Attack SureThing! Offensive security assessment of a location certification system

José Miguel de Brito Alves Ferrão
Instituto Superior Técnico, Universidade de Lisboa, Portugal

***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 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.**

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

We also prepared all 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

I. 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*" [1].

With this work, we propose an offensive security assessment of SureThing [2] by assessing the security of the architecture and the implementation of CROSS (*loCation pROof techniqueS for consumer mobile applicationS*) [3], 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 techniques. For that we deployed a virtualized environment with specific requirements to make it as real as possible, meaning we had to simulate different networks and multiple machines. 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 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.

The remainder of this document is structured as follows: Section II gives a background overview of security concepts and commonly used attack tools, and works on *Location Certification Systems*; Section III describes CROSS - the target system of this work - and our offensive approach to its security assessment; Section IV presents the results and our evaluation of CROSS; and, finally, Section V concludes this article, and presents some future work directions.

II. Background and Related Work

A *threat* consists in any intention to cause and inflict damage to a system [1]. A threat can only be considered as a potential action when performed by a *threat-actor*, motivated and capable enough to do it. In this context, a threat-actor can either be a group of people or just a single individual. If successful, a threat has the potential of having a negative impact on the targeted system. An *attack path* relates to the steps a threat-actor has to go through to plan, prepare, and later execute an attack. An *attack vector* is then used to refer to 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 gain access to a system [4], i.e., the total number of attack vectors a threat-actor can use to gain access to a system.

A system is likely to have vulnerabilities: both in code and 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. If and when exploited, such vulnerabilities may result in the violation of the security assumptions of a system [5]. In essence, vulnerabilities have to be individually assessed. Furthermore, Tripathi and Singh [5] refer that after the identification of all existing vulnerabilities, each individual vulnerability needs to be evaluated based on its *risk level*[1]. That is why the correct use of security tools is important as they will help detect and even mitigate vulnerabilities.

A. Offensive Security

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 offensive security can help. Offensive security can be seen as a proactive strategy of protecting a system, with an emphasis in an adversarial approach.

Vulnerability assessment is the process of analysing a system to find, identify, and prioritize vulnerabilities in terms of their risk. Once identified these vulnerabilities can be mitigated, leading to the reduction of the attack surface of the system. Vulnerability assessment is not concerned with exploiting a vulnerability, even if found in the process [1].

In turn, penetration testing consists in the execution of an attack targeted at a specific system in order to identify and measure the risks associated with the possible exploitation of the attack surface. In other words, penetration testing adds to vulnerability assessment by performing exploitation [1]. 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.

One way to perform an offensive security is through the use of a *Red Team*. Usually independent of the organization whose system is being tested, a Red Team is a group of security professional tasked with testing a system in terms of its security. The operations taken by a Red Team are designated as *Red Teaming*. Red Teaming is the process of using TTPs[3] to emulate an actual threat. This process - known as engagement - 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. Red Teaming always assumes there is an active *Blue Team*. A Blue Team is usually composed by in-house members and is responsible to defend a system from outside threats.

B. 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 vulnerability exploitation.

[1]Metric based on the (negative) impact the exploitability of a vulnerability may have in a system.

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

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

1) Information Gathering Tools: Information Gathering Tools (or OSINT[4]) tools are used to gather some initial information about a target. From a vast list of other existing tools, we highlight the following: *theHarvester*[5], *Metagoofil*[6], *Automater*[7].

theHarvester focus on exposing information about a target 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 related with the target.

Metagoofil helps in finding publicly accessible documents of the targeted organization through a *Google*[8] 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.

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) Enumeration Tools: These tools are used to get to know the target system better. Following are some of those tools: *ident-user-enum*[9], *Nmap*[10], *arp-scan*[11], *AMAP*[12].

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*[13], 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.

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 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)[14].

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.

3) Vulnerability Analysis Tools: These tools are designed to assess computer systems, networks or applications for known weaknesses. Vulnerability analysis tools can help identify

[4]https://github.com/inesc-id/SecurityTaxonomy/blob/main/Definitions/OSINT.md

[5]https://github.com/laramies/theHarvester

[6]http://www.edge-security.com/metagoofil.php

[7]www.tekdefense.com/automater

[8]Search engine. Available at https://www.google.com

[9]http://pentestmonkey.net/tools/user-enumeration/ident-user-enum

[10]Network Mapper. Available at https://nmap.org

[11]https://github.com/royhills/arp-scan

[12]Application MAPper. Available at https://www.thc.org/

[13]https://tools.ietf.org/html/rfc1413

[14]https://tools.ietf.org/html/rfc826

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*[15] and *Google Tsunami*[16].

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.

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 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 [6].

We highlight some freely available fuzzers: *FFUF*[17], *WebSlayer*[18], and *Wfuzz*[19].

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.

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

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

[17]Fuzz Faster U Fool. Available at https://github.com/ffuf/ffuf

[18]The web application bruteforcer. Available at http://www.edge-security.com/webslayer.php

[19]The Web Fuzzer. Available at http://www.edge-security.com/wfuzz.php

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

C. Location Proof Systems

Many applications rely on the location of its users in order to provide them with services, but many times the information is not verified. Without proper verification, the applications are susceptible to *location spoofing attacks* [3]. Users of a system that know about this problem may use it for their own benefit. This is where location proof systems can help. Unlike only relying on the location of, e.g., the GPS signal or the geographical source of the IP address of the device of an end-user, these systems provide additional security to applications by issuing location certificates [3]. These systems use the definition of location proof made by Saroiu and Wolman [7] and implement verification procedures. Zhu and Cao [8] proposed APPLAUS and introduced users themselves as witnesses to verify claims. Many other systems follow a similar approach, such as VeriPlace [9], and, SureThing [2], to name a few. These systems, just like any others, rely on the isolation of processes and on the absence of vulnerabilities to make sure that the defined security policies are correctly enforced.

III. Attack Approach

We start by describing the target system of our attacks. Followed by a description of the deployed testbed and the considered network topologies. Next, we present our toolset. We end with a description of the methodology we followed in our attack approach.

A. Target System

The SureThing [2] framework defines a model for location certification, and provides libraries and services to develop systems that issue, verify and store location certificates. One of those systems is CROSS [3] (*loCation pROof techniqueS for consumer mobile applicationS*), a location proof system that supports a smart tourism application that rewards its users after they visit a set of predefined points of interest in a city.

1) Architecture: The architecture of CROSS is composed by four main components, represented in Figure 1. Those are server, client, *Wi-Fi Access Point* (used as part of the TOTP strategy, described in III-A2), and *kiosk* (used as part of the Kiosk strategy, described in III-A2). The client, represented in the lower-left side of Figure 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 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.

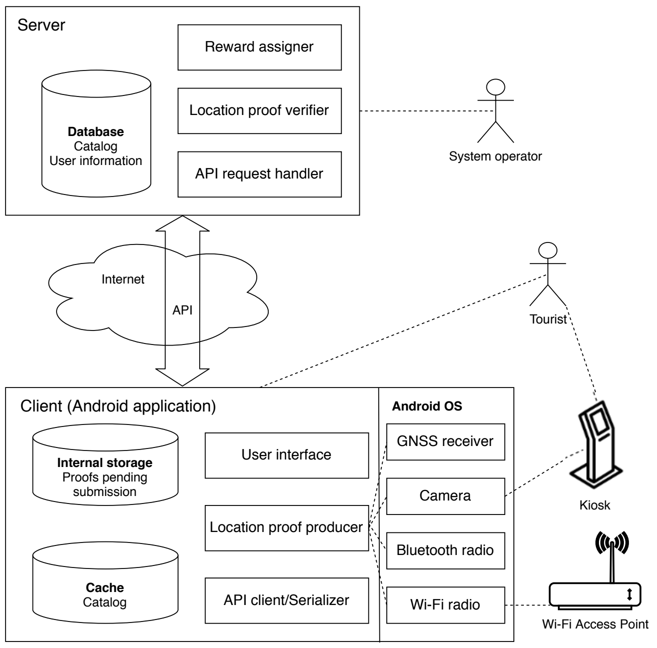
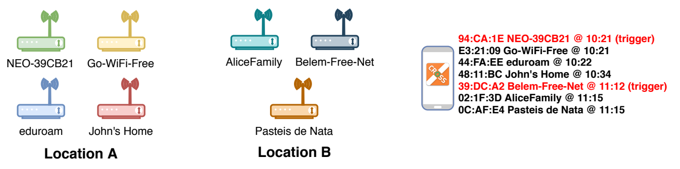


Fig. 1: Architecture of the CROSS system.

2) Location Certification Strategies: The CROSS system makes location proofs with one of three different strategies: *Scavenging*, *Time-based One-time Password* (TOTP), and *Kiosk*.

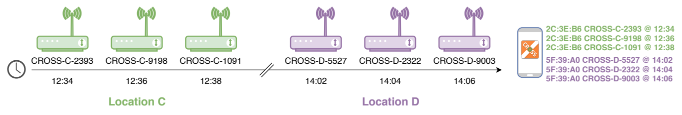
The scavenging strategy, represented in Figure 2, simply relies on the already existing Wi-Fi networks infrastructure in an urban area - either public or private - where each network is associated with a *timestamp*. Meaning a user just needs to be at a certain location and the client mobile application will scan the nearby networks.



```
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
```

Fig. 2: Representation of a typical usage of the scavenging strategy.

The TOTP strategy, represented in Figure 3, relies on special Access Points (APs) that broadcast one-time values as SSID (Service Set Identifier), the identifier a user sees for each Wi-Fi network as its name. Each value is determined by the current time and by a keyed hash function.



```
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
```

Fig. 3: Representation of a typical usage of the TOTP strategy.

This strategy prevails over the scavenging strategy if both strategies are in place at a given location.

Finally, the *kiosk* strategy relies on interactions of the user with a physical device placed at a specific location that, in practice, requires the user to be physically present. The kiosk acts as a trusted witness and is more effective than the previous strategies at preventing *Sybil attacks* [10], i.e., avoiding users with 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) Potential Attacks: An user can cheat in the scavenging strategy once the existing networks at a given location are known and do not change. The scavenging strategy is also subject to *network/Wi-Fi spoofing attacks* [11] because it relies on an unmanaged infrastructure, formed by the already existing Wi-Fi networks. This third-party infrastructure is not controlled and not authenticated, making it exploitable by what is also known as an *evil twin attack* [12].

The TOTP strategy offers a stronger security guarantee than the scavenging strategy. 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 still vulnerable to a user, now being an attacker, that relays values to another user.

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

B. Testbed

The deployed testbed consisted in several virtual machines, provisioned by VirtualBox[20].

The core of the testbed is composed by the CROSS server and client (Android mobile application), the attacker machine, and additional virtual machines to simulate the rest of the network. The CROSS server runs 64-bit Ubuntu Linux 20.04. The CROSS client runs Android 5.1, emulated using Genymotion. The attacker machine is a 64-bit Kali Linux version 2020.4. Following what a real attacker would do, we allowed the machine of the attacker to run in promiscuous mode[21]. The routers run pfSense CE 2.5.0. We deployed another Ubuntu 20.04 machine to hold the CROSS client mobile application source code. Thus, the code of the Android application can be provisioned through Android Studio to the virtual machine running the emulator of the CROSS client when needed.

We deployed two different network topologies, but the core of the testbed, as described in III-B, remained the same. The main difference is where the virtual machine of the attacker is positioned in the network.

We opted to use the *internal network* mode composed only of virtual machines. One of the network adapters of the "Internet" router uses *bridged network* mode, allowing all virtual machines to have Internet access Only the virtual machine with the code of the CROSS client mobile application and the Genymotion emulator are connected to each other

[20]Virtualization *hypervisor*. Available at: https://www.virtualbox.org/

[21]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.

through the *host-only* network mode of VirtualBox. This host-only network mode allows virtual machines to communicate with other virtual machines, as well as with the host system. Allowing these two virtual machines to be connected without the need to create and have a separate (internal) network. The virtual machine with the code of the CROSS client then uses the *Network Address Translation (NAT)* mode in order to have Internet access, e.g., to perform software updates.

1) Attacker Inside the Network: One topology, represented in Figure 4, places the *attacker inside the network*. It is either part of the network of the CROSS client or has already managed to gain access to it. In the figure, "S" stands for server and represents the router of the subnetwork where the CROSS server is placed. Also in the figure, "I" stands for ISP (Internet Service Provider) of the CROSS client.

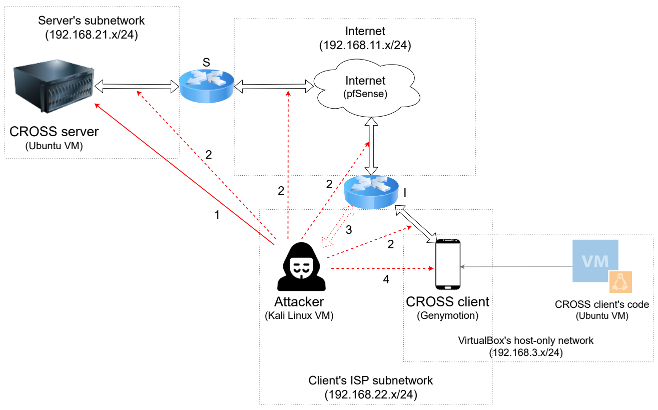


Fig. 4: Topology of the network when the attacker is inside the network of the client. The solid red arrows represent the attacks that were performed, and the dashed red arrows represent the attacks that were considered but not performed.

In terms of each subnetwork shown in Figure 4. The "Internet" subnetwork works in the 192.168.11.0/24 IP addresses range. More specifically, the "Internet" router, the router of the subnetwork of the CROSS server, and the router of the subnetwork of the CROSS client are statically assigned with the 192.168.11.1, 192.168.11.11 and the 192.168.11.12 IP addresses, respectively. The IP addresses of the remaining virtual machines are assigned through DHCP[22]. The subnetwork where the CROSS server is in runs in the 192.168.21.11/24 - 192.168.21.255/24 range. With the subnetwork of the ISP of the CROSS client is working in the 192.168.22.11/24 - 192.168.22.255/24 range.

2) Attacker Outside the Network: In the other topology, the *attacker is outside the network*, i.e., it has no access to the network where the CROSS client is placed, as shown in Figure 5. In the figure, "A" stands for attacker and represents the ISP of the attacker. Again, "S" stands for server and is used to represent the router that belongs to the subnetwork of the CROSS server. With "I" standing for the ISP of the CROSS client.

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

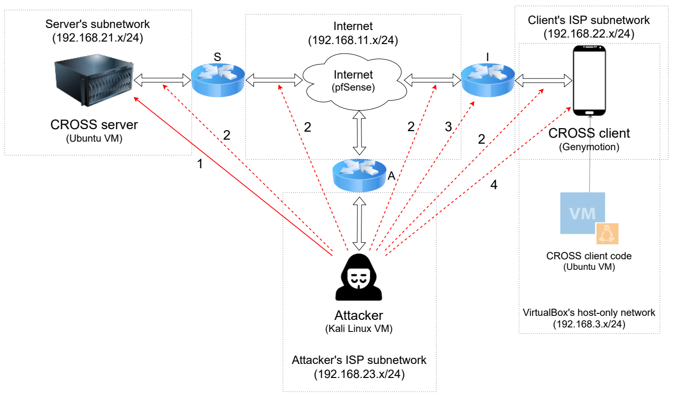


Fig. 5: Topology of the network when the attacker is outside the network of the client. The solid red arrows represent the attacks that were performed, and the dashed red arrows represent the attacks that were considered but not performed.

Concerning each subnetwork of Figure 5 and corresponding network IP addresses ranges. The "Internet" subnetwork works in the range of the 192.168.11.0/24 IP addresses. Namely, the "Internet" router is statically assigned the 192.168.11.1 IP address. The routers of the subnetworks of the CROSS server and of the CROSS client are statically assigned the 192.168.11.11 and the 192.168.11.12 IP addresses, respectively. Finally, the router of the subnetwork where the machine of the attacker is is statically assigned the 192.168.11.13 IP address. While the IP addresses of the rest of the virtual machines were assigned through DHCP. 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 works in the 192.168.22.11/24 - 192.168.22.255/24 range. Finally, the subnetwork where the attacker is in uses the 192.168.23.11/24 - 192.168.23.255/24 IP addresses range.

It is worth mentioning there was a need to add an "any-to-any rule" to the routers to allow the required communication between the virtual machines.

C. Attack Tools

Here we describe our toolset, that included: enumeration, vulnerability assessment, and fuzzing tools.

1) Enumeration Tools: 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.

2) Vulnerability Assessment Tools: For vulnerability analysis, we used *OpenVAS* and *Google Tsunami*. We opted to use OpenVAS because of its popularity and connection with large vulnerabilities databases. Google Tsunami was used as a network security scanner. While quite new, Google Tsunami is part of a Google which was the main reason for choosing it.

3) Fuzzing Tools: We used fuzzing techniques to verify if the CROSS server (III-A) would stop its normal operation to the point where it would no longer respond to the requests of the deployed client (III-B). 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*[23] 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.

D. 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 that entered in a terminal/console, known as terminal logs. With that in mind, we opted to use the Linux *script* command[24] 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 III-A) 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, respectively described in III-B1 and in III-B2. We followed an incremental approach, by introducing a new tool (either offensive or not) 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 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 II-B2 and OpenVAS (II-B3), in this order. In the second iteration we used Google Tsunami (II-B3). 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 (II-B4) fuzzing tool.

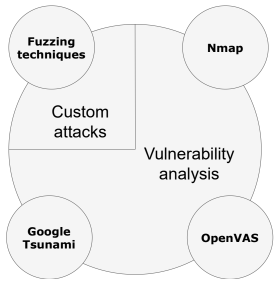


Fig. 6: Different iterations of the performed attacks.

IV. Evaluation

As we described in III-D, we performed the attacks from the two vantage network positions we assumed an attacker can be. That is with the attacker inside the network of the client or outside of it, respectively described in III-B1 and in III-B2. The following presented results correspond to only one of the cases, as the gathered results were the same in both cases as we were expecting.

We use and assess each attack tool listed in Section III-C individually. The output of several tools can be combined in the future to perform more advanced attacks.

We started by finding the IP address of the CROSS server, even though we knew the IP of all deployed virtual machines beforehand. Next, we present the steps taken to find it:

1) Initialization of Wireshark[25];
2) Start capture of packets on the *eth0* interface;
3) Use the CROSS client application to request the rewards page (`/v1/users/@me/rewards`);
4) Infer that the IP address of the server is 192.168.21.14, by inspecting the exchanged packets shown in Figure 7;
5) Conclude that the CROSS server service is running at TCP port 13000, by inspecting the packets exchanged between the CROSS client application and the CROSS server;
6) Infer all available paths are sub-directories of the root `/v1` path.

A. Nmap Results

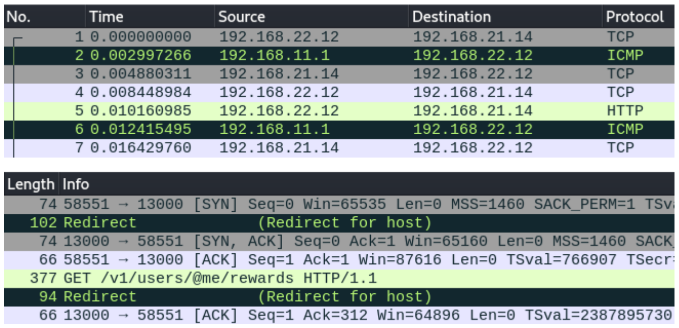
We used Nmap to perform several ports scans of the target system with the 192.168.21.14 IP address. Starting with the simple default port scan, we used the option to probe any open ports to determine the service and version running at each port. By default Nmap only scans the most commonly used 1000 TCP ports. As a result, this initial port scan did not detect any open ports; we only gathered the information that the target system is at a distance of 4 hops from the attacker machine.

We next moved on to perform a broader port scan including all TCP and UDP ports, again probing any open ports for

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

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

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



```
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
```

Fig. 7: Wireshark packet captures done by the attacker machine that shows some of the exchanged packets between the CROSS client mobile application and the CROSS server after a user requests its rewards. The 'length info' rows correspond to rows 1-7 of the captures.

their service. This time Nmap was able to detect that TCP port 13000 was open, as we would expect. Yet Nmap wrongly detected its service, detecting it as a DAAP service[26]. 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 information. UDP ports 631, 5353 and 53574 were also detected but as *open|filtered*, meaning Nmap was not able to determine if these ports were actually open or simply filtered since it did not get a response from them. A port that Nmap reports as *open|filtered* may also mean that the probe used by Nmap was dropped by a packet filter, which was not the case, given the testbed configuration (described in Section III-B) and deployed network topologies. UDP port 631 was detected by Nmap and is usually used for running the IPP[27] service. In the used Linux machine, the service running at port 631 was CUPS[28]. UDP port 5353 was detected by Nmap and is registered as the port to run the mDNS[29] protocol. The third UDP port Nmap detected as *open|filtered* - 53574 - has no common service associated with it. Perhaps for that same reason, Nmap was not able to detect the service listening at that port.

Overall, Nmap correctly detected the operating system of the virtual machine where the CROSS server is running. Yet, it did not detect its Linux kernel version nor the exact Linux distribution (Ubuntu). These results could be related with the fact that the target system is running inside a virtual machine.

Furthermore, Nmap was also able to detect the PostgreSQL database used by the CROSS server, listening at port 5432, but only after we explicitly configured PostgreSQL to accept outside connections.

We decided to scan port 13000 also with AMAP, since Nmap was unable to correctly determine the service running there. After making sure AMAP also reported TCP port 13000 was open, we tried to map the service running at that port by sending some triggers and analysing their responses, which did produce any relevant information. AMAP was also not able to gather useful data about the specific service running at TCP port 13000.

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

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

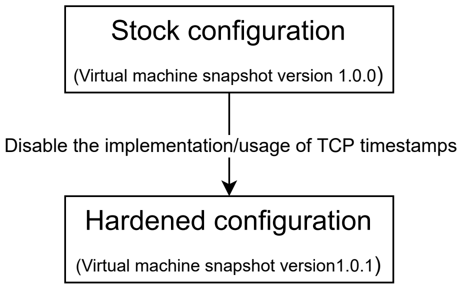
[28]The Common Unix Printing System allows a Unix-based computer to operate as a printing server.

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

B. OpenVAS Results

OpenVAS was used to perform various scans. These included scans to all ports of both TCP and UDP, all ports of one of the protocols (TCP or UDP), or only targeted at the TCP port 13000. The only vulnerability OpenVAS was able to detect was the implementation of TCP timestamps[30] active in the CROSS server virtual machine. This vulnerability is scored 2.6 in the base group of CVSS[31] and is therefore considered a low severity vulnerability.

1) Changes: Based on the results gathered after running OpenVAS, we moved on to make the suggested changes in order to harden the target system. The solution involved editing the */etc/sysctl.conf* file, which required root privileges, to add the line *net.ipv4.tcp_timestamps = 0* and then execute the command *sysctl -p* to apply the change to Linux. The equivalent procedure would be different for other operating systems. Figure 8 represents the change on the configuration of the target system.



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

Fig. 8: Changes made to the configuration of the CROSS server to harden it according to the changes suggested by OpenVAS.

2) Avoided Attacks: The TCP timestamps feature can be used by an attacker to compute the *uptime* of a system, i.e., compute for how long that system has been booted. An attacker may then be able to infer if a system is still running a previous version of its operating system or installed software, as some updates or patches require a system reboot. By changing this setting, these attacks are made more difficult.

C. Tsunami Results

By default, Tsunami only uses Nmap to scan the most common 1000 TCP ports. Therefore the port where the CROSS server is listening at (13000) was not detected. Tsunami did

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

[31]Scoring system used to describe and rate IT vulnerabilities. More information available at: https://www.first.org/cvss/.

not proceed to the next step of vulnerability verification. We decided to change the configuration Tsunami uses by default, as we knew TCP port 13000 was open. The port was promptly detected once we overrode the default values that specify which ports Nmap should scan. This allowed the process employed by Tsunami to continue since it detected an open port. Namely, it moved to use fingerprinting techniques to try to identify which service is running on the port. As expected because of the use of Nmap, it was again detected as a DAAP service. Finally, Tsunami used its default vulnerability detection plugins targeted at the TCP port 13000, but they did not find vulnerabilities.

D. Fuzzing Results

FFUF was used with two collections of wordlists: *SecLists*[32] and *RobotsDisallowed*[33]. SecLists contains a collection of multiple wordlists: from usernames and passwords to data pattern matching. RobotsDisallowed holds a list of the most commonly disallowed directories present in *robots.txt* from top websites.

Directory discovery was done assuming the existence of the `/v1` path, considering previously gathered information described in the beginning of this section. We used two different wordlists, with the second being larger than the first one which allowed us to get further results. We were able to find the following seven directories under the `/v1` path:

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

Using a wordlist from the RobotsDisallowed list we discovered a sub-directory of the `/trips` path - `/trips/upcoming` - that also returned a 401 HTTP code.

We decided to send some POST requests as both `/users` and `/pairs` paths and the server returned a 405 (Method Not Allowed) HTTP code to the requests. We used different wordlists, each containing a list of values meant to disrupt the normal function of a system. Both paths accepted the *null* value, returning a 200 (OK) as the response status code. In the following requests containing the *null* value we got a 409 (Conflict) HTTP code as response. Namely, the `/users` path responded with the following: *"message": Username in use*. This meant that we were trying to insert it in the same table of the database. The remaining values either failed or could not be decoded, all returning 400 (Bad Request) status codes. None of these values managed to stop the normal operation of the CROSS server. Yet, we managed to write the *null* value to the table where the existing users and corresponding usernames are kept using the `/users` path. As for the `/pairs` path, the *null* value was written to a different table of the database.

We tried to do database enumeration by sending some POST requests, using two different wordlists. Yet, we were not able to do any evident damage. Nor were we able to get any further information through database enumeration. Actually, all responses returned a 400 (Bad Request) HTTP code followed by the *"invalid character"* response.

We also tried to do file discovery targeted at the `/users` and `/meta` paths. Together with a wordlist designed to perform directory discovery, we used two other wordlists containing different known file extensions. One of these wordlists contained the most commonly used file extensions, while the other contained common web file extensions.

We also tried to do a directory discovery for any sub-directories of the `/users` and `/meta` paths but we were not able to find any. Considering these results, we suspected the results would be the same for the remaining discovered paths. Therefore we did not try to continue with the directory discovery for any of the other 5 previously discovered paths.

Performing these fuzzing techniques produced quite a large number of requests, almost creating a Denial of Service (DoS) attack. We could attest this by using the CROSS client application at the same time and seeing the CROSS server indeed took much more time to respond to the requests.

V. Conclusion

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.

A. 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.

[32]https://github.com/danielmiessler/SecLists
[33]https://github.com/danielmiessler/RobotsDisallowed

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.

B. 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 *https://github.com/inesc-id/SureThingTournament/tree/main/tournament_materials*.

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).

References

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

[2] J. Ferreira and M. L. Pardal, "Witness-based location proofs for mobile devices," in *17th IEEE International Symposium on Network Computing and Applications (NCA)*, Nov. 2018.

[3] M. L. P. Gabriel A. Maia, Rui L. Claro, "CROSS City: Wi-Fi Location Proofs for Smart Tourism," 2020.

[4] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 371–386, 2010.

[5] A. Tripathi and U. K. Singh, "Towards standardization of vulnerability taxonomy," in *2010 2nd International Conference on Computer Technology and Development*. IEEE, 2010, pp. 379–384.

[6] P. Oehlert, "Violating assumptions with fuzzing," *IEEE Security & Privacy*, vol. 3, no. 2, pp. 58–62, 2005.

[7] S. Saroiu and A. Wolman, "Enabling new mobile applications with location proofs," in *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, 2009, pp. 1–6.

[8] Z. Zhu and G. Cao, "Applaus: A privacy-preserving location proof updating system for location-based services," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 1889–1897.

[9] W. Luo and U. Hengartner, "Veriplace: a privacy-aware location proof architecture," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2010, pp. 23–32.

[10] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[11] L. Tamilselvan and D. V. Sankaranarayanan, "Prevention of impersonation attack in wireless mobile ad hoc networks," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 7, no. 3, pp. 118–123, 2007.

[12] D. Mónica and C. Ribeiro, "Wifihop-mitigating the evil twin attack through multi-hop detection," in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 21–39.