Department of Electronics, Telecommunications and Informatics of the University of Aveiro

André Morais
Eduardo Santos
Gonçalo Matos
Isadora Loredo
Margarida Martins
Pedro Bastos

2021

secureUAll

Project report in Informathics in Computer Science of the University of Aveiro,
held by André Morais, Eduardo Santos, Gonçalo Matos,
Isadora Loredo, Margarida Martins and Pedro Bastos

Under the guidance of
João Paulo Barraca
Assistant Professor of the Department of Electronics, Telecommunications and Informatics of the University of Aveiro
André Zúquete
Assistant Professor of the Department of Electronics, Telecommunications and Informatics of the University of Aveiro
Ricardo Martins
Responsible for the Cybersecurity Office of the University of Aveiro
Vitor Cunha
Student of the PhD Program in Computer Science

1. Keywords

Cibersecurity, vulnerabilities, Network Scrapping, Awareness culture, Security breach

2. Abstract

A security test is a method of evaluating the security of a computer system or network by methodically validating and verifying the effectiveness of application security controls. A web application security test focuses only on evaluating the security of a web application. The process involves an active analysis of the application for weaknesses, technical flaws, or vulnerabilities. A threat is anything (a malicious external attacker, an internal user, a system instability, etc) that may harm the assets owned by an application (resources of value, such as the data in a database or in the file system) by exploiting a vulnerability. The security posture of an application can be characterized from the perspective of the effect, such as number of vulnerabilities and the risk rating of the vulnerabilities. By developing a system that integrates the assessment of a host and giving back a report on it's web application security, that points out the flaws and weaknesses, is how we approached the problem of having the University of Aveiro great amount of public domains serving as a potential security breach. Through this system we expect to provide it's owners and administrators a proposal for mitigation or a technical solution to prevent and reduce the risk.

3. Acknowledgements

We would like to show our very great appreciation to the following people who were essential for this project:

- Our peer groups for filling the usability form and giving us useful feedback:
  - Group 9, CrowdWire - Massive Online Meetings;
  - Group 6, Electronic Exam Lab;
  - Group 2, Fire Lab;
  - Group 11, 5G Mobility;
- Lara Fahla, a Design student at University of Aveiro, for the creation of our logo;
- Our advisors for helping us throughout the project's development.

4. Abbreviations

ASN	Autonomous System Numbers
CD	Continuous Delivery
CI	Continuous Integration
CPE	Common Platform Enumeration
CSIRT	Computer Security Incident Response Team
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DNS	Domanin Name System
FIRST	Forum of Incident Response and Security Teams
HTTPS	Hypertext Transfer Protocol Secure
IdP	Identity Provider
IEETA	Institute of Electronics and Informatics Engineering of Aveiro
IT	Telecommunications Institute
IP	Internet Protocol
JSON	JavaScript Object Notation
ORM	Object-relational mapping
OS	Operative System
OWASP	Open Web Application Security Project
SSL	Secure Socket Layer
UA	University of Aveiro
UI	User Interface
UX	User Experience
XML	Extensible Markup Language
ZAP	Zed Attack Proxy

Contents

List of Figures

5. Introduction

With cyber attacks increasing over the time, organizations with a high level exposure to the outside world are starting to put a greater effort in making their systems more secure.

The University of Aveiro domain, `ua.pt`, has more than 1500 public domains [16]. If not tracked, each one of these can be a potential security breach. It is therefore important to frequently search for vulnerabilities as well as encouraging the owners of the hosts to develop more secure software while raising awareness for this problem.

In the field of web applications, security includes all tasks that introduce a secure software development life cycle, and its final goal is to improve security practices and, through that, to find, fix and preferably prevent security issues within applications. It encompasses the whole application life cycle from requirements analysis, design, implementation, verification as well as maintenance.

Web application security is a branch of information security that deals specifically with security of websites, web applications and web services. At a high level, web application security draws on the principles of application security but applies them specifically to internet and web systems, and guided by the Open Web Application Security Project (OWASP) Foundation, free and open resources are developed and provided to improve the security of software.

This work focuses exactly on the development of a system, consisting of an enumeration system, an analysis system and a dashboard that allows to assess the current vulnerability risk of a host. Security issues will be detected and pointed out to be resolved using current state-of-the-art tools, with the work being focused on building the platform itself and implementing the functional logic to perform periodicals scraping.

Vulnerabilities can be classified according to different criteria. The most commonly used vulnerability severity metric is the Common Vulnerability Scoring System (CVSS), a standard maintained by the Forum of Incident Response and Security Teams (FIRST). Following the criteria of the OWASP Foundation, when reporting security test data, the best practice is to include the following information:

- a categorization of each vulnerability by type;
- the security threat that each issue is exposed to;
- the root cause of each security issue, such as the bug or flaw;
- each testing technique used to find the issues;
- the remediation, or countermeasure, for each vulnerability; and
- the severity rating of each vulnerability (e.g., high, medium, low, or CVSS score).

By describing what the security threat is, it will be possible to understand if and why the mitigation control is ineffective in mitigating the threat, and reporting the root cause of the issue can help pinpoint what needs to be fixed.

Taking this into consideration, we will start, on *Chapter 6*, by discussing the current state-of-the art of our project's concept as well as making a comparison between products. On *Chapter 7*, we will present a detailed conceptual modeling, describing how we intend to solve the problem proposed and who will use our system. Following, on *Chapter 8*, we bring in a technical level the product developed, it's structure, features and technologies, discussing the architecture, domain model, protocols, testing

and usability. In *Chapter 9* we talk about the results obtained in terms of the final product achieved alongside the data collected by real user testing done by our peers with their hosts in UA's network, and presenting the results of an usability report. At last, *Chapter 10* draws a conclusion for this report and project, with some final remarks on what we learned and the overall experience during this cycle of work developing this project, as well as some improvements that could be done in the future.

6. State of the art

In this chapter we will explore the current state of the art regarding applications similar to ours. These research helped us in several parts of our application concept, development. Analysing the application's functionalities was specially relevant in the concept modeling part of the project. It also was particularly important on the development of the front-end, because none of our group members were previously familiarized with a typical administration dashboard applied to our context, seeing multiple dashboard examples helped us in designing a more usable and intuitive front-end where the information is compacted but readable.

6.1. Shodan

Figure 1: Shodan Logo

Shodan [26] is a "Search Engine for the Internet of Things". It gathers information about all types of devices connected to the internet. It's main purpose is to "allow organizations to monitor their network, assess 3rd-party cyber risk, gather market intelligence and understand the global Internet landscape in real-time ". Shodan is used by more than 3 million registered users including universities and big companies. It's products can be accessed throughout a paid account. We will focus on the **Shodan Monitor** [25] which is Shodan's product most similar to our application.

Shodan Monitor consists on a web application that allows its users to register a network or a single host, through a range of IP addresses or a single one respectively.

After the registering process Shodan will scan the host(s).The Shodan scanning consists of analyzing the banners that appear on the hosts response to its requests. With these banners Shodan can retrieve information about the software used by the host as well as their version.

In figure 2 bellow, we can see an example of a dashboard. The most common open ports appear, as well as the ones that typically are not open (which might be a security flaw). With the version of the software found Shodan will search if there are CVEs associated and present them in the dashboard. Hosts with found vulnerabilities will be highlighted in red, where a stronger red means that higher risk vulnerabilities were found.

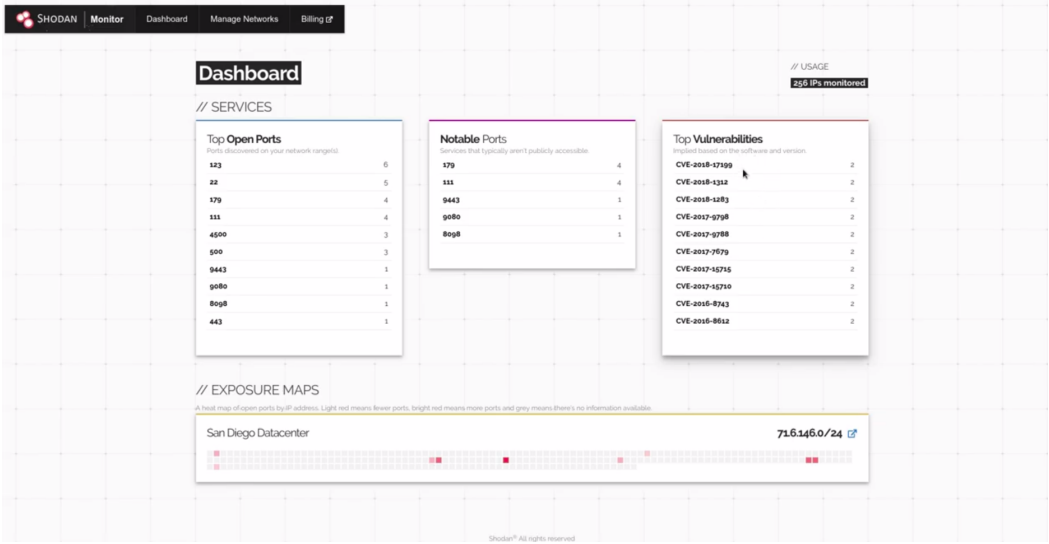


Figure 2: Shodan Monitor Dashboard

Shodan will periodically scan the IPs provided, but it also allows its users to request an immediate scanning. When configuring an network/host, users can set multiple notification triggers that will send a notification via email every time a host matches a condition. Examples of Shodan notification triggers:

- ssl_expired: Sends a notification when the service's SSL certificate expired
- open_database: Sends a notification if the a database from a service does not require authentication.
- uncommon: Sends a notification if a service would normally not be available publicly.

6.2. BinaryEdge

Figure 3: BinaryEdge Logo

BinaryEdge [3] is a cibersecurity company created in 2015 co-founded by a Portuguese engineer after its participation in the *SAPO Codebits* hackathon.

BinaryEdge has a web application and an API in which its users can search for hosts using multiple parameters such as: IP, port, ASN, country, etc. Figure 4 shows an exemple of the results obtained when searching a network.

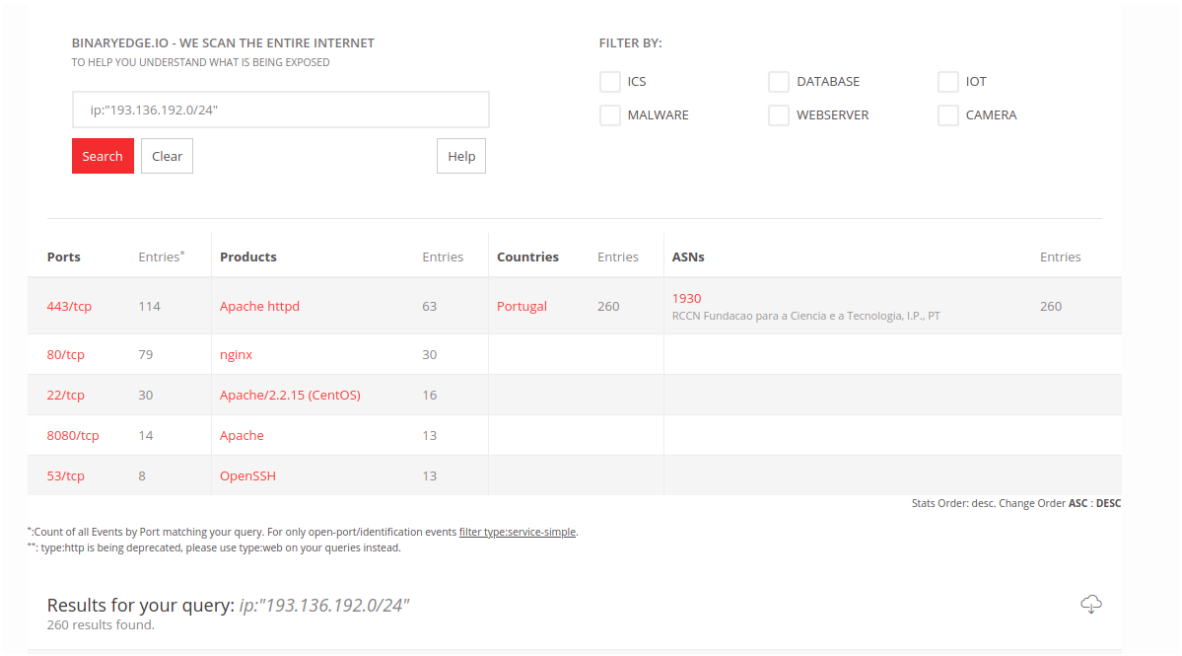


Figure 4: BinaryEdge Search Result of a network

First an aggregation of the results is shown: number of ports, software, localization of the hosts and ASNs. Secondly, a list of all IPs in the network is displayed. By clicking in one of them we are redirected to the host page (figure 5).

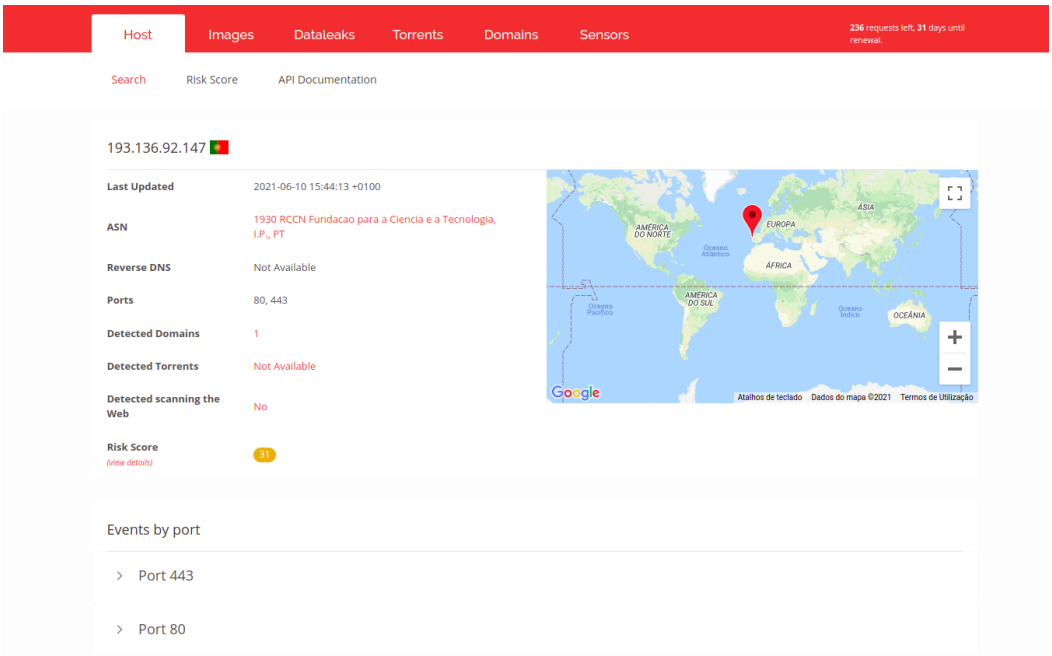


Figure 5: BinaryEdge Host Dashboard

BinaryEdge gives information on the hosts ASN, open ports, DNS, localization and risk score. BinaryEdge risk score ranges from 0 (low risk) to 100 (high risk). It is calculated using multiple parameters such as: number of open ports, use of unencrypted services, CVEs found, etc.

For each open port an extensive report is displayed containing the protocol used, the service software and version and the CPEs found.

6.3. Spyse

Figure 6: Spyse Logo

Spyse [28] is a "Internet assets registry for every cybersecurity professional". Spyse provides several tools that can be used through its API or a search engine in its web page.

Spyse's search engine allows to search by multiple parameters such as: IP, DNS, CVE, Technology, etc. We will focus on the first two.

When entering an IP or DNS address Spyse's displays a dashboard with information found about the host searched. In the figure 7 below we can see an example of a dashboard when a search was made for the DNS `xcoa.av.it.pt`

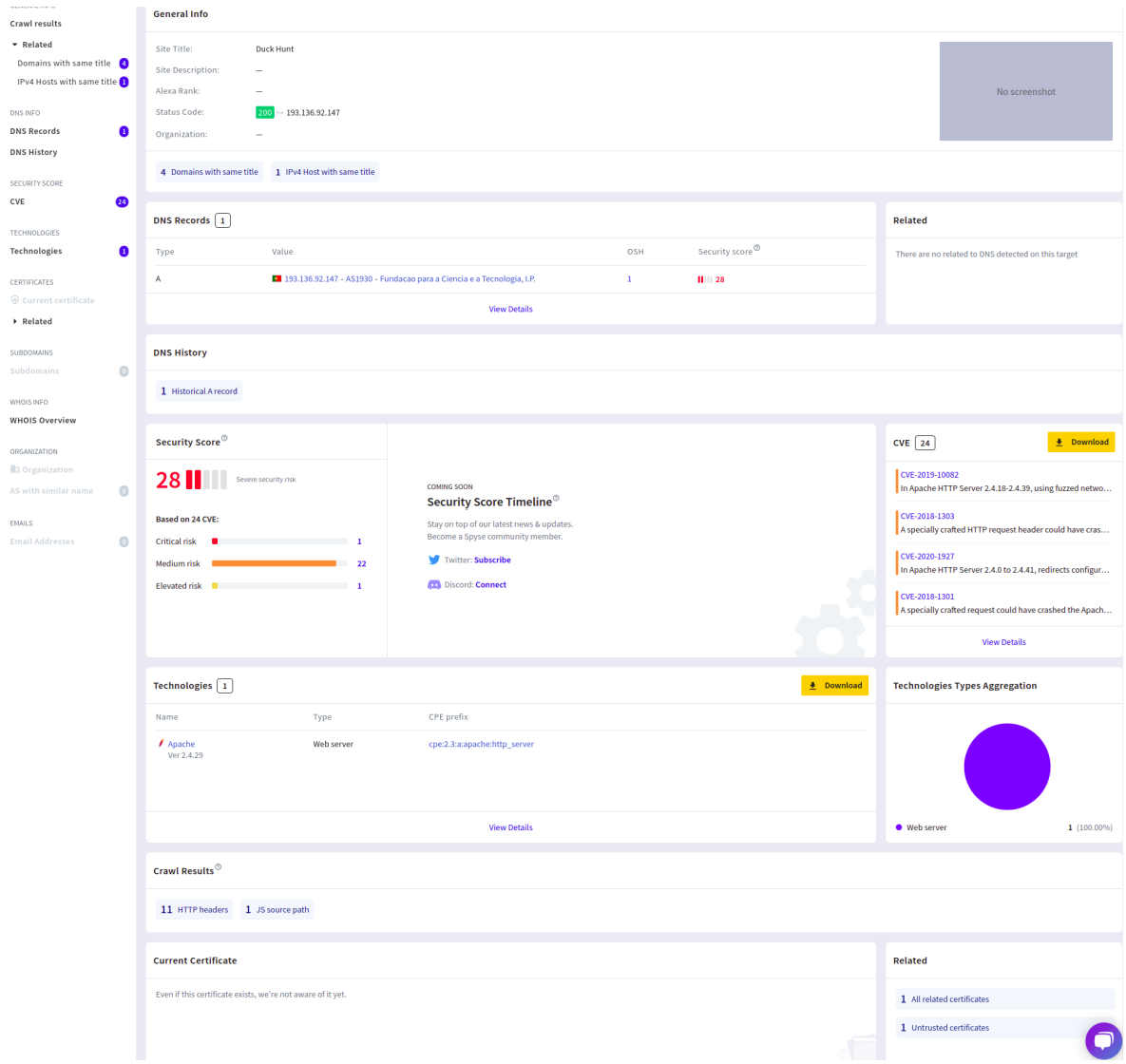


Figure 7: Spyse Dashboard

Spyse's dashboard gives general information about the host, such as: IP address, organization name and DNS records and history. No information regarding open ports is given.

Spyse's calculates a "Security Score" ranging from 0 (high risk to attacks) to 100 (low risk to attacks) based on the vulnerabilities' CVSS found for that host. The CVE's found are displayed and the user can click on them in order to display more information about it.

Technologies used by the hosts are enumerated. We can know the software and its versions as well as their CPE prefix. The technologies are divided in types to allow an easier perception of what it is exposed in the hosts.

Spyse's also tries to obtain a host's certificate in order to see whether it has expired or not.

Information regarding the parent domain is also shown.

6.4. Comparison between products

6.4.1. Information Displayed

Spyse is the only product which does not give information regarding the number of open ports.

Both Spyse and Shodan give special importance to the validity of SSL certificates, this is not explored in BinaryEdge.

Similarly to our project all the products display information about the technologies used by the services in each port.

6.4.2. Risk Score Calculation

Both Spyse and Shodan base their risk score in the CVE's found in the host while BinaryEdge has a much larger range of parameters. All the tools risk score goes from 0 to 100. In our project we narrowed the risk score (1 to 5) and calculated it not only based in the CVE's found but also in the number of vulnerabilities found by the tools as well as their respective risk associated with each one (which is the case of Zap tool).

6.4.3. User notifications

Shodan users can set notification triggers of different types. The notifications on our project are more rigid, as the subscriber of a host will always receive a notification when a scan is finished. In this notification the user will receive possible solutions for some of the vulnerabilities found, which is not done by any of the products above.

6.4.4. Other Functionalities

Like Shodan, our project also allows its users to do an immediate scanning request.

BinaryEdge's API allows to search for historical data up to 6 months while in our project the user can also see previous reports.

7. Conceptual modelling

7.1. System Requirements

7.1.1. Requirements Elicitation

This project, as well as all of those developed in this curricular unit, have been proposed to the students that have developed them by teachers of the department. In our case, we were presented with a pre-conceived description of the system, that provided us with the following context:

- University of Aveiro, as a public university with its big dimension, has thousands of domains exposed to the internet, with each and every one of them representing a potential threat to the organization. If explored by anyone with bad intentions it could represent a leak of personal information of its members, directly if those are managed by the system or indirectly if the system is used as an entry point in the institution network to carry out parallel attacks;
- This unknown possible vulnerabilities exist but are not controlled by the ICT services. It is crucial to introduce a system capable of providing this monitoring, ending with unawareness culture and fostering good practices of regularly updated and better deployed systems;
- The system must incorporate existent tools for scrapping, enumerate the vulnerabilities and present them in a dashboard. The focus of the project is developing the platform and implements its functional logic.

Despite having a general idea of what the system should do, the development process required a better understanding of who would be using the system, in what terms, with what objectives and from an architectural point of view, the components that it would be made of, how they would interact and how we would implement each one.

This process was carried out during the first weeks, in which we have searched, read and started documenting each one, discussing them in regular team meetings to ensure a collective knowledge and participation. The weekly meetings with our advisor, co-advisor and collaborators were decisive to this process, allowing us to clarify topics that were not clear yet and providing us with knowledge of experts in the area: teachers with expertise in the Security area, members of the IEETA and IT research units, a PhD student and the coordinator of the University CSIRT team. This diverse group allowed us to have inputs from multiple perspectives and in our opinion, to have developed a better system.

When we finished this process we had defined the three major components of our project: the user interface that provides the user with information about their hosts, the workers that execute the scrapping tools and the colector, the backbone that manages the workload of the last, processes their outputs and populates the database with information to be displayed to the user by the first component.

We have also defined the core actors and user stories, prioritizing the vertical implementation of the scrapping process and after that administrative and other improvements to ease the system management.

During the development process we have kept with the weekly meetings with our advisors, that gradually grew into a space for summing up the work done, discuss it, define improvements and the next steps to follow.

7.1.2. Context Description

This subsection presents a description on how the system is expected to be used by its different stakeholders.

The **system administrator** manages the entire system with all the permissions, being able to:

- See each host's details;
- See the statistics of the system and its hosts;
- Approve/disapprove requests to subscribe an host;
- Assign host roles to users (owners e subscribers);
- Change the scrapping level for a specific host;
- See the workers' page and manage them.

The **owner** of a host (or a set of hosts) can:

- Add new owners/subscribers to a host;
- See the statistics for the hosts which he subscribes/owns, also receiving an email each time a scan is finished;
- Change the scrapping level for a host he owns.

A **regular user** can:

- Ask for authorization to subscribe to a host he does not own;
- See the statistics for the hosts which he subscribes;

7.1.3. Actors

7.1.3.1 Admin

The admin has access to the entire system, as well as its features. Has the permission to see the list of all the hosts analysed on the system. He/she can also see the global statistics of the system, add/change/delete workers and its hosts associated and accept/decline hosts subscriptions requests.

7.1.3.2 Owner

The owner can manage his/her hosts, check statistics/vulnerabilities found, define a scrapping level for a specific host, and receive notifications whenever a vulnerability is found. He can also request for a subscription to a different host that can either be accepted or declined by an admin. Is important to state that the owner is only able to see information of hosts it is assigned, being unable to access information of other hosts.

7.1.3.3 User

The user can check global statistics about the system, and can request to receive notifications about a specific host. If the request is accepted, the user will be able to see the host's page and its information.

7.1.4. Personas

	Name: Rafaela Fernandes Occupation: Server Manager at STIC Age: 31 Gender: Female Scholarity: Master in Software Engineering
Objectives: Rafaela is responsible for STIC servers. Her objective is to assist machines to recover from failures, mitigate risks, search for/eliminate threats.	
Stories: As a Server Manager I want to be able to keep the servers up to date in terms of security, preventing possible attacks. As a Server Manager I want to understand which machines are obsolete.	

Figure 8: Persona 1

	Name: Carla Pereira Occupation: Teacher at DETI Age: 42 Gender: Female Scholarity: PhD in Software Engineering
Objectives: In addition to teaching at DETI, Carla is also a researcher at IT. Both in teaching and research, over the years she has created several web services that she makes available internally and externally through the UA network. However, the workload prevents her from keeping all these systems up to date and free from vulnerabilities, as she doesn't have time to check them one by one regularly.	
Stories: As a teacher and researcher, I want to focus on my new projects and maintain existing ones only when strictly necessary.	

Figure 9: Persona 2

	Name: Ricardo Ferreira Occupation: Member of the UA's Cybersecurity Office Idade: 38 Gender: Male Scholarity: Mestrado em Engenharia de Computadores e Telemática
Objectives: Ricardo is a member of the UA's Cybersecurity Office, whose mission is to promote awareness and the adoption of safer behavior with regard to the use and treatment of devices and digital information.	
Stories: As a member of the UA's Cybersecurity Office, I want to manage and ensure that security controls are well implemented.	

Figure 10: Persona 3

7.1.5. Use Cases

Figure 11 presents the use case model of the whole system, containing both the use cases and their respective actors.

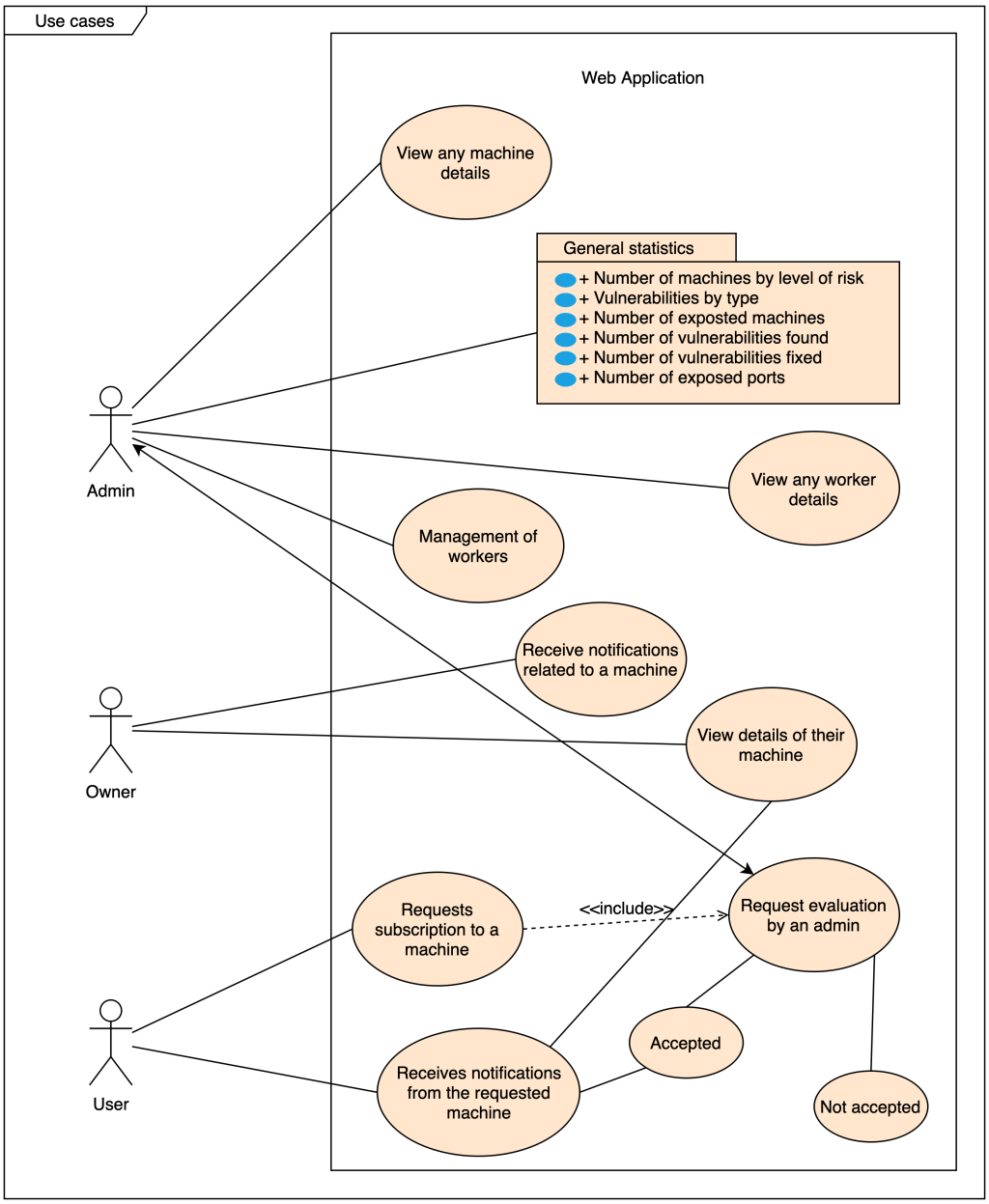


Figure 11: Use Case Model

7.1.6. Functional Requirements

- **Login:** To safeguard sensitive information that will be shown by the system, it will be required from users (subscriber or owner of UA's hosts or system's administrator) to have an active UA account, as the login process will be made through UA's IdP. Every new user, at first access, must request permission to the system's administrator informing the host(s) of subscription, to justify the need to have the permission granted. **Priority:** Maximum
- **Dashboard:** Web platform available 24/7 through UA IdP access that allows the system management and hosts vulnerability monitoring. **Priority:** Maximum
- **Alerts:** Email sent every time a scrapping is performed on the user hosts. It informs the manager of its existence and suggests possible corrections. **Priority:** Maximum
- **Scrapping:** With a configurable periodicity and with several levels (from the less to the most evasive), it recurs to external pluggable tools. **Priority:** Maximum
- **Vulnerability Analysis:** Based on the data gathered by scraping tools, the risk is computed based on CVSS found on the vulnerability scans.

7.1.7. Non-functional Requirements

In addition to the functional requirements mentioned before we have also defined some non-functional ones to make sure that more than responding to what the user needs, it does so within a proper and effective manner.

- **Security:** Despite that the information that our system collects is accessible to anyone with a scrapping tool in the same network as our workers, it must be visible only to those that own, have an authorized subscription or administrate the system. Therefore, it is fundamental to have defined roles with different clearances. **Priority:** Maximum
- **Privacy:** The system must not store any sensitive or personal data about the hosts, even if it is retrieved by any of the scrapping tools during the scans. **Priority:** Maximum
- **Usability:** The management interface must be intuitive, following the latest principles of User Experience and User Interaction, providing a good experience to all its users. **Priority:** Medium
- **Scalability:** The system should implement a fair and efficient distribution of the workload through its workers in order to respond seamlessly to high workloads. **Priority:** Medium
- **Portability:** In order to make the deployment process easy and avoid compatibility issues that tend to increase with the time, this process can be fulfiled with containerization tools. **Priority:** Low

7.1.8. Assumptions and Dependencies

In order for the application to work as expected, the following assumptions are made. The first requirement to operate this product is that the user must have a stable connection to the Internet, as well as the programs needed to access it (web browser, etc.). It is also expected that the user has an active UA account, because the login process is made through UA's IdP and the owners and subscribers of the hosts registered to the system will receive primary through UA's email account the vulnerability reports after each scan whenever a vulnerability is detected.

7.2. System Architecture

7.2.1. Domain Model

The following image represents the desired architecture for Domain Model, which aggregates Front-end, Back-end, Tools and Probe Hosts. Front-end is referring to the interactive platform in which users (host's owners) can manage their hosts.

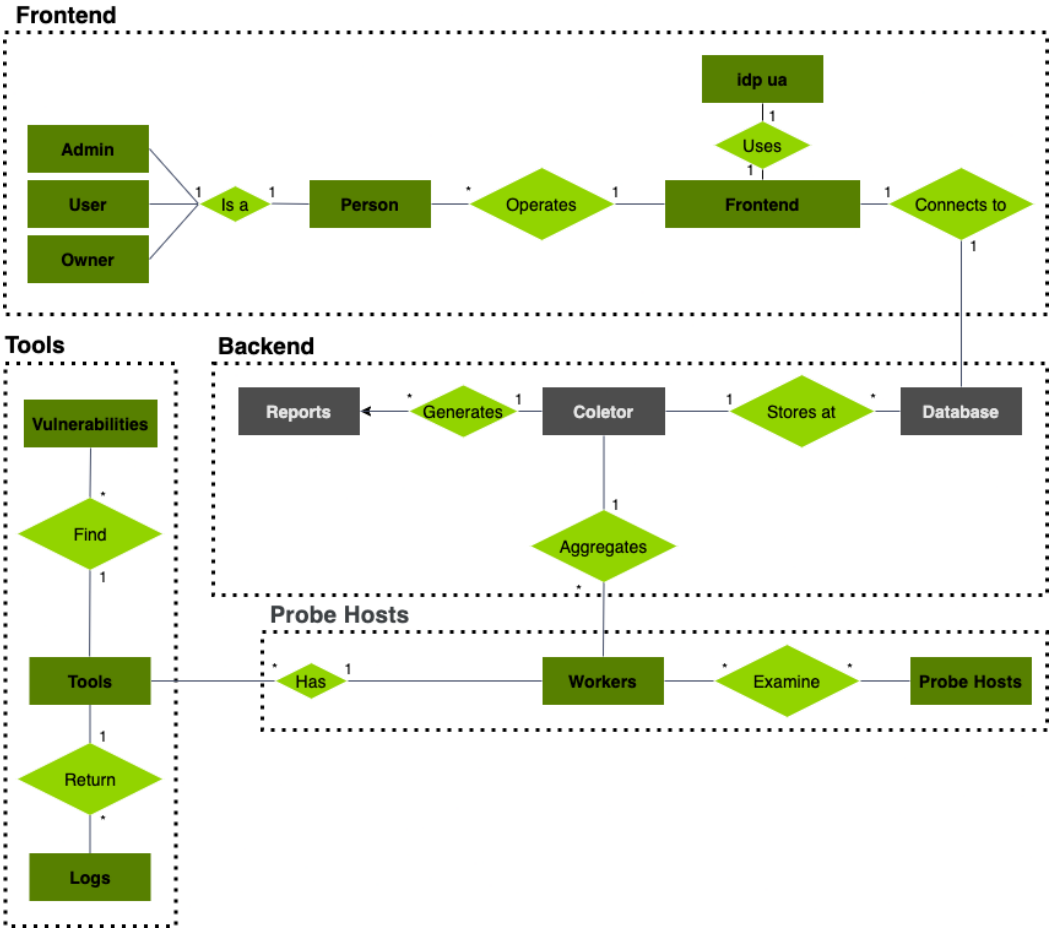


Figure 12: Domain Model

Back-end is responsible for ensuring the communication between front-end and the database, which is important so the user can get their devices information, such as scan reports, host's status and vulnerabilities, worker's logs, fix suggestions, etc. Probe Hosts are all the identifiers of machines to be examined by workers. Back-end is also important to collect the data sent by worker, which is done by the 'collector' entity. This entity collects the data (vulnerabilities and logs), compiles everything in one report and send a notification saying that user's host have been scanned. Is also responsible for scheduling automatic scans for all registered hosts.

7.2.2. Physical Model

The figure below gives an idea about how are the different components built. We chose python due to our group previous experience and its versatility.

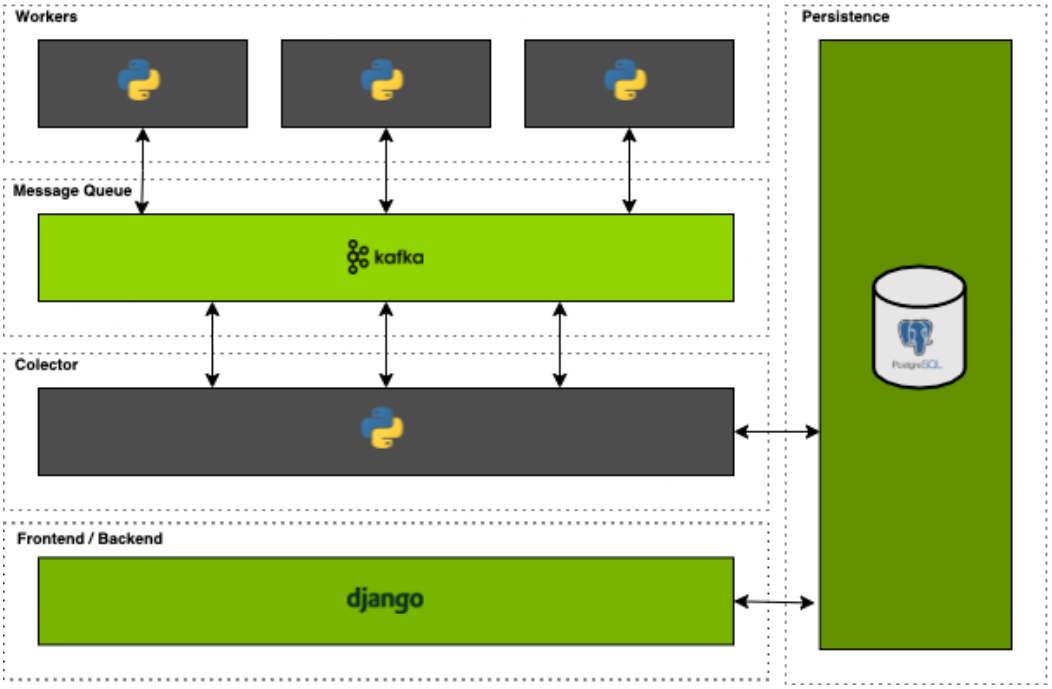


Figure 13: Physical Model

Kafka was implement to ensure the communication between Workers and the Collector, so it can send the scan requests to an available worker. This worker can also send the vulnerabilities, logs and his heartbeats over the queue. The topics adjacent to their communication are : 'INIT', 'SCAN_REQUEST', 'FRONTEND', 'LOG', 'HEARTBEAT' and 'UPDATE'. Respectively, they allow the secure connection between one Worker and the Collector; the request of a scan for a domain/IP from the Collector to a Worker; the request, from an user (using the platform) to scan a host as soon as possible; sending the logs from Worker to Collector; the verification that a Worker is online by sending an heartbeat message and waiting for the response; Lastly, 'UPDATE' is used to change some worker's configuration.

7.2.3. Deployment Model

Figure 14 below represents the deployment diagram of our system.

User's browser connects via HTTPS to the web server which was implemented in Django. The web server is connected to a PostgreSQL database and also publishes Kafka messages with the purpose of communicating with the Collector.

Worker-Collector communication is done via Kafka. The Collector is also connected to the PostgreSQL database.

The Workers have access and use the docker images of the tools which are contained in our docker registry.

All this components are in docker containers started from the host `deti-vuln-mon.ua.pt`.

We chose to dockerize our components in order to isolate each one and making the deployment easier.

Only port 443 and 5000 are exposed to the exterior as a security measure.

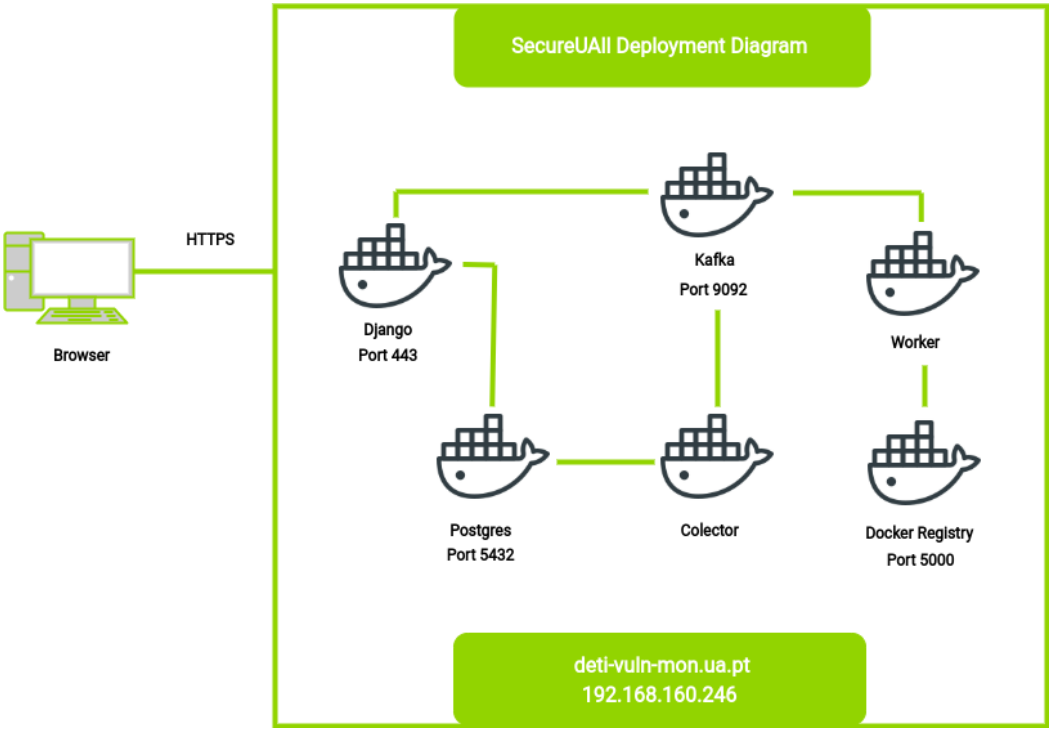


Figure 14: Deployment Model

8. Implementation

8.1. Dashboard

For the users to interact with SecureUAll system, we have created a web based application with Django.

8.1.1. Prototyping

8.1.1.1 Mock up

After the requirements elicitation and before starting the implementation phase we have created a mock up with Figma editor [14] to establish a skeleton for the main interfaces such as the home page (Figure 15), the host details (Figure 16), the workers management (Figure 17) and the requests review (Figure 18).

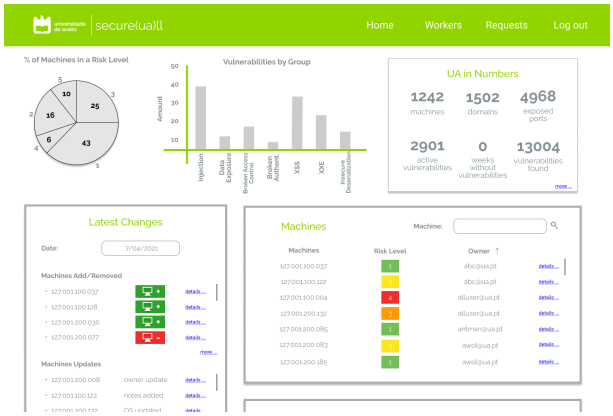


Figure 15: Home page mock up

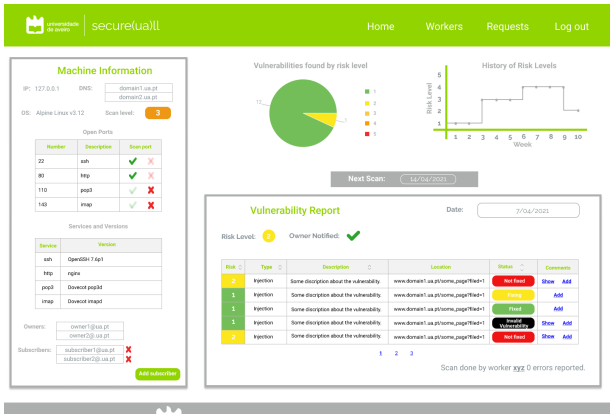


Figure 16: Host details mock up

Figure 17: Workers management mock up

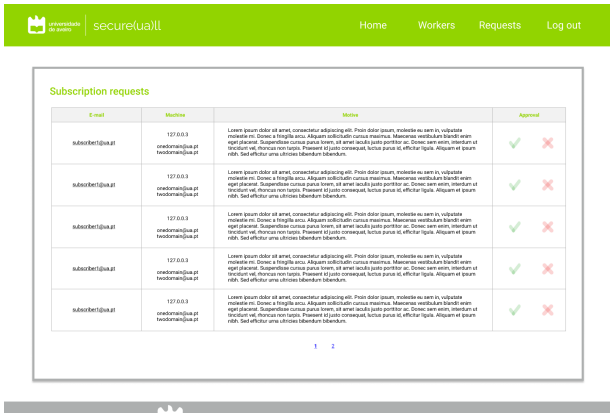


Figure 18: Requests review mock up

8.1.1.2 Visual styling

During the mock up construction we have also studied color combinations. We have experimented several of complementary, triadic and analogous colors [15], trying to achieve the most comfortable and appealing design. Eventually, after some discussion with our advisors we have chosen to use a pallet based on the UA visual identity [1], illustrated on Figure 19.

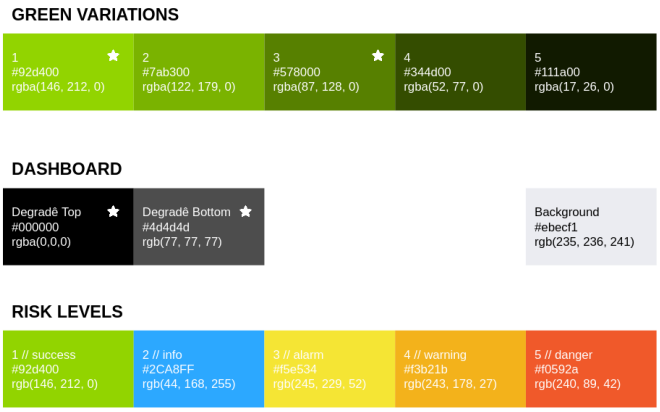


Figure 19: secureUAll color pallet

8.1.1.3 Template

Since we already had the visual skeleton and the colors defined, the next task related to the dashboard was creating the template. We used Now UI Dashboard by Creative Team [29] (represented in Figure 20) as basis, adapting it to our UI and UX vision and context. Figure 21 shows the final result. The main changes were the colors and the sidebar, that we decided to remove as the dashboard does not have a number of pages that justifies it. We have used a top bar instead.

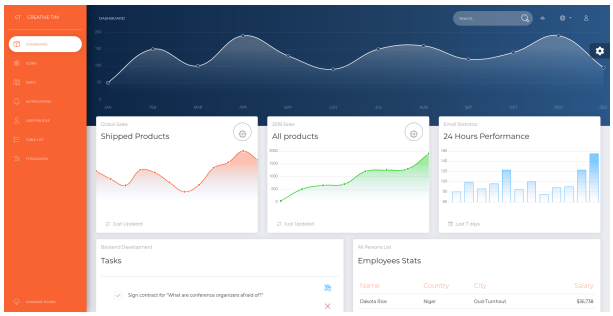


Figure 20: Now UI Dashboard template

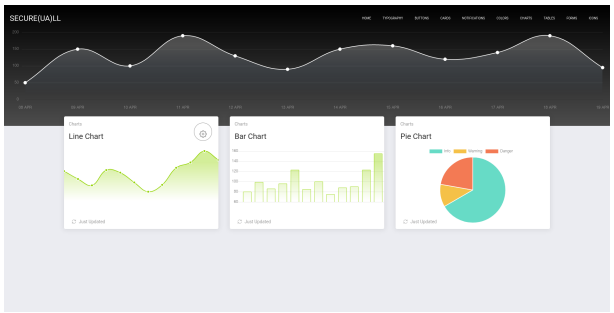


Figure 21: Customized template

In this template we have also created several pages with examples of the main components, such as buttons, forms, notifications, tables... Figure 22 shows some examples.

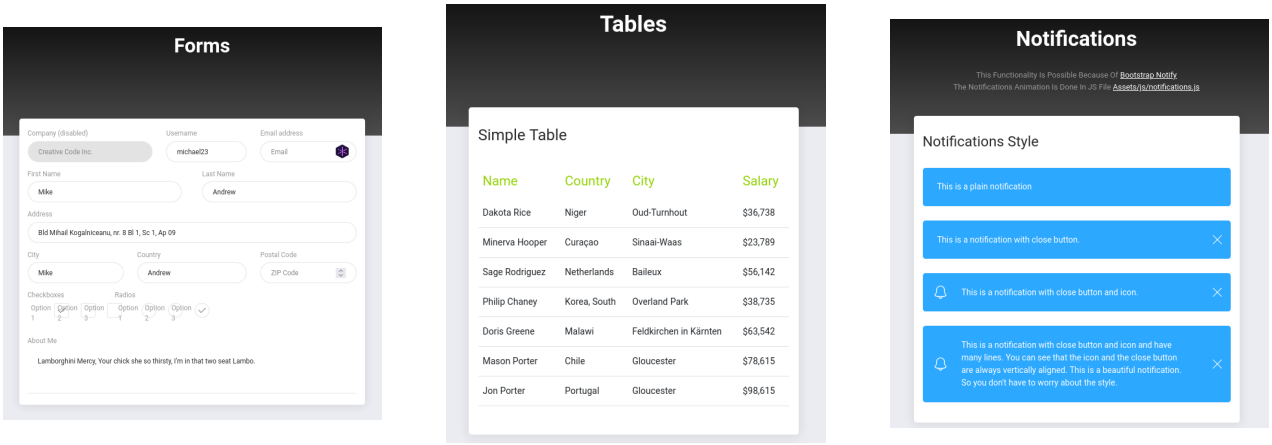


Figure 22: Template main components examples

8.1.2. Architecture

In order to reduce the complexity and ease the development process we have divided our project into several applications [8] that encapsulate the different business logics associated with our project, namely:

- **Login and djangosaml2 [27] (external library)** To handle user authentication;
- **Machines** That works with everything related to the hosts, their attributes and scans results;
- **Workers** To manage and monitor the system workers;
- **Dashboard** An app created for the system homepage, an aggregator of the previous.

Apart from the external library that was integrated in our project without changes, every other application was developed with a standardized internal structure.

This structure is visible in the project architecture diagram, illustrated at Figure 23.

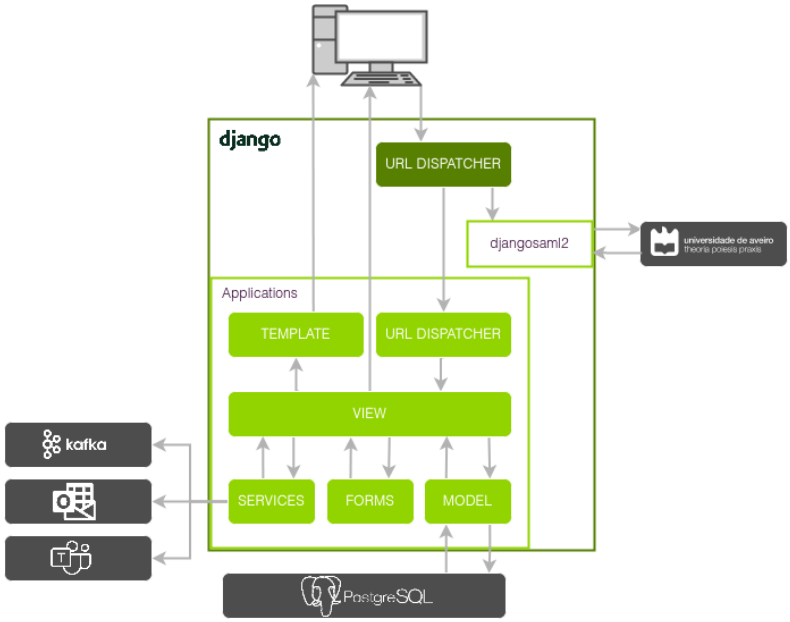


Figure 23: Dashboard architecture

The user HTTPS requests enter the Django through Url Dispatcher. The main one (presented in dark green) forwards it to the in app dispatcher (presented in light green), which will map the requested URL to a view function and calls it. The View, then, processes the user request and performs all the actions associated, using Forms to analyse data input, Models to manage data and interact with the database and Services to connect with external components. Finally, after the processing is done, the response is returned in one of two forms: the first and most common, a HTML page rendered from a Template, and the second which is a JSON response for asynchronous calls with Javascript.

The external services are detailed in the next topics.

8.1.3. Domain Model

When we first tought about the system architecture, we defined that we were going to use two databases: one document oriented and the other relational. The first to store the host logs, as they were expected to have a big dimension and variable structure, and the second for everything else.

However, during the development process we decided to create a standardized output for every scrapping tool that we were using, which reduced the entropy and filtered the unnecessary information. This, the fact that Django is ORM (inefficient with NoSQL databases) and that Postgres can store big amounts of data in text fields [20] led us to drop the documental database and use only a relational.

As already mentioned, the database system we used was PostgreSQL [21]. This choice was based on several factors, such as that it is open source, supported by Django [9] and our experience in previous projects.

The domain model was created right after the requirements elicitation was finished, having suffered small improvements during the development process, correcting small mistakes and introducing

new models for features that we did not predict to implement, like the logs being stored in the relational database. It is represented at Figure 24.

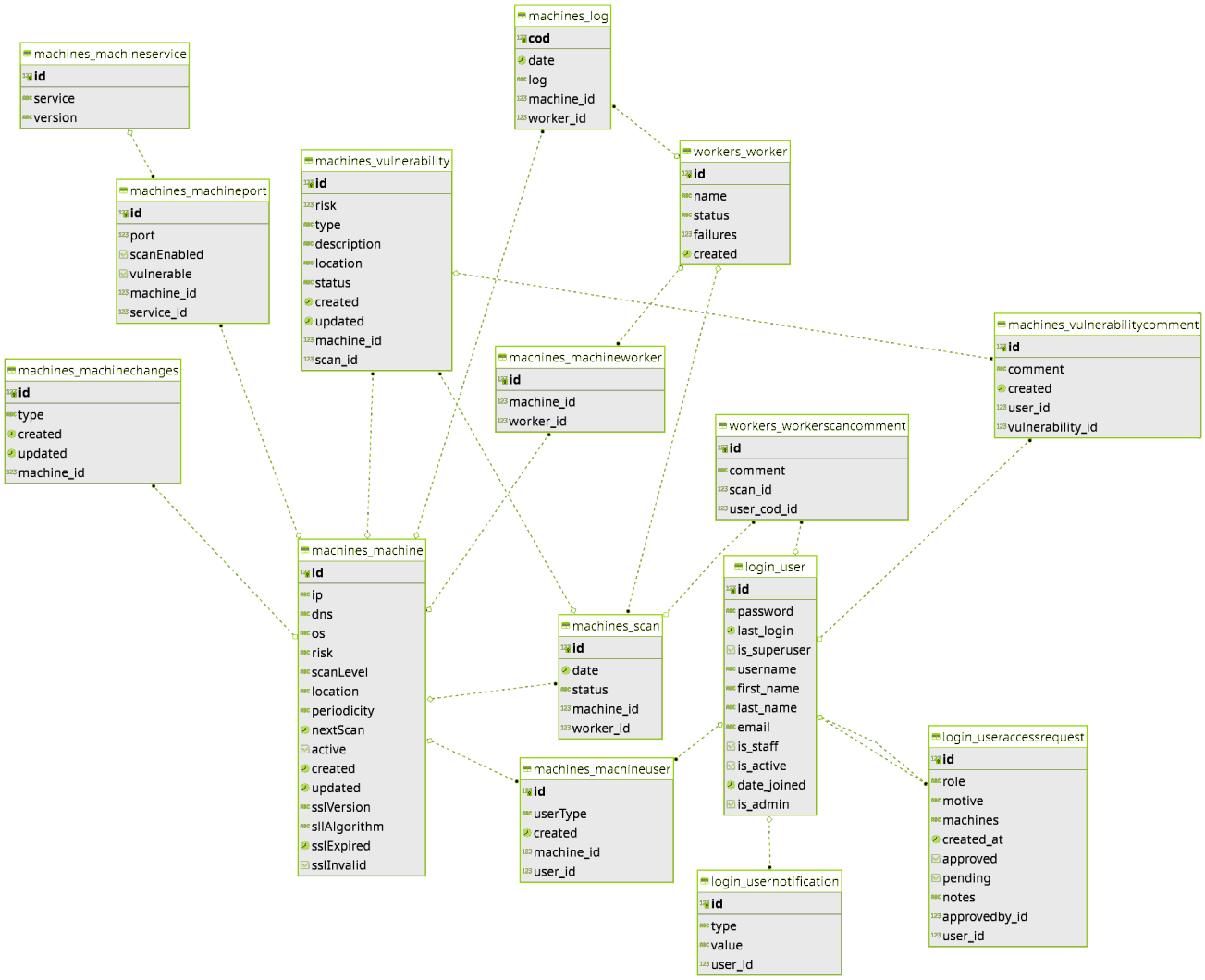


Figure 24: Domain model

The database structure was defined with Django Models [10], which maps Django classes and its variables to database tables and attributes.

8.1.4. Project applications

8.1.4.1 Login

This application integrates most of the logic related to user management.

Because our system has UA community as target group, we use this institution Identity Provider as authentication mechanism. For that purpose, we recur to an external library, djangosaml2, that we have

configured with the necessary settings to work with this specifid IdP.

After logging in, the user can be redirected to the Dashboard if he/she is the system administrator or is associated to some host(s) as subscriber or owner. If not, he/she is presented with a welcome page with a form that they can fill an access request, that will be reviewed by an administrator. The follow up on the request status is also possible on this view, paired with an email notification that is sent to the petitioner when it is reviewed.

Finally, and for authenticated users with access to the dashboard, this application also offers a profile page, where the user can check their information (can't change it because it is provider by the IdP) and change their notification preferences. We have two ways of reaching our users:

- **Email notifications** Sent to the user's institutional email, provided by the IdP;
- **Microsoft Teams** Sent to a web hook that is configured in this external service by the user following instructions that we provide.

When the user needs to be notified of something, these preferences are mapped to a service that encapsulates both the options available, making it transparent in the code that uses it through the implementation of factory and builder creational patterns.

8.1.4.2 Dashboard

The Dashboard is the main page, it has a friendly interface and aggregates the general information for that user, which can be a system administrator or a subscribing user of hosts, and for each type of user the information shown is different. By clicking on any host reference on this page, it will redirect to the host's specific page.

An administrator will be able to view the system's global information of all registered hosts, with a compilation of relevant numbers, some statics and updates, and alerts concerning the status of workers and pending requests from users. Listed below are the features for this type of user:

- **Vulnerabilities evolution graph** - It shows the number of vulnerabilities registered in all scanned hosts in the past days.
- **Alerts** - In this area are listed the workers that are down and it can redirect the administrator to the worker's page, and also the pending requests awaiting for approval.
- **secureUAll in Numbers** - This one gives some global stats, the number of all registered hosts, the total amount of open ports from all hosts, the active vulnerabilities in all hosts, the total number of performed scans by all workers together, the total of vulnerabilities (including active, fixed and invalid vulnerabilities), and a counter giving the number of weeks all scans returned no vulnerabilities.
- **Hosts added/removed** - It lists the last updates, in the past week, showing the hosts added to the system and the hosts turned inactive.
- **Hosts updates** - It lists hosts that had updated information, in the past week, this information can be change of scan level, change of scan periodicity, new risk rating or Operating System version changed.
- **Unresolved vulnerabilities** - It lists hosts that have unresolved vulnerabilities, from the past week.

- **Hosts** - A table listing all the active hosts in the system, showing the main relevant information of each host, ip/dns, risk level and owners. The table can be ordered by any of the columns and can be filtered by any of the information on the hosts.
- **Hosts per risk level chart** - It shows the percentage of hosts by risk level rating, ranging from 1 to 5, and when clicking on a given risk it filters the host's table by it.

A subscriber type of user will be able to view the global information of the hosts he/she has subscription or owns, with a compilation of relevant numbers related only with these hosts. Listed below are the features for this type of user:

- **Hosts** - A table listing all the active hosts that user has subscription to, showing the main information of each host, ip/dns, risk level and owners. The table can be ordered by any of the columns and can be filtered by any of the information on the hosts.
- **Hosts per risk level chart** - It shows the percentage of hosts, from the hosts the user has subscription to, by risk level rating, ranging from 1 to 5, and when clicking on a given risk it filters the host's table by it.
- **Hosts in Numbers** - This one gives some stats, the number hosts this user has subscription to, the total amount of open ports from these hosts, the active vulnerabilities these hosts have, the total number of performed scans on these hosts, the total of vulnerabilities (including active, fixed and invalid vulnerabilities), and a counter giving the number of weeks all scans returned no vulnerabilities.

8.1.4.3 Workers

Workers application allows for the management of the system workers and because it is an administrative operation, is only accessible to the system administrator.

The main page is the one that lists the workers, where the administrator can check their status, logs, see the hosts associated to each one, edit them and add new ones.

There are two ways to associate new hosts. The "batch mode", in which the user can insert a list of host names (IP and/or DNS) separated by any of the supported symbols (new line, comma and/or semicolon). This functionality was created thinking of users that want to insert long lists of hosts from spreadsheets. The other is "IP Range", a simple form where the user writes the base IP address with a range, and associates all the hosts among that range.

Both of this alternatives are just the beginning of the association process, after which they converge to a page where the user is shown a form with all the hosts inserted and for each a group of settings that he can configure, like the scan level or its periodicity.

This final view is also used for the editing process, with an extra option to disassociate the host from the worker.

Every time a worker sees its associated hosts change (through addition or edition), a message is sent to the collector through Kafka to notify of this changes, so that it can schedule the new scans in case a new host is paired with the worker.

For each host, the logs are in a page grouped by scans. When the user clicks a scan he is presented with the log (in JSON format) indented and with syntax highlighting to eas the analysis.

Finally, we have consider a different paradigm. Instead of managing the hosts for each worker, we created a view that presents all the system hosts and for each the workers associated. This allows the administrator to have a perception of the hosts that are not associated to a worker and correct this failure.

8.1.4.4 Machines

Machines is the main page of a host. It aggregates the specific information for that host and also allows the management of parameters related to the scanning of the host. The users that have access to a specific host are only the system administrators and the owners/subscribers of that host, the other users that has access to the system but are not subscribers of that host don't have access to the page. The difference between the user's roles are related to the management parameters, that can be made by administrators and owners but are read-only for the regular subscribers that are not owners. Listed below are the features for this page:

- **Host general information** It shows the main information of the host, ip address, dns, date registered to the secureUAll system, the Operational System installed and the risk level rating.
- **SSL information** It shows the SSL configurations detected during the scans, the version, the algorithm, the expiration date and the validity.
- **Workers** This area shows the workers associated with the host, that perform the scanning. It allows system's administrators and owners to manage this information.
- **Open ports** This table lists the open ports detected during the last scan, giving it's number, service, service version and vulnerability.
- **Owners and subscribers** This one lists the owners and subscribers to the host, which system's administrators and owners are able to manage this information by adding or removing them.
- **Vulnerabilities per risk level chart** It shows the percentage of vulnerabilities found on scanning, by risk level, ranging from 1 to 5, and when clicking on a given risk it filters the vulnerabilities report's table by it. The chart can filter by showing the last scan, last moth scans or all time scans information, which also applies the filter to the vulnerabilities report's table.
- **Vulnerabilities per scan chart** It shows the amount of vulnerabilities found on each scan, by the date it was performed. The chart can filter by showing the last scan, last moth scans or all time scans information.
- **Vulnerability report** A table listing all the vulnerabilities found on scanning, showing the main relevant information of each vulnerability, risk, type, date of scan, description of the vulnerability, location, status and comments. The table can be ordered by any of the columns and can be filtered by any of the information on the hosts. The tables can also filter by showing the vulnerabilities on the last scan, last moth scans or all time scans. It also shows some general information, the date of next scheduled scan, the periodicity that host is scanned and the scan level, in which the periodicity and scan level can be managed by the system's administrator and owners type of users. The scanning logs and the request for an instant scan are only visible for the system's administrator and owners. In addition, all users can add comment to the vulnerabilities and change it's status, by clicking on the vulnerability row on the table.

Apart from the host page, there is also the requests one, where the users can check their hosts

access requests once they were authorized to use the system and eventually submit new ones. This is also where the administrator reviews the pending requests.

8.1.5. Usability

In order to improve the experience of our system users we have tried to make the functionalities the most intuitive possible and with a reduced learning curve.

A good example are the options for associating hosts to a worker, with the batch and the IP range modes, which allow to perform the same task, but for different scenarios.

To simplify the access to the information, we have also implemented several filters that are accessible both on the top of the page to change its full context (for example in the workers list the user can filter by their status - Figure 25) or on the top of the cards to change their specific context (Figure 26).

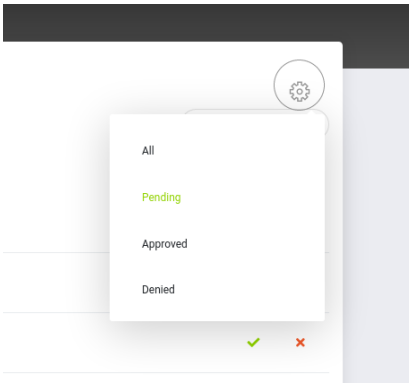


Figure 25: Workers page filters

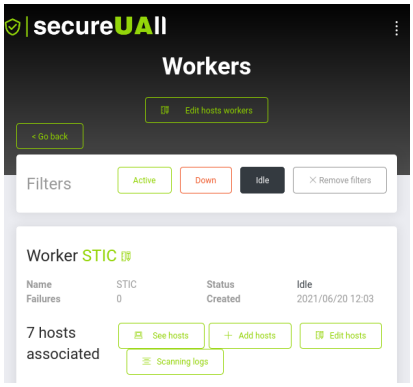


Figure 26: Requests page card filters

Paired with the filters, are the dynamic tables, that are paginated, allow the user to sort it for different columns and to search for any text.

Lastly, we have decided not to create extensive documentation. We consider that our platform is not that difficult that it would demand that kind of paperwork. Instead and to make it easier for the user to understand each interface, we have developed a help Javascript plug-in from scratch, based on Bootstrap popovers [4].

This help is accessible through a button that is available on the top right corner of every page (Figure 27) and when clicked starts a guided tutorial that shows popovers sequentially next to every important element of the page, showing for each a brief description about its purpose and/or functionality. Figure 28 shows an example of this help.

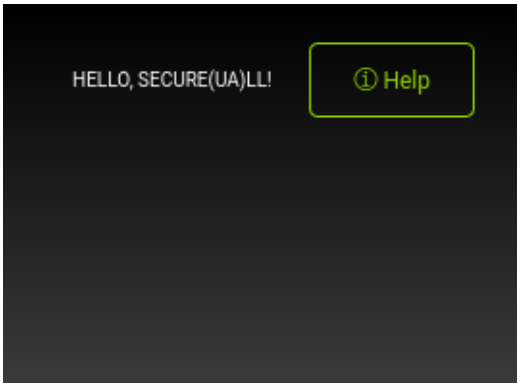


Figure 27: Help button

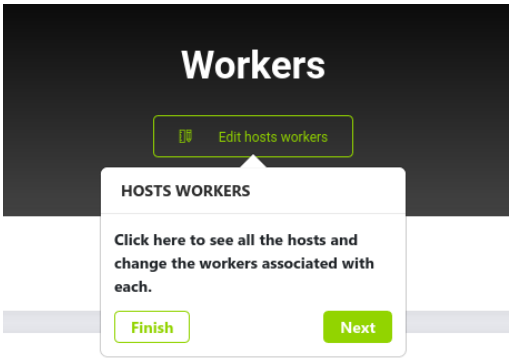


Figure 28: Help popover

8.1.6. Testing

For testing our Django project we have tried to implement a complete tests suite. In order to accomplish that, we have used Django testing tools [11] and developed it based on a Mozilla tutorial [17].

Our testing strategy was founded on the principle that Django features are valid and working, as they are tested already by their developers. So, we tried to test every other functionality that was coded by us:

- **Models** For every model created, we have tested the custom methods we have defined;
- **Forms** Apart from the forms fields default validations, we have created lots of methods that provide a customized one and for every one there are several tests that check both the positive and the negative scenarios;
- **Views** We tested the most important views for basic configurations, like the URL, its reverse and if it is available for every user that must have access to it;
- **Full context** To test the application full context we have developed some integration tests for the most important workflows. These tests use the automation ecosystem Selenium [18] and the behaviour-driven development tool Behave [2], which allowed us to write tests in plain text.

8.1.7. CI/CD

For CI and CD, we have implemented custom workflows directly in our GitHub repository with GitHub Actions.

CI using GitHub Actions runs custom tests on push or pull request events, the last only for the main branch. The workflow runs on a GitHub-hosted virtual machine using Linux, and run the following steps:

- Checkout the code
- Install project requirements
- Set up Django project
- Run Django tests

If all CI tests in a workflow pass, the changes pushed were reviewed by a team member, other than the member who commit the changes, and merged. When a test fails, the commits are not pushed to the main without fixing.

CD using GitHub Actions pulls from git repository specific branch with the changes ready to be deployed, install all the requirements and restart the server in charge of running our system. For that, GitHub Actions runs the following steps:

- Set up job, prepare permissions, workflow directory and required actions
- Run actions, sync and fetches the repository and checks out the branch 'deploy'
- Start docker, runs the project's docker compose file with its dependencies, and builds the deployment container
- Post Run actions, deploys the code from the checked out branch on the built service

8.2. Workers

Workers are responsible for scanning the requested hosts, using a defined set of tools. The output from each tool is parsed, standardized and sent back to the collector. The main operations that workers perform are:

- Receive host domains/IPs and run a set of tools, depending on the requested scrapping level;
- Update its domains' list;
- Hearbeat;
- Parse each tool output into a standardized output.

8.2.1. Architecture

In this section it is given an overall architecture of the Worker component.

8.2.1.1 General

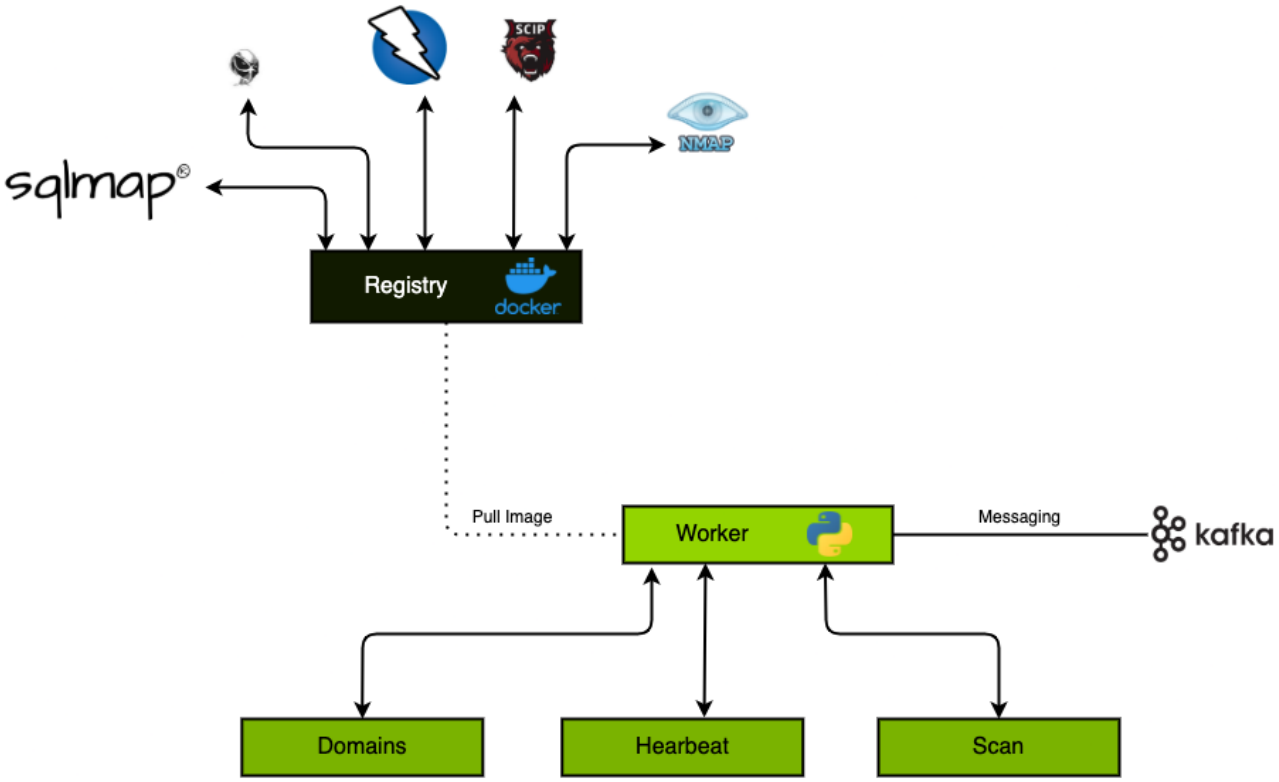


Figure 29: Worker's general architecture

Firstly, there is a local docker registry that contains all the tools' images. This allows any Worker on any host that connects to the Collector to pull the necessary images to be used in the scans. Our registry contains images of:

- Nmap
- Vulscan (Nmap NSE Script)
- Zaproxy
- Nikto
- Sqlmap

This is also a good practice because the Workers do not need to pull the images directly from external sources, which could be a gateway for dangerous software.

All the communications between the Workers and the Collector are made through Kafka Consumers and Producers. The Worker uses 5 topics:

- 'INIT' - When the Worker connects for the first time, a series of messages are exchanged in order to identify the worker correctly and assign it an ID.

- 'HEARTBEAT' - This topic works on a different thread. Its purpose is to make sure that the Worker is still active.
- 'UPDATE' - Updates the Worker's domains list of hosts to scan.
- 'SCAN_REQUEST' - This is our main topic. A host domain/IP is received, as well as the scrapping level. The Worker proceeds to scan the host and parse/standardize the outputs from each tool.
- 'LOG' - After scanning the host, the constructed output is sent to the Collector in this topic.

8.2.1.2 Technologies

In this section we will explore the technologies used in the Worker component.

Kafka

Figure 30: Kafka logo

Kafka is our inter-module communication component. As it was said before, we have a few topics that make the needed operations between the Worker and the Collector. Essentially, Kafka is our communicator.

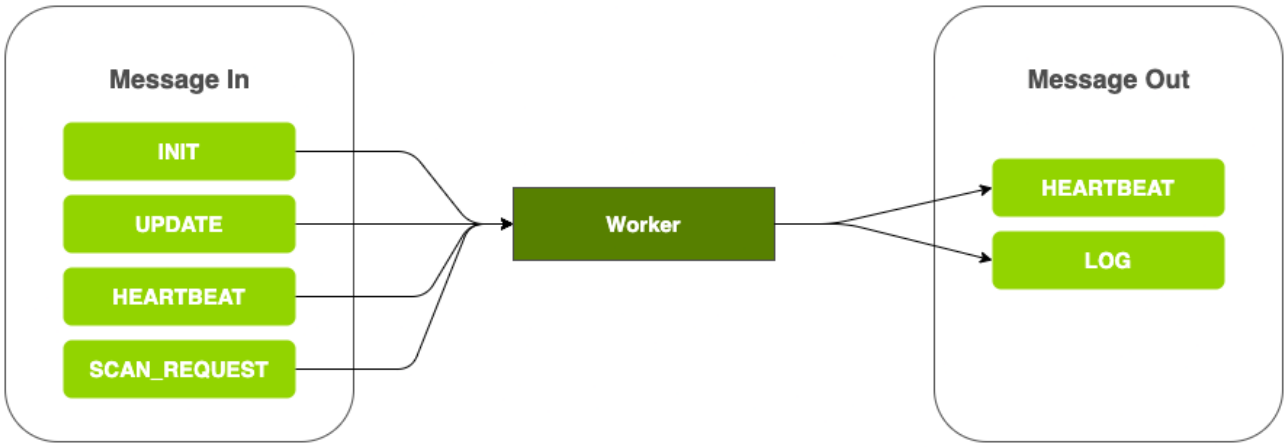


Figure 31: Worker Messaging Model

Link to the software's page: `https://kafka.apache.org/`

Docker Registry

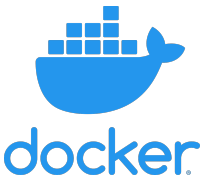


Figure 32: Docker logo

The open-source software Docker Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images. In our system, we have a local docker registry. This means that is completely independent from the network. We have our tools' images stored in it. When any Worker wants to use an image, it just needs to pull from the registry and is ready to be used.

This registry also supports a version system, that we used for some of the tools. For example, we have two vulscan images stored. In the early stages of the project, we had it under '/vulscan:v1'. But then we realized that it was not the latest version of the tool, so we added another version as '/vulscan:v2'. This way, the first version is still in the registry if needed.

Link to the software's page: `https://docs.docker.com/registry/`

Nmap

Figure 33: Nmap logo

Nmap ("Network Mapper") is a free and open source software, used for network discovery and security auditing. Nmap used raw IP packets to determine what hosts are available on the network, what services (name and version) are those hosts are offering, what operating system they are running, what type of packet filters/firewalls are in use, etc.

This tool was designed to rapidly scan large networks, but it works fine with simple hosts.

Some of Nmap characteristics being:

- **Flexible** - Supports dozens of advanced techniques for mapping out networks filled with IP filters, firewalls, routers, and other obstacles. This includes many port scanning mechanisms (both TCP & UDP), OS detection, version detection, ping sweeps, and more.
- **Powerful** - Nmap has been used to scan huge networks of literally hundreds of thousands of machines.
- **Portable** - Most operating systems are supported by Nmap.

- **Easy** - While Nmap offers a rich set of advanced features for power users, you can start out as simply as "*nmap -v -A targethost*".
- **Well Documented** - Significant effort has been put into comprehensive and up-to-date man pages, whitepapers, tutorials, and even a whole book.
- **Supported** - While Nmap comes with no warranty, it is well supported by a community of developers and users.

Docker image created from the following repository: `https://github.com/nmap/nmap`

Vulscan

Figure 34: Vulscan logo

The Nmap Scripting Engine (NSE) is one of the most powerful and flexible features in Nmap. It allows users to use scripts to automate network tasks. This allows users to make Nmap a more powerful tool. Vulscan is an NSE script which enhances Nmap to a vulnerability scanner. It uses a list of vulnerability databases to scan for potential flaws and CVEs.

Docker image created from the following repository: `https://github.com/scipag/vulscan`

Zaproxy

Figure 35: Zaproxy logo

ZAP is a free and open source penetration testing tool being maintained under the OWASP. ZAP is designed specifically for testing web applications and is both flexible and extensible.

At its core, ZAP is what is known as a “man-in-the-middle proxy.” It stands between the tester’s browser and the web application so that it can intercept and inspect messages sent between browser and web application, modify the contents if needed, and then forward those packets on to the destination. It can be used as a stand-alone application, and as a daemon process.

Docker image created from the following repository: `https://github.com/zaproxy/zaproxy`

Nikto

Figure 36: Nikto logo

Nikto is an open-source web sever scanner that performs comprehensive tests on web servers in order to find potentially dangerous files, outdated versions, HTTP options, etc. Its goal is to find possible problems/ security vulnerabilities in a web server, such as:

- Server and software misconfigurations;
- Default files and programs;
- Insecure files and programs;
- Outdated servers and programs;
- Pointers to lead a human tester to better manual testing;

Docker image created from the following repository: `https://github.com/sullo/nikto`

Sqlmap

Figure 37: SQLmap logo

Sqlmap is a free and open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers.

It comes with a powerful detection engine, many niche features for the penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

Some of sqlmap features are:

- Full support for many of the most used databases.
- Full support for six SQL injection techniques: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band.
- Support to directly connect to the database without passing via a SQL injection, by providing DBMS credentials, IP address, port and database name.
- Support to enumerate users, password hashes, privileges, roles, databases, tables and columns.

- Automatic recognition of password hash formats and support for cracking them using a dictionary-based attack.
- Support to dump database tables entirely, a range of entries or specific columns as per user's choice. The user can also choose to dump only a range of characters from each column's entry.
- Support to search for specific database names, specific tables across all databases or specific columns across all databases' tables.
- Support to download and upload any file from the database server underlying file system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
- Support to execute arbitrary commands and retrieve their standard output on the database server underlying operating system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
- Support to establish an out-of-band stateful TCP connection between the attacker machine and the database server underlying operating system.

Docker image created from the following repository: `https://github.com/sqlmapproject/sqlmap`

8.2.2. Functionalities

Here we will explore how the Worker's main functionalities were implemented.

8.2.2.1 Docker Registry

In order to have all the tool images available at any point in any worker, we have a local docker registry that stores all of them. When any worker wants to use a specific tool, just needs to pull from the registry and it is ready to be used:

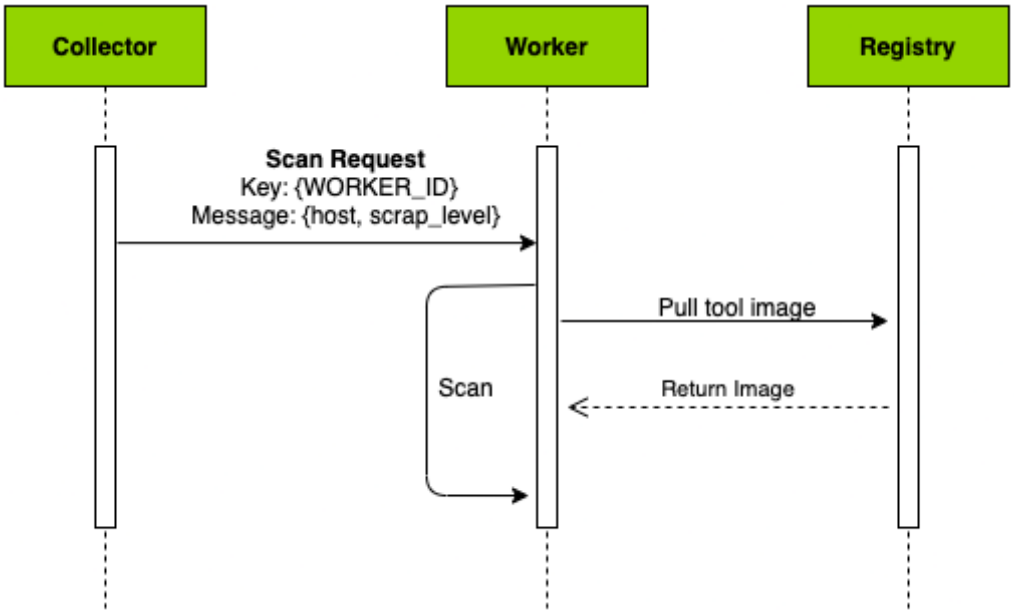


Figure 38: Registry protocol

8.2.2.2 Parsing

Each tool returns very different outputs. In order to surpass this problem, we have a parser for every tool that standardize each tool's output so that we can expect the same JSON structure at every scan.

For example, the Nmap tool only supports XML format in the output. Because of this, we had to convert it to JSON, while standardizing the output:

```
<port protocol="tcp" portid="22"><state state="open" reason="syn-ack" reason_ttl="0"/><service name="ssh" product="OpenSSH" version="7.6p1 Ubuntu 4ubuntu0.3" extrainfo="Ubuntu Linu
x; protocol 2.0" ostype="Linux" method="probed" conf="10"><cpe>cpe:/a:openbsd:openssh:7.6p1</cpe><cpe>cpe:/o:linux:linux_kernel</cpe></service><script id="ssh-hostkey" output="&#xa
;  2048 3e:97:e9:72:bb:9d:66:4c:17:60:1d:10:a7:a1:8c:2f (RSA)&#xa;  256 3f:0c:62:f8:27:d4:dd:65:16:7f:79:a5:1e:ff:18:5a (ECDSA)&#xa;  256 f3:cf:a2:73:d1:48:c0:75:ca:34:e4:bb:dc:0f:
a5:11 (ED25519)"><table>
<elem key="bits">2048</elem>
<elem key="fingerprint">3e97e972bb9d664c17601d10a7a18c2f</elem>
<elem key="key">AAAAB3NzaC1yc2EAAAADAQABAAABAQDAROFB07DUnu+pzJaAdqHdRvePmcUMR2SaWFO7TzQambRHrnTR39CQzG+HEjyX8Qo8TKHGhLsq8asGmbuwXorfsJ7jzt85qwUaE5+EhKm9UKeryKqKbHtMkX8kwgVlOS0N4Yqz
cGJHqT62Uzf+tLlVQIpRchI+Zwja43rHyIIVoeY3bTfxGxQhX3Bwx8xjrP3v9vv3ENlJgrR1FdrdDFKgrVCH2DYVqJHjlfQHwP4BRgC6sFnO6qcNM1SRRRnUn7gA85B0xKCjrzj9T6fPCP/15johNTd7K3t4sCuqRv4i8yKIoxAQUHhCvzbN
c739lmceGILUN4atcvoLhxs3IHGP</elem>
<elem key="type">ssh-rsa</elem>
</table>
<table>
<elem key="bits">256</elem>
<elem key="fingerprint">3f0c62f827d4dd65167f79a51eff185a</elem>
<elem key="key">AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBDATtaNWuLtrUcEbnOko8dnQeuEbeoDzNzqxFMzLPXeBfLDKIoNWta7sYPTU8slJT7/0DH7a0EIGvFWVRjRMkbM=</elem>
<elem key="type">ecdsa-sha2-nistp256</elem>
</table>
<table>
<elem key="bits">256</elem>
<elem key="fingerprint">f3cfa273d148c075ca34e4bbdc0fa511</elem>
<elem key="key">AAAAC3NzaC1lZDI1NTE5AAAAIJpEcifz7XYmuANGC0DrvucYt36slUJFOQ7hN021F5my</elem>
<elem key="type">ssh-ed25519</elem>
</table>
</script></port>
```

Figure 39: Nmap XML output example

```
  "ports": [
    {
      "protocol": "tcp",
      "id": "22",
      "name": "ssh",
      "product": "OpenSSH",
      "version": "7.6p1 Ubuntu 4ubuntu0.3",
      "os": "Linux",
      "method": "probed",
      "script": {
        "id": "ssh-hostkey",
        "output": "\n  2048 3e:97:e9:72:bb:9d:66:4c:17:60:1d:10:a7:a1:8c:2f (RSA)\n  256 3f:0c:62:f8:27:d4:dd:65:16:7f:79:a5:1e:ff:18:5a (ECDSA)\n  256 f3:cf:a2:73:d1:48:c0:75:ca:34:e4:bb:dc:0f:a5:11 (ED25519)",
        "keys": [
          {
            "bits": "2048",
            "fingerprint": "3e97e972bb9d664c17601d10a7a18c2f",
            "key": "AAAAB3NzaC1yc2EAAAADAQABAAABAQDAROFB07DUnu+pzJaAdqHdRvePmcUMR2SaWFO7TzQambRHrnTR39CQzG+HEjyX8Qo8TKHGhLsq8asGmbuwXorfsJ7jzt85qwUaE5+EhKm9UKeryKqKbHtMkX8kwgVlOS0N4YqzcGJHqT62Uzf+tLlVQIpRchI+Zwja43rHyIIVoeY3bTfxGxQhX3Bwx8xjrP3v9vv3ENlJgrR1FdrdDFKgrVCH2DYVqJHjlfQHwP4BRgC6sFnO6qcNM1SRRRnUn7gA85B0xKCjrzj9T6fPCP/15johNTd7K3t4sCuqRv4i8yKIoxAQUHhCvzbNc739lmceGILUN4atcvoLhxs3IHGP",
            "type": "ssh-rsa"
          },
          {
            "bits": "256",
            "fingerprint": "3f0c62f827d4dd65167f79a51eff185a",
            "key": "AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBDATtaNWuLtrUcEbnOko8dnQeuEbeoDzNzqxFMzLPXeBfLDKIoNWta7sYPTU8slJT7/0DH7a0EIGvFWVRjRMkbM=",
            "type": "ecdsa-sha2-nistp256"
          },
          {
            "bits": "256",
            "fingerprint": "f3cfa273d148c075ca34e4bbdc0fa511",
            "key": "AAAAC3NzaC1lZDI1NTE5AAAAIJpEcifz7XYmuANGC0DrvucYt36slUJFOQ7hN021F5my",
            "type": "ssh-ed25519"
          }
        ]
      }
    },
```


Figure 40: Nmap output after being standardized

As we can see, the first Figure 39 is the raw output from the nmap tool. After getting that, a function is called (in this example, *nmap_converter* in the *nmap_converter.py* file) that will convert XML to JSON and mold the output into a standard structure that will always be the same, returning something like the second figure 40. Then, the output from all the tools is concatenated and sent to the Collector.

8.3. Collector

The Collector is responsible for multiple jobs in our system such as:

- Distributing the scanning tasks between Workers;
- Scheduling the scanning tasks based on the respective host scan periodicity;
- Receive, process and store information about the logs received from a Worker as result of a host scan;
- Verifying the availability of the Workers through heartbeat messages;
- Sending notifications to the owners or subscribers of the host.

8.3.1. Architecture

In this section it is given an overall architecture of the Collector component.

8.3.1.1 General

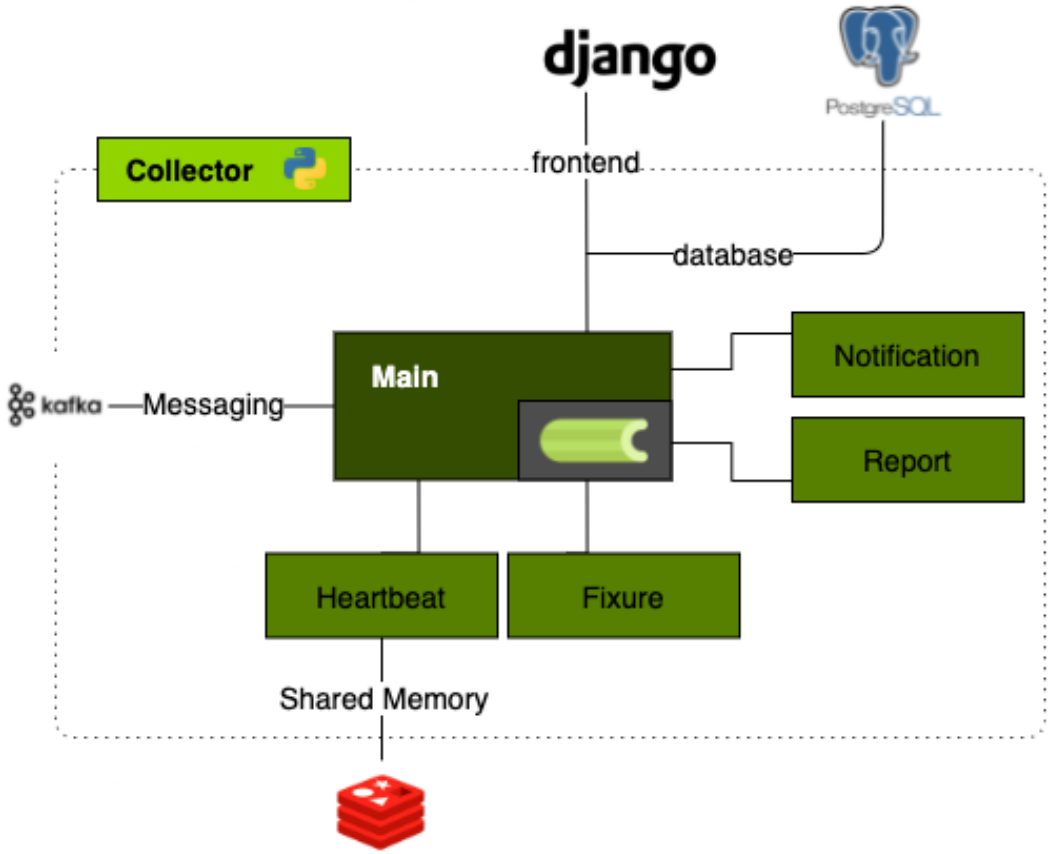


Figure 41: Collector's General Architecture

Collector has it's main component which handle all communications via Kafka, since both Frontend and Workers publish/consume messages from Collector. All requests are received by Kafka's message queue divided by topics: 'INIT','SCAN_REQUEST','FRONTEND','LOG', 'HEARTBEAT' and 'UPDATE'. Among these, 'INIT', 'SCAN_REQUEST', 'FRONTEND' and 'UPDATE' are handled by Main, 'LOG' by Fixure, Notification and Report, and 'HEARTBEAT' is handled by Heartbeat module.

It was also needed some other tools like PostgreSQL, Kafka and Redis to ensure the database, communication and threaded shared memory systems, respectively.

8.3.1.2 Technologies

In this section we will explore the technologies used in the Collector component.

Celery

Celery [5] is an open-source distributed Task Queue implemented in Python. It is highly maintained with almost a thousand contributor's on its GitHub repository [6].

Figure 42: Celery Logo

Celery supports scheduling of tasks, where we can specify a time to run a task or create periodic tasks for recurring events based on simple time intervals or more complex ones such as solar schedules. This feature was specially important in the scheduling of scans and heartbeat messages.

For this project it was used Celery's Python library [22], in its most recent version 5.1.1

RabbitMQ

RabbitMQ [23] is an open-source message broker widely used. It has good documentation with many tutorials which cover the basics of creating messaging applications.

RabbitMQ supports multiple programming languages, Python included.

Figure 43: RabbitMQ Logo

In this project RabbitMQ was used as a message transport required for the usage of Celery.

The RabbitMQ software used was the most recent alpine official docker image available in Docker Hub [13].

Redis

Redis [24] is an open source in-memory database that supports a variety of data structures.

Redis has multiple clients for support in multiple programming languages with Python being one of them.

Figure 44: Redis Logo

In this project Redis was used for the heartbeat messages as a cache of the workers who succeeded in replying and therefore are active.

For this project it was used bitnami/redis:6.2 docker image available in Docker Hub [12].

Zookeeper & Kafka

Apache Kafka is an open-source streaming platform which can handle a lot of data per unit of time. It was originally developed at LinkedIn as a messaging queue, but now Kafka is much more than a messaging queue. It is a powerful tool for working with data streams and it can be used in many use cases.

It also has low latency, which allows for the processing of data in a real-time mode.[30]

Figure 45: Kafka Logo

We use Kafka as main component for inter-module communication. There are a few topics that describe how each message will be passed:

- INIT: topic used to ensure the signup of new workers trying to connect to collector. The first message is from Worker with a random numeric 12bit message and the return from Collector will be the Worker's id.
- FRONTEND: from this topic we can expect two types of message. Or it is for 'SCAN' or 'UPDATE', which corresponds to an 'as soon as possible' scan and to an update to worker's configuration, respectively.
- LOG: when Worker sends the LOG from the scan, it is read by Collector and generated a Report. After the report generated, it is send a notification to the host owner with this same report.
- HEARTBEAT: this is the way to get to know which Workers are online and which are not. There is an interval of 300 seconds between messages containing a request to Workers so it can send a reply to the Collector informing that it's alive.
- SCAN_REQUEST: topic Collector uses to send a request to workers start scanning a host. This is used in two contexts, when machine have scheduled a scan and when Collector received from topic 'FRONTEND' a SCAN.
- UPDATE: topic with the purpose of updating Worker's address list. This communication starts at topic 'FRONTEND' asking for an 'UPDATE'. Then Collector will send a message to the respective

worker with the updated address list the worker shall scan.

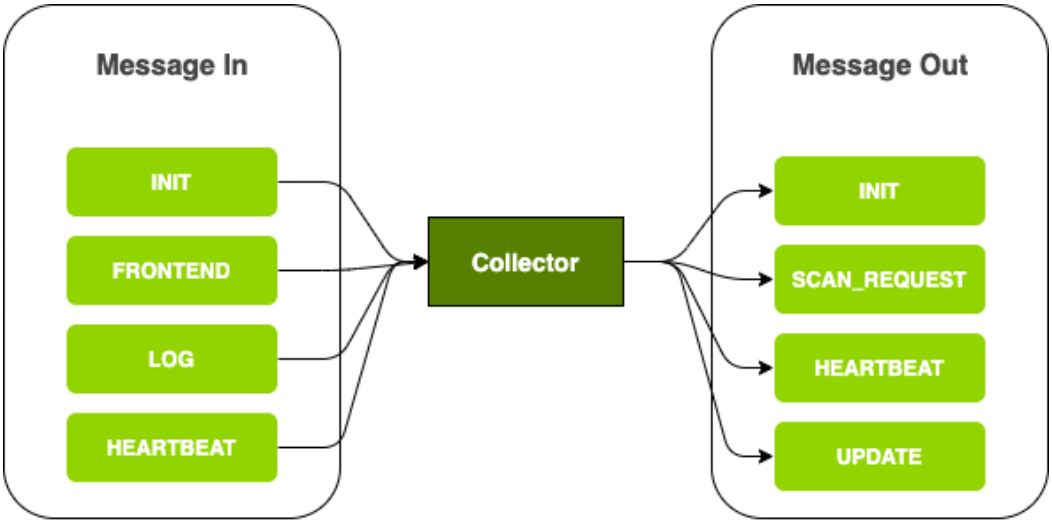


Figure 46: Messaging Model

8.3.2. Functionalities

Here we will explore how the Collector's main functionalities were implemented.

8.3.2.1 Worker Register

In order for the Collector to send scan requests to a worker it first needs to know that the worker exists and is available.

In figure 48 bellow we can observe the messaging protocol in order to register a worker.

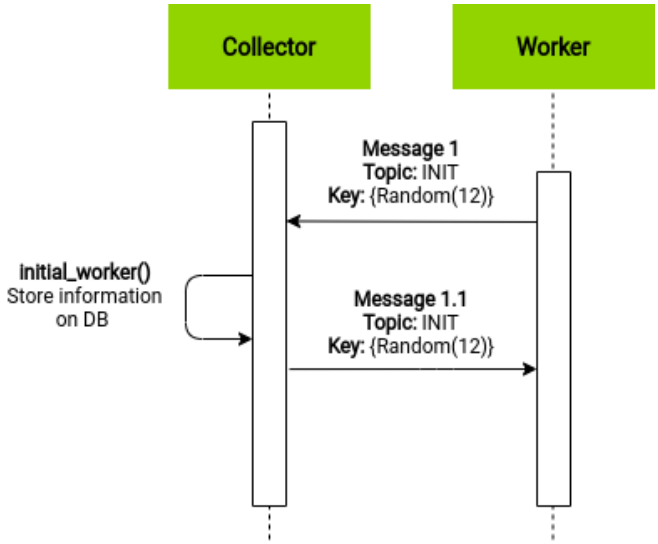


Figure 47: Signup Worker Protocol

First the Worker publishes an initial message to Kafka topic INIT with a key consisting of a 12 byte length random generated by the worker. This message may also contain a list with the IP's or DNS of hosts to be assigned to the worker.

When the Collector consumes the INIT message he will add the worker to the database as well as its hosts if they do not exist already. Afterwards it will publish a INIT message with the same key so that the worker can know that this message is designated to it. The message will contain the worker id from the database. From now on all the messages published by the worker will have as keys its id, as a mean of identification.

8.3.2.2 Heartbeat

User needs to know if the workers that were active are still available to continue working. So, with an interval of 300 seconds, Collector sends a broadcast to all known Workers asking for a response. If there is an answer in 30 seconds, then it means Worker is available, otherwise Worker is considered down. Redis was needed as memory tool, since heartbeat is working in parallel with the rest of the system. So, it's stored on Redis a json-like list with all workers who respond to the message. In the end, after 30 seconds, all workers who aren't in that list are excluded from database and are considered offline.

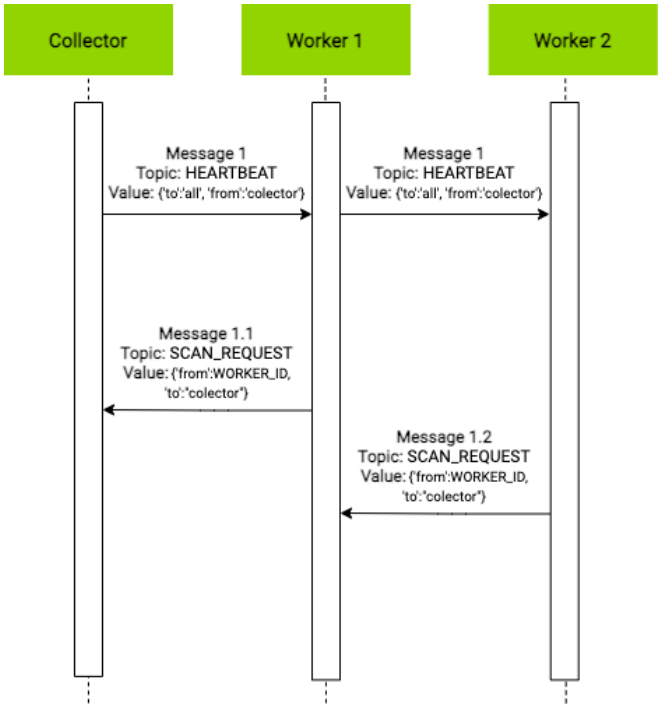


Figure 48: Heartbeat Protocol

8.3.2.3 Scan

The Collector can send a scan request to a worker in two different situations. When an user asks for a scan, figure 49, or when the Collector detects that a host needs to be scanned, figure 50.

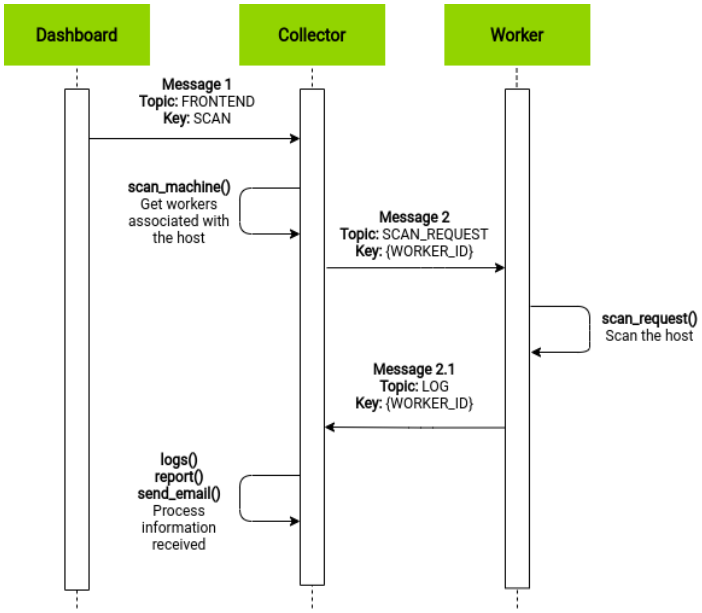


Figure 49: User Request Scan Protocol

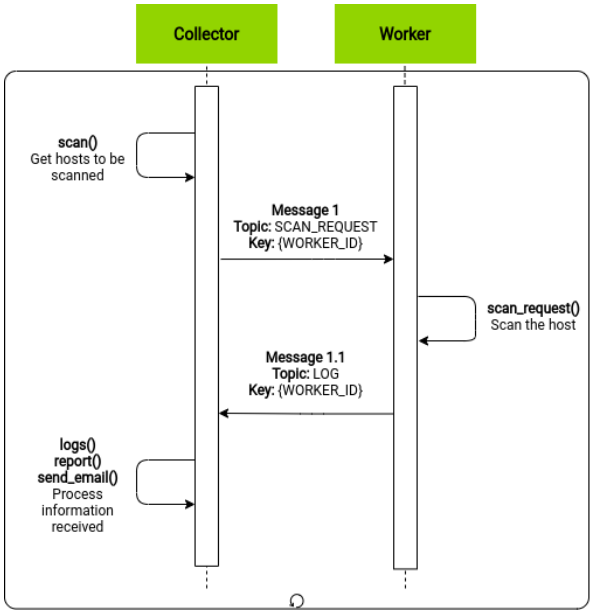


Figure 50: Periodic Scan Protocol

When an user requests an immediate scan the dashboard will publish a message to Kafka topic FRONTEND with key 'SCAN' containing the id of the host to be scanned. Like this, the Collector knows which host needs to be scanned and can publish a request message to the respective workers.

In order for the Collector to detect that a host needs to be scanned a periodic task *scan* was set using Celery. This task consists in retrieving all the machines from the database in which the *nextScan* date property of a machine is less than the current date and publishing the scan request message to the respective workers.

The scan request message is published to Kafka topic SCAN_REQUEST with the key corespondent to the assigned Worker id. The message contains the id of a host and its IP address or DNS if the IP is not in the database as well as the host scan level. With this information the Worker can start scanning the host.

After the scanning finishes the Worker will publish to Kafka topic LOG with the results of each tool used in the scan. This message contains the host id, allowing the collector to know which host was scanned by which Worker.

Upon consuming this message the Collector is now ready to generate the scan report and send a notification to all owners and subscribers of the host.

8.3.2.4 Report and Notification

Since Worker has finished its job, it sends the scan logs back to Collector who has the job of compose the logs into an useful report about the Host.

It's created an instance of Report's module. In the beginning, Collector is getting Host's state by checking the message returned by Worker and searching on database for the Host's instance. Then a new record is saved on database with the information related to this scan. This is useful as a log.

In the case of being successful, information related to Host's scan, such as IP address, DNS, OS and open ports scanned along with their software version are extracted. If different tools give different information results, the most frequent information is chosen. If the host detects that the OS is different from the previous scan, the update is stored in the database so as to appear highlighted in the main dashboard page.

Then, Collector will get the vulnerabilities, their risks and possible solutions. Depending on the tool and/or the type of vulnerability it might not be possible to retrieve a solution or associate a risk to a vulnerability. With zap tool the solution and the risk associated to a vulnerability are given, the risk ranges from 0 to 4 so the final vulnerability risk will be the risk returned by zap plus one. There are 3 different types of vulnerabilities in which we associated a risk:

- Software outdated - This vulnerability is detected by nikto when there is a more updated version for a software found. The risk level associated was 3.
- Certificate - This vulnerability is detected by certigo when the host certificate chain is invalid. The risk level associated was 3.
- Sql injection - This vulnerability is detected by sqlmap. Injections are very serious vulnerabilities being the first ones in the "OWASP Top 10 Application Security Risks" [19]. Therefore the risk level associated was the maximum of 5.

Having processed all the vulnerabilities found, the final risk level associated with a host is:

- 1 - If no vulnerabilities were found or all the vulnerabilities had a risk level of 1 and there were no more than 5 vulnerabilities detected.
- 2 - If no vulnerabilities were found with a risk level greater than 2 and there were no more than 10 vulnerabilities detected.
- 3 - If no vulnerabilities were found with a risk level greater than 3 and there were no more than 20 vulnerabilities detected.
- 4 - If no vulnerabilities were found with a risk level of 5 and there were no more than 50 vulnerabilities detected.
- 5 - If there was at least one vulnerability with risk level of 5 or more than 50 vulnerabilities were found.

All this information will be stored in the database in order to be displayed in the dashboard.

With the report completed the Collector starts a task which creates an instance of Notification-Sender's module and receives a list with the solutions found, the scanned host id as well as the number of vulnerabilities found.

There are three different types of notifications:

- Host down - This notification will be sent if all the tools could not reach the host thus considering it down. In the message there will be a warning in order for the user to check if its host is up (he might have not been aware of the situation).
- Host up with no vulnerabilities found - This notification will be sent if the number of vulnerabilities found is 0. The message will warn the user if its host scan level is small, meaning that it is possible for the host to have vulnerabilities that were not detected.

- Host up with vulnerabilities - This notification will be sent if the number of vulnerabilities found is greater than 0. The message will contain a list of solutions for some of the vulnerabilities found.

All messages contain the IP or DNS of the host, the time of the scan as well as a link redirecting to the host page containing the full report.

Users can receive notifications via two different ways with the default one being throughout the institutional email and the other via Microsoft Teams. The content of the notification is the same for both ways.

9. Results and discussion

9.1. User acceptance tests

9.1.1. Introduction

Due to the complexity of our system we have chosen to perform user acceptance tests with real users. To do so, we have created an online form with a step by step description of the main functionalities we wanted to test and sent it to every group of our course. This form is publicly available (in Portuguese) at `https://forms.gle/yvahogFSeNGBSVi2A`.

Because user acceptance tests should be performed in the final phase of testing, it collided with the due date of the project delivery, reason we believe justifies the low number of entries. We only got 4 submissions.

9.1.2. Result types

From the 4 groups that participated in this test, there were two different outcomes.

9.1.2.1 Host not found

Our system marked two of the four hosts as down. This means IT private network, inaccessible to our workers, that were placed in the UA's network.

These failures can be misleading, originating false negatives when the host is flagged as down when he is actually available. On the other hand this can be helpful to identify hosts that have unexpectedly failed, as the owner is immediately notified in this situations and can quickly address the issue.

9.1.2.2 General example

Our system was able to scan the other two hosts. We will focus on the result of the host `https://deti-eel.ua.pt`, that was successfully found and scanned.

Starting with the basic information shown in Figure 51, we can see the host name, OS and computed risk level. On te right there is also information about the SSL certificate. All these fields were corroborated by the owner group. The certificate error was expected, as they stated that it was only issued for development purposes.

Moving to Figure 52, we can see the list of the open ports identified and respective services and their versions. In this case, all ports detected have been confirmed by the group. They even referred that we had identified a port that they did not expect to be exposed. This is a really positive outcome, as it meets our project's purpose, to raise awareness regarding what is exposed to the outside world. However, there was a open port that was not identified: 8883, used by Message Queuing Telemetry Transport.

Finally, on Figure 53 its shown the report of the scan. Here we can see the multiple vulnerabilities found, their risk and location.

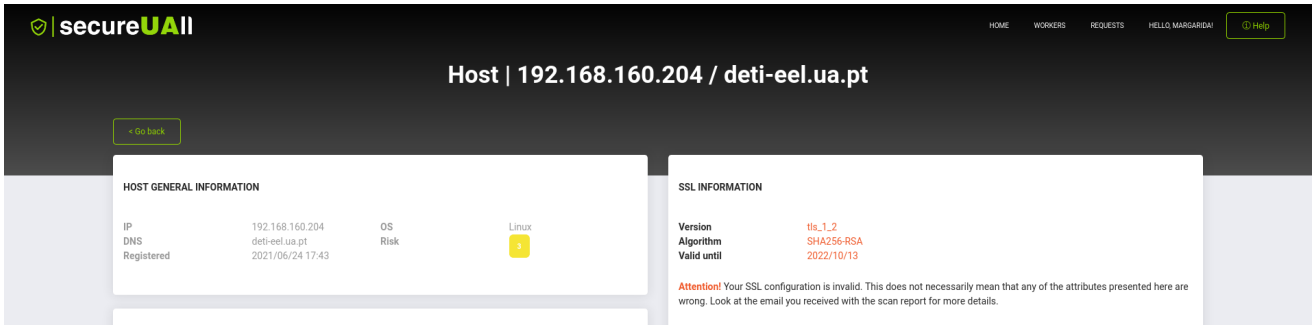


Figure 51: Generic scan info

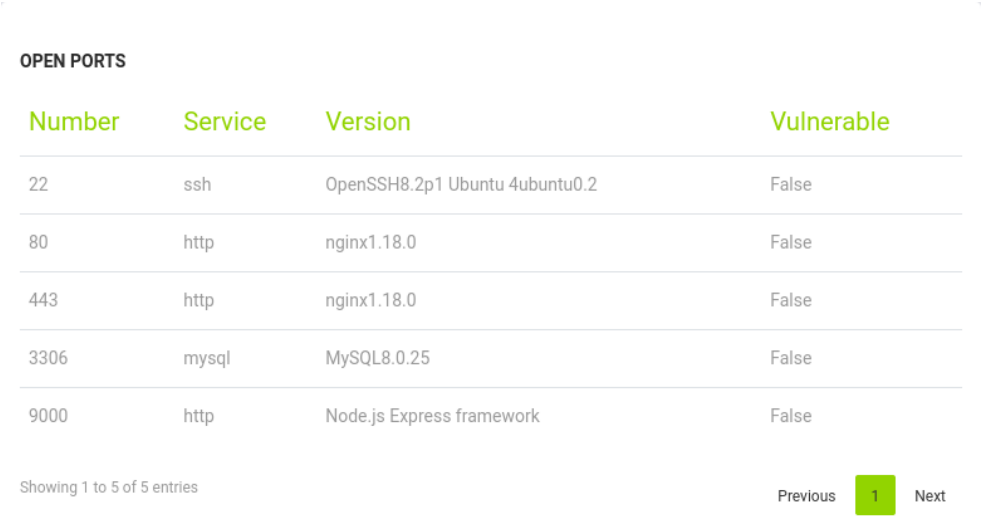


Figure 52: Generic scan ports

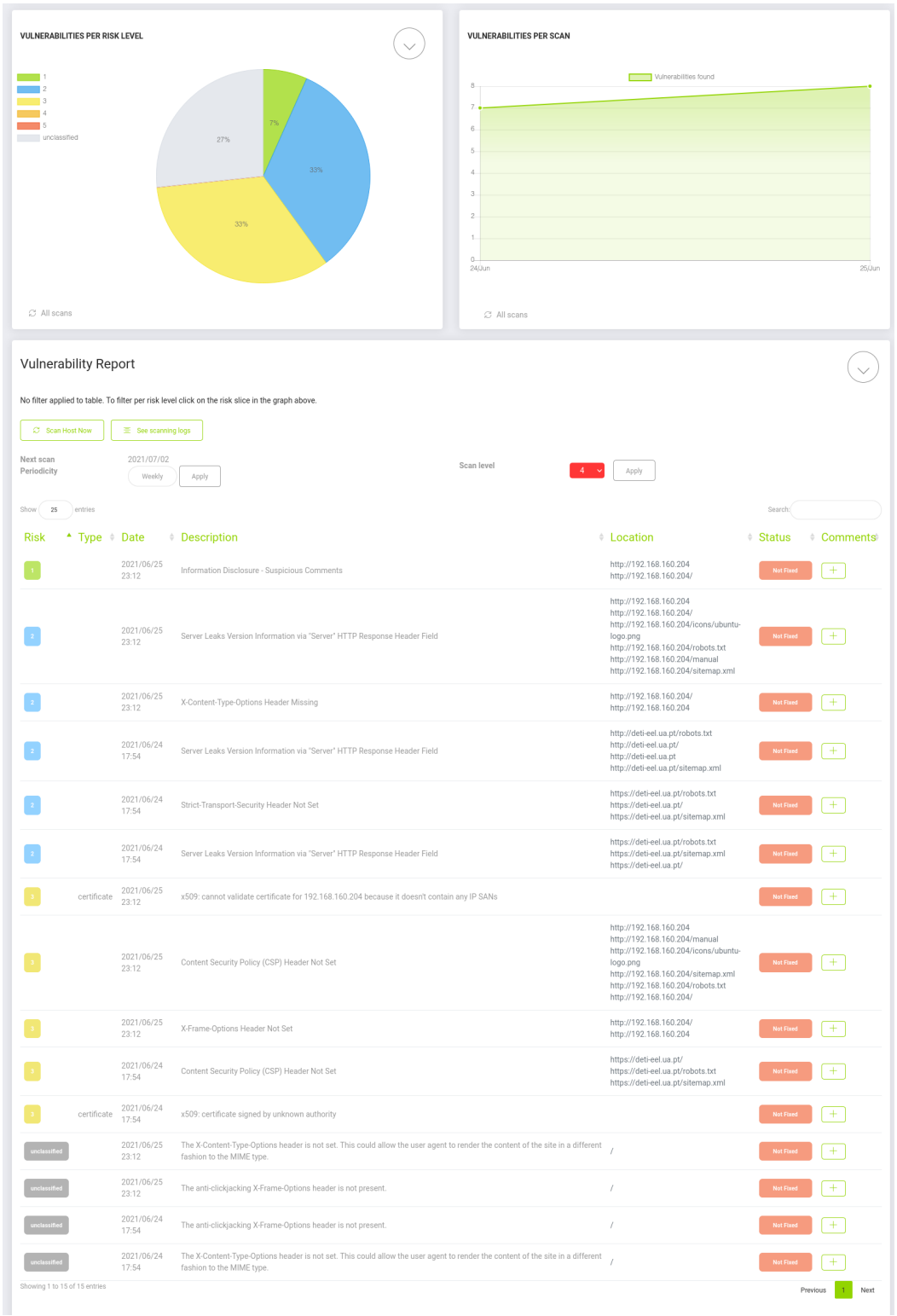


Figure 53: Generic scan vulnerabilities

9.1.3. Usability results

Besides evaluating our system scanning accuracy, we have also made some questions about its usability. Most of them were to evaluate in a scale of 1 to 5 how easy it was to find some information or to perform a certain action. In these questions (there were 3), we got an average classification of 4.66, with the lowest being 4.

For some tasks that we were not sure about the perceived difficulty, we defined multiple-choice questions with options that would allow us to have a clearer understanding about the difficulties felt by the users. The user feedback was also very positive. In the question "Could you access the host page without reading the help?" [from the dashboard], most of the respondents answered "yes, at first try". We have also asked about "How difficult was it to find the information to answer the previous questions?" [in the host page], having all the answers been that it was "intuitive" and they "had no difficulties". Figures 54 and 55 show the graphical results for these two questions.

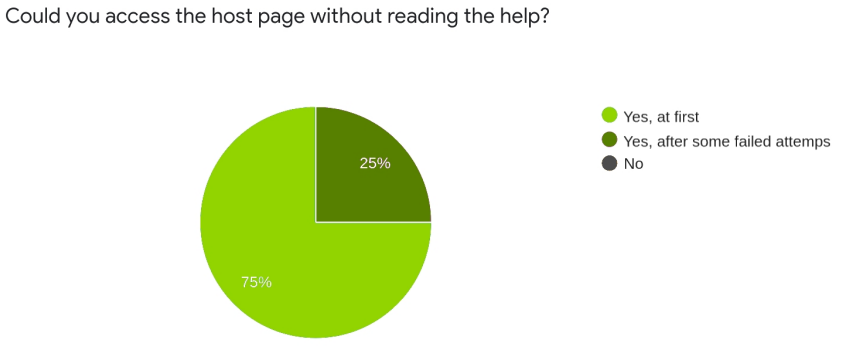


Figure 54: Question about the machine page access

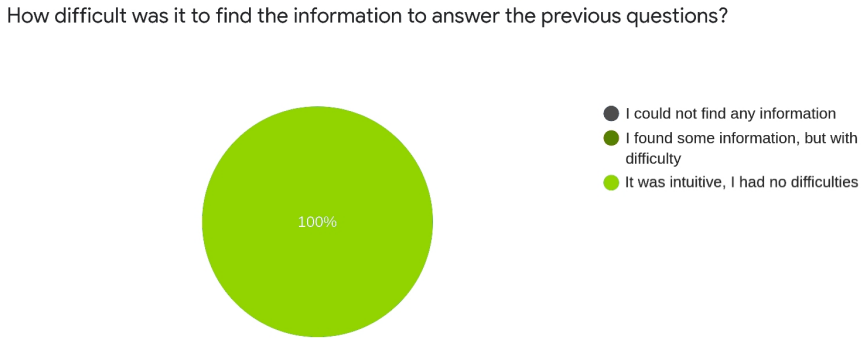


Figure 55: Question about the machine page information

9.2. Workflow tests

9.2.1. Specific vulnerabilities

9.2.1.1 Certificate detection

Our system is able to detect multiple cases regarding the host's certificate, such as the existence of it, its validation date, the chain and the CA that signs it. For example, the host 'app.gov.pt', a government website, has some serious problems with its certificate.

We have chosen this host because it has a peculiar problem: the root CA has been recently revoked. Camerfirma has several issues identified since 2017 [31] and Chrome and Firefox have removed their trust in its issued certificates in the latest versions [7].

Figure 56: Certificate alert

Figure 57: Certificate vulnerability

As we can see, in the host's page, an alert is shown at the top of the page and then the vulnerability is displayed in the table. All the certificate issues are evaluated with 3 as the risk level.

9.2.1.2 Outdated Software detection

There is a detection for outdated software in the host. Therefore, if there is any software that isn't in its latest version, the system will detect and display it. For example, scanning the 'deti-cismob.ua.pt' host, we found some outdated software/libraries:

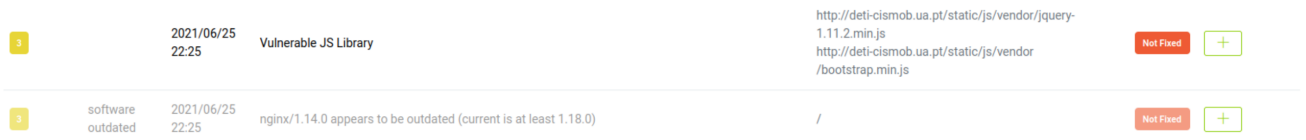


Figure 58: Outdated Software

As we can see, when a software is detected as outdated and the vulnerability is shown in the table.

9.2.1.3 Sql Injection detection

Because this is a very serious and compromising vulnerability, it is natural that almost every host does not show any problems of SQL injections. However, we tested with some specific hosts that are vulnerable to it and found some interesting results with SQLmap. For example, the host 'test-php.vulnweb.com' is vulnerable to a lot of SQL Injections:

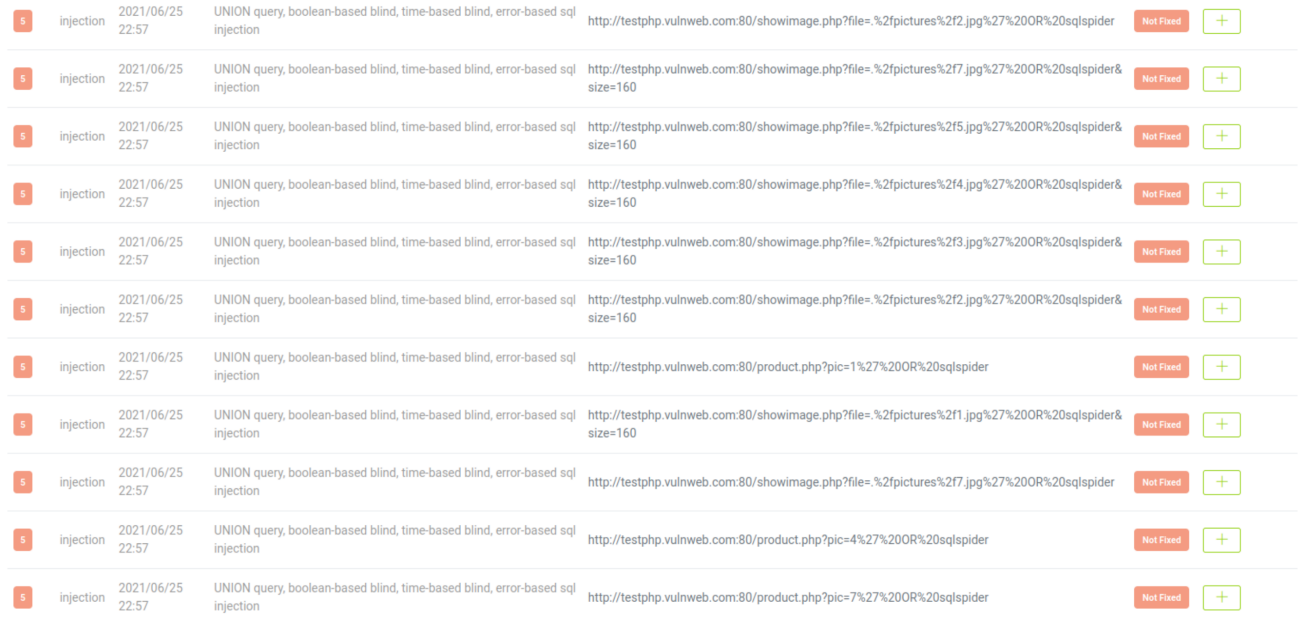
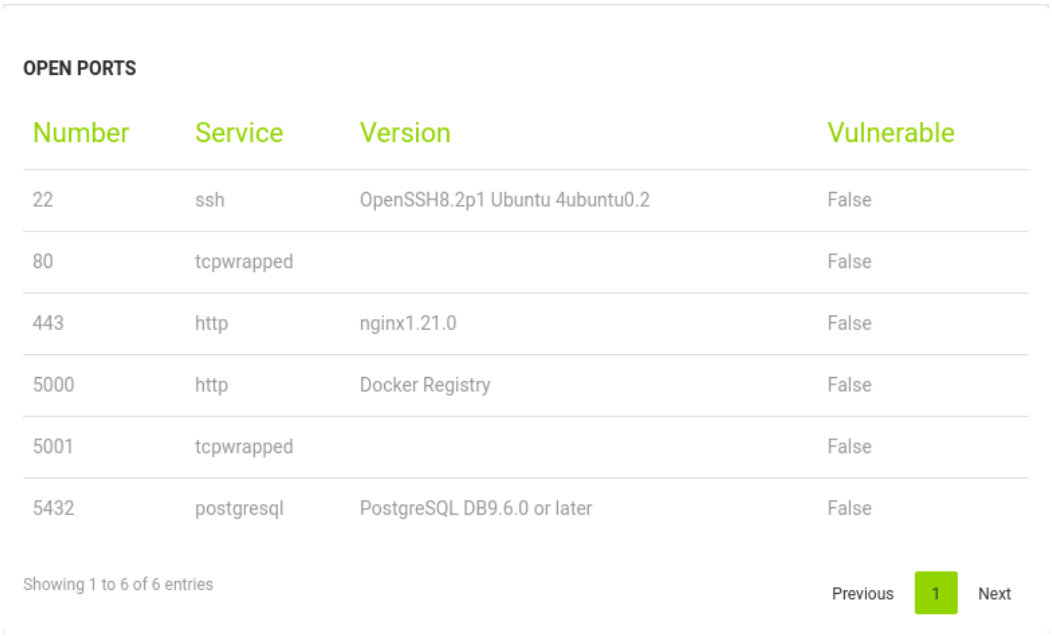


Figure 59: SQL Injections

9.2.2. Ports and services identification

This is our most common feature because it is performed on every scanned host. The goal is to make a non-intrusive scan to retrieve every open port and its corespondent services. For example, we tested our own host to check for open ports and services:



OPEN PORTS

Number	Service	Version	Vulnerable
22	ssh	OpenSSH8.2p1 Ubuntu 4ubuntu0.2	False
80	tcpwrapped		False
443	http	nginx1.21.0	False
5000	http	Docker Registry	False
5001	tcpwrapped		False
5432	postgresql	PostgreSQL DB9.6.0 or later	False

Showing 1 to 6 of 6 entries

Previous 1 Next

Figure 60: Ports and Services

All the ports and services detected in our host are valid, the system detected exactly what we had. We can see the port numbers, services, versions and if is vulnerable or not.

9.2.3. Solutions

Although the solutions are not shown in the table, the owner receives a notification with the full report and the generic solutions for each vulnerability, when available:

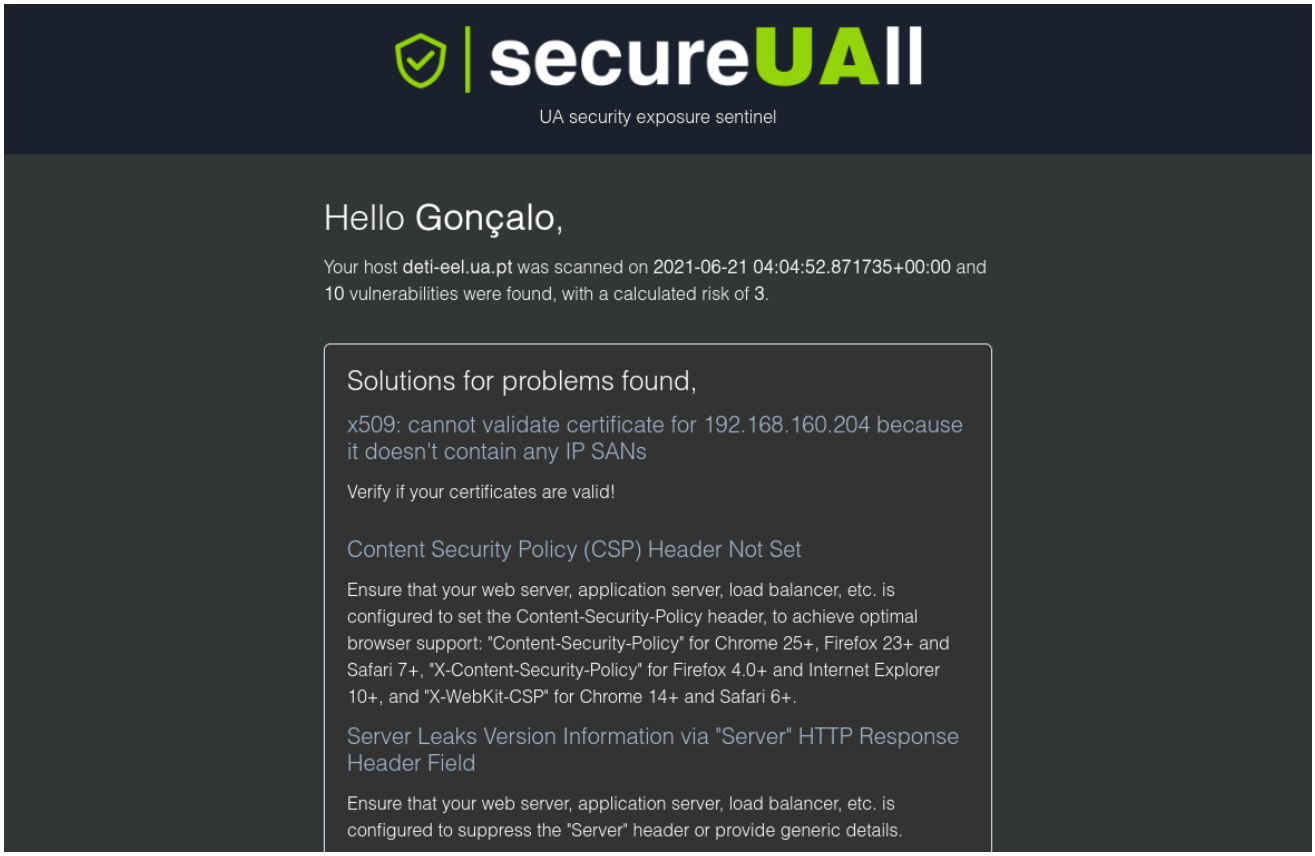


Figure 61: Email with solutions

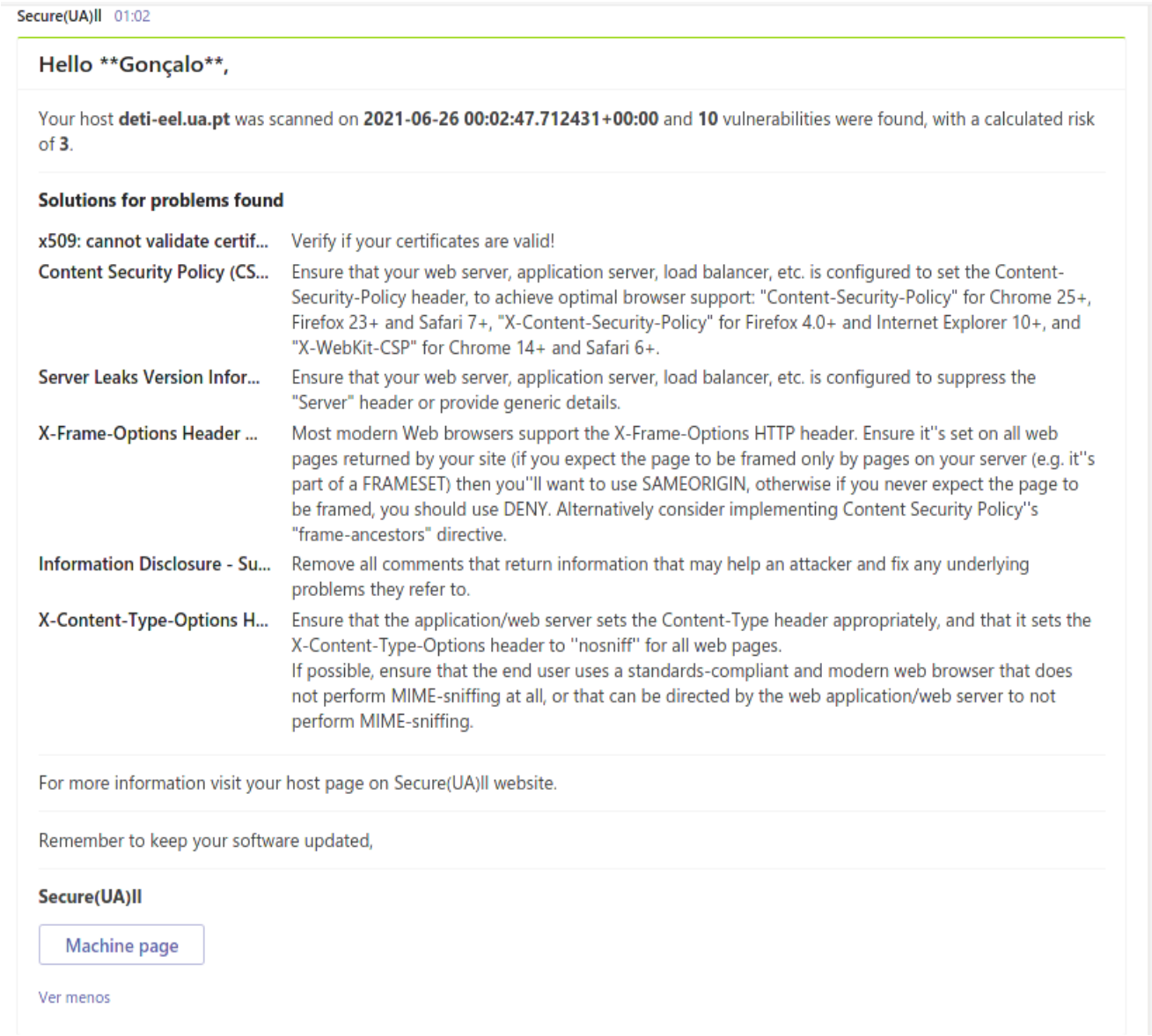
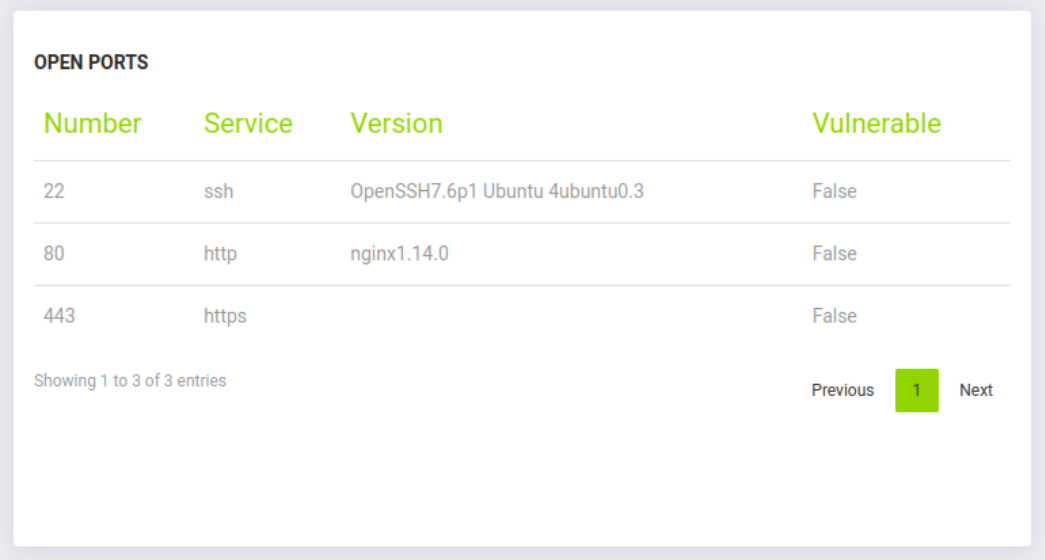


Figure 62: Message in Microsoft Teams with solutions

9.3. Global results

We decided to compare our system's results to some of the state of the art applications. For example, when scanning the host 'deti-cismob.ua.pt', we got the following ports and services:



OPEN PORTS

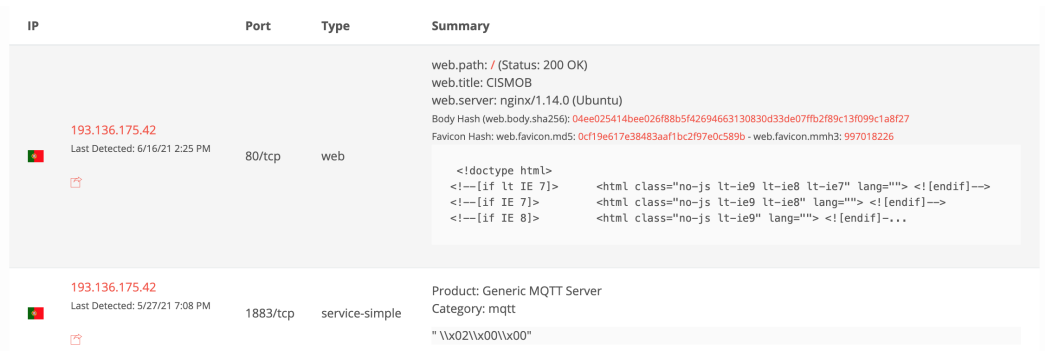
Number	Service	Version	Vulnerable
22	ssh	OpenSSH7.6p1 Ubuntu 4ubuntu0.3	False
80	http	nginx1.14.0	False
443	https		False

Showing 1 to 3 of 3 entries

Previous 1 Next

Figure 63: Ports and Services from secureUAll

By scanning the same host on some of the SOA applications, BinaryEdge and Shodan, we get the following result:



IP	Port	Type	Summary
193.136.175.42 Last Detected: 6/16/21 2:25 PM	80/tcp	web	web.path: / (Status: 200 OK) web.title: CISMOB web.server: nginx/1.14.0 (Ubuntu) Body Hash (web.body.sha256): 04ee025414bee026f88b5f426946631308 30d33de07ffb2f89c13f099c1a8f27 Favicon Hash: web.favicon.md5: 0cf19e617e38483aaf1bc2f97e0c589b - web.favicon.mmh3: 997018226 <!doctype html> <!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang=""> <![endif]--> <!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang=""> <![endif]--> <!--[if IE 8]> <html class="no-js lt-ie9" lang=""> <![endif]-...
193.136.175.42 Last Detected: 5/27/21 7:08 PM	1883/tcp	service-simple	Product: Generic MQTT Server Category: mqtt " \\x02\\x00\\x00"

Figure 64: Ports and Services from BinaryEdge

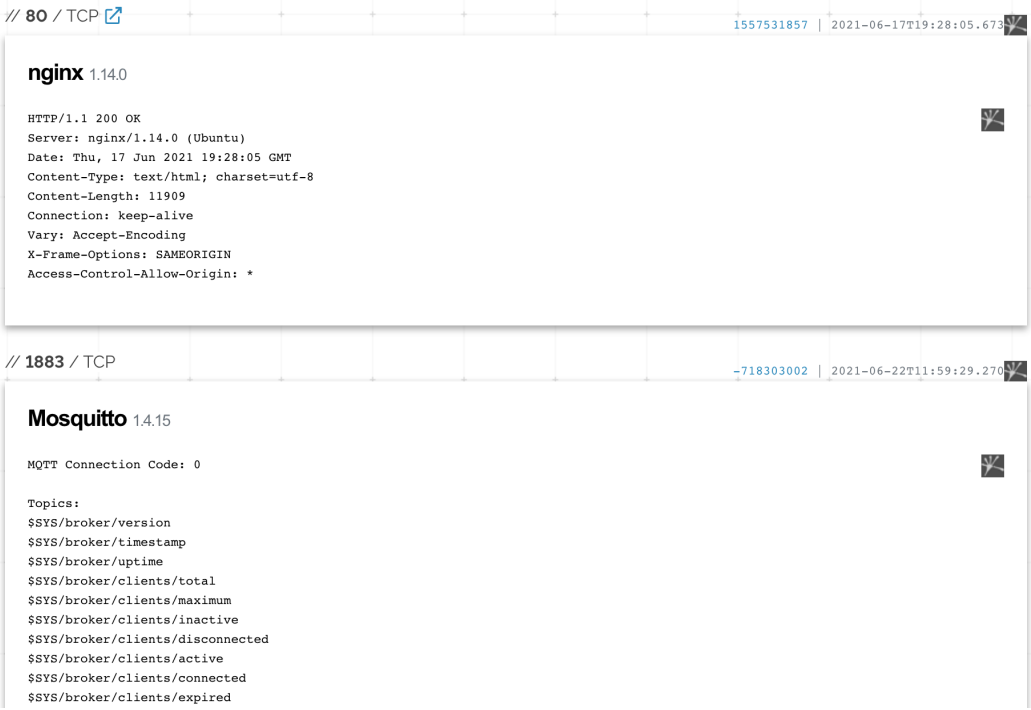


Figure 65: Ports and Services from Shodan

As we can easily see, our system detected the ports 22, 80 and 443. In the other hand, the SOA applications detected 80 and 1883. Although our system detected more ports, we have to take in consideration that our application is hosted inside the UA's network, which can affect the detection of open ports.

Another important point is that these applications only showed the ports and services, while our system detects vulnerabilities, operative systems, certificates, etc.

10. Conclusion and Future Work

10.1. Conclusion

SecureUAll is a system developed for scraping University of Aveiro's public domains, that can monitor a wide range of systems, detecting and alerting to potential security issues, accomplished by the integration of free and open source tools, a tool manager and a dashboard that allows the assess of the the domains current situation. Based on the outlines of the project proposal, the final product achieved fulfills the objective of identifying potential problems and vulnerabilities.

The development process of this project provided us with a deepening in design methodology in computer engineering, applying and integrating skills learned throughout our cycle of studies, foster skills in teamwork, get in touch with real-world issues and strengthen communication capabilities.

10.2. Future Work

Some improvements can be done to our system that were not implemented due to lack of time. In this section we will explore some of these extra features and why we think they would add value to our system.

Workers in multiple networks - Inside UA there are multiple sub-networks that our current system can not reach. One example is the already mentioned IT. In order to tackle this issue, multiple workers could be placed in different networks, to improve the system network coverage and solve this problem. This would also be useful to better understand how exposed the hosts are from the perspective of different locations. This would also allow for better scalability and fault tolerance due to a greater redundancy.

Improved method of deprecated software detection - Currently, our system can only detect deprecated software via Nikto. A major improvement would be to identify deprecated software for all the services detected. This could be achieved by having a constantly updated database with the latest software versions. Whenever a service was detected, the version would be compared with the one on the database and a deprecated software would be flagged. This would also allow for better solution proposals.

Extra tools integration - The job of adding scanning tools is a continuous one, that should follow the state of the art.

Associate scanning configuration to worker - This functionality was proposed by the subject leading teacher and was considered valid by us. Currently, the system only allows to configure the scanning level and periodicity per machine. This can be a heavy task for the administrator to do manually. With this approach it would be possible to associate this configuration to the worker, being applied to every host scanned by it.

Massive host creation with owner associated - Currently, both ways of associating hosts with workers only allow for the host name insertion, being the owner association a manual process only allowed in the machine page. In a future version, it would be more user-friendly to allow the massive input of hosts paired with the correspondent owner, reducing the effort for the system administrator.

Initial network scan - When the system is started, it has to wait for a manual host insertion to be populated. With this new paradigm, we could automate this process, by performing a periodic network

scan that would identify the network topology. More than automating a manual process, this change would allow to identify unknown hosts and follow the network evolution.

11. References

References

[1] Universidade de Aveiro. *Manual de Identidade e Normas Gráficas*. https://www.ua.pt/file/64049. Accessed on 2021-06-25.

[2] Richard Jones Benno Rice and Jens Engel. *Behave*. https://behave.readthedocs.io/en/stable/. Accessed on 2021-06-25.

[3] BinaryEdge. *BinaryEdge*. https://binaryedge.io/. Accessed on 2021-06-12.

[4] Bootstrap. *Popovers*. https://getbootstrap.com/docs/4.6/components/popovers/. Accessed on 2021-06-25.

[5] Celery. *Celery 5.1.1 Documentation*. https://docs.celeryproject.org/. Accessed on 2021-06-25.

[6] Celery. *Celery Source Code*. https://github.com/celery/celery. Accessed on 2021-06-25.

[7] CNCS. *CNCS aconselha entidades à substituição dos certificados TLS/SSL Camerfirma*. https://www.cncs.gov.pt/recursos/noticias/cncs-aconselha-entidades-a-substituicao-dos-certificados-tlsssl-camerfirma/. Accessed on 2021-06-25.

[8] Django. *Applications*. https://docs.djangoproject.com/en/3.2/ref/applications/. Accessed on 2021-06-25.

[9] Django. *Databases*. https://docs.djangoproject.com/en/3.2/ref/databases/. Accessed on 2021-06-25.

[10] Django. *Models*. https://docs.djangoproject.com/en/3.2/topics/db/models/. Accessed on 2021-06-25.

[11] Django. *Testing in Django*. https://docs.djangoproject.com/en/3.2/topics/testing/. Accessed on 2021-06-25.

[12] Docker. *Bitnami Redis Docker Image*. https://hub.docker.com/_/rabbitmq/. Accessed on 2021-06-25.

[13] Docker. *RabbitMQ Docker Image*. https://hub.docker.com/_/rabbitmq/. Accessed on 2021-06-25.

[14] Figma. *Figma*. https://figma.com/. Accessed on 2021-06-25.

[15] Jimena. *How to pick the best colors for your presentation slides*. https://www.slidescarnival.com/how-to-pick-the-best-colors-for-your-presentation-plus-10-color-combination-ideas-to-get-you-started/14686. Accessed on 2021-06-25.

[16] Omnisint Labs. *University of Aveiro public domains*. https://sonar.omnisint.io/subdomains/ua.pt. Accessed on 2021-06-25.

[17] Mozilla. *Testing a Django web application*. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing. Accessed on 2021-06-25.

[18] Baiju Muthukadan. *Selenium with Python¶*. https://selenium-python.readthedocs.io/. Accessed on 2021-06-25.

[19] OWASP. *Top 10 Web Application Security Risks*. https://owasp.org/www-project-top-ten/. Accessed on 2021-06-25.

[20] PostgreSQL. *Character Types*. https://www.postgresql.org/docs/9.1/datatype-character.html. Accessed on 2021-06-25.

[21] PostgreSQL. *PostgreSQL*. https://www.postgresql.org/. Accessed on 2021-06-25.

[22] PyPI. *Celery PyPI*. https://pypi.org/project/celery/. Accessed on 2021-06-25.
[23] RabbitMQ. *RabbitMQ*. https://www.rabbitmq.com/. Accessed on 2021-06-25.
[24] Redis. *Redis*. https://redis.io/. Accessed on 2021-06-25.
[25] Shodan. *Shodan Monitor*. https://monitor.shodan.io/. Accessed on 2021-06-11.
[26] Shodan. *Shodan Search Engine*. https://beta.shodan.io/. Accessed on 2021-06-11.
[27] Yaco Sistemas. *djangosaml2*. https://pypi.org/project/djangosaml2/. Accessed on 2021-06-25.
[28] Spyse. *Spyse Internet Assets Search Engine*. https://spyse.com/. Accessed on 2021-06-12.
[29] Creative Team. *Now UI Dashboard*. https://www.creative-tim.com/product/now-ui-dashboard. Accessed on 2021-06-25.
[30] Upsolver Team. *pache Kafka Use Cases: When To Use It & When Not To*. https://www.upsolver.com/blog/apache-kafka-use-cases-when-to-use-not. Accessed on 2021-06-24.
[31] Mozilla Wiki. *CA:Camerfirma Issues*. https://wiki.mozilla.org/CA:Camerfirma_Issues. Accessed on 2021-06-25.