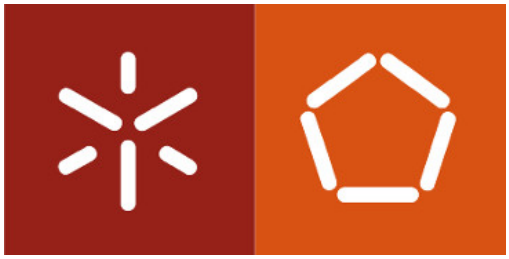

Universidade do Minho

Mestrado integrado em Engenharia Informática

Trabalho Prático 1

Diogo Pinto Ribeiro, A84442

Luís Pedro Barbosa Ferreira, A86265

Segurança de Sistemas Informáticos
4th Year, 1st Semester
Departmento de Informática

November 24, 2020

Contents

1 Abstract

In this report we analysed the *mID* in order to search for vulnerabilities that may affect it's operation. To do so, we used a mix of different strategies including Threat Modelling, Risk Analysis and CVE. We also included some suggestions to fix or mitigate the problems we listed. Given the constraints related to resources and time, we also specified which problems should be addressed first.

2 mID

The **mID** is a digital and mobile personal identification system. It's built with trustworthiness and privacy in mind. Some of the main requirements of this system consist in it being secure by design, trustworthy, interoperable, flexible and able to work while offline.

2.1 Entities Envolved

This system is composed by three entities:

- **mID Application**
- **Reader Application**
- **System Backend(Emitting Entity)**

3 Analysis

In this chapter we are going to apply different strategies in order to find security threats and vulnerabilities that this system may be exposed to. We are going to analyze each component of this system so that we can reduce complexity and use different strategies, since each component is independent. In our analysis we will also try to suggest possible solutions to mitigate problems.

3.1 mID Application

For this component of the system we will be using **Threat Modelling**, **focusing on Assets**. We decided that this approach is the best due to the overall state of the system, specially of this component, that is not yet finished. Alongside this strategy, we are going to use the **STRIDE** approach complemented with a **Data Flow Diagram**.

3.1.1 Focusing on Assets

When we are Focusing on Assets while searching for potential threats in our system, it is important to think about 3 topics: **Things attackers want**, **Things you want to protect**, and **Stepping stones to either of these.** Considering our mID Application, we answered these topics:

Things attackers want

- Steal personal information (Identity Theft for example)
- Disrupt/Sabotage service
- Change personal information

Things you want to protect

- Personal Information
- Personal Information(data) integrity
- Service Reliability/Trustworthiness

Stepping stones to either of these

- Encrypt local data
- Use secure authentication
- Unique and Secure Identifiers for all entities
- Safe and Unbreakable logging (for auditing)

3.1.2 System Modelling

The best way to understand what can go wrong with the mID Application is to understand what it does. To do so, we will be using a **Data Flow Diagram** to represent this part of the system as well as it's interactions.

We represented other elements of the system such as the Reader and Emitting Entity as external entities in order to better represent our communication with them, and, the local storage of the device was represented as a Data store.

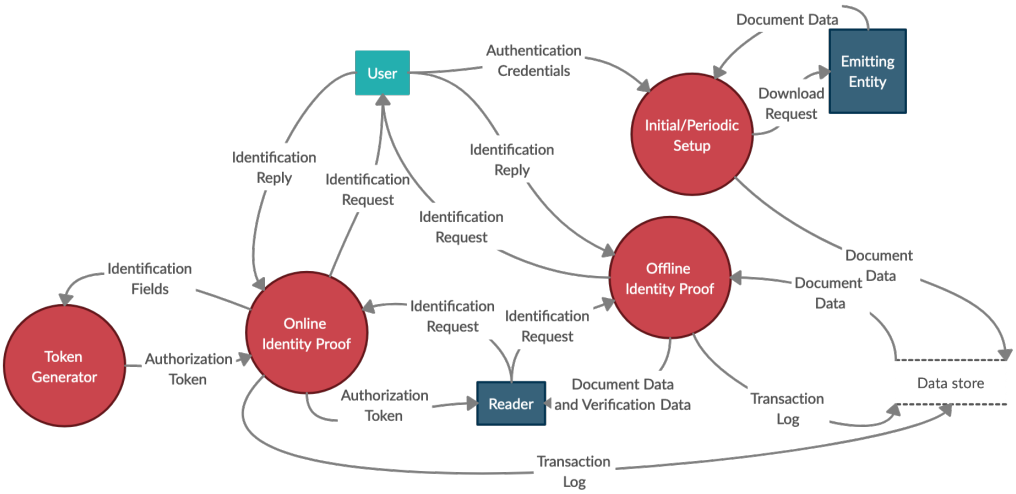

Figure 1: mID Application Data Flow Diagram

3.1.3 Finding Threats

With our Data Flow Diagram done, we are now able to analyze it and find threats in a more streamlined way. In order to find the relevant threats, we will be using the STRIDE methodology:

3.1.4 STRIDE

Spoofing

- The lack of User authentication may lead to an attacker posing as someone being able to download their personal data. This can be fixed by using a secure authentication system, such as **Two Factor Authentication**. The integration of this service is highly recommended.
- The communication with the backend without any kind of authentication allows an attacker being able to pose as an Emitting Entity. A good idea would be to implement a **Public Key Infrastructure** in our system. A PKI is a infrastructure that allow to identify users in a system. The idea is to have one or more trusted parties digitally sign documents certifying that a particular cryptographic key belongs to a particular user or device. The key can then be used as an identity for the user in digital networks.[11]

- The communication with the Reader is also vulnerable to spoofing. An attacker could pose as an trusted reader and start the connection process with a user without the user knowing that he is given his information to an attacker and not a trusted reader. This could be also be prevented with the use of a PKI[11] in our system. This way, the application has the means to check the signature of the reader.

Tampering

- The local storage of the device has sensitive information, which is a target to attackers. These can be external attackers or even the user of the application. An attacker may, for example, try to alter information stored. Encrypting this local storage can limit these actions, but, it would also be useful to have a PKI[11] and use it to sign the information that is stored. With this, we have a way of knowing if a record has been tampered, improving our systems integrity and the authenticity of the information stored.

- The fact of some communication being made over the internet leaves the possibility of an attacker tampering our system. It would be a good idea to use secure communication channels for all communications over the Internet. This could be achieved using, for example, **TLS**. TLS stands for **Transport Layer Security** and it can be used to encrypt the communication between web applications and servers, emails, messaging, and more.[13] Between the user and the reader we have two choices: focusing on making the channels secure or encrypting the message. Since we have different kinds of channels, it would be a good idea to focus on **encrypting the message**, managing to have a solution that works on all channels. On top of the message encryption we could also **sign the message** with the help of our PKI, allowing us to mitigate this problem.

Repudiation

- At the moment, we have no way of guaranteeing that a entity is who it claims to be. This is a serious threat to our system that also could be resolved with the addition of a Public Key Infrastructure.[11]

- The lack of certain fields in a transaction may lead to the possibility of an attacker forging one. In order to avoid that, all transaction logs should contain an unique identification tag and information such as the date in which the transaction occurred. It should also be implemented a system to monitor alterations of these logs.

Information Disclosure

- The local storage containing logs and personal information is vulnerable to an attacker who may try to access that information. This situation could be mitigated by encrypting the local storage, preventing unwanted accesses.
- The communication with the emitting entity is not protected, and, vulnerable to being listened to by an attacker that can collect user information. By using TLS[13] and HTTPS this problem could be mitigated.
- The connection process with the reader is done using three different technologies, BLE, NFC and WiFi-Aware. All of these technologies are vulnerable to being listened to by an attacker. As seen before, a good solution would be to encrypt and sign(using our PKI) all the messages sent.
- The communication with the reader also lacks protection, leaving the door open to an attacker reading information such as the Authorization Token or personal information. This could the same way as on the connection process, by encrypting messages and signing them.
- The generated Authorization Token has no guarantee that after being sent to the reader it will only be used once. A good solution for this would be the implementation of a **TTL**(Time To Live) for the token. With this, for example, after a certain time, the token would be rendered useless.

Denial of Service

- When receiving or updating information from the emitting entity, there is no verification if the incoming data is in fact from whom it claims to be. By not checking this, an attacker could be posing as an emitting entity and try to flood the application with fake data filling the local storage of the device.

This could be solved using a PKI[11], discarding all packets which are not of interest.

- When receiving data, there is no verification if the data is coming from whom we think it does. This could be exploited by an attacker trying to flood the user with incoming data. By filtering the incoming information using a PKI[11] we could use the signature of the data to decide if we want to discard it, and thus decrease the odds of flooding.

Elevation of Privilege

- The local storage on the device is vulnerable to an attacker modifying bits on disk in order to run unauthorized commands. This could also be solved by encrypting local storage.
- The content of incoming data is not being validated which is a problem. There should be implemented an module to validate incoming data to ensure that there is no malicious code hidden. An attacker could, for example, hide malicious code hidden in data fields and run them on the users device.

3.1.5 Risk Analysis

With the help of the STRIDE methodology we were able to search for vulnerabilities that the mID Application may encounter when in use. Unfortunately, resources are limited and there are vulnerabilities that are more important and should be addressed first. With that in mind, we tried to evaluate what should be implemented/focused on first. For that process of evaluation we tried to make a **Risk Analysis**, which involves analyzing the probability of something happening with the impact that is could have. We added also the weight of one solution fixing multiple problems. Given this, and after making the proper analysis, we decided that from the previous solutions we proposed, the ones that should be focused on first are: the implementation of **Two Factor Authentication**, the addition of a **Public Key Infrastructure** to the system both for encryption and data signature, and the use of **Transport Layer Security** and **HTTPS** in communications. By adding these functionalities to the system, a big part of the listed vulnerabilities could be mitigated.

3.2 Reader Application

For this component of the system we will also be using **Threat Modelling**, **focusing on Assets**. We decided that this approach is the best due to the same reasons as the mID application. Alongside this strategy, we are going to use the **STRIDE** approach complemented with a **Data Flow Diagram**.

3.2.1 Focusing on Assets

When we are Focusing on Assets while searching for potential threats in our system, it is important to think about 3 topics: **Things attackers want**, **Things you want to protect**, and **Stepping stones to either of these.** Considering our mID Reader, we answered these topics:

Things attackers want

- Steal information that the reader requests
- Disrupt/sabotage service
- Add/Remove data to the identification requests

Things you want to protect

- Data Integrity
- Service Reliability/Trustworthiness

Stepping stones to either of these

- Use secure authentication
- Use secure communication channels
- Data encryption

3.2.2 System Modelling

The best way to understand what can go wrong with the mID Reader is to understand what it does. To do so, we will be using a **Data Flow Diagram** to represent this part of the system as well as it's interactions.

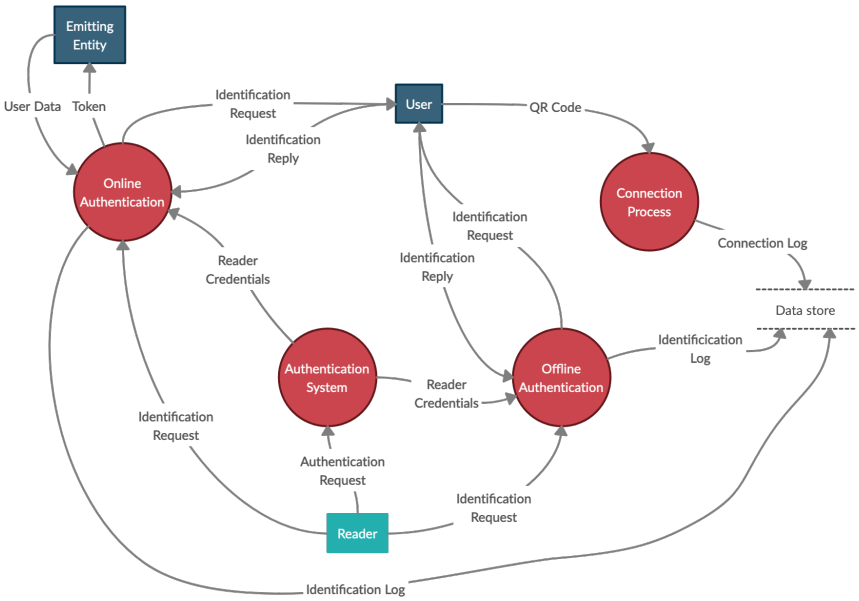

Figure 2: mID Reader Data Flow Diagram

We represented other elements of the system such as the User and Emitting Entity as external entities in order to better represent our communication with them, and, the local storage of the device was represented as a Data store.

3.2.3 Finding Threats

With our Data Flow Diagram done, we are now able to analyze it and find threats in a more streamlined way. In order to find the relevant threats, we will be using the STRIDE methodology:

3.2.4 STRIDE

Spoofing

- The communication with the User is vulnerable to Spoofing. This occurs because when we receive identification fields, we have no way of guaranteeing that they belong, in fact, to the User that is in front of us. This situation could also be mitigated with the addition of a Public Key Infrastructure[11]. With the PKI we can verify the signature of the incoming identification data.

- The communication with the Emitting Entity is also lacking protection against Spoofing. For example, when we use a token to get a certain set of identification data we have no guarantee that it has been, in fact, sent by a trusted entity, and not by an attacker. Since this problem consists on the same nature of the last one, it can also be solved by using a PKI[11].

Tampering

- The local storage on the device where the mID Reader is located lacks protection since these devices are generic Android or iOS smartphones. These lack of specific protection could lead to an attacker being able to make changes in the logs, compromising our systems integrity. These attacker can be external attackers or even the user of the Reader Application. By encrypting the local storage on the mID Reader we could prevent these unwanted actions.
- Since we mostly rely on wireless communication there exists the possibility of an attacker modifying data flowing over the network. To prevent this, a good idea would be to work on encryption and signature of messages sent to the user, and use TLS[13] and HTTPS for communications with the backend. With this, we could prevent an attacker from adding/modifying data that is being transmitted.

Repudiation

- Since we have no way of guaranteeing the authenticity of an entity's identity, we have no way of maintaining non-repudiation in our system. To avoid that, and as seen before, a PKI[11] would be useful to have signatures that uniquely identify an entity.
- The lack of protection on our local storage allows an attacker to modify our log files, which compromises the non-repudiation property of our system that we want. Despite also being a Tampering threat, this also falls in the Repudiation category. Luckily, it can be fixed with the same solution, by encrypting local storage.

Information Disclosure

- Following on the topic of local storage and its lack of protection, this also leads to the possibility of an attacker accessing the logs and extracting information from our system. In order do prevent this, encrypting the local storage would be a good idea.
- The communication channels with the emitting entity and the mID Application of the user are done over the network or BLE, NFC and WiFi-Aware. These communication channels do not offer protection against an attacker intercepting traffic and accessing information. To prevent this, the implementation of a security layer on these channels would be recommended. TLS[13] and HTTPS would be a good suggestion for TCP/IP channels, while for the others we could encrypt messages and sign them using our PKI[11].
- One thing that we want to prevent in our system is that the reader or any kind of attacker may be able to add fields to the Authorization Token. To prevent this it would be a good idea to make use of the PKI[11] that we already mentioned, and encrypt the token after its creation, and decrypt it when it arrives on the backend only. Given this, we prevent the modification of the token to get more information that the user authorized.

Denial of Service

- By not checking the origin of incoming information from users, or responses from the emitting entity, we are vulnerable to an attacker trying to flood the reader. This could be avoided by implementing content filtration. We could use the already suggested PKI[11] to discard incoming data with invalid signatures.
- In addition to flooding the reader, an attacker may try to fill the data store on the reader device in order to disrupt the service. The above mentioned filtration using the PKI[11] could be used to mitigate this problem. By adding a rule that checks the incoming data size, we could discard suspicious sizes.

Elevation of Privilege

- As seen on the mID Application, the local storage on the device is vulnerable to an attacker modifying bits on disk with the purpose of running unauthorized commands. This could also be solved by encrypting local storage.

3.2.5 Risk Analysis

With the help of the STRIDE methodology we were able to search for vulnerabilities that the mID Reader may encounter when in use. Unfortunately, resources are limited and there are vulnerabilities that are more important and should be addressed first. With that in mind, we tried to evaluate what should be implemented/focused on first. For that process of evaluation we tried to make a **Risk Analysis**, which involves analyzing the probability of something happening with the impact that this could have. We added also the weight of one solution fixing multiple problems. Given this, and after making the proper analysis, we decided that from the previous solutions we proposed, the ones that should be focused on first are: the addition of a **Public Key Infrastructure** to the system both for encryption and data signature, and the use of **Transport Layer Security** in communications. By adding these functionalities to the system, a big part of the listed vulnerabilities could be mitigated.

3.3 System Backend

For this component of the system we will also be using **Threat Modelling**, **focusing on Assets**. We decided to also take this approach since it's the most important entity of the system, in terms of security and functionality. One attack here could compromise the whole system concerning Availability, Integrity and Confidentiality.

3.3.1 Focusing on Assets

When we are Focusing on Assets while searching for potential threats in our system, it is important to think about 3 topics: **Things attackers want**, **Things you**

want to protect, and **Stepping stones to either of these.** Considering our backend, we answered these topics:

Things attackers want

- Steal information
- Disrupt/sabotage service
- Tamper with stored data

Things you want to protect

- Data Integrity
- Service Reliability/Trustworthiness

Stepping stones to either of these

- Use secure authentication
- Use secure communication channels
- Data encryption

3.3.2 System Modelling

The best way to understand what can go wrong with the backend is to understand what it does. To do so, we will be using a **Data Flow Diagram** to represent this part of the system, as well as its interactions.

We represented other elements of the system, such as the User and Reader, as external entities, in order to better exhibit our communication with them, and, the storage of the backend was displayed as a Data store.

3.3.3 Finding Threats

With our Data Flow Diagram done, we are now able to analyze it and find threats in a more streamlined way. In order to find the relevant threats, we will be using the STRIDE methodology:

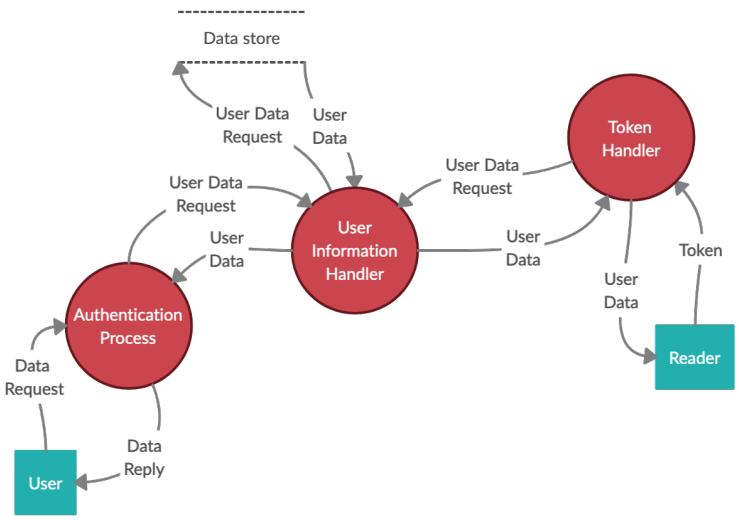

Figure 3: Backend Data Flow Diagram

3.3.4 STRIDE

Spoofing

- When receiving a token from a supposed reader, we have no way of knowing if it was sent from a trusted source or not. In order to know if a token was sent from a reader and not an attacker, we should use a PKI[11], so that every token has the signature of the reader who made the request, as well as the signature of the user whom information is being fetched.
- Considering the possibility of a backend user having his credentials stolen, an attacker is able to pose as one, and access our system. To prevent or hinder these kind of problems, it would be advised to implement a login system including **Two Factor Authentication**.

Tampering

- As stated on the other entities, since we mostly rely on wireless communication, there exists the possibility of an attacker modifying data flowing over the network. To prevent this, a good idea would be to use TLS[13] and HTTPS for communications. With this, we could prevent an attacker from adding/modifying data that is being transmitted.

- If an attacker is able to enter into the operating system via spoofed credentials or lack of protection, the local Database can be compromised. These lack of specific protection could lead to an attacker being able to tamper with the information stored or make changes in the logs, compromising our system integrity. By implementing **Two Factor Authentication** and encrypting local data, we could prevent these unwanted activities, respectively.

Repudiation

- Since we have no way of guaranteeing the authenticity of an entity's identity, we have no way of maintaining non-repudiation in our system. To get around that problem, and just like before, a PKI[11] would be useful to have signatures that uniquely identify an entity.
- Without keeping track of the transactions, from other entities, in a log, it's impossible for the system to guarantee that certain request came from an entity in particular. This could be solved adding a logging system that stores all the requests divided per entity.
- Changes to the system can be made without anyone knowing who it was. The system should have an encrypted logging system, in order to keep track of modifications and the responsible behind them. This would also help finding a compromised account.

Information Disclosure

- The connection process with the other entities is done via TCP/IP. This technology allows an attacker to intercept the data being transmitted. To solve this, it's suggested the use of TLS[13] and HTTPS to provide integrity and privacy to the data being transmitted.
- The fact of not having any way of knowing the authenticity of an entity's identity, allows an attacker to pose as other entity in order to get information that otherwise wouldn't have access. This problem can be solved by implementing a PKI[11] to have signatures that uniquely identify an entity.
- Lack of authentication on the backend results in undesirable people having

access to critical information. To solve this issue, it should be implemented a secure authentication method.

- In case of a process returning an error, an attacker can take advantage from this error message and extract information.
- We also considered the possibility of a person being able to physically access the backend and steal information. This can't be fixed via software but is still possible. Although, strong data encryption could be implemented in order to make it more difficult.
- Since the backend has a database, a set of bad permissions or configurations may lead to an attacker being able to explore the database without our consent. There should exist an effort to verify if the database has all configurations and permissions in order.

Denial of Service

- An attacker with the objective of disrupting our service could try to flood the backend either with reader or user requests, occupying available resources. We could use the mentioned PKI[11] to discard requests with an invalid signature.

Elevation of Privilege

- The lack of proper verification of the token received from supposed reader could give the possibility to an attacker to embed malicious code that can harm our system. Checking the tokens before doing anything with them is highly advised.
- The lack of proper authentication in our system backend allows a potential attacker to access our backend and gain unwanted privileges within the system. This can be prevented by implementing a strong authentication system.
- The lack of encryption and access control to the data store of our backend allows an attacker to modify bits on disk to things other than what a backend user wants. This can be mitigated by encrypting local data.

- The lack of a firewall increases the odds of an attacker gaining unauthorized access to our system. By including a **firewall**[12] to our system we could prevent this and also inspect more closely suspicious traffic.
- We also considered the possibility of a person being able to physically access the backend and run malicious code or gain privileged access. This can't be fixed via software but is still possible.

3.3.5 CVE

Since the system backend has specific specified, we used that to search for **CVE** that affect our system. Not only did we explore the current known vulnerabilities, but also vulnerabilities from the past. This approach will help to rectify ongoing problems and to be prepared for future threats.

CentOS 7.8.2003

The Operating System CentOS doesn't have many vulnerabilities at its core, and it couldn't be found any that was critical to our project. Although, some of the tools that can be used with this OS might have it, this problem gets out of our work scope, since the solution would vary with every tool. Before the use or installation of every program it should be made some search about past and current problems. Some of the programs with problematic versions are CentOS Web Panel, Docker Engine and Piranha Configuration Tool.

PostgreSQL 12.1 & PostgreSQL 12.4

After a vast search of vulnerabilities which occurred over the past years with this **relational database management system (RDBMS)**, we managed to find some attack patterns. This threats normally took advantage of flawed commands and functions, code injections or badly defined user permission, these would allow the attacker to have access and modify data, or even shutdown the whole database. To prevent this developers should have special care with external functions and commands used, deal with external data with caution to avoid code injections and attention with the different users of the database and their permissions. The

known CVEs to the versions that will be used are:

CVE-2020-25695 - An attacker having permission to create non-temporary objects in at least one schema can execute arbitrary SQL functions under the identity of a superuser. The highest threat from this vulnerability is to data confidentiality and integrity as well as system availability.[8] **Updating to versions 12.5 or 13.1** fixes this issue. **This CVE has not CVSS score yet.**

CVE-2020-25694 - If a client application that creates additional database connections only reuses the basic connection parameters while dropping security-relevant parameters, an opportunity for a man-in-the-middle attack, or the ability to observe clear-text transmissions, could exist. The highest threat from this vulnerability is to data confidentiality and integrity as well as system availability.[7] **Updating to versions 12.5 or 13.1** fixes this issue. **This CVE has not CVSS score yet.**

CVE-2020-14349 - An authenticated attacker could use this flaw in an attack in order to execute arbitrary SQL command in the context of the user used for replication.[3] **Updating to version 12.4** fixes this issue. **This CVE has a CVSS score of 7.1, being classified as High.** This means that it is highly advised to fix it.

Docker 19.03.6

PostgreSQL has a Docker container running. Docker has some vulnerabilities specially if taking into account the runC container that is used by Docker. Most of the past vulnerabilities came from lack of content verification, such as code injection, spoofing, denial of service due to repeated join and quit actions, sharing of secrets on account of the debugging mode, being negligent on executing trojan horse programs and the use of commands as root to gain access to certain privileges and execute critical code. With this information we know that we need to be careful with how all external data is managed, and the use of some functionalities and commands.

CVE-2020-13401 - An attacker in a container, with the CAP_NET_RAW ca-

pability, can craft IPv6 router advertisements, and consequently spoof external IPv6 hosts, obtain sensitive information, or cause a denial of service.[2] **Updating to version 19.03.11** fixes this issue. **This CVE has a CVSS score of 6.0, being classified as Medium.** This means that it is advised to fix it, but not crucial.

Ubuntu 20.04

This version of Ubuntu is used for the management system of the backend. Despite being a stable version of this OS, there are still vulnerabilities to be found. In our search, we found out that most of the problems associated to this OS are related to utilities and tools that it ships with. Given the fact that we don't know if these tools will ever be used, we only considered CVE's directly related to the OS.

CVE-2020-15708 - This CVE created a control socket with world read and write permissions. An attacker could use this to overwrite arbitrary files or execute arbitrary code. [4] **This CVE has a CVSS score of 7.8, being classified as High.** This means that it is highly advised to fix it.

Django v3.0

The **Web Framework** Django has to deal with a lot of user input, because of this most of the vulnerabilities are related with lack of input validation, such as code, sql and malicious script tags injection, and key validation which can lead to data leakage. This problem can be mitigated, if from the beginning of the development of this project, all the input data is processed carefully. Some of the CVEs found to this version show this problem.

CVE-2020-9402 - This allows SQL Injection if untrusted data is used as a tolerance parameter in GIS functions and aggregates on Oracle.[10] **Updating to version 3.0.4** fixes this issue. **This CVE has a CVSS score of 8.8, being classified as High.** This means that it is highly advised to fix it.

CVE-2020-24584 - With this CVE, the intermediate-level directories of the filesystem cache had the system's standard umask rather than 0o077.[6] **Updat-**

ing to version 3.0.10 fixes this issue. **This CVE has a CVSS score of 7.5, being classified as High.** This means that it is highly advised to fix it.

CVE-2020-24583 - FILE_UPLOAD_DIRECTORY_PERMISSIONS mode was not applied to intermediate-level directories created in the process of uploading files. It was also not applied to intermediate-level collected static directories when using the collectstatic management command. [5] **Updating to version 3.0.10** fixes this issue. **This CVE has a CVSS score of 7.5, being classified as High.** This means that it is highly advised to fix it.

CVE-2020-13254 - In this CVE, In cases where a memcached backend does not perform key validation, passing malformed cache keys could result in a key collision, and potential data leakage. This can be solved **updating to version 3.0.7**.[1] **This CVE has a CVSS score of 5.9, being classified as Medium.**. If not possible updating to that version, it's important to have special care with key validation.

CVE-2020-7471 - This vulnerability allows SQL Injection if untrusted data is used as a StringAgg delimiter. By passing a suitably crafted delimiter to a contrib.postgres.aggregates.StringAgg instance, it was possible to break escaping and inject malicious SQL.[9] This can be solved **updating to version 3.0.3**. **This CVE has a CVSS score of 9.8, being classified as Critical.**. This means that it is extremely advised to fix it.

Flask 1.0

Just like Django, Flask is a **Web Framework** and it has similar vulnerabilities issues. For the given version, it wasn't found any known vulnerability. Past vulnerabilities revolves around improper input validation and denial of service due to crafted encoded JSON data. Even though, this issues were solved in the current version, all data should be treated carefully.

UWSGI & Gunicorn

UWSGI and Gunicorn are **Web Servers**. It's not explicit which version of them

will be used, but these server didn't have many vulnerabilities in the past. These vulnerabilities are still important. They were based in flawed functions leading to overflows and return of wrong HTTP headers and inadequate calling and manipulation of files. To avoid this problems to happen in our project, we should have caution with the function used and code quality.

3.3.6 Risk Analysis

With the help of the STRIDE methodology we were able to search for vulnerabilities that the backend may encounter when in operation. Unfortunately, resources are limited and there are vulnerabilities that are more important and should be addressed first. With that in mind, we tried to evaluate what should be implemented/focused on first. For that process of evaluation we tried to make a **Risk Analysis**, which involves analyzing the probability of something happening with the impact that is could have. We added also the weight of one solution fixing multiple problems and we also used the **CVE** of the listed version of software used to build the backend to have an idea of what kind problems could be more common. Given this, and after making the proper analysis, we decided that from the previous solutions we proposed, the ones that should be focused on first are: the addition of a **Public Key Infrastructure** to the system, the use of **Transport Layer Security and HTTPS** in communications, the implementation of a **Firewall** and the implementation of an **input validation module**. Tanking into account the CVE's that we analyzed, we decided that input validation is critical and a strong firewall could mitigate some of them. By adding these functionalities to the system, a big part of the listed vulnerabilities could be mitigated.

3.3.7 Official Emitting Entity

In our analysis we considered that the communication with the entity that provides user information and emits official documents is out of our scope, given that this communication would be entity dependent and would probably diversify a lot, so we considered that our backend has the information necessary in order to operate properly.

4 Conclusions

After analyzing each of the three entities of our system and applying our strategies in order to find vulnerabilities, we were able to list them and use a Risk Analysis to see which of these vulnerabilities should be addressed first. With this analysis we were able to understand the amount of problems existing in a complex system, and the amount of effort required to solved them. We also learned that it is almost impossible to fix every single vulnerability or security issue that we found because they are so many and some of them depend on the software we use. The best we can do is to understand the problems that exist and try to implement measures that mitigate them and have good practices/security measures.

References

[1] *CVE-2020-13254*. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-13254 (visited on 11/14/2020).

[2] *CVE-2020-13401*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-13401 (visited on 11/14/2020).

[3] *CVE-2020-14349*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-14349 (visited on 11/14/2020).

[4] *CVE-2020-15708*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-15708 (visited on 11/21/2020).

[5] *CVE-2020-24583*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24583 (visited on 11/14/2020).

[6] *CVE-2020-24584*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24584 (visited on 11/14/2020).

[7] *CVE-2020-25694*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25694 (visited on 11/14/2020).

[8] *CVE-2020-25695*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25695 (visited on 11/14/2020).

[9] *CVE-2020-7471*. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-7471 (visited on 11/14/2020).

[10] *CVE-2020-9402*. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-9402 (visited on 11/14/2020).

[11] *PKI - Public Key Infrastructure*. URL: https://www.ssh.com/pki/ (visited on 11/11/2020).

[12] *What Is a Firewall?* URL: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html (visited on 11/20/2020).

[13] *What is TLS (Transport Layer Security)?* URL: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/ (visited on 11/11/2020).