Attack SureThing!
Offensive security assessment of a location certification system

José Miguel de Brito Alves Ferrão

Thesis to obtain the Master of Science Degree in

Information Systems and Software Engineering

Supervisor(s): Prof. Dr. Miguel Filipe Leitão Pardal

Examination Committee

Chairperson: Prof. Dr. Pedro Tiago Gonçalves Monteiro
Supervisor: Prof. Dr. Miguel Filipe Leitão Pardal
Member of the Committee: Prof. Dr. Rui Miguel Soares Silva

November 2021

Acknowledgements

First of all I have to thank my advisor, Professor Dr. Miguel Pardal, and Dr. Samih Eisa for all their help and support throughout these yet somewhat uncertain times, and their invaluable comments which greatly improved this work.

I also have to dedicate some thank you words to João Machado for his very well detailed taxonomy that was used as a reference.

Last, but not least, I would also like to thank my mother for her constant support. Even if not always aware of what was going on, by my choice, I know I could always count on her.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID) and through project with reference PTDC/CCI-COM/31440/2017 (SureThing).

Resumo

O projeto SureThing está a desenvolver uma estrutura de certificação para evitar a falsificação de localização em serviços. Os protótipos existentes permitem a emissão e a verificação de provas de localização a partir de dispositivos móveis e suportam diversos casos de uso, como CROSS, uma aplicação de turismo inteligente. Apesar dos esforços dos programadores e administradores de sistemas, um sistema não pode ser considerado realmente seguro e robusto até ter sido exposto a ataques de atacantes qualificados e motivados. Partindo desta premissa, realizou-se uma avaliação de segurança ofensiva do sistema CROSS, constituído por uma aplicação móvel de turismo inteligente que permite a emissão de certificados de localização e um servidor que expõe uma interface na Internet. A nossa avaliação consistiu no uso de técnicas de ataque, nomeadamente na identificação de vulnerabilidades e testes de penetração. Usaram-se ferramentas genéricas, a partir de diferentes pontos na rede, sempre na perspectiva de um atacante.

Nesta dissertação apresentamos as ferramentas e as técnicas usadas para atacar o servidor do sistema CROSS, os resultados obtidos, e os procedimentos para tornar o servidor mais seguro. A nossa avaliação de segurança consistiu no uso de diferentes ferramentas. Ao usá-las, foi-nos possível encontrar uma vulnerabilidade anteriormente desconhecida que permitia fazer escritas não autorizadas diretamente na base de dados do servidor.

Foram também preparados os materiais necessários, permitindo que a mesma abordagem ofensiva seja replicada com a organização de futurs torneios competitivos, onde os mesmos ataques podem ser reproduzidos num ambiente de jogo permitindo a descoberta de novos ataques.

Palavras-chave: Avaliação de Vulnerabilidades, Testes de Penetração, Cibersegurança Ofensiva, Sistemas de Certificação de Localização

Abstract

The SureThing project is developing a location certification framework that can prevent location spoofing on Location Based Services (LBS). The current prototypes using the framework allow to issue and verify location certificates using mobile devices in a variety of use cases, including the CROSS smart tourism application. Despite the best efforts of designers, a system cannot be said to be truly secure and robust until it has experienced attacks from skilled and motivated attackers. With that in mind, we performed an offensive security assessment of CROSS, composed by a smart tourism mobile application that issues location certificates and a server with a publicly exposed interface on the Internet. Our assessment involved exercising offensive security techniques, in the form of vulnerability assessment and penetration testing. We used generic tools, from different vantage points in the network, always in the perspective of an attacker.

In this dissertation we present the tools and the techniques that were used to attack the server of the CROSS system, along with our findings and the procedures taken to harden the server after the detection of vulnerabilities. Our security assessment included the use of different tools. We were able to find a previously unknown vulnerability that allowed to make unauthorized writes to the database of the server.

We also prepared the necessary materials, allowing the same offensive approach to be replicated by organizing future offensive security tournaments, so that the same attacks can be reproduced in a gamified environment allowing for the future discovery of new attacks.

Keywords: Vulnerability Assessment, Penetration Testing, Offensive Security, Location Certification Systems

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Most organizations only rely on certifications for the security of their system. It is true a certification can be very thorough and detailed but it does not cover all security aspects, and may give a false sense of security. Some organizations go further and subject their system to *Vulnerability Assessment* and/or *Penetration Testing.* While they might be able to detect some flaws, both lack the (negative) impact a real attack might have in a system. That is why we fully agree with the following statement: "*A system cannot be said to be truly secure until it has experienced an attack from a real threat*" [TV20].

The need for security assessment is even more important when we are considering a deployed mobile application, as the impact of security incidents is even greater when there are physical world consequences. These devices inherit the security-related issues common to computer networks but are even more challenging because these devices are characterized by being limited by its small memory, lack of processing power, and its battery capability [KS18].

A specific application that uses such deployable mobile applications is SureThing[1] [FP18], a location certification system that allows devices to prove that they were at a particular location at a specific time. The architecture of the SureThing lacked a security assessment. A security assessment is required to ensure the proper security measures of a system are in place, to ultimately prevent attackers to target and use unknown entry points.

With this work, we propose an offensive security assessment of SureThing [FP18] by assessing the security of the architecture and the implementation of CROSS (*loCation pROof techniqueS for consumer mobile applicationS*) [GAM20], a location certification system made to support a smart tourism application, where people go around a specific route in a city and get rewarded for it. We based our approach in the steps an attacker would perform, by exercising vulnerability assessment and penetration testing techniques. For that we deployed a virtualized environment

[1]More information about the SureThing research project can be found at: `http://surething-project.eu`

with specific requirements to make it as real as possible, meaning we had to simulate different networks and multiple machines.

1.1 Objectives

The main goal of this work was to assess the security of an existing deployment of CROSS. We evaluated the security of CROSS by performing an offensive security assessment targeted at its server. We used different tools, distributed over three iterations of attacks. These attack iterations ranged from exercising simple vulnerability analysis tools to fuzzing techniques. We used and assessed each tool individually. The results of this security assessment allowed us to improve and harden the security of the existing deployment.

Even though this work is specific to a particular system, we provide some insight and guidelines to perform an offensive security test so that the assessment can consider the (negative) impact an attacker might have in a system.

1.2 Contributions

The main contributions of this work are the following:

- Full preparation of a virtualized environment of a CROSS deployment, including:
  - The server, a web server programmed in Go[2] with a REST[3] API and a relational database with PostgreSQL;
  - The client, a mobile application for the Android platform;
  - Network, sub-networks and routers.
- Security assessment of the CROSS server and improvements to its configuration;
- Survey of existing and freely available offensive security tools.

1.3 Dissertation Outline

The remainder of the document is structured as follows: Chapter 2 gives a background overview of security concepts and commonly used attack tools, and also outlines some previous works in *Location Certification Systems*. Chapter 3 describes CROSS - the target system of this work - and the offensive approach to security assessment. Chapter 4 presents our evaluation of CROSS. Finally, Chapter 5 concludes the document, and also presents future work directions.

[2]`https://golang.org/`
[3]REpresentational State Transfer

Chapter 2

Background and Related Work

A *threat* consists in any intention of to cause and inflict damage to a system [TV20]. When aimed to an organization as a whole, a threat may therefore have to target several systems that belong to that organization. The targets of a threat may not only be systems but also the staff of the targeted organization, as we will study in more detail in 2.2. If successful, a threat has the potential of having a negative impact on the targeted organization. A threat can only be considered as a potential action when performed by a *threat-actor*, the motivated and skilled enough entity involved or that is responsible for a threat [TV20]. In this context, a threat-actor can either be a group of people or just a single individual. An *attack path* relates to the steps a threat-actor takes or goes through to plan, prepare and later execute an attack. An *attack vector* is then the attack path or the method a threat-actor may use to gain unauthorized access to a network or a system. In turn, the *attack surface* corresponds to the different ways through which a threat-actor is able to access a system [MW10], i.e., the total number of attack vectors a threat-actor can use to gain access to a system.

The two main entities involved in any security testing process are called *Blue Team* and *Red Team*[1]. These two entities play different roles and their end goal are quite distinct. In fact they perform on opposing sides. The goal of the *Red Team* is to test a system. A *Red Team* is usually independent of the organization whose system is being tested. This helps prevent the members of the group do not have direct contact with the rest of the organization. It is quite essential for reasons we will see later in 2.2. On the other hand, a *Blue Team* is tasked with defending a system against outside threats. Usually composed by members that are part of that organization [TV20]. Other important entities that also take part in a security testing process are the target, the channels that are used to have access to the target, and the team responsible for the management side.

[1] *Red Cell* is another term commonly used to refer to a *Red Team*. This term originally comes from the military and is used to define every and all components that make the team performing the offensive tasks.

In the following sections we present security concepts and tools that are used in security testing, as well as some works done in the field of location certification systems. We start by defining offensive security in 2.1. Namely, we describe what are the differences between *Vulnerability Assessment* and *Penetration Testing.* In 2.2 we present *Red Teaming* - another way to perform an offensive security assessment. We then consider the importance of having a standardized way of describing vulnerabilities, followed by some of the existing databases, in 2.3. Following that, we present some commonly used attack tools in 2.4. We finish this Chapter with some location certification works, described in 2.5.

2.1 Offensive Security

Vulnerability Assessment, *Penetration Testing* and *Red Teaming* are often used interchangeably. Yet they have distinct connotations. *Red Teaming* will be described in detail later in 2.2.

Vulnerability Assessment is the process of analysing a system to find, identify and prioritize vulnerabilities in terms of their risk. Once any flaws are identified the target system can have those vulnerabilities mitigated, leading to the reduction of its attack surface. By reducing the attack surface, the ability of an attacker to take advantage of the (previously) identified flaws diminishes. The idea behind a Vulnerability Assessment is to decrease the existing attack paths. It is not concerned with exploiting a vulnerability, even if it was found in the process [TV20].

In turn, Penetration Testing consists in the execution of an attack targeted at a specific system to identify and measure the risks associated with the exploitation of the attack surface of that specific system. In other words, Penetration Testing adds to the process of performing a Vulnerability Assessment by introducing exploitation to the equation [TV20]. Any Penetration Testing execution follows a set of basis sections, as defined by the Penetration Testing Execution Standard[2]: Pre-engagement Interactions, Intelligence Gathering, Threat Modelling, Vulnerability Analysis, Exploitation, Post Exploitation, and Reporting.

Even if there were not detected any vulnerabilities in a given system, that system is still prone to have its security broken. Another way to circumvent the security assumptions of a system is through the exploration of the human factor. An example of this situation is when an attacker would take advantage of naive employees to gain access to a system. Another way is by taking advance of weak security configurations being employed in a system.

[2]`http://www.pentest-standard.org`

2.2 Red Team Models

Some business areas require certification, thus organizations usually only rely on audits and compliance to comply with the law. Even though it gives organizations the opportunity to have their systems, policies and procedures certified, it does not cover all security aspects [TV20].

Prevention has an important role in the security of a system but it does not guarantee protection against every attack. A *Blue Cell* has to know and understand the motivations of the threat-actor but also know its capabilities and how it operates. It is only then that it is possible to have an idea of the real threat and the impact it may have in a system. A system cannot be said to be truly secure until it has experienced an attack from a real threat. It is then necessary to include and have in mind the perspective of a threat-actor. This is where a *Red Team* comes in [TV20].

The main objective of a *Red Team* is to train and measure the effectiveness of the *Blue Team*. The operations taken by a *Red Team* are usually designated as *Red Teaming*. *Red Teaming* is the process of using *Tactics*, *Techniques* and *Procedures* (TTPs[3]) to emulate an actual threat. An engagement is the process that includes all activities performed by a *Red Team* when testing a system from the perspective of an attacker. These activities are not limited to but can include *Vulnerability Assessment* and *Penetration Testing* techniques. A *Red Team* is supposed to gather a set of evidences throughout an engagement and at the end produce a list of everything that was tested, along with any recommendations and/or aspects that should be addressed. *Red Team*s will eventually discover vulnerabilities and use them to progress and achieve their goals. While this may happen, it is not the central goal of a *Red Team*. The focus of a *Red Team* is to capture the broader picture and do an accurate assessment of a system as a whole. A *Red Team* should be able to provide an unbiased and impartial perspective of a system. This is why it is imperative to have a *Red Team* free of any connections with the system being tested or with any of the personnel involved in its design and conception. For that, *Red Team*s have to follow a set of guidelines and rules that must be respected when testing a system [TV20]. Such rules and guidelines are known as *Rules of Engagement* (ROE), which are described in detail in 2.2.1.

In 2.2.2 we describe in more detail each group that is normally involved in any engagement. Following that we present the life-cycle of a *Red Team*, as shown in figure 2.1.

2.2.1 Rules of Engagement

The *Rules of Engagement* is the document that defines and contains all agreed-upon rules and guidelines that a *Red Team* has to follow and adhere to for the duration of an engagement. The

[3]`https://github.com/inesc-id/SecurityTaxonomy/blob/main/Definitions/TTPs.md`

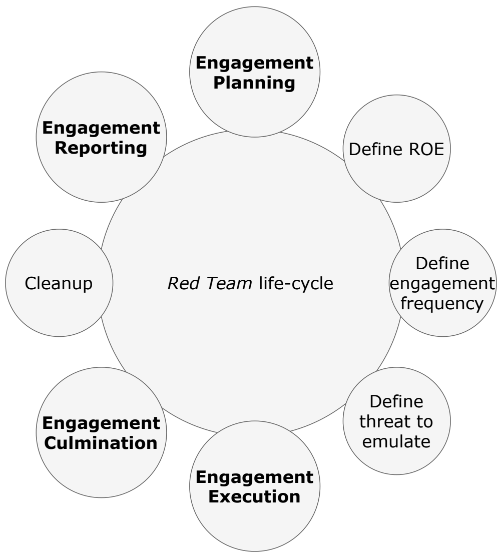


Figure 2.1: Life-cycle of a *Red Team*.

ROE establishes the responsibility that each and every entity involved in an engagement has. This document can be seen as a formal agreement between both parties, giving a *Red Team* the authorization to perform any (offensive) actions during an engagement. On one side is the *Red Team*, while the other side is composed by all entities related to the organization and the targeted system. A ROE shall document the target information, the threat to implement/emulate and any foreseen activities the *Red Team* will execute. Here the target can either be a system, a process, a currently used technology or even the network of the organization. In particular, a ROE comprises [TV20]:

- Methodology usually employed by the *Red Team*;
- Description of the activities that may be executed;
- Overview of the hardware and software that may be used;
- Threat levels available;
- Roles and responsibilities of each group that is involved in the engagement (discussed below);
- Legal requirements and responsibility disclaimers the *Red Team* must follow.

Other relevant information like detailed information about the target, specific objectives of the engagement, approved activities and any restricted activity should be documented in the document annexes. The annexes should also clearly state the authorized *target space*, either physical or network-related. There must be defined any physical boundaries if the engagement is to be conducted in a specific site, namely authorized areas to access and also any off-limits areas, if applied. The ROE document must be updated accordingly if the authorized actions, the scope or the objectives of the engagement, or any other relevant information changes [TV20].

2.2.2 Groups and their Roles

Apart from the *Red Team* itself and its opposing team - the *Blue Team* - there are other groups involved in an engagement. Namely *White Cell*, *Trusted Agents*, *Engagement Control Group*, and *Observers*, each with their respective role.

White Cell

The *White Cell* is primarily tasked with enforcing that the ROE are strictly followed. This also includes making sure that neither the activities performed by the *Red Team* nor by the *Blue Team* cause any problems to the target system or to the organization. A *White Cell* can be seen as a referee between the *Red Team* activities and the corresponding defensive responses from the *Blue Team.* More specific tasks a *White Cell* is trusted with may comprehend:

- Establish the metrics for the engagement;
- Guarantee the engagement goals are met;
- Coordinate the activities of both *Red* and *Blue* teams;
- Provide any needed information to make sure the engagement is efficiently conducted;
- Elaborate a list of learned lessons according to its observations at the end of the engagement;
- Score the engagement, if applicable.

Trusted Agent

A *Trusted Agent* (TA) is a member of the target organization who knows an engagement is under way. A TA is supposed to have privileged knowledge and detailed information about the undergoing engagement. These detailed information could consist in the activities, conditions and the overall status of the engagement. The role of a TA is to make sure (or at least limit) any irreversible damage to any personnel or equipment of the organization. Such damage could be caused by the activities performed by the *Red Team* but a TA is also responsible to prevent any member of the *Blue Cell* to cause any self-inflicting damage, i.e., cause damage to the organization that he/she is protecting.

Engagement Control Group

An *Engagement Control Group* (ECG) is composed by managers from the target organization and liaisons from both the *White Cell* and the *Red Team*. Often the ECG and the *White Cell* are merged into a single group. Either way, all members must be TAs. The ECG is responsible by all and any activities conducted during an engagement. More specifically, an ECG is committed to the following tasks:

- Approve the engagement, which particularly includes defining the schedule and the objectives of the engagement;
- Provide all the necessary management and direction for the correct execution of the engagement;
- Determine if and subsequently when and what information to provide and share with the defence team.

Observers

If required, it is possible to have *Observers* during an engagement execution. If present, *Observers* are responsible to document the actions and any reactions taken by each opposing cell during an engagement. Their sole purpose is to observe the engagement. An *Observer* is not supposed to give any type of information to none of the teams but may eventually report potentially damaging actions to the *White Cell*, if deemed necessary [TV20].

2.2.3 Planning Phase

A good planning is as important as the execution phase itself. *Engagement Planning* is the term normally used to denominate the phase that precedes an engagement. An engagement can only be effective in terms of achieving the desired goals if all involved entities take the time and effort needed in the design and planning phase. The planning phase ought to outline any cost related to the execution of an engagement. There is the indispensable personnel, with the number of required personnel mainly dependent on the size, the length, the scope, and the desired goals of the engagement. A *Red Team* may have to customise its own toolset/toolkit according to the engagement objectives, which will also add up to the total cost. In addition, a *Red Team* may need to do some *intelligence gathering* and passive reconnaissance (OSINT) of the target system prior to the engagement. Moreover, a *Red Team* may be required to conduct the engagement at a specific remote site, e.g., at the organization site or at the target system deployment site. Therefore, any travel costs have to be accounted for when planning an engagement [TV20].

Engagement Frequency

Another aspect to consider when planning a future *Red Team* engagement is how frequent an engagement is supposed to take place. If a system is tested not often enough then the team responsible for the system may start to be complacent, disregarding some critical security operations. On the other hand, testing a system too often may not give the *Blue Cell* the necessary time to apply the recommendations outlined at the end of an engagement. The frequency with which engagements should be performed is dependent and subject to the specific targeted system and the goals the organization wishes to achieve with the engagement. With that said, we can distinguish three time intervals an engagement can be executed on:

- *One-time (or Single) Engagement*;
- *Periodic Engagement*;
- *Continuous Engagement.*

A one-time engagement type gives an organization the ability to get introduced to *Red Team* engagements and its advantages. In this case the *Red Team* has to work closely with the team behind the system that will be tested. Both sides have to take its time to understand the goals and what are the results the *Red Team* is supposed to deliver with the engagement. In a periodic type of engagement a system is periodically under test. These engagements could either be annual or biannual. *Red Team*s have to be particularly organized when planning and executing security testing for an organization that follows this testing approach. Each of these engagements needs to be as challenging and engaging as if it was a single engagement. For that, *Red Team*s should focus on and integrate new attack scenarios and currently existing threats when performing each different engagement. In a continuous engagement type, *Red Team*s are not constrained by time issues. Instead of being limited by a couple of weeks, a *Red Team* can achieve the engagement goals over some weeks, months or even years, depending on the case. This way, a *Red Team* is capable of positioning itself and remain more time inside the targeted system or its network. Having a *Red Team* constantly testing a system gives an organization the ability to understand the risks when exposed to a long-term threat. Allowing a *Blue Team* to adequately manage its security operations and defences [TV20].

Engagement Notification

Other aspect to take into account throughout an *Engagement Planning* is whom to inform about the undergoing engagement. An engagement can either be an *Announced Red Team Engagement* or an *Unannounced Red Team Engagement.* In an announced engagement the whole organization is aware that an engagement will take place. Having an announced engagement might make the *Blue Cell* increase its security procedures and the overall security of the system being tested, which may have a negative impact in the results of the engagement. However, an announced engagement allows a better planning. In turn, in an unannounced engagement the staff of the targeted organization is not informed about the engagement. This scenario is beneficial in terms of the final results since the organization will act and respond to the attack as it would in any other day. Letting the *Red Team* accurately measure the real posture of the *Blue Cell* and consequently gather more realistic results. However, critical assets and desirable targets to test may not be included in the *Red Team* planning when performing an unannounced engagement. The decision of whom to inform about an engagement cannot be taken lightly and is directly limited by the desired goals of the engagement. An *Unannounced Red Team Engagement* is more suitable if the goal is to measure the overall effectiveness and the procedures employed by an organization against a real threat and the (negative) impact it may cause in a system. Whereas an *Announced Red Team Engagement* is more suitable to measure the effectiveness of a specific operation, process, or tool [TV20].

Emulation of a Threat

When emulating a threat, a *Red Team* has to consider the perspective of the threat it is tasked with mimicking. There are three types of perspective from which to base an engagement [TV20]:

- Outsider: An entity who has neither access to the organization nor to its system;
- "Nearsider": An entity that has no legitimate access to the system but has physical access to the assets of the organization;
- Insider: An entity that has both physical and legitimate access to the assets of an organization and its system.

Once the perspective of the threat to impersonate is picked, the next step is to define the profile and the scenario of that threat.

The profile of the threat will help both the *Red* and the *Blue Team*s. Concerning the *Red Team*, it establishes the actions the *Red Team* will have to perform and how to act in order to correctly emulate the threat. As for the *Blue Team*, it allows the *Blue Team* to have a clear definition of which type of threat will be emulated and how that specific threat can (negatively) impact its organization. When emulating a threat, a *Red Team* needs to have in consideration the Indicators of Compromise (IOCs) that threat is known for leaving behind. If an unintentional

IOC is left in the target system the *Blue Team* may find it before the end of the engagement, thus exposing the *Red Team* before it reaches the planned goals. A well-known framework that helps in the description of the profile of a threat is MITRE ATT&CK®. The MITRE ATT&CK is not only complete but also does a good job of decomposing a threat into TTPs. TTPs are the Tactics, the Techniques and the Procedures (or sub-procedures, as it is referred in the MITRE ATT&CK framework) a threat-actor may use during an attack [SAM+18]. The tactics represent the objective the threat-actor is trying to achieve with an attack, i.e., the motive behind the attack. In other words, the tactics in the MITRE ATT&CK framework give an overview of the intentions of the threat-actor by covering some of the things the attacker might do. Techniques are used to represent the actual actions an attacker has to perform in order to achieve a given tactic. In turn, there are the sub-techniques. They depict the needed procedures to perform a given technique, i.e., how and what an threat-actor has to do to perform a given technique. The knowledge base the MITRE ATT&CK framework provides ensures the profile of the threat to emulate is valid and well defined [TV20]. Table 2.1 shows an example on how to describe in detail the profile of a threat.

Description	Skilled enough user of the system with access to common offensive tools and techniques
Goal and Intent	Issue location certificates for arbitrary locations and perform impersonation and replay attacks
Key IOCs	Presence and diffusion of a fraudulent AP when performing network spoofing attacks
C2 Overview	HTTPS on port 80
TTPs	Enumeration/Reconnaissance via OSINT gathering, Gather Victim Host Information and Gather Victim Network Information
	Resource Development via Develop
	Initial Access via Valid Accounts, namely using Local Accounts
	Discovery through Network Sniffing
	Collection via Man-in-the-Middle attacks
	Impact via Data Manipulation, namely Transmitted Data Manipulation
Exploitation (scenario model)	Assumed Breach Model
Persistence	Exist on RAM

Table 2.1: Threat profile to emulate.

Other existing models include the STRIDE and the DREAD models. These two latter models are more oriented towards threat modelling, providing a number of categories that can be used to rate a threat in terms of its security risk; whereas the MITRE ATT&CK is more directed at the actual steps a threat-actor goes through when targeting a system. Yet, the MITRE ATT&CK

encompasses both STRIDE and DREAD. A mapping between MITRE ATT&CK and each of the other two models is, respectively, presented in tables A.1 and A.2 in appendix A.

In turn, the threat scenario sets out how a threat should interact with the system and the network within the targeted environment. In other words, a scenario-based engagement allows a *Red Team* to focus on key areas of a system by emulating a specific threat.

A *Red Team* engagement also has to follow a scenario model, which is based on the operational impacts an organization expects to achieve and measure. Existing scenario models include [TV20]:

- Full Engagement Model;
- Assumed Breach Model;
- Custom Breach Model.

A Full Engagement Model consists in a complete emulation of a threat. The *Red Team* starts the engagement as if the threat is outside of the targeted organization, i.e., the emulated threat has yet no access to the network. Thus, the *Red Team* has to first do information gathering (OSINT) and reconnaissance and then perform an enumeration of the system and its network. The *Red Team* will then continue with its plan until it achieves its goals or until it is stopped. In an Assumed Breach Model the threat already has some type of access to the system or to its network. This way a *Red Team* has no need to spend time and resources trying to gain access to the system in the first place. It should be used if the goal is not to understand if a threat is able to gain access to the network. In a Custom Breach Model a *Red Team* is free to start the engagement at any phase, therefore focusing on certain stages of the engagement. A *Red Team* is able to design any scenario that allows testing specific key areas of the targeted organization and its system. The selection of the scenario model to follow is quite decisive due to the fact that the chosen model will largely shape the balance of the engagement. If a *Red Team* gets to reach its goals quite fast, the observations taken from the engagement may not bring much value. On the other hand, if a *Red Team* is stopped during an engagement then the targeted organization may not get the desired exposure to the threat to emulate [TV20].

2.2.4 Culmination Phase

After an engagement execution - which can be a couple of hours, or just a few days - there are a set of activities to make sure the proper cleanup takes place. Such activities are part of an *Engagement Culmination*.

A red team should use this time to make sure all performed activities are correctly logged. Namely, make sure all collected data - terminal logs, screenshots, and other collected artifacts - were correctly stored. The culmination phase should also be used to have all evidences properly sanitized. In practical terms, all produced evidences left behind by a red ream have to be completely removed. Also, as important, a red team must make sure to revert any and all modifications that may have been made to the security control of the target system. Otherwise the system may be left more vulnerable to outside attacks. The cleanup process ought to be documented in the ROE (2.2.1) to ensure the sanitization process goes accordingly to what was agreed upon.

If any of the tools that were used by a red team did not include self-destructing code - both time-based and target-based - then these tools should be individually removed and its removal documented for future reference. Self-destructing code can either be time or target-based. Where in time-based self-destructing code prevents code from running outside of the time-window an engagement takes place. Whereas in target-based self-destructing code, code is prevented from running and target any hosts outside of the defined environment for an engagement. As important, a red team must also provide a Trusted Agent (2.2.2) with a list of all produced artifacts, along with the status of their cleanup process. [TV20]

2.3 Vulnerability Databases

Any system is subject to have vulnerabilities: both in code and/or in the infrastructure. Vulnerabilities may include any errors that are present in a given system. These errors could have been introduced either during the design or during the implementation phase. In turn, if and when exploited, such vulnerabilities may result in the violation of the security assumptions of a system [TS10]. In essence, vulnerabilities have to be individually assessed. Furthermore, Tripathi and Singh [TS10] refer that after the identification of all existing vulnerabilities, each individual vulnerability needs to be evaluated based on its *risk level*. When talking about vulnerabilities, the level of risk is used to give some type of metric based on the level of exploitability and the (negative) impact a vulnerability might have in a system. When properly used, a taxonomy[4] will provide a useful and consistent way of classifying attacks [TS10]. Hansman and Hunt [HH05] propose a list of properties a taxonomy must follow. Table 2.2 lists some of those properties and their corresponding definitions. The *Terminology complying with established Security terminology* property is also commonly referred as *Backwards Compatible* in the literature (e.g. [TS10]).

[4]A taxonomy of cybersecurity standards and tools. Available at `https://github.com/inesc-id/SecurityTaxonomy/`

Property	**Definition**
Accepted	Structured in a way that it can be approved
Comprehensible	Understandable by those that work in the Security field as well by those that do not but may have to use it
Complete	Should take into account any attack
Deterministic	Its classification has to be well defined
Mutually Exclusive	Should categorize any given attack into only one category
Repeatable	Classifications should be repeatable
Terminology complying with established Security terminology	Should use already existing terminology
Terms well defined	Its terms should not generate any confusion
Unambiguous	Each category has to be clear and well defined to not cause any ambiguity
Useful	Should be able to be easily used in the security industry

Table 2.2: Desired properties a taxonomy should have and their definitions.

Some widely known and used vulnerability databases are NVD, Security Focus, and Exploit Database.

Part of NIST, NVD offers a repository that aims at helping manage vulnerabilities through standards. Apart from providing an analysis of published CVEs, it also scores vulnerabilities using CVSS.

Security Focus provides information with vulnerability-related announcements.

Maintained by Offensive Security, Exploit Database is a CVE compliant archive that contains public information about exploits and its corresponding vulnerable software.

2.4 Attack Tools

There are a wide variety of available tools. Each tool usually only focus on a specific goal that can be information gathering and reconnaissance, resources/assets enumeration, vulnerability identification, or even fuzzing techniques.

2.4.1 Information Gathering Tools

Information Gathering Tools (or OSINT) tools are used to gather information about the target system prior to the actual execution of an engagement. From a vast list of other existing tools, we highlight the following: *theHarvester*[5], *Metagoofil*[6], *goofile*[7], *Automater*[8].

theHarvester can help in the early stages of an engagement. It focus on exposing any information of a given organization that is publicly available on the Internet, gathered from a variety of public data sources, e.g., search engines. It gathers and enumerates emails, names, IPs, URLs and any subdomains belonging to that organization.

Metagoofil helps in finding publicly accessible documents of the targeted organization through a *Google*[9] search. The supported file types include .doc, .xls, .ppt, .odp, .ods, .docx, .xlsx, .pptx and .pdf files. The tool extracts *metadata* from those files. It then generates a report with several resources, like *usernames* and servers names.

The goofile tool can search a given domain for a specific type of file, e.g., PDF files. After searching a domain for the requested file type, the tool returns a list with all the files that were found and their location in the domain, i.e., its corresponding URL.

Automater is a tool aimed at gathering relevant results about a specific target from various sources. The specified target can either be a domain, an IP address or a MD5 hash. Automater relies on ipvoid.com, robtex.com, fortiguard.com, unshorten.me, urlvoid.com, ThreatExpert, VxVault and VirusTotal for getting the results.

2.4.2 Enumeration Tools

These type of tools are used with the goal of getting to know the target system better. Following are some of those tools: *ident-user-enum*[10], *InTrace*[11], *Nmap*[12], *DMitry*[13], *arp-scan*[14], *AMAP*[15].

The ident-user-enum tool can determine the owner of the process that is listening on each TCP port of a given system by querying the *Identification Protocol*[16], helping in the prioritization of processes. For instance, it might be more worth it to target a system where the authenticated user is already running with privileged access, e.g., *root* or *superuser* in *Unix*-like systems.

InTrace is a traceroute-like[17] tool that allows the enumeration of IP hops. It exploits existing TCP connections, either initiated from the local network or from a remote host.

[5]https://github.com/laramies/theHarvester
[6]http://www.edge-security.com/metagoofil.php
[7]https://code.google.com/archive/p/goofile
[8]www.tekdefense.com/automater
[9]Search engine. Available at https://www.google.com
[10]http://pentestmonkey.net/tools/user-enumeration/ident-user-enum
[11]https://github.com/robertswiecki/intrace
[12]Network Mapper. Available at https://nmap.org
[13]Deepmagic Information Gathering Tool. Available at https://mor-pah.net/software/dmitry-deepmagic-information-gathering-tool
[14]https://github.com/royhills/arp-scan
[15]Application MAPper. Available at https://www.thc.org/
[16]https://tools.ietf.org/html/rfc1413
[17]https://tools.ietf.org/html/rfc1393

Nmap is a tool that can determine what hosts are available in a network. It also finds out what services and Operating System a host is running. To achieve this, Nmap relies on raw IP packets.

The DMitry tool gives the ability to gather various types of information about a host. Namely, a sub-domain and an email search at the target host, a *WHOIS*[18] lookup and a TCP port scan at the target host. Furthermore this tool is extensible, i.e., its users can add new functions to it.

The arp-scan allows its users to know about any currently active devices in a local network by discovering and identifying all active devices in a local subnet. For that, the *arp-scan* tool relies on the *Address Resolution Protocol* (ARP)[19].

AMAP is a scanning tool that tries to identify applications running on a giving host, even if they are running on a different port than usual.

2.4.3 Vulnerability Analysis Tools

These tools are designed to assess computer systems, networks or applications for known weaknesses. Vulnerability analysis tools can help identify outdated software versions, missing patches, and misconfigurations. This is done by identifying the Operating System and major software applications running on a system and matching it with information about known vulnerabilities. Two of them are *OpenVAS*[20] and *Google Tsunami*[21].

OpenVAS is a vulnerability assessment tool that helps identify a system for known weaknesses. It does so by matching known vulnerabilities of a specific system with the version of the software running on the scanned system.

Google Tsunami is a general purpose network security scanner that aims at detecting high severity vulnerabilities with high confidence, by offering an extensible engine through the use of *plugins*. It uses a two-step process: reconnaissance and vulnerability verification. In the first step it uses Nmap as a port scanner to detect any open ports. Only then uses some fingerprinting techniques to try to identify the services running on each of the previously scanned open ports.

2.4.4 Fuzzing Tools

Checking that an application or a system does what it was designed to do is rather straightforward. On the other hand, validating that an application or a system does not have an unexpected behaviour or that it does not do something that it is not intended to do is more

[18]`https://tools.ietf.org/html/rfc3912`

[19]`https://tools.ietf.org/html/rfc826`

[20]Open Vulnerability Assessment System. Available at `https://www.openvas.org`

[21]`https://github.com/google/tsunami-security-scanner`

difficult to test. This is directly related to the fact that the amount of possible combinations of invalid test cases is considerably larger than the number of so-called positive tests. Generating invalid input data addressed to a given application or system is a difficult task, not to mention it is time consuming. This is where *fuzzing* techniques can help. The main idea behind fuzzing is to generate and submit malformed data to a given application or system. Any malformed data consists of *semi-valid* data, i.e., data that is valid enough to be accepted by the targeted application but that is still invalid enough to have a negative impact. A fuzzer starts by submitting the malformed data to the application or system that is being tested. If the used data manages to cause errors or problems of any kind in the targeted application, the fuzzing tool saves this data for subsequent analysis. This process continues until it reaches the last iteration of malformed data. The fuzzer may still save data even if in any of the iterations the malformed data does not affect the application in question. Otherwise the fuzzer may just exclude the data and continue with the iteration process [Oeh05].

We highlight some freely available fuzzers: *FFUF*[22], *WebSlayer*[23], *sfuzz*[24], and *Wfuzz*[25].

FFUF is an open-source web fuzzer written in Go, with an emphasis on speed.

WebSlayer is designed to *bruteforce* web applications, that comes with a payload generator.

sfuzz was built to be a simple and easy to use fuzzer. It allows to develop command line fuzzing scripts.

Wfuzz provides a framework that allows to automate security assessments of web applications.

2.5 Location Certification Systems

In this section we focus on a specific kind of information system: *Location Certification Systems* and their security and privacy mechanisms. The goal is to prevent *location spoofing attacks* [GAM20]. It is a relevant problem because many applications rely on the location of the device of its users to provide them with services without verifying this information. Therefore making themselves susceptible to such attacks.

There are several reasons for users to cheat, which means systems should deploy the proper measures to defeat the malicious intentions of these users [ZC11]. Instead of only relying on the location of, for example, the GPS signal or the geographical source of the IP address of an end-user device, Location Certification Systems provide additional security to applications by

[22]Fuzz Faster U Fool. Available at `https://github.com/ffuf/ffuf`
[23]The web application bruteforcer. Available at `http://www.edge-security.com/webslayer.php`
[24]Simple fuzz. Available at `https://github.com/foreni-packages/sfuzz`
[25]The Web Fuzzer. Available at `http://www.edge-security.com/wfuzz.php`

issuing Location Certificates [GAM20].

In this section we present systems that provide Location Certification by issuing location certificates (2.5.1). At the end, we highlight one of the most sought after requirements of these systems - the privacy of its users - by studying two other location certification systems (2.5.2).

2.5.1 Location Certification Systems

The location of mobile devices is one area where the use of location certificates has been used for some time. In fact, back in 2009, researchers at Microsoft [SW09] had already been working on a solution that allowed mobile devices to prove their location to an application that might require it. Saroiu and Wolman [SW09] refer that with some adjustments their solution could also be applied to ensure the owner of the device is in fact at the same location as the device itself. The meaning of location in this context is sometimes crucial to the correct use of some applications. With that in mind, the authors mention some examples of applications and their use cases where that is clearly visible. The format used for the Location Certificate the system works with is composed by the Public Key of the issuer, the Public Key of the user to whom the Location Certificate is issued to, the (approximate) location of the device, a timestamp with the current time when the certificate was issued, and a digital signature of the content of the certificate, except the Public Key of the issuer. The authors chose to use the Wi-Fi communication for their solution because it is a well known and widely used standard. It also has the advantage that Access Points (APs) are constantly broadcasting their presence, allowing devices within reach to scan them. Whenever clients need a location certificate they can simply request it to an AP from which they received a beacon. Clients must sign and send the sequence number they previously received from an AP along with their Public Key in the request. On the other end, the AP only issues and sends back a location certificate to the client if the request signature and the sequence number are both valid. Otherwise the AP simply does not reply to the client. The AP adds the identification of that specific client as the recipient of the certificate and a current timestamp, according to the previously mentioned format. To use this certificate the client just needs to sign it and add his/her Public Key before sending it to the application that is asking for a proof of location. Finally, the application just needs to verify if the location certificate has not been tampered with. This last step may vary from application to application, depending on its implementation and security requirements. For that, the application checks the signature of the issuing AP and verifies if the client is indeed the recipient of the location certificate.

There is a downside: the location present in the location certificate is the location of the issuing AP, not the exact location of the device itself. Making the solution proposed by Saroiu and Wolman not suitable to all uses.

2.5.2 Privacy Prevention of Location Certificates

The privacy of the users is an important requirement for Location Certification Systems. These systems not only have to deal with the location of its users but also their identity. Another challenge, as Luo and Hengartner [LH10] state, is that users may request a Location Certificate without the intent of using it right away. Luo and Hengartner refer two possible reasons. One is the fact that users may request a certificate just to have it ready to present to an application at a moment's notice without having to request one when needed. The other reason is that users may also request a proof without knowing the application where they will use it. Which makes Location Certification Systems have to issue certificates without a specific application as the "certificate consumer". Leading these systems to issue more generic certificates which may mean each certificate will contain more information about the user than what is actually needed.

Zhu and Cao propose Privacy-Preserving Location Proof Updating System (APPLAUS) in [ZC11]. It offers users of mobile devices the ability to request and exchange Location Proofs between them. It only requires that a mobile device is *Bluetooth* enabled. Since it uses a Peer-to-peer (P2P) [Sch01] approach there is no need to deploy or rely on an external infrastructure. It uses a server that can be considered as untrusted to store the location certificates after they are issued. When needed, an external application can later retrieve the necessary location certificates from that server, providing it is an authorized verifier. One particular aspect of the APPLAUS system is the fact that the identity of the users is preserved by the use of *pseudonyms* assigned to each mobile device. This way guaranteeing the protection of the identity of the users, not only among themselves but also from the untrusted server.

Luo and Hengartner propose *VeriPlace* [LH10] that focuses on the detection of cheating users without compromising their identity and information privacy, namely from the issuers of proofs. By making the assumption that it is impossible for someone to be at more than one place at a time, *VeriPlace* is able to detect when users try to collect proofs for places where they are not located, i.e. try to fool an application with a false location proof. The proposed solution is divided in two steps: first, a user needs to request an *Intermediate Proof* from an AP; second, the user has to present this proof to a trusted third-party that is responsible for issuing the *Final Proof*. In the first step, similar to the model used in [SW09], users request a proof from a nearby AP. In short, after receiving the request the AP issues a proof by adding its Public Key to it and digitally signing it, issuing the requested location proof. With the *Intermediate Proof* in the possession of the user he/she can now contact a trusted third-party, known in the system as Trusted Third Party for managing Location information (TTPL). When contacted, a TTPL takes the *Intermediate Proof* and replaces the identity of the AP, i.e., the Public Key of AP, with the corresponding geographical location. Lastly, a TTPL removes the signature of the AP and adds its own signature, generating the *Final Proof*. Apart from the TTPL, *VeriPlace* also uses two other entities. One of them - Trusted Third Party for managing User Information (TTPU) - is responsible for storing information associated with the location where users requested the proofs. Aside from the identity of the user, such information is comprised of sets formed by the (encrypted) identity of the proof's issuing AP and the time when a proof was requested. The identity of the AP can then be "translated" to its location, used by a TTPL, as it was described above. The other entity is the Cheating Detection Authority (CDA), used by the system to verify if an user, given by his/her Public Key, is associated with two geographically different AP entries. If it is the case then it means the user is trying to trick an application. As it is one of the main aspects of *VeriPlace*, neither of the three trusted entities - TTPL, TTPU and CDA - knows both the identity of the user and his/her location. *VeriPlace* also offers applications the ability to ask a TTPU for the verification of a given *Final Proof*, presented by an user. The process involves both a TTPU and the CDA. After getting a proof verification request by an external application, a TTPU sends the proof information to the CDA to check if the user to whom the proof was issued is/was cheating. The solution proposed by Luo and Hengartner gives paramount importance to the privacy of the information of the users. *VeriPlace* is able to avoid colluding, when users collaborate with each other in a *wormhole attack* [HPJ03]. Yet, it fails to detect it if the users do it in two diverging periods of time.

Chapter 3

Attack Approach

This Chapter is dedicated to describe how we approached and performed our proposed offensive security assessment. It is worth to mention the attacks were only targeted at the server-side of the target system (3.1) that is connected to the public Internet. We can consider this threat as an alleged outsider in terms of the perspective of the threat itself, as described in 2.2.3.

We start by describing the target system of our attacks in Section 3.1. In Section 3.2 we present and describe the testbed we deployed, where the attacks took place. Section 3.3 presents the selection of attack tools that we gathered and used as our toolset, along with the reasons for each tool choice. In Section 3.4 we describe the steps and the methodology followed in our attack approach, along with a description of the first steps taken before performing any offensive-related attacks. Finally, in Section 3.5 we summarize our proposed attack approach, highlighting some key aspects of our solution.

3.1 Target System

The SureThing [FP18] framework defines a model for location certification, and provides libraries and services to develop systems that issue, verify, and store location certificates. Such systems have in mind deployable mobile applications devices. One of those systems is CROSS (*loCation pROof techniqueS for consumer mobile applicationS*) [GAM20]. Proposed with the tourism use case, CROSS rewards its users after they visit a set of predefined points of interest by following a certain route. CROSS follows a *client-server* model, composed by a server and a client application, respectively *CROSS server* and *CROSS client.*

These mobile applications devices can be described as a very large network where several devices interact with each other [KS18]. These could either be sensors, wearables, like smart watches, or even appliances, like washing machines and refrigerators that can be remotely monitored and/or operated. There are several typical uses for sensors. For instance, a sensor could be tasked with automatically deciding if (and when) it is time to increase or decrease the temperature inside an house.

There are normally two ways of building a network: *Centralized* or *Distributed Approach.* A *Centralized Approach* has a central application platform, focused on acquiring information from the devices that are scattered across the network and then providing data (namely services) to them [RZL13]. On the other hand, in a *Distributed Approach* the data and services provisioning is located at the edge of the network [RZL13]. Both approaches present security challenges. Some of those challenges are protocol and network security, identity, authentication and authorization management, access control, privacy of the data of the users, the trust users have in the system, non-repudiation, accountability, and fault tolerance [RZL13]. A *Centralized Approach* has the disadvantage of being subject to the existence of a single point of failure. Whereas in a *Distributed Approach* each single attack will be smaller and therefore will have less impact. Yet, in this latter approach the number of *attack vectors* is higher, i.e., an attacker has more paths/alternatives of gaining access to the device [RZL13].

The remaining of this Section is dedicated to describe the target system of this work: CROSS. We start by presenting an overview of its architecture in Section 3.1.1. Section 3.1.2 shows the three strategies employed by CROSS. Finally, in Section 3.1.3 we identify some potential attacks.

3.1.1 Architecture

The architecture of CROSS is composed by four main components, represented in Figure 3.1. Those are server, client, *Wi-Fi Access Point* (used as part of the strategy described in 3.1.2), and *kiosk* (used as part of the strategy described in 3.1.2). The client, represented in the lower-left side of Figure 3.1, is an Android application for smartphones. The server is available over the Internet and can accessed by its users with their smartphones through a REST API. Its internal structure is represented on the upper-left side of Figure 3.1. The server is the central component of the system and is responsible to assign rewards, verify location certificates received from clients, provide information about the available points of interest (that consist in tourism routes), and manage the authentication of users and provide information about each specific user.

3.1.2 Location Certification Strategies

The system can be used in one of three different strategies, each with increasingly stronger security guarantees:

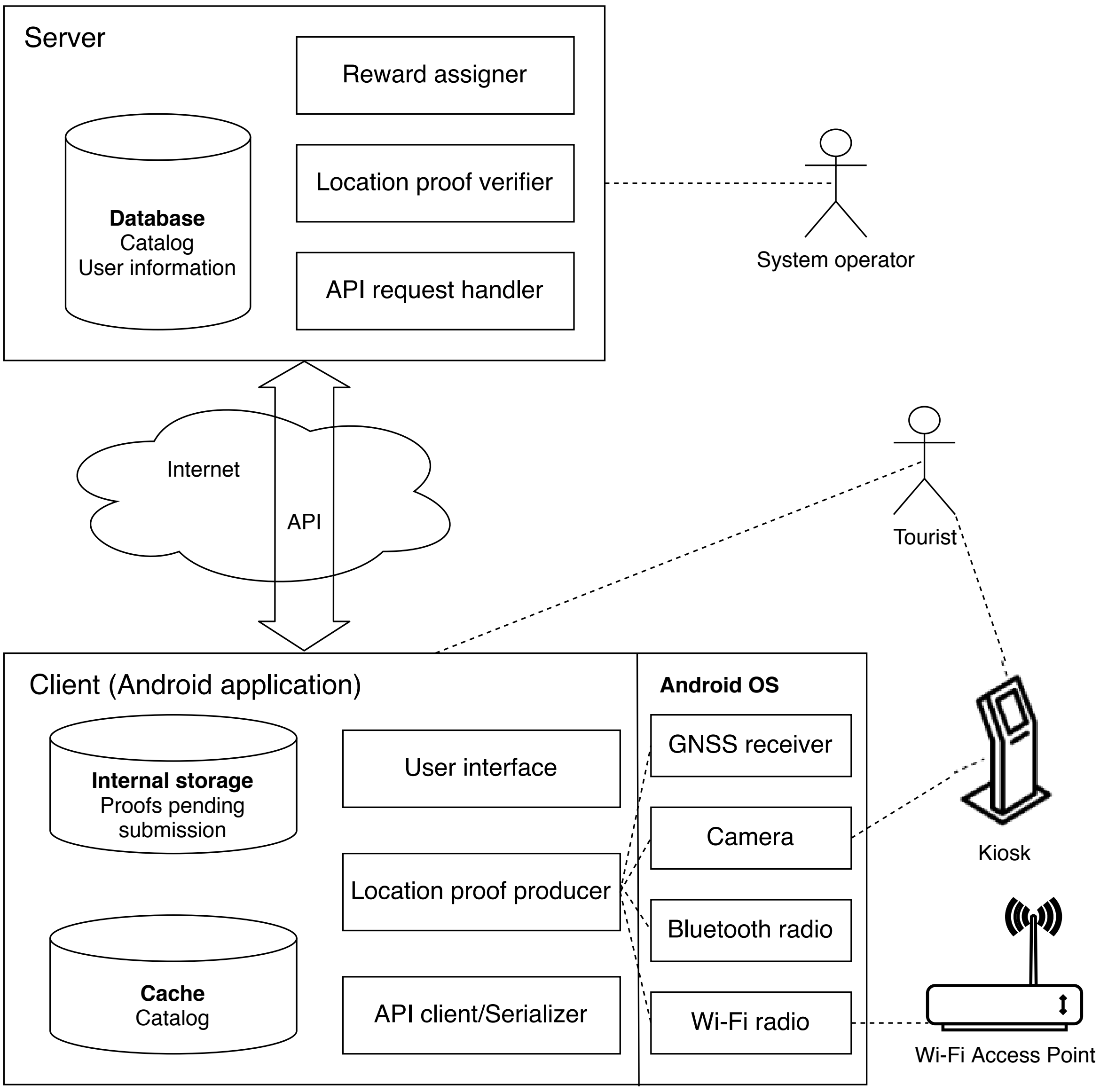


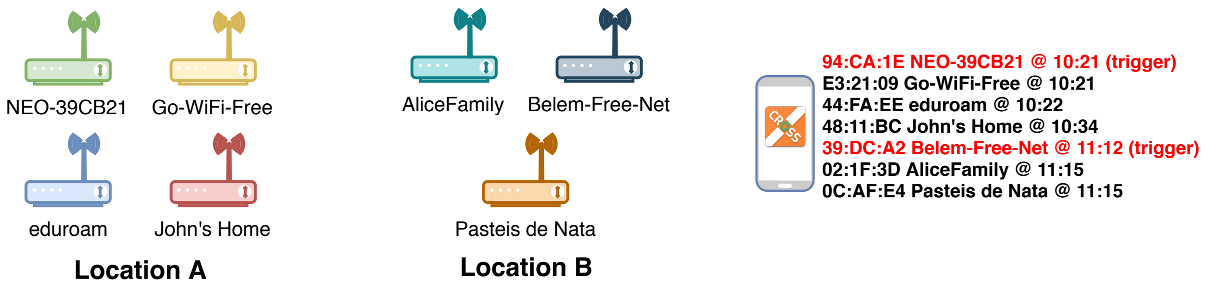
Figure 3.1: Architecture of the CROSS system.

- *Scavenging*;
- *Time-based One-Time Password* (TOTP);
- *Kiosk*.

All of these strategies can be used to verify the location of an user and issue the corresponding certificate.

Scavenging Strategy

The scavenging strategy, represented in Figure 3.2, simply relies on the already existing Wi-Fi networks infrastructure in an urban area - either public or private. An user just needs to be at a certain location and the mobile application scans all nearby networks. On the other side, the server stores a list of previously scanned networks that are known to exist at each location. Each network is associated with a *timestamp* and this data is later submitted to the server for validation. When the server receives the scan results from a client it compares the results with the list it holds. If they match then the server issues a certificate to that user for that precise location.



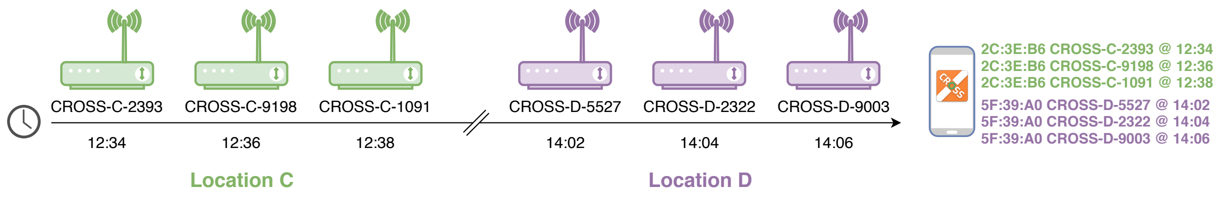
```
94:CA:1E NEO-39CB21 @ 10:21 (trigger)
E3:21:09 Go-WiFi-Free @ 10:21
44:FA:EE eduroam @ 10:22
48:11:BC John's Home @ 10:34
39:DC:A2 Belem-Free-Net @ 11:12 (trigger)
02:1F:3D AliceFamily @ 11:15
0C:AF:E4 Pasteis de Nata @ 11:15
```

Figure 3.2: Representation of a typical usage of the scavenging strategy.

This strategy is restricted by the number of third-party networks available at a certain location. Also, it has to be taken into account the Wi-Fi capabilities of the mobile devices of its users.

TOTP Strategy

The second strategy - TOTP - uses an infrastructure specifically deployed for this purpose, as depicted in Figure 3.3. This strategy adds a one-time value, based on the current time and keyed with an hash function, to the SSID[1] transmitted by each AP that comprises the deployed infrastructure. For the end users this or the scavenging strategy is the same since the client application is prepared to detect which strategy is operating at each location and adapt accordingly without the user ever knowing about it. This strategy prevails over the scavenging strategy if both strategies are in place at a given location.



```
2C:3E:B6 CROSS-C-2393 @ 12:34
2C:3E:B6 CROSS-C-9198 @ 12:36
2C:3E:B6 CROSS-C-1091 @ 12:38
5F:39:A0 CROSS-D-5527 @ 14:02
5F:39:A0 CROSS-D-2322 @ 14:04
5F:39:A0 CROSS-D-9003 @ 14:06
```

Figure 3.3: Representation of a typical usage of the TOTP strategy.

Like the scavenging strategy, the TOTP strategy is also limited by the Wi-Fi capabilities of each mobile device.

[1]Service Set Identifier, the ID a user sees for each Wi-Fi network as its identifying name.

Kiosk Strategy

The *kiosk* strategy requires a physical machine to be placed at a location. A kiosk acts as a trusted witness[2], similar to the role each user plays in [ZC11]. When users are at a location with an available kiosk they need to send their *username* to it in the form of a QR code. The kiosk then uses its Private Key to sign a message that contains the *username* of the user, the ID of the kiosk, the current time, and a nonce, i.e., a one use only randomly generated number. The client application then stores this signed message which can then be used as a proof that the user was at the same location where a given kiosk is. Once this proof is submitted to the server, the server uses the Public Key associated with the kiosk that signed the proof and verifies if the contents of the previously signed message are correct. This strategy requires an user to interact with a kiosk placed at a specific location, therefore requires an user to be physically present, that way preventing *Sybil attacks* [Dou02], where a malicious user tries to cheat the system by creating multiple accounts. It is worth mentioning that this strategy was not available in the implementation of the CROSS system that was tested but it is an ongoing work.

3.1.3 Potential Attacks

An user can cheat in the scavenging strategy once the networks that exist at a given location are known. The scavenging strategy is also subject to *network/Wi-Fi spoofing attacks* because it relies on an unmanaged infrastructure. This third-party infrastructure, formed by any of the already existing Wi-Fi networks, is not controlled and neither is authenticated which can be exploited by what is also known as an *evil twin attack.*

The TOTP strategy offers a stronger security guarantee than the scavenging strategy since the TOTP value changes from time to time and is only known by each AP and by the server (for verification purposes). This strategy is yet vulnerable to an user (now posing as an attacker) to replay priorly issued legitimate certificates later in time. Using this strategy an attacker is also able to send certificates to another user.

The kiosk strategy is able to prevent the two formerly mentioned attacks but is still vulnerable to *Denial of Service* (DoS) attacks.

3.2 Testbed

The deployed testbed consisted in several virtual machine, provisioned by VirtualBox[3]. We opted to use VirtualBox: first, for its widely usage but also because it was previously used in

[2]Device that, as part of a whole system in place, works to testify a user was at a given location.
[3]Virtualization *hypervisor*. Available at: `https://www.virtualbox.org/`

different academic projects. VirtualBox offers the option to create and have snapshots[4], that way we could keep an history of the changes and go back to a previous state of any deployed virtual machine.

The core of the tesbed is composed by the CROSS server and client (Android mobile application), and a machine that would be controlled an by attacker. We also deployed additional virtual machines to simulate the rest of the network. The CROSS server runs inside a virtual machine running 64-bit Ubuntu[5] 20.04. The CROSS client mobile application runs in Android 5.1, emulated using Genymotion[6]. The attacker machine is a 64-bit Kali Linux[7], version 2020.4. Following what an attacker would do, we allow the machine of the attacker to run in promiscuous mode[8]. We also deployed another virtual machine running Ubuntu 20.04 that was used to hold the CROSS client mobile application code and provision it to the Android emulator through Android Studio[9], when needed.

3.2.1 Network Topology

We assumed an attacker can be at two different points in the network: inside the network of the client, or outside of it. In 3.2.1 we describe the two network topologies that were deployed with that end.

Both network topologies were deployed using the *internal network* mode of VirtualBox. The internal network mode allows to create a software-based network composed of only virtual machines, making it the most suitable mode for the testbed we needed to deploy. Only the virtual machine that holds the code of the CROSS client mobile application and the Genymotion emulator are connected to each other through the *host-only* network mode of VirtualBox. This network mode allows virtual machines to communicate with other virtual machines and with the host system. We chose this mode since there was no need to create and have a separate (internal) network solely for connecting these two virtual machines. We then added another network adapter using the *Network Address Translation* (NAT) mode to allow the virtual machine where the code of the CROSS client is kept to have Internet access, e.g., to perform software updates. One of the network adapters of the router posing as the Internet is using the *bridged network* mode, in order to allow all virtual machines deployed using the internal network mode to have Internet access.

[4]Saved state of a particular virtual machine in a specific point in time.
[5]Linux distribution. Available at: `https://ubuntu.com/`
[6]Android emulator. Available at: `https://www.genymotion.com/`
[7]Linux distribution. Available at: `https://www.kali.org/`
[8]In promiscuous mode the wired or wireless network interface controller is allowed to intercept and pass all network traffic it receives to the central processing unit (CPU), even if not specifically meant to it.
[9]`https://developer.android.com/studio/`

The additionally deployed virtual machines to simulate the rest of the network, that were working as routers, were running pfSense[10] CE 2.5.0. We chose pfSense as the software to run inside the virtual machines working as routers because of its versatility. It does not only work as a dedicated router but also as a firewall, which will can come in handy to add some network security to the deployed network. Another advantage of using pfSense instead of using, e.g., iptables[11], is that it has an easy to use web-interface through which users can configure it.

Attacker Inside the Network

In this topology we assume the attacker either already managed to get access to it or is already part of the network where the CROSS client mobile application is. Figure 3.4 shows the topology of such scenario.

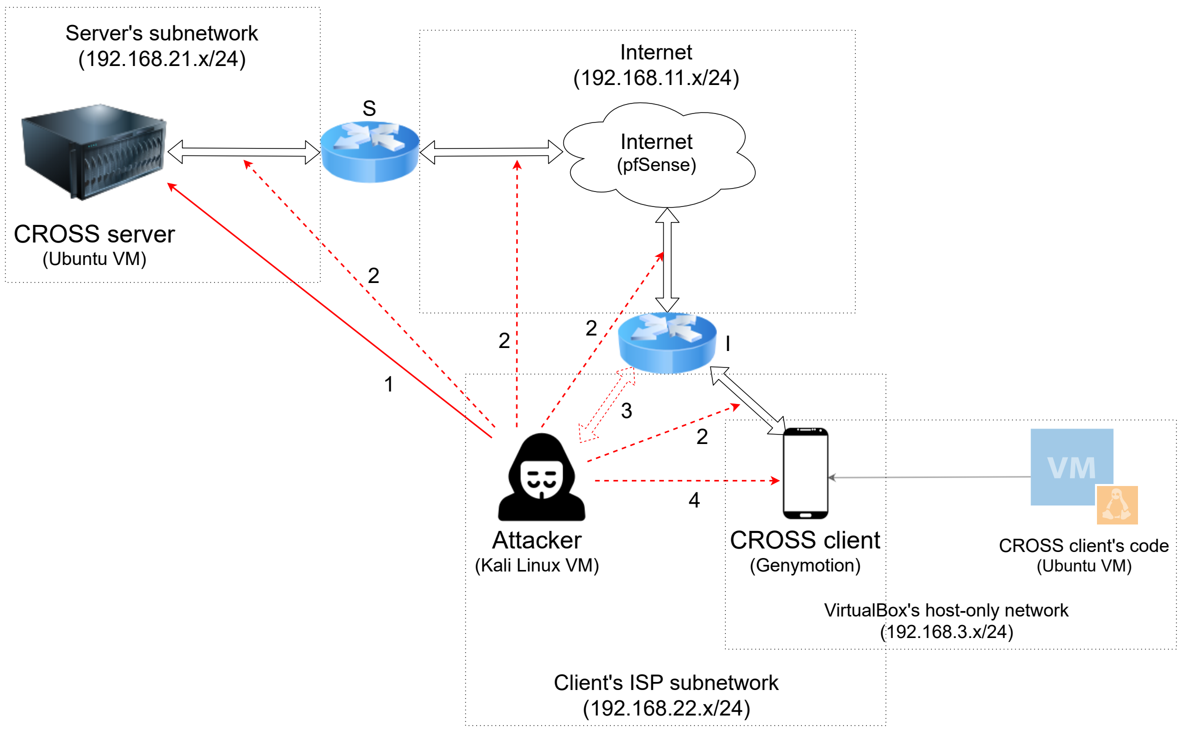


Figure 3.4: Topology of the network when the attacker is inside the network of the client. The solid red arrows represent the attacks that were performed, and the dashed red arrows represent the attacks that were considered but not performed.

In the figure, "S" stands for server and represents the router that belongs to the subnetwork the CROSS server is behind. Also in the figure, "I" stands for ISP[12]. In this case the ISP of the (CROSS) client.

In terms of each network shown in figure 3.4. The IP addresses of the "Internet" router, the

[10]FreeBSD-based firewall distribution. Available at `https://www.pfsense.org/`
[11]`https://www.netfilter.org/`
[12]Internet Service Provider.

router of the subnetwork of the CROSS server, the router of the subnetwork of the CROSS client, and the router of the subnetwork of the machine of the attacker were statically assigned. The "Internet" subnetwork works in the 192.168.11.0/24 range. More specifically, the "Internet" router is statically assigned with the 192.168.11.1 IP address, the router of the subnetwork of the CROSS server with IP address 192.168.11.11, and the router of the subnetwork of the CROSS client has the 192.168.11.12 IP address. The subnetwork where the CROSS server is in runs in the 192.168.21.11/24 - 192.168.21.255/24 range. In turn, the subnetwork of the ISP of the CROSS client is in the 192.168.22.11/24 - 192.168.22.255/24 range. The remaining virtual machines had their IP addresses assigned through DHCP[13]. While permissive, there was the need to add an "any to any rule" to both routers to allow the required communication between the virtual machines.

As for the attacks we assume an attacker can perform following this topology, "1" represents the attacks targeted at the CROSS server itself. The "2"s represent where the attacker can perform passive attacks. Such attacks include replay and impersonation/spoofing attacks. The "3" is used to represent where the attacker may perform network/Wi-Fi spoofing attacks. Finally the attacker may also target and perform attacks specific to the Android application, represented with a "4".

Attacker Outside the Network

In a different scenario, the attacker has no access to the network where the CROSS client application is. Figure 3.5 shows the network topology of this scenario.

In the figure, "S" stands for server and represents the router that belongs to the subnetwork the CROSS server is behind, and "I" stands for ISP. In this case the ISP of the (CROSS) client. Also in the figure, "A" represents the ISP of the attacker.

In the figure, "1" represents the attacks targeted at the CROSS server itself. The "2"s represent the nodes where an attacker may perform passive attacks. The "3" is used to represent where an attacker may perform network/Wi-Fi spoofing attacks. Finally, the attacker may also target and perform attacks specific to the Android application, represented with a "4".

In terms of each network shown in Figure 3.4. The "Internet" router, the router of the subnetwork of the CROSS server, the router of the subnetwork of the CROSS client, and the router of the subnetwork of the machine of the attacker were assigned static IP addresses. The "Internet" subnetwork works in the 192.168.11.0/24 range. More specifically, the "Internet" router is statically assigned the 192.168.11.1 IP address, the router of the subnetwork of

[13]The Dynamic Host Configuration Protocol can automatically assign and distribute IP addresses within a network.

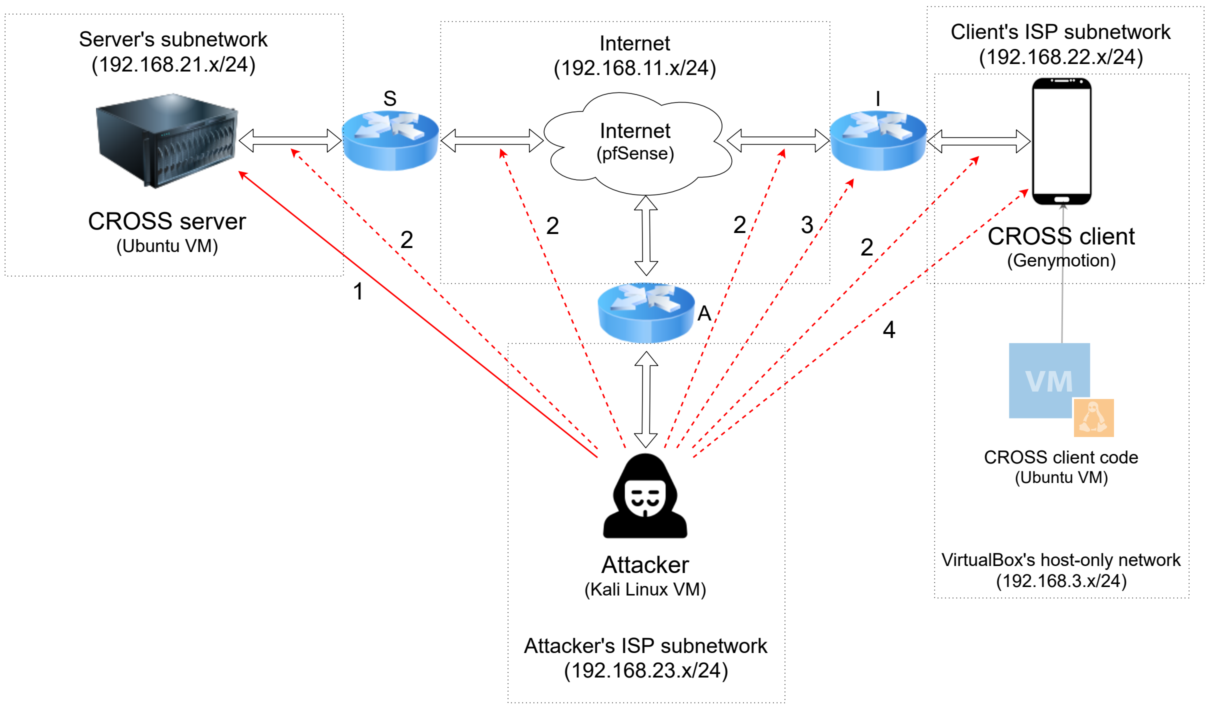


Figure 3.5: Topology of the network when the attacker is outside the network of the client. The solid red arrows represent the attacks that were performed, and the dashed red arrows represent the attacks that were considered but not performed.

the CROSS server with IP address 192.168.11.11, the router of the subnetwork of the CROSS client has the 192.168.11.12 IP address, and the router of the subnetwork of the attacker has the 192.168.11.13 IP address. The subnetwork where the CROSS server is in is in the range 192.168.21.11/24 - 192.168.21.255/24. The subnetwork of the ISP of the CROSS client is in the 192.168.22.11/24 - 192.168.22.255/24 range. The IP addresses of the remaining virtual machines were assigned through DHCP. In turn, the subnetwork of the machine of the attacker is in the range 192.168.23.11/24 - 192.168.23.255/24. Like in the topology described in 3.2.1, there is an "any to any rule" in place in all three routers.

3.3 Attack Tools

This section describes the tools that comprised our toolset. Such tools included enumeration (2.4.2), vulnerability analysis (2.4.3), and fuzzing tools (2.4.4).

We used *Nmap* and complemented it with *AMAP*, as enumeration tools. We opted for *Nmap* as it is a widely known and used port scanner/mapper. Another advantageous aspect is its well-structured and helpful documentation. While not as popular or without a service version detection database as large as Nmap, AMAP is still a good choice to use as a second option.

For vulnerability analysis, we used *OpenVAS* and *Google Tsunami*. We opted to use Open-

VAS because of its popularity and connection with large vulnerability databases (2.3). Google Tsunami was used as a network security scanner. While quite new, Google Tsunami is part of an open source project run by Google which was the main reason for choosing it.

We used fuzzing techniques to verify if the CROSS server (3.1) would stop its normal operation to the point where it would no longer respond to the requests of the deployed client (3.2). We had the constrained that we needed a fuzzer that could be used through the network. After comparing several available fuzzers, we decided to use *FFUF*. Among the fuzzers that were considered, we highlight the following ones: *SIPArmyKnie*, *WebSlayer*, *sfuzz*, *Wfuzz*, and *Powerfuzzer*. We compared those fuzzers through a series of steps, for instance:

- We did several a Google searches and compared the number of returned results for each fuzzer that was initially considered;
- Used *Google Trends*[14] to compare the search volume of each different considered fuzzer throughout the time;
- Took into consideration the novelty and the advertised speed at which each fuzzer performs.

3.4 Attack Iterations

Once we deployed our testbed, we started our approach by using the target system without the intention of performing any offensive actions. The system was first used as a possible non-malicious end user would use it. This way we were able to get an overview of the whole system while putting together some possible attack scenarios. Allowing us to gather some evidence about the attack surface of the target system.

Before starting any offensive-focused activities we had to make sure all data produced during the attacks was properly and automatically logged. It includes commands or any raw data entered in a terminal/console, known as terminal logs. With that in mind, we opted to use the Linux *script* command[15] when performing the attacks from the machine of the attacker.

Our approach consisted in three sequential steps of testing the CROSS server - the target system (Section 3.1) of our work. These steps will be hereinafter referred as iterations or loops. We performed the attacks with the machine of the attacker placed at the two previously defined network positions it can be: inside of the network of the client and outside of it, described in 3.2.1. We followed an incremental approach, by introducing a new tool (either offensive or not)

[14]Google Trends analyzes and compares the popularity of Google search queries. Available at `https://trends.google.com/trends`.

[15]`https://man7.org/linux/man-pages/man1/script.1.html`

at each new iteration. The results gathered from previous iterations were used as the basis for the next ones, that way we could incrementally harden our target system.

The three iterations were divided in two main stage: vulnerability analysis and custom attacks. Figure 3.6 shows a representation of the attack iterations that were performed. As part of the vulnerability analysis stage, we did two separate iterations. The first iteration involved using Nmap (2.4.2) and OpenVAS (2.4.3), in this order. In the second iteration we used Google Tsunami (2.4.3). As part of the second stage, we did a single iteration where we introduced some custom attacks in the form of fuzzing techniques. Those fuzzing techniques were performed by exercising the FFUF (2.4.4) fuzzing tool.

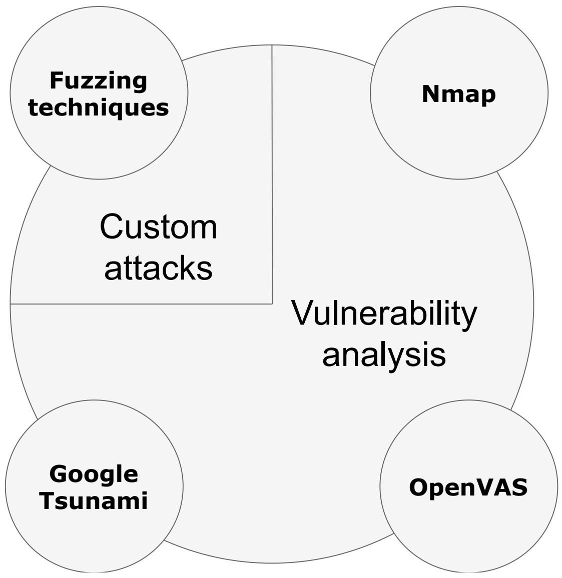


Figure 3.6: Representation of the attack iterations performed.

3.5 Summary

We described our attack approach to perform an offensive security assessment of our target system, by discussing our attack methodology. In Section 3.2 we described our testbed and some of the considerations that were taken into account when deploying it. Namely we detailed the required components, how they were connected, in particular the two deployed types of network topologies and the reason behind those decisions. Section 3.3 then showed the survey of existing attack tools that was done and some of the considerations behind each tool choice. Finally, we presented our attack approach methodology in Section 3.4.

Chapter 4

Evaluation

In this Chapter we present and discuss our evaluation of the target system. Section 4.1 describes the methodology we used to evaluate the CROSS server: both in terms of the security of its architecture and its implementation. In Section 4.2 we present the results we gathered from performing the attacks. Finally, we use Section 4.3 to analyze and discuss the gathered results: how they were used to help harden the CROSS server, and how its harden version compare with its initial one.

4.1 Methodology

Our assessment of the security of CROSS was done by performing an offensive security test. Our approach consisted in three separate attack iterations, discussed in Section 3.4. Each attack iteration then consisted in the exercise of a different tool. We used and assessed each attack tool listed in Section 3.3 individually. The output of several tools could be combined to perform more advanced attacks.

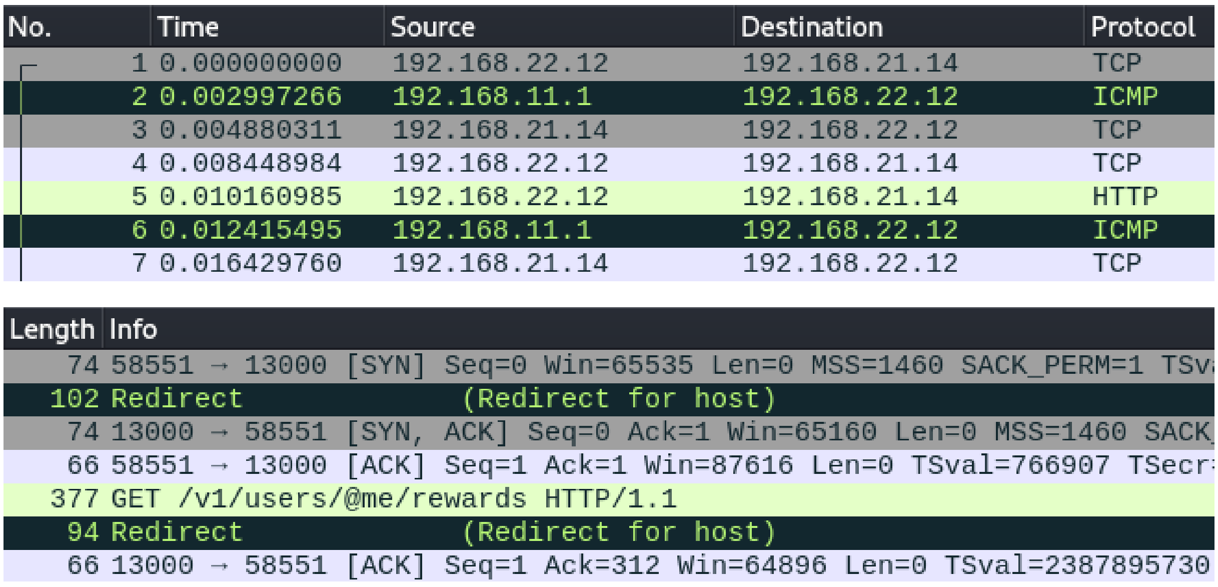
4.2 Attack Results

In this Section we describe the results that were gathered from performing our assessment of CROSS. We start by describing the first reconnaissance steps, before actually performing actual attacks. Next, we present the results from running the different tools. To recall, the attacks were done with the machine of the attacker in each of the two network positions we defined earlier (Section 3.2.1) an attacker can be.

4.2.1 Initial Steps

We started by finding the IP address of the CROSS server, even though we knew the IP of all deployed virtual machines beforehand. Following, we present those steps:

1. Initialization of Wireshark[1];
2. Start capture of packets on the *eth0* interface of the machine of the attacker;
3. Use the CROSS client application to request the rewards page (`/v1/users/@me/rewards`);
4. Infer the IP address of the server is `192.168.21.14`, by inspecting the exchanged packets shown in Figure 4.1;
5. Conclude that the CROSS server service is running at port 13000, by inspecting the packets exchanged between the CROSS client application and the CROSS server;
6. Infer all available path are sub-directories of the root `/v1` path.



No.	Time	Source	Destination	Protocol
1	0.000000000	192.168.22.12	192.168.21.14	TCP
2	0.002997266	192.168.11.1	192.168.22.12	ICMP
3	0.004880311	192.168.21.14	192.168.22.12	TCP
4	0.008448984	192.168.22.12	192.168.21.14	TCP
5	0.010160985	192.168.22.12	192.168.21.14	HTTP
6	0.012415495	192.168.11.1	192.168.22.12	ICMP
7	0.016429760	192.168.21.14	192.168.22.12	TCP

Length	Info
74	58551 → 13000 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM=1 TSv
102	Redirect (Redirect for host)
74	13000 → 58551 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_
66	58551 → 13000 [ACK] Seq=1 Ack=1 Win=87616 Len=0 TSval=766907 TSecr
377	GET /v1/users/@me/rewards HTTP/1.1
94	Redirect (Redirect for host)
66	13000 → 58551 [ACK] Seq=1 Ack=312 Win=64896 Len=0 TSval=2387895730

Figure 4.1: Wireshark packet capturing on the attacker machine that shows packets exchanged between the CROSS-client mobile application and the CROSS-server after requesting the rewards of the client. The "Length Info" columns correspond to rows 1-7 of the captures.

4.2.2 Nmap Results

We used Nmap to perform several port scans of the target system. Some of the commands that were used to run Namp can be seen in appendix (B.1). Starting with a simple port scan, we used the option to probe any detected open ports to determine the service and version running

[1]Wireshark is a free and open-source network packet analyzer. Available at `https://www.wireshark.org/`.

at each port. By default, Nmap only scans the most common 1000 TCP ports. As a result, this initial port scan did not detect any open ports; we only gathered the information that the target system was at a distance of four hops from the machine of the attacker.

Running some broader scan - that included all TCP and UDP ports - we were able to detect that TCP port 13000 was open, as expected. Again, using the option to probe any detected open ports for their service. The service detection of Nmap wrongly detected the service running at the TCP port 13000 as a DAAP[2] service. Yet, we used the *daap-set-library* script from the Nmap Scripting Engine (NSE) to try to get more information about this service but we did not get any further results.

As a result of all the port scans - targeted at all TCP and UDP ports - Nmap detected the UDP ports 631, 5353, and 53574, but as *open|filtered*. Which means Nmap was not able to determine if these ports were actually open or were simply filtered, since it did not get a response from them. Nmap may also report a port as *open|filtered* if the probe used by Nmap was dropped by a packet filter. This was not the case given our testbed configuration (3.2)and deployed network topologies (3.2.1). The UDP port 631 that Nmap detected is usually used for running the IPP[3] service. In our case, the service running at port 631 was CUPS[4]. The UDP port 5353 detected by Nmap is registered as the port to run the mDNS[5] protocol. The third UDP port Nmap detected as *open|filtered* - 53574 - has no common service associated with it. Perhaps for that reason Nmap was not able to detect the service running/listening at that port.

Furthermore, in a different configuration of the virtual machine, Nmap was able to detect the PostgreSQL database service used by the CROSS server was listening at the port TCP port 5432. In this configuration, we explicitly configured PostgreSQL to accept outside connections to confirm the previous gathered results.

Overall, Nmap correctly detected the Operating System of the virtual machine where the CROSS server service is running as Linux. Yet, it did not detect its exact Linux distribution (Ubuntu), nor its exact version. Also, Nmap could not accurately detect its version of the Linux kernel, even running the scans with the Operating System detection option enabled. These results could be related with the fact that the target system was running inside a virtualized environment.

After exploring several options and configurations with Nmap, we decided to complement it by using AMAP. The commands that were used to run AMAP can be seen in appendix (B.2).

[2]The Digital Audio Access Protocol is a proprietary protocol introduced by Apple in its iTunes software to share media across a local network.

[3]The Internet Printing Protocol is used to allow client devices, such as computers, to communicate with printers.

[4]The Common Unix Printing System allows a Unix-based computer to operate as a printing server.

[5]The multicast DNS protocol can resolve hostnames to the corresponding IP addresses within a small network.

We used AMAP to scan the TCP port 13000, since Nmap was unable to correctly determine the service running there. We started by making sure AMAP also detected TCP port 13000 as open. We then tried to map the service at that port by sending some triggers and analyzing their responses, which did not produce any relevant information. AMAP returned the following response: *HTTP/1.0 404 Not Found\r\nDate Wed, 26 May 2021 152516 GMT\r\nContent Length 0\r\n\r\n.*

4.2.3 OpenVAS Results

OpenVAS was used to perform several different scans: scans to all ports of both TCP and UDP, all ports of only one of the protocols (TCP or UDP), or only targeted at the TCP port 13000 of the virtual machine where the CROSS server service was running.

OpenVAS only detected one vulnerability during all performed scans. The only vulnerability OpenVAS was able to detected was the implementation of TCP timestamps[6] in the CROSS server virtual machine. This vulnerability (OID 1.3.6.1.4.1.25623.1.0.80091) is scored 2.6 in the base group of CVSS and is therefore considered a low severity vulnerability.

Changes

Here we describe the changes that were made in order to harden the target system. Given the nature of the detected vulnerability, OpenVAS suggested disabling the usage of TCP timestamps in the CROSS server virtual machine as a mitigation strategy. The solution involved editing the */etc/sysctl.conf* file, which requires root privileges. There we had to add the line *net.ipv4.tcp_timestamps = 0* and then execute the *sysctl -p* command in order to apply the changes in Linux. The equivalent procedure would be different for other operating systems. Figure 4.2 shows a representation of the changes that were made on the configuration of the target system in order to harden its security.

Avoided Attacks

The implementation of the TCP timestamps feature can be used by an attacker to compute the *uptime* of a system, i.e., compute for how long a system has been booted. An attacker can achieve this by comparing the response of a system with responses known to be particular to that type of system, either common to the Operating System running or installed software. An attacker may then be able to infer if a system is still running a previous version of its Operating

[6]TCP timestamps are defined in RFC 7323, TCP Extensions for High Performance. Available at `https://tools.ietf.org/html/7323`.

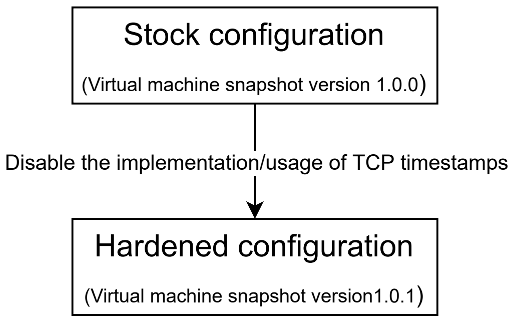

Stock configuration
(Virtual machine snapshot version 1.0.0)
Disable the implementation/usage of TCP timestamps
Hardened configuration
(Virtual machine snapshot version1.0.1)

Figure 4.2: Changes made to the configuration of the CROSS-server to harden it according to the changes suggested by OpenVAS.

System or installed software, as some updates or patches require a system reboot. By disabling this feature these attacks are made more difficult.

4.2.4 Google Tsunami Results

By default, Google Tsunami only uses Nmap to scan the most common 1000 TCP ports as part of its reconnaissance process. Therefore, the port where the CROSS server is listening at (13000) was not detected. Thus, Google Tsunami did not proceed to the next step - vulnerability verification (2.4.3). We knew TCP port 13000 was open so we decided to change the configuration Google Tsunami uses by default. Launching Google Tsunami again, that port was promptly detected as open once we overrode the default values that specify which ports Nmap should scan. The process employed by Google Tsunami continued normally, since it detected an open port in the scanned system. During the vulnerability verification step, Google Tsunami used fingerprinting techniques to try identify which service was running on that port. As expected, because of the use of Nmap, TCP port 13000 was again detected as a DAAP service (4.2.2). Finally, Google Tsunami used its default vulnerability detection plugins targeted at that port but did not find any vulnerabilities.

As presented, Google Tsunami did not return any results that we could use. Therefore, we made no changes to harden the target system.

4.2.5 Fuzzing Results

As part of the second stage of attacks (3.4), we performed an iteration where we introduced some fuzzing techniques. The results that we gathered are presented and discussed below.

FFUF (3.3) was used with two wordlists[7]: *SecLists*[8] and *RobotsDisallowed*[9].

[7]Wordlist is a text file with a list of words: possible or known passwords, usernames, fuzzing payloads, or data pattern matching.

[8]Collection of multiple wordlists: from usernames and password, to data pattern matching. Available at `https://github.com/danielmiessler/SecLists`.

[9]Contains lists with the most commonly disallowed directories present in *robots.txt* from top websites. Available

All fuzzing techniques were done assuming `/v1` as the root path, based on the previously gathered information (4.2.1). We started by doing a directory and file discovery, followed by sending some POSTS requests, and database enumeration. A list with some of commands that were used to run FFUF can be seen in appendix (B.3).

With directory directory discovery were able to find seven main directories (under the `/v1` root path):

- `/users`, that returned the 405 (Method Not Allowed) HTTP code;
- `/meta`, which returned a 200 (OK) HTTP code;
- `/rewards`, which returned a 200 HTTP code;
- `/trips`, that returned the 401 (Unauthorized) HTTP code;
- `/routes`, which returned a 200 HTTP code;
- `/datasets`, which returned a 200 HTTP code;
- `/pairs`, that returned the 405 HTTP code.

Using a different wordlist from RobtosDisallowed, we were able to find a sub-directory of the `/trips` path - `/trips/upcoming` - that returned a 401 HTTP code.

We were not able to find any sub-directories of the `/users` path, nor of the `/meta` path. Given the results (none) we did not try to find sub-directories of the other discovered main directories. Instead decided to pursue other ways of fuzzing.

We also tried to do file discovery by targeting the same two `/users` and `/meta` endpoints, as before. We used two different types of wordlists: directory discovery and commonly used file extensions. As worlists for common file extensions, we used one wordlist with common file extensions and other with common web files extensions. Together with a wordlist designed to perform directory discovery, we used each of the two other wordlists that contained known file extensions. Yet, we were not able to get any results. Following the same strategy as before, we decided to not continue with file discovery targeted at the other discovered endpoints.

We decided to send some POST requests as both `/users` and `/pairs` endpoints returned a 405 HTTP code to the previous requests. We used different wordlists that contained values meant to disrupt the normal function of a system, and sent some POST requests to those two endpoints. Of the hundred values that were sent, only the *null* value was accepted by the `/users` and the `/pairs` endpoints. With both endpoints returning a 200 HTTP code to said requests.

at `https://github.com/danielmiessler/RobotsDisallowed`.

We got 409 (Conflict) HTTP codes for the following POST requests sent to those paths, also with the *null* value. Namely, the `/users` path responded with the following: *"message": Username in use.* Meaning we were trying to insert the same value in the same table of the database. Later we could verify the *null* was in fact inserted in the database, by inspecting the two table of the database of the server from where the `/users` and the `/pairs` endpoints fetch information. We could also verify these two unauthorized writes to the database did not manage to do much harm. Nonetheless, it were two unauthorized writes made to two quite important tables of the database used by the server. In the case of the *null* value accepted by the `/users` endpoint, the table where the users of the system and corresponding usernames are stored. The remaining values sent in POST requests to both paths either failed or could not be decoded by the server, all returning 400 (Bad Request) HTTP status codes.

We used two different wordlists to send some POST requests as part of database enumeration. Yet, we were not able to get any further information. Actually, all requests returned a 400 HTTP code followed by the *"invalid character"* response.

The amount of requests sent to the server while exercising the mentioned fuzzing techniques almost worked as a DoS attack. We confirmed it by using the CROSS-client mobile application at the same time and verifying the server was indeed taking much more time to respond to the requests sent using the application.

4.3 Discussion

We used Nmap and AMAP to run several port scans but did not find any unnecessary open ports that could be used by an attacker as an entry point. Nmap and AMAP both did detect the port where the CROSS server was running in the deployed virtual machine, but none were able to correctly detect its service. OpenVAS was used to run several vulnerability analysis scans yet, it only detected a low severity TCP-related vulnerability. We were able to discover several endpoints available in the server through directory discovery. The most important discovery was the discovery of an unknown vulnerability. Using a specific value, FFUF was able make unauthorized writes directly in the database of the server, even though these writes were not able to do much damage. While none of the performed fuzzing techniques managed to stop the server completely, it was enough to disrupt its normal operation. These fuzzing techniques produced quite a large number of requests, almost creating a DoS attack. The obtained results were then used to harden the configuration of the target system, consequently reducing its attack surface.

As expected, the results were the same for either of the two network positions so we only presented the gathered results for one of them. It is worth noting the initial configuration was left untouched. In this configuration we allow the free communication between all virtual machines, as previously noted for both network topologies in 3.2.1. Having full access to them allows for more accurate results when running the vulnerability assessment and security scanning tools. In a real scenario this could not be the case or we could not make this assumption but our objective was to know if and which vulnerabilities were present in the target system.

The tools that were used did not manage to produce a large amount of results that could then have been used to further harden the target system. Nonetheless, we believe we can attest the harden version of the CROSS server is more secure, comparing to its initial configuration that was subject to our security assessment.

Chapter 5

Conclusion

There are many components in a deployed mobile application, each with its own complexity. The mobile application interacts with mobile operating systems, runs on user-owned devices, and relies on communication with remote servers through a remote call API, with calls made over cellular and other networks. The API is exposed to the public Internet and receives requests both from legitimate clients but also from other sources, many of them malicious in nature. The server leverages a programming language run-time and needs to implement complex application logic, that evolves over time, and where there may be bugs. The data storage relies on databases for persistent storage and transaction support, i.e., making sure that changes are made in consistent ways and with lasting effects. Each of these components, that we deployed in a full virtualization, can be the target of attacks.

Different systems will apply diverging security-related defensive mechanisms, yet there are common ways and strategies that can be followed in order to have a system tested in terms of the (negative) impact a given threat might have in it. This work showed an offensive assessment of a deployed system that involved vulnerability assessment and penetration testing techniques from the perspective of an attacker.

5.1 Achievements

Our whole attention was on the security of a specific location certification system, namely CROSS (*loCation pROof techniqueS for consumer mobile applicationS*) but our approach can be adapted to other systems. We suggested some general guidelines to perform vulnerability assessment and penetration testing techniques when applied and including the perspective of a real attacker.

Our security assessment was done with different tools, including *Nmap*, *AMAP*, *OpenVAS*, *Google Tsunami*, and *FFUF*. *Nmap* and *AMAP* ran several port scans but were not able to find unnecessary open ports that could be used as an entry point. *OpenVAS* was only able to detected a low severity TCP-related vulnerability. Using a specific value, we were able to make unauthorized writes in the database using *FFUF*. Even though it did not make much damage, it was an unknown vulnerability, and showed that this approach has merits. The results we obtained were then used to improve and harden the target system and reduce its attack surface.

Once all the ground work was done - full virtual environment and tool survey - we were able to prepare all the materials for organized tournaments, where multiple red teams can compete. Allowing a system to be under the evaluation of quite different backgrounds and ways of conducting an offensive security test.

5.2 Future Work

This work would benefit from the addition of more attack tools to the toolset, namely exploitation tools.

Apart from that, there are some specific attacks that could enrich this work. Those include:

- *Impersonation/spoofing* attacks, where an attacker assumes the identity of an already existing user of the system;
- *Replay* attacks - where an attacker captures a given message (or part of it) intended to a specific user and, after saving it for a certain period of time, sends it to the recipient at a later time;
- *Network spoofing* attacks;
- Ultimately trying to disrupt the main contribution of CROSS and SureThing by claiming false location certificates, i.e., claim a certificate for a location where the attacker is not in fact, achieving location spoofing.

The whole architecture of CROSS could only be considered as more secure after the mobile client Android application is also assessed in terms of its security through the same offensive security assessment proposed in this work.

Finally, organizing several tournaments where multiple teams will have the chance to perform the same attacks we proposed but in a gamified environment, allowing the discovery of new attacks. These tournaments can be gradually opened to more teams/players at each new tournament. The results from these tournaments could then be collected and analyzed to further enrich the offensive security assessment. This way allowing CROSS to be under the evaluation of quite different backgrounds and ways of conducting an offensive security test. We include materials and guidelines for organizing such tournaments in appendix C.

The materials in appendix C were prepared to be used as part of this work but can easily be modified to work as guidelines to perform offensive security assessment of other systems. For instance, we suggest following a continuous type of results submission but a one-time type of submission can also be adopted, among others, that can include a mix of these two. In a continuous type of submission, teams (or players) can submit their results as soon as they successfully perform an attack. This way leaving the reporting stage (and corresponding overview of the gathered results) for last, allowing teams to solely focus on it at the time. In the other mentioned way of submitting the results - one-time submission - teams are asked to submit their results one single time at the end, before the submission deadline.

5.3 Final Words

Recalling to our initial citation: "*A system cannot be said to be truly secure until it has experienced an attack from a real threat*" [TV20]. With this work, we contribute to the promotion of better security practices by assessing the security of a system from the perspective of an attacker, while taking into account the defensive strategies. At the end of this work, we believe we can claim that the default approach to a security assessment should include both a defensive perspective - blue team - and an offensive perspective - red team. This way guaranteeing a information system can be built taking into consideration the knowledge each perspective can add to the security of a system.

Bibliography

[Dou02] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[FP18] João Ferreira and Miguel L. Pardal. Witness-based location proofs for mobile devices. In *17th IEEE International Symposium on Network Computing and Applications (NCA)*, November 2018.

[GAM20] Miguel L. Pardal Gabriel A. Maia, Rui L. Claro. CROSS City: Wi-Fi Location Proofs for Smart Tourism. 2020.

[HH05] Simon Hansman and Ray Hunt. A taxonomy of network and computer attacks. *Computers & Security*, 24(1):31–43, 2005.

[HPJ03] Y-C Hu, Adrian Perrig, and David B Johnson. Packet leashes: a defense against wormhole attacks in wireless networks. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, volume 3, pages 1976–1986. IEEE, 2003.

[KS18] Minhaj Ahmad Khan and Khaled Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.

[LH10] Wanying Luo and Urs Hengartner. Veriplace: a privacy-aware location proof architecture. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 23–32, 2010.

[MW10] Pratyusa K Manadhata and Jeannette M Wing. An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386, 2010.

[Oeh05] Peter Oehlert. Violating assumptions with fuzzing. *IEEE Security & Privacy*, 3(2):58–62, 2005.

[RZL13] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57(10):2266–2279, 2013.

[SAM$^+$18] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. *Technical report*, 2018.

[Sch01] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 101–102. IEEE, 2001.

[SW09] Stefan Saroiu and Alec Wolman. Enabling new mobile applications with location proofs. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6, 2009.

[TS10] Anshu Tripathi and Umesh Kumar Singh. Towards standardization of vulnerability taxonomy. In *2010 2nd International Conference on Computer Technology and Development*, pages 379–384. IEEE, 2010.

[TV20] J. Tubberville and J. Vest. *Red Team Development and Operations: A Practical Guide*. Independently Published, 2020.

[ZC11] Zhichao Zhu and Guohong Cao. Applaus: A privacy-preserving location proof updating system for location-based services. In *2011 Proceedings IEEE INFOCOM*, pages 1889–1897. IEEE, 2011.

Appendix A

STRIDE and MITRE ATT&CK, and DREAD and MITRE ATT&CK Mappings

STRIDE categories	MITRE ATT&CK	
	Tactic(s)	Technique(s)/Sub-technique(s)
Spoofing	Impact	Data Manipulation
Tampering	Impact	Disk Wipe
		Data Destruction
		Data Manipulation
Repudiation	Defense Evasion	Subvert Trust Controls: Code Signing
Information Disclosure	Credential Access	Input Capture
		Man-in-the-Middle
		Network Sniffing
		Unsecured Credentials
	Collection	Man in the Browser
		Man-in-the-Middle
		Input Capture
		Email collection
	Exfiltration	Automated Exfiltration
		Exfiltration Over Alternative Protocol
	Impact	Data Manipulation
Denial of Service	Impact	Service Stop
		Network Denial of Service
		Endpoint Denial of Service
Elevation of Privilege	Privilege Escalation	Abuse Elevation Control Mechanism
		Boot or Logon Autostart Execution
		Boot or Logon Initialization Scripts
		Create or Modify System Process

Table A.1: STRIDE categories and corresponding MITRE ATT&CK Tactics and Techniques.

DREAD categories	MITRE ATT&CK	
	Tactic(s)	Technique(s)/Sub-technique(s)
Damage	Impact	Data Destruction
		Data Manipulation
		Disk Wipe
		Endpoint Denial of Service
		Network Denial of Service
Reproducibility	Initial Access	Replication Through Removable Media
		Drive-by Compromise
		Hardware Additions
	Execution	Inter-Process Communication
		Scheduled Task/Job
	Persistence	Boot or Login Autostart Execution
		Boot or Logon Initialization Scripts
		Browser Extensions
		Event Triggered Execution
		Pre-OS Boot
		Scheduled Task/Job
	Defense Evasion	Rootkit
		Template Injection
		Valid Accounts
Exploitability	Initial Access	Exploit Public-Facing Application
	Execution	Exploitation for Client Execution
	Privilege Escalation	Exploitation for Privelege Escalation
	Defense Evasion	Exploitation for Defense Evasion
	Credential Access	Exploitation for Credential Access
	Lateral Movement	Exploitation of Remote Services
Affected users	Impact	Account Access Removal
		Data Destrcution
		Disk Wipe
		Network Denial of Service
Discoverability	Defense Evasion	Exploitation for Defense Evasion
		File and Directory Permissions Modification
		Hide Artifacts
		Indicator Removal on Host
		Masquerading
		Modify Registery
		Obfuscated Files or Information
		Valid Accounts

Table A.2: DREAD categories and corresponding MITRE ATT&CK Tactics and Techniques.

Appendix B

Command-line Tools Commands

B.1 Nmap Commands

- *nmap -v -r -A -sV 192.168.21.14*;
- *nmap -v -p 13000 192.168.21.14*;
- *nmap -v -r -A -sV -Pn –version-all -p- 192.168.21.14*;
- *nmap -v -r -A -sV –osscan-guess -sS -sU -p- -O -T5 192.168.21.14*;
- *nmap -v -sV –script=daap-set-library -p 13000 192.168.21.14*;
- *nmap -vv -sV –version-all –version-trace -sT -sC -p 13000 192.168.21.14*;
- *nmap -vv -sV –script=cups-info -sU -p 631 192.168.21.14*;
- *nmap -vv -sV –script=dns-service-discovery -sU -p 5353 192.168.21.14*.

B.2 AMAP Commands

- *amap -Pbqvv 192.168.21.14 13000*;
- *amap -Abqv 192.168.21.14 13000.*

B.3 FFUF Commands

- *./ffuf -v -u http://192.168.21.14:13000/v1/FUZZ -w /usr/share/sec-lists/Discovery/Web-Content/directory-list-2.3-small.txt -r -recursion -recursion-depth 2 -ic*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/FUZZ -w /usr/share/sec-lists/Discovery/Web-Content/directory-list-2.3-big.txt -r -ic*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/FUZZ -w /usr/share/robots-disallowed-lists/top10000.txt -r*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users/FUZZ -w /usr/share/sec-lists/Discovery/Web-Content/directory-list-2.3-small.txt -r -recursion -recursion-depth 2 -recursion-strategy greedy -ic*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users/FUZZ -w /usr/share/sec-lists/Discovery/Web-Content/directory-list-2.3-small.txt -e $(cat /home/pen-test/Documents/CROSS_server_attacks/fuzzing/SecLists_comma_separated_most_common_extensions.txt) -r -ic*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users/FUZZ -w /usr/share/sec-lists/Discovery/Web-Content/directory-list-2.3-small.txt -e $(cat /home/pen-test/Documents/CROSS_server_attacks/fuzzing/SecLists_comma_separated_web_extensions.txt) -r -ic*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/big-list-of-naughty-strings.txt -mc 200, 201, 204, 301, 302, 307, 401, 403, 405, 409, 413, 415 -ic*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/Metacharacters.fuzzdb.txt -mc 200, 201, 204, 301, 302, 307, 401, 403, 405, 409, 413, 415*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users -X POST -H "Content-Type: application/json" -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/JSON.Fuzzing.txt -mc 200, 201, 204, 301, 302, 307, 401, 403, 405, 409, 413, 415*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/Databases/Postgres-Enumeration.fuzz.db.txt -mc all*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/users -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/Databases/MySQL.fuzzdb.txt -mc all*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/meta/FUZZ -w /usr/share/sec-lists/Discovery/Web-Content/directory-list-2.3-small.txt -r -recursion -recursion-depth 2 -recursion-strategy greedy -ic*;

- *./ffuf -v -u http://192.168.21.14:13000/v1/meta/FUZZ -w /usr/share/sec-lists/Discovery/Web-Content/directory-list-2.3-small.txt -e $(cat /home/pen-test/Documents/ CROSS_server_attacks/fuzzing/SecLists_comma_separated_most_common_extensions.txt) -r -ic*;
- *./ffuf -v -u http://192.168.21.14:13000/v1/meta?FUZZ=unknown_value -w /usr/share/sec-lists/Discovery/Web-Content/burp-parameter-names.txt -ac*;
- *./ffuf -v -u http://192.168.21.14:13000/v1/pairs -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/big-list-of-naughty-strings.txt -mc 200, 201, 204, 301, 302, 307, 401, 403, 405, 409, 413, 415 -ic*;
- *./ffuf -v -u http://192.168.21.14:13000/v1/pairs -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/Metacharacters.fuzzdb.txt -mc 200, 201, 204, 301, 302, 307, 401, 403, 405, 409, 413, 415*;
- *./ffuf -v -u http://192.168.21.14:13000/v1/pairs -X POST -H "Content-Type: application/json" -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/JSON.Fuzzing.txt -mc 200, 201, 204, 301, 302, 307, 401, 403, 405, 409, 413, 415*;
- *./ffuf -v -u http://192.168.21.14:13000/v1/pairs -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/Databases/Postgres-Enumeration.fuzz.db.txt -mc all*;
- *./ffuf -v -u http://192.168.21.14:13000/v1/users -X POST -d "FUZZ" -w /usr/share/sec-lists/Fuzzing/Databases/MySQL.fuzzdb.txt -mc all.*

Appendix C

Red Team Tournament Materials

This appendix includes the content of a set of web pages written to introduce and describe the organization of a game to be played by competing red teams. The remaining materials are available at `https://github.com/inesc-id/SureThingTournament/tree/main/tournament_materials`.

C.1 Foreword

This tournament is organized as part of the research work being developed by José Ferrão[1] for the Master's Thesis titled *Attack SureThing! Offensive security assessment of a location certification system*, which is part of the SureThing project[2].

The main goal of this work is to assess the security of the architecture supporting the CROSS application. CROSS is a *smart tourism* application that rewards its users after they visit a set of predefined points of interest in a given city. Location certificates are issued for each point of interest. The mobile application interacts with the server through a REST API exposed on the public Internet.

The following figure illustrates the overall system:

C.2 Requirements

In order to be part of this tournament, players needs to have access to a computer with a minimum of 8GB of RAM (16GB recommended), and at least 110GB of free hard disk space for the virtual machines.

[1]José Ferrão Fénix page: `https://fenix.tecnico.ulisboa.pt/homepage/ist426143`
[2]SureThing project website: `http://surething-project.eu`

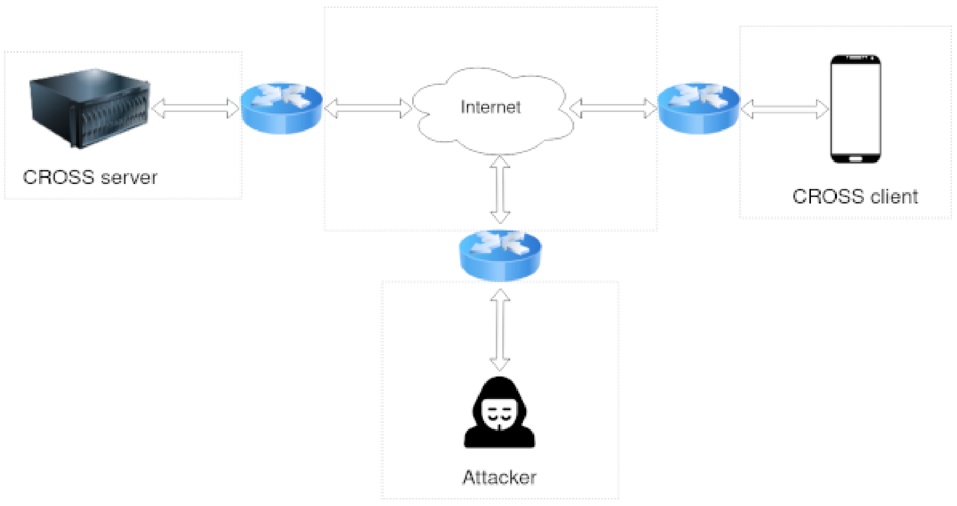


Figure C.1: Diagram of the topology of the network used as part of tournaments.

C.3 Objectives of the Tournament

This tournament is meant to act as a way of evaluating the work done so far. Players are asked to perform a set of attacks that will then be used to compare with the attacks already performed. Players will also be asked to perform new attacks.

C.4 Target System

The target system of this tournament is CROSS (lo**C**ation p**RO**of technique**S** for consumer mobile application**S**), a smart tourism application that uses the SureThing[3] framework to reward its users after they visit a set of predefined points of interest in a given city. CROSS is composed by a server (CROSS-server[4]) and a mobile client application (CROSS-client[5]):

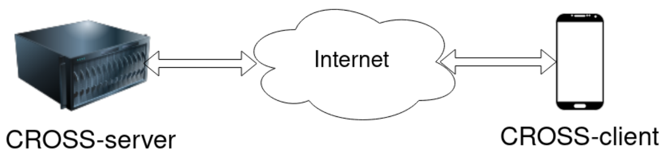


Figure C.2: Diagram with the components of the CROSS system.

[3]SureThing project website: `http://surething-project.eu`

[4]CROSS-server setup folder: `https://github.com/inesc-id/SureThingTournament/tree/main/tournament_materials/1-setup/3-CROSS_server_VM`

[5]CROSS-client mobile application setup folder: `https://github.com/inesc-id/SureThingTournament/tree/main/tournament_materials/1-setup/2-CROSS_client/1-CROSS_client_mobile_application`

C.5 Stages

Each folder describes a different **stage** of the tournament. The numbers preceding the names of the folders were added to state in which order each folder should be seen.

C.5.1 Rules

The rules[6] describe how the tournament will be played, as well as how the attacks will be scored. The way to submit results will work is also described there.

C.5.2 Setup

The setup folder[7] contains all the necessary steps needed to setup and deploy the network where the attacks are to be performed. Namely, it describes the steps in order to download, install and configure all the virtual machines to be used for the duration of the tournament. This folder is, in turn, divided into several subfolders, each for a different virtual machine detailing its configuration.

No real machines will be **harmed**, only virtual machines :)

C.5.3 Attack Rounds

The rounds folder[8] describes in detail how and what to perform during each different attack round. Inside this folder there is a subfolder for each round with all the necessary information about that round.

C.6 Help

If you need any help following the instructions, do not hesitate to contact us. Either send an email to *jose.alves.ferrao@tecnico.ulisboa.pt* or leave a message at the SureThing Project Discord server.

[6]Rules folder: `https://github.com/inesc-id/SureThingTournament/tree/main/tournament_materials/0-rules`

[7]Setup folder: `https://github.com/inesc-id/SureThingTournament/tree/main/tournament_materials/1-setup`

[8]Attack rounds folder: `https://github.com/inesc-id/SureThingTournament/tree/main/tournament_materials/2-rounds`