatic network discovery. In some of them, problems with memory exhaustion were reported.

From the analysis conducted on all the tools described before, 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, or on specific services.
- Most of the solutions require inbound network permissions to access the target hosts, if run from outside of their network, to the exception of the solutions that allow local agents to be installed. Some of them also require that target host credentials are known.
- Most of the solutions don’t automatically discover infrastructure information. 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. 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.

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. The work presented in this document differs from these projects, as it condenses a set of characteristics that solve the current shortcomings in a single open-source artifact that requires little to no specialized staff to enable its installation and operation, as the next section will detail in terms of architecture and implementation.

A final note to mention [23]. Its authors consider that there is a failure in current studies/works in analyzing "the inner relationships between different vulnerabilities". Their work consisted in using graph-driven intelligence to predict co-exploits between multiple vulnerabilities (CVE). Such intelligence can potentially be applied on top of the results of this work, to make vulnerability remediation more efficient by system administrators (as part of the suggestion “AI/ML”, in the “Conclusions and Future Work” section).

3. System Design and Implementation

This section starts by giving an overview of the preliminary design choices that were made, for the developed artifact to attain its goal. Then, it provides details on the architecture of the artifact, tooling choices, and the implementation process.

3.1. Design Choices

The main goal of this work was to develop a vulnerability scanning and reporting system that is free to use and does not require any cybersecurity expertise to operate. To achieve this, 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's 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 agent hardware is mostly focused on the *Raspberry Pi* board, which uses the ARM architecture, but the solution should be flexible enough to even run on a regular computer/server with a x86 architecture.
- *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), Python (programming language), Nmap (open-source port scanning tool), *MongoDB* (database).
- *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.
- *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 [24], 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).

The developed artifact not only builds from the current state-of-the-art but, as mentioned before, also solves some of its current shortcomings.

3.1. Implementation

SIAAS Agent and *SIAAS Server* are the two core software pieces of the developed work. Figure 2 depicts a diagram of the SIAAS architecture, from a high-level perspective.

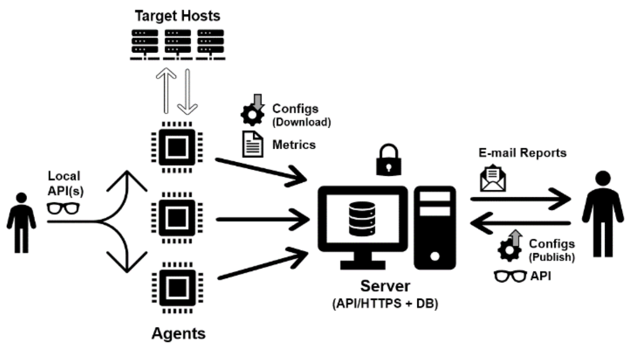


Figure 2. SIAAS high-level architecture

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 are Raspberry Pi boards – 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. This is done, firstly, by checking their local ARP tables. Then, by doing an ARP scan of the local IPv4 networks (IPv6 networks are too big to be scanned this way). 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. This scan is divided in two parts: first is identifying the running OS in the target and running services in all the open UDP and TCP ports, and the second part is running a vulnerability scan using Nmap scripts against those ports, which – as per the default configured script ("vuln") – returns a list of found CVEs. 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, and the agent will scan the new network with no new configurations being needed.

The collected metrics are merged into a single JSON object and then, periodically, sent to the server. Figure 3 shows a graphical representation of the structure of this JSON object.

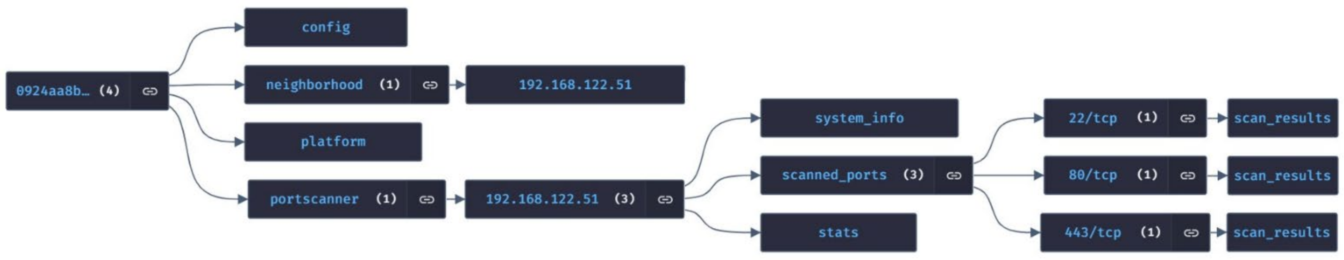


Figure 3. Graphical representation of the JSON's object schema

In this figure, it can be observed that the JSON object contains information about the main modules of the agent: the first module shows the currently active configuration of the agent (see next paragraph), the second module shows the IPs of the hosts discovered in the agent's neighborhood (in this example: 192.168.122.51), the third module contains information about the platform of the agent itself (like CPU and memory usage) and, finally, the fourth module contains system information, service list, vulnerability scanning results, and stats, of the targeted hosts.

After this information is sent to the server (upload phase), the agents then check if any new configurations are published to them (download phase). 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*, a web browser like *Firefox* (which has a JSON parser that cascades the API data in a human-readable way, as shown in Figure 6), 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's 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's 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, and 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 paragraphs will drill down on specific choices that were made regarding the software platforms and tools chosen for the deployment of the artifact.

Nmap [25] was chosen as the port/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). Nmap supports NSE scripts, using the Lua language, allowing Nmap to perform enumeration, vulnerability scanning, and penetration testing (or, simply, *pentesting*).

The systematic research also found some commercial alternatives to Nmap, like the already mentioned OpenVAS and Nessus, but also *Nexpose*, *Scanner-VS*, *Cybot*, *Xspider*, and *Qualys*, as per [26][27][28]. Other tools were also found in academic works, like *Faraday*. None of these were considered for use because they either have one of the issues detailed at the end of Section 2, or they are paid and therefore do not fit in the paradigm of this work.

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 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. 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 [29]), 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.

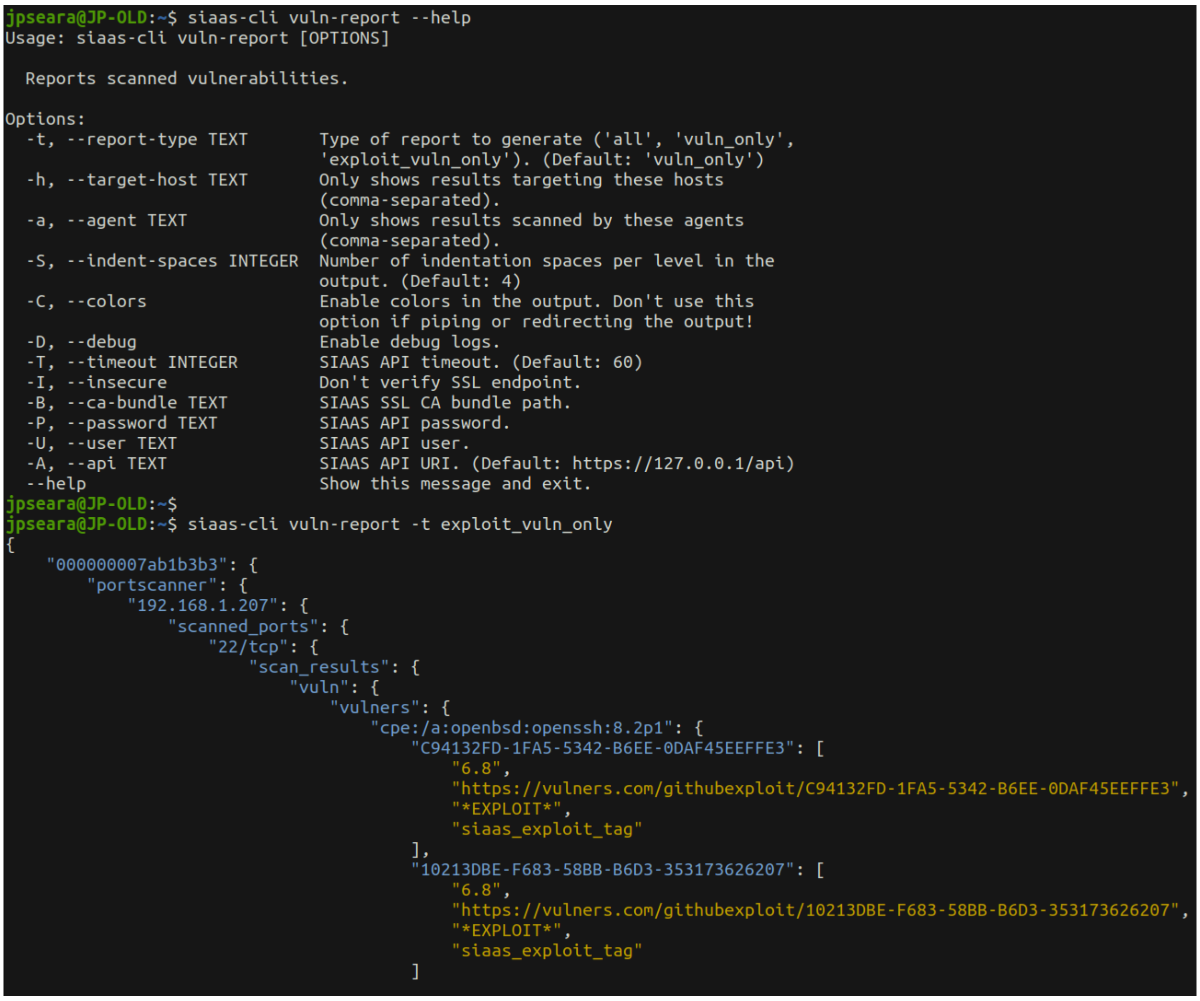
Scapy is a packet manipulation Python library. It is useful to perform ARP-related operations, like scanning the neighborhood of a network adapter. This is especially useful to implement an automated scanning of neighborhood hosts. Project in [30] implements such a network scanner to find hosts in the 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*, *Flask*, and *FastAPI*. 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. It is responsible for authenticating 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. 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 and MongoDB. These solutions are fundamentally different, as MySQL is a structured DB, whereas MongoDB is a document-oriented DB (usually known as a NoSQL DB). 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. The flexibility and ease of integration with Python data structures made MongoDB become the choice.

A CLI was developed as part of this work (SIAAS CLI), 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, which was used. Figure 4 shows two examples of CLI outputs (the complete list of available commands, as well as a vulnerability report showing only exploits).



```
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 4. SIAAS CLI showing help and vulnerability report outputs

As seen in the mentioned figure, the output of the CLI commands is in colored JSON format, so it allows the operator to use a JSON parser like *jq* 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. All of the development was done in *Ubuntu* 20.04 "Focal", but it was also installed and tested in Ubuntu 22.04 "Jammy" (Server edition, or ARM edition in the agents), *Debian* 11 "Bullseye" (main edition, or ARM edition in the agents), and *Raspberry Pi OS* (previously known as "Raspbian") 11 "Bullseye" (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.

4. Validation

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 performance (reliability and 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.

Regarding local tests, 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/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 protocol/service level.

While selecting the external testers, it was attempted to diversify their 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, one organization in the telecommunications business with specialization in cybersecurity, one organization in the intersection of the IT and financial businesses, and an individual IT freelancer in the open-source field (more details about the testers in the acknowledgments at the end of this paper).

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, whose links are available at the end of this document). 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: 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. The surveys were answered via video call, to give the testers the chance to elaborate on each point, if needed, as well as to let the conversation flow to different perspectives and opinions from them on current shortcomings and possible improvements. Testers were also given the option to answer via a written form, if they preferred, but none opted for this method.

3.1. Local Tests

The hardware used for the local tests used consisted of a Sony *Vaio* E11 laptop (2013) with the hostname "JP-OLD" running as server/agent, and two Raspberry Pi running as agents ("RPI4" being a 4th generation Model B (2019), and "RPI1" being a 1st generation Model B (2012)). A VM named "SIAAS", running in an external hypervisor, was also used for development and testing purposes.

Specifications of this environment:

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

Figure 5 shows a photo of the setup of the local laboratory.

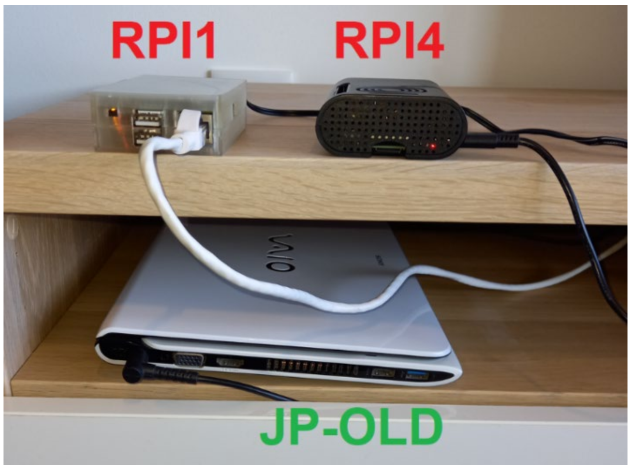
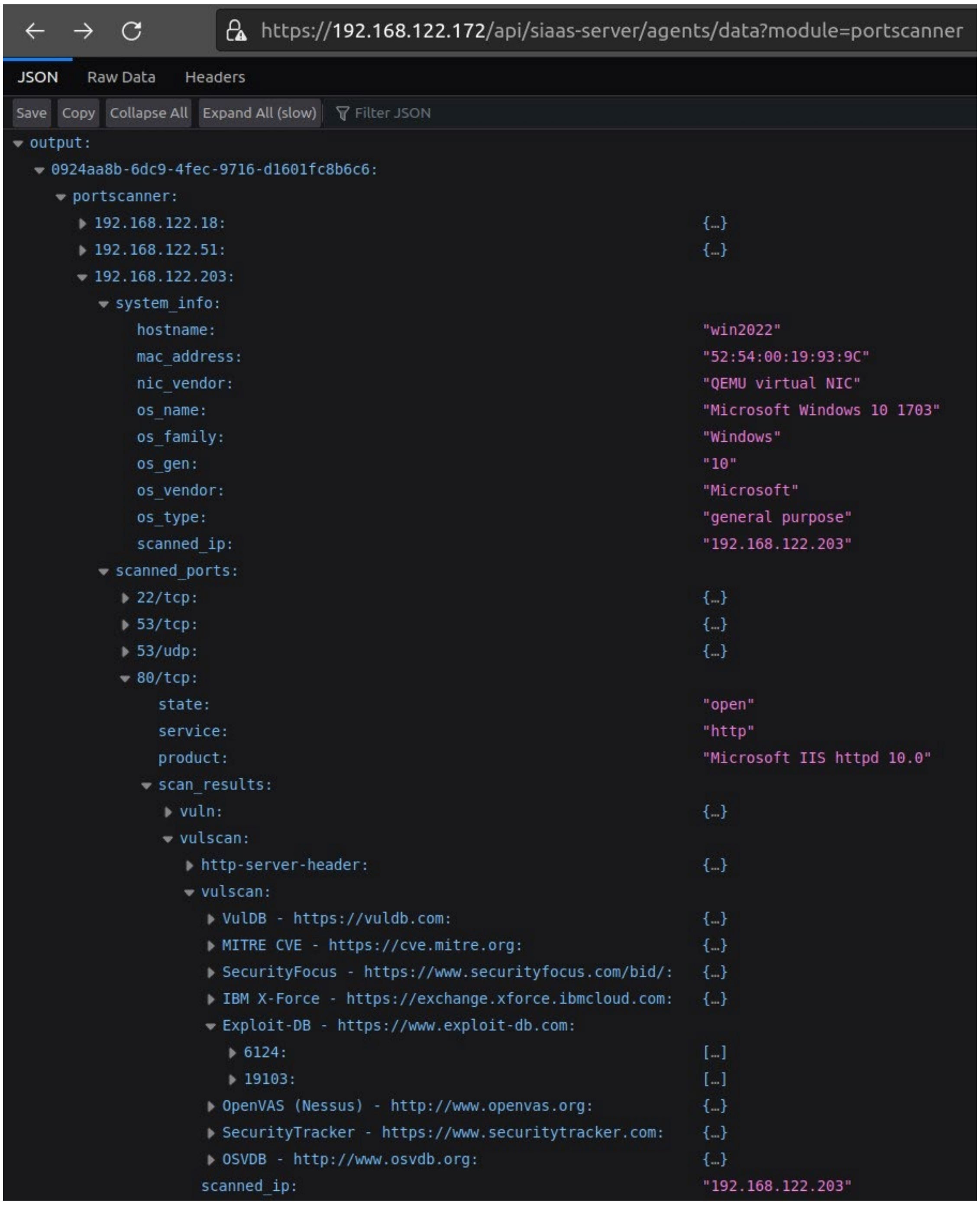


Figure 5. Local laboratory

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.

In terms of vulnerability scanning accuracy, the developed artifact successfully detected 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.

Figure 6 shows the SIAAS API output from a scan done against a Microsoft Windows Server 2022 VM, using the script "vulscan" (not active by default), showing a list of found vulnerabilities from different databases.


https://192.168.122.172/api/siaas-server/agents/data?module=portscanner
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All (slow)
Filter JSON

```
▾ output:
  ▾ 0924aa8b-6dc9-4fec-9716-d1601fc8b6c6:
    ▾ portscanner:
      ▸ 192.168.122.18:                                              {…}
      ▸ 192.168.122.51:                                              {…}
      ▾ 192.168.122.203:
        ▾ system_info:
            hostname:                                                "win2022"
            mac_address:                                             "52:54:00:19:93:9C"
            nic_vendor:                                              "QEMU virtual NIC"
            os_name:                                                 "Microsoft Windows 10 1703"
            os_family:                                               "Windows"
            os_gen:                                                  "10"
            os_vendor:                                               "Microsoft"
            os_type:                                                 "general purpose"
            scanned_ip:                                              "192.168.122.203"
        ▾ scanned_ports:
          ▸ 22/tcp:                                                  {…}
          ▸ 53/tcp:                                                  {…}
          ▸ 53/udp:                                                  {…}
          ▾ 80/tcp:
              state:                                                 "open"
              service:                                               "http"
              product:                                               "Microsoft IIS httpd 10.0"
            ▾ scan_results:
              ▸ vuln:                                                {…}
              ▾ vulscan:
                ▸ http-server-header:                                {…}
                ▾ vulscan:
                  ▸ VulDB - https://vuldb.com:                       {…}
                  ▸ MITRE CVE - https://cve.mitre.org:               {…}
                  ▸ SecurityFocus - https://www.securityfocus.com/bid/: {…}
                  ▸ IBM X-Force - https://exchange.xforce.ibmcloud.com: {…}
                  ▾ Exploit-DB - https://www.exploit-db.com:
                    ▸ 6124:                                          […]
                    ▸ 19103:                                         […]
                  ▸ OpenVAS (Nessus) - http://www.openvas.org:       {…}
                  ▸ SecurityTracker - https://www.securitytracker.com: {…}
                  ▸ OSVDB - http://www.osvdb.org:                    {…}
                  scanned_ip:                                        "192.168.122.203"
```

Figure 6. Scanning results for a Windows Server VM during local tests

There's an inherent limitation to the type of scanner this work implements. 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. This can eventually be solved by developing a module to be installed in the target hosts (this is suggested as future work, in the "Conclusions and Future Work" section).

Finally, an extra test was done regarding API security. User authentication is implemented using only simple HTTP authentication, so the login credentials are hard coded in the host (even though they can be changed in the SIAAS Server installation script). Therefore, only minimalistic tests were done, to validate that the HTTPS protocol and simple HTTP authentication were correctly implemented. These tests passed successfully.

The raw outputs from all local tests can be consulted online (check the data availability links at the "Acknowledgements" section, at the end of this document).

4.2 User Tests

As mentioned above, a total of 4 organizations and 1 freelancer tested and replied to the testers' survey. Figure 7 shows a graphical distribution of the agreeableness replies obtained.

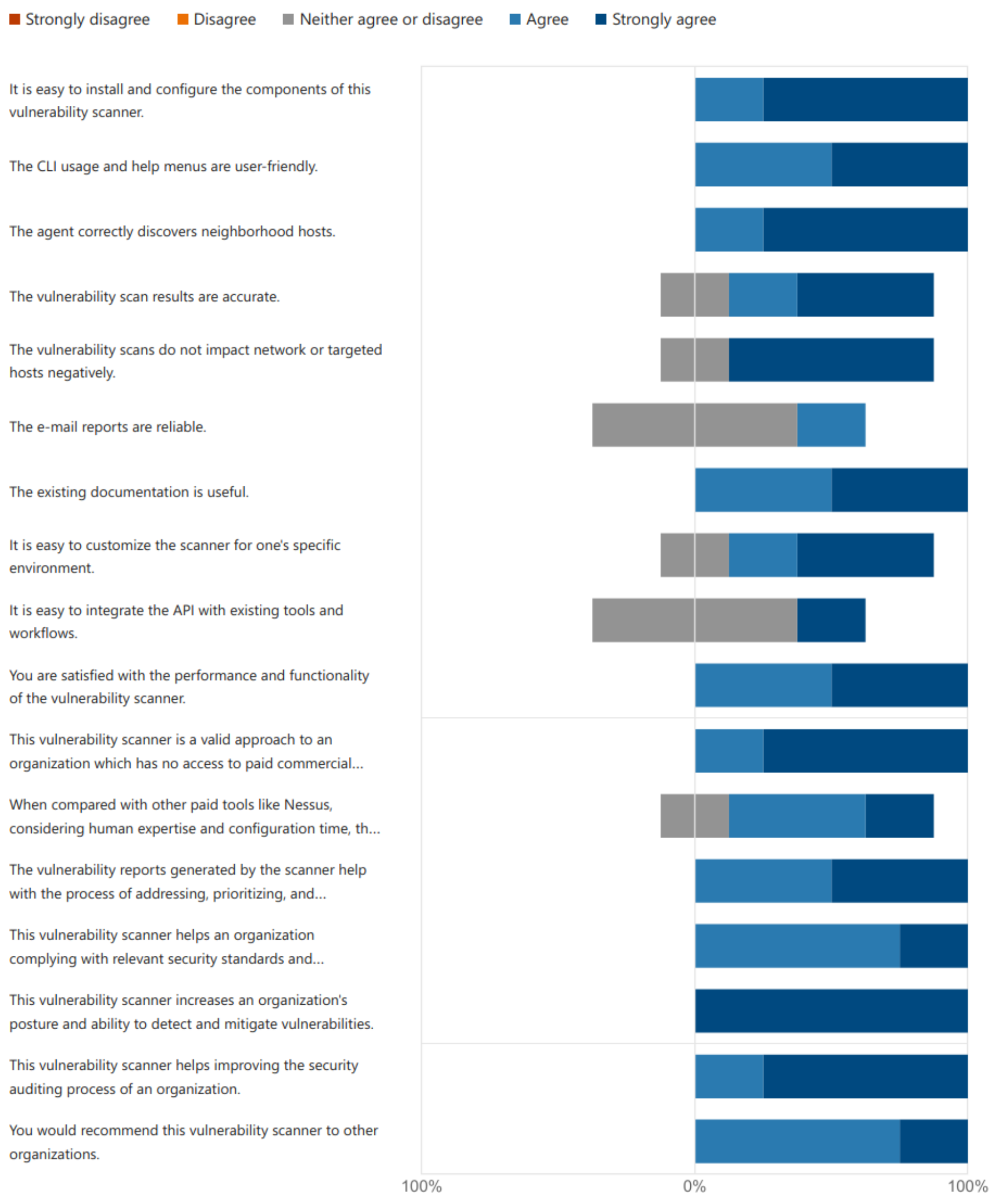


Figure 7. Graphical distribution of the agreeableness replies given by the testers

As seen in this figure, the large majority of the replies are in the "agree" and "strongly agree" area, which denotes a strong satisfaction overall from all the testers.

Regarding open-ended questions, testers were asked for problems found and suggestions for improvement. The following problems were reported: the inability of the vulnerability scanner to detect backported vulnerabilities (as explained in the last sub-section), a limitation in the object size that the agent can upload to the server (this is rarely observed, but a known documented issue nevertheless), and the possibility of the agents being sensitive to ARP spoofing (as they rely on the ARP protocol to discover hosts in the neighborhood). When it comes to suggestions of future work, these were focused mostly on creating a web interface, having further details from the target's OS distribution versions and lifecycle, and containerization (all of these are incorporated as suggestions of future work, in the next section).

The final open-ended question was about final comments. All of the testers gave very positive feedback in terms of their perception of added value to the organizations, and some even considered adopting this artifact for their own toolkit.

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.

5. Conclusions and Future Work

The main goal of this work was to 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.

The authors consider that this goal was completely accomplished. The main aspects to consider about the developed artifact are the following:

It provides port scanning performed by Nmap, which was chosen for its simplicity, portability, efficiency, possibility of using varied scripts that can do tasks that go beyond checking vulnerability DBs (like penetration testing or vendor-specific service verification), and easy integration with Python.

It automatically discovers hosts in the neighborhood, which are fed automatically to the port scanner by default, allowing a system administrator with no cybersecurity-related skills to operate it with minimal configuration effort. Advanced configuration is still supported (like restricting which scripts to run, or hosts/ports to scan), if needed.

It operates in an agent-server architecture, giving the operator the flexibility to distribute load and/or connect agents directly to internal LANs.

It is fully modular. An agent can work independently from a server, or both server and agent can run together in an AIO environment. The fully featured API can be consumed by web interfaces, mobile clients, or even AI systems, implemented by the community.

It provides vulnerability reporting reports via e-mail using the well-known CSV format. The granularity of these reports can be configured, allowing system administrators to focus on only exploitable vulnerabilities that need to be fixed on priority.

Testing the artifact 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, it effectively and accurately detects hosts in the network, their OSes and services, and vulnerabilities. Nmap scripts allow 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 adds value in automating the security audit processes of organizations, even if no other software options are available, as it was easy to install and operate, didn't impact current infrastructure, and gave reliable results.

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

The authors provide some suggestions for future work, based on own observations and feedback got from testers:

Related to UI/UX, a web frontend that interfaces with the server's API. Some features that this frontend could implement: AAA/SaaS multi-tenant cloud, advanced graphical reporting, stateful metadata information (like marking vulnerabilities as resolved or already seen), improved e-mail reports, advanced scheduling of the agent's and server's modules' runs. 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.

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 (like identifying backported vulnerability fixes for older versions of services, or obtaining more detailed information about the OS distribution, version, and lifecycle);

And, finally, the possibility of adding extra tools/modules to the agent (suggestion: using specialized tools like *Metasploit* to perform extensive pentesting on specific services/applications, by probing found exploits and then running post-exploitable code).

Acknowledgements.
GitHub repository containing the source code, installation scripts, and "readme" files, of the SIAAS Agent artifact: https://github.com/jpseara/siaas-agent (accessed 6 Jun 2023). GitHub repository containing the source code, installation scripts, and "readme" files, of the SIAAS Server artifact: https://github.com/jpseara/siaas-server (accessed 6 Jun 2023). GitHub repository containing the source code, installation scripts, and "readme" files, of the SIAAS CLI artifact: https://github.com/jpseara/siaas-cli (accessed 6 Jun 2023). GitHub repository containing complete configuration and API

references, outputs of local tests, sample API outputs, and user survey form and response transcriptions: https://github.com/jpseara/siaas-research (accessed 6 Jun 2023). Original recordings of the user surveys can be requested by contacting the corresponding author(s) via e-mail. Finally, we would like to thank all the testers of this work: Matt Golden, Customer Success Engineer at Trilio Data (USA); Ricardo Ramalho, Head of Cybersecurity Behaviour and Automation at Altice Portugal; Jorge Teixeira, IT Team Lead at VTXRM – Software Factory (Portugal); and David Negreira (IT Freelancer, active member of the Ubuntu community).

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

[3] 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)

[4] 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

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

[6] 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)

[7] 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

[8] 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

[9] 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)

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

[11] 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)

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

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

[21] 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

[22] 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

[23] J. Yin, M. Tang, J. Cao, M. You, H. Wang, and M. Alazab, “Knowledge-Driven Cybersecurity Intelligence: Software Vulnerability Coexploitation Behavior Discovery”, IEEE Transactions on Industrial Informatics, 2023, vol. 19, no. 4, pp. 5593-5601, doi: 10.1109/TII.2022.3192027

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

[25] 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)

[26] 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

[27] 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

[28] 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

[29] 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)

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