Offensive security assessment of a REST API for a location proof system

José Ferrão[0000−0002−6405−9453], Samih Eisa[0000−0003−0972−4171], and Miguel L. Pardal[0000−0003−2872−7300]

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
{jose.alves.ferrao,miguel.pardal}@tecnico.ulisboa.pt
samih.eisa@inesc-id.pt

Abstract. Despite the best efforts of designers, a system cannot be said to be truly secure and robust until it has experienced - and thwarted - attacks from skilled and motivated attackers. With that in mind, we performed an offensive security assessment of CROSS, a smart tourism application that uses location proofs. The server is exposed on the public Internet and offers a REST-based API. We performed a vulnerability assessment and penetration testing on the server, using generic attack tools, and from different vantage points in the network, always in the attacker perspective. We present the tools and techniques used to attack the REST API, a detailed presentation of the findings, and the procedures for hardening the server. The security assessment used five different tools and we were able to find a previously unknown vulnerability that allowed unauthorized writes to the database of the target system.

Keywords: Vulnerability Assessment · Penetration Testing · Offensive Security · Location Proof Systems · REST API Security.

1 Introduction

The Internet of Things (IoT) can be described as a very large network where many and diverse devices interact with each other [4]. IoT devices inherit the security-related issues common to computer networks but add challenges because they are limited by their small memory, lack of processing power [4], and battery capabilities. To deal with security challenges, some organizations subject their systems to *vulnerability assessment* and *penetration testing* to detect flaws, having in mind the perspective of an attacker [13]. In this paper we evaluate the security of an exposed REST API for the CROSS smart tourism application [6] that uses location proofs developed in the SureThing [3] project[1]. We used different tools, distributed over three iterations of attacks. The attack iterations ranged from simple vulnerability analysis tools to fuzzing techniques. We use and assess each tool individually. In the end we improved and hardened the existing deployment. Even though this work is specific to a particular system, we provide some insight on how to perform offensive security testing.

[1] http://surething-project.eu.

The remainder of this document is structured as follows: Section 2 gives an overview of concepts and works on location proof systems; Section 3 describes the target system of our work; Section 4 describes our approach; Section 5 presents the results and, finally, Section 6 concludes this article.

2 Background

A *threat* consists in any intention to cause and inflict damage to a system [13]. A threat can only be considered as a potential action when performed by a *threat-actor*, motivated and capable enough to do it. An *attack path* relates to the steps a threat-actor has to go through to plan, prepare, and later execute an attack. In turn, the *attack surface* corresponds to the different ways through which a threat-actor is able to gain access to a system [7].

A system is likely to have vulnerabilities: both in code and in the infrastructure. If and when exploited, such vulnerabilities may result in the violation of the security assumptions of a system [12]. In essence, vulnerabilities have to be individually assessed. That is why the correct use of security tools is important as they will help detect and even mitigate vulnerabilities.

2.1 Offensive Security

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

2.2 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* [5]. Users of a system that know about this problem may use it for their own benefit. This is where location proof systems can help. These systems use the definition of location proof made by Saroiu and Wolman [10] and implement verification procedures. Zhu and Cao [14] proposed APPLAUS and introduced users themselves as witnesses to verify claims. Many other systems follow a similar approach, such as CREPUSCOLO [1], SureThing [3], and PASPORT [9], 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.

3 Target System

The selected target system is CROSS [6], a smart tourism application that rewards its users after they visit a set of predefined points of interest in a city, and uses the SureThing [3] framework for location proofs. A CROSS deployment is represented in Figure 1 and is composed by a server exposing a REST API to a mobile client application.

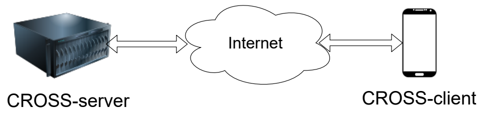


Fig. 1. CROSS system components.

The CROSS system makes location proofs with one of three different strategies: *scavenging*, *time-based one-time password* (TOTP), and *kiosk*. The scavenging strategy simply relies on the already existing Wi-Fi networks infrastructure in an urban area. The TOTP strategy 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. 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* [2], i.e., avoiding users with multiple accounts.

Regarding more potential attacks, a 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* [8]. 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.

Providing its users with a REST API, CROSS is therefore exposed to the outside, being a vulnerable entry point for an attacker to exploit.

4 Attack Approach

We start by describing the deployed testbed and the considered network topologies. We end with a description of the toolset and the methodology.

4.1 Testbed

The deployed testbed consists in several virtual machines. The core of the testbed is made of the CROSS server and client, the attacker machine, and additional virtual machines to simulate the rest of the network. With the exception of the client, all the virtual machines are provisioned with VirtualBox. 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. The routers run pfSense. We opted to use the *internal network* mode composed only of virtual machines. One of the network adapters of the router uses *bridged network* mode, allowing all virtual machines to have Internet access.

4.2 Network Topologies

We deployed two different network topologies, but the core of the testbed, as described in 4.1, remained the same. The main difference is where the virtual machine of the attacker is positioned in the network.

One topology, represented in Figure 2, 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.

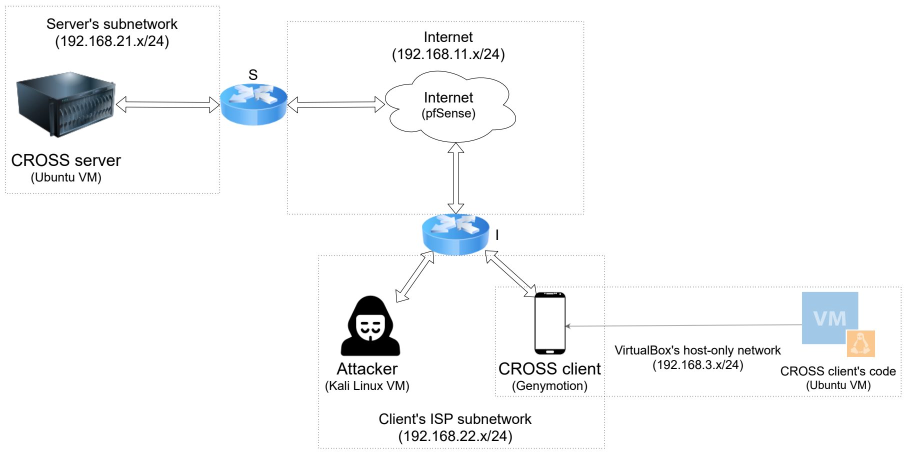


Fig. 2. Topology of the network when the attacker is inside the network of the client.

The "Internet" router, the router of the subnetwork of the CROSS server, and the router of the subnetwork of the CROSS client have statically assigned IP addresses. The IP addresses of the remaining virtual machines are assigned through DHCP[2].

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 3. In the figure, "A" stands for attacker and represents the ISP of the attacker.

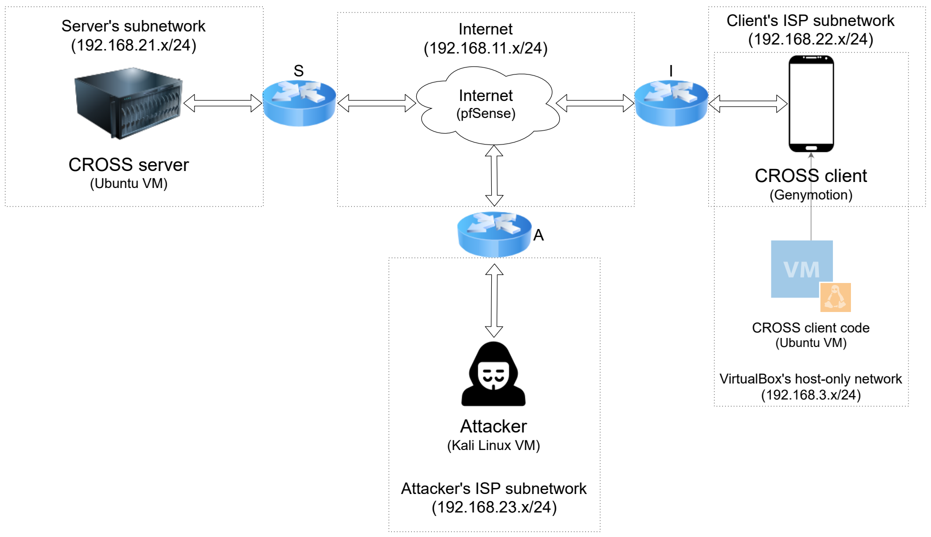


Fig. 3. Topology of the network when the attacker is outside the network of the client.

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.

4.3 Attack Tools

Here we describe the toolset, including: enumeration, vulnerability assessment, and fuzzing tools.

Enumeration Tools *Nmap*[3] is a tool that can determine what hosts are available in a network. It can also find which services and operating system a host is running.

AMAP[4] is another scanning tool that tries to identify applications running on a given host, even if running on a different port than usual.

[2] The Dynamic Host Configuration Protocol can automatically assign and distribute IP addresses within a network.
[3] Network Mapper. Available at `https://nmap.org`.
[4] Application MAPper. Available at `https://hackerschoice.github.io/`.

Vulnerability Assessment Tools *OpenVAS*[5] is a 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 that system.

Google Tsunami[6] is a general purpose network security scanner, offering an extensible engine through the use of plugins. It uses Nmap as a port scanner during the first step, and then it uses some fingerprinting techniques to try to identify the services running at each of the scanned open ports.

Fuzzing Tools We used fuzzing tools to verify if the CROSS server would stop its normal operation, to the point where it would no longer respond to the requests of clients. We had the constraint that we needed a fuzzer that could be used through the network. After comparing some available fuzzers we decided to use *FFUF*[7].

4.4 Attack Iterations

We performed the attacks with the machine of the attacker placed at both of the two defined network positions: inside or outside, as described in 4.2. Having full access to all deployed virtual machines from any virtual machine 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. Yet, our objective was to know if and which vulnerabilities were present in the target system.

We introduced a new tool at each new iteration, following an incremental approach. The results gathered from previous iterations were used as the basis for the next ones, and provided information for later hardening the target system. As part of the vulnerability analysis stage, we did two separate iterations. The first one involved using Nmap and OpenVAS. In the second we used Tsunami as a network security scanner. As a third attack iteration, we introduced some customs attacks in the form of fuzzing techniques. Figure 4 gives an overview of the attack iterations.

5 Results

We use and assess each attack tool listed in Section 4.3 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[8];

[5] Open Vulnerability Assessment System. Available at `https://openvas.org/`.
[6] Tsunami. Available at `https://github.com/google/tsunami-security-scanner`.
[7] Fuzz Faster U Fool. Available at `https://github.com/ffuf/ffuf`.
[8] Wireshark is a free and open-source network packet analyzer. Available at `https://www.wireshark.org/`.

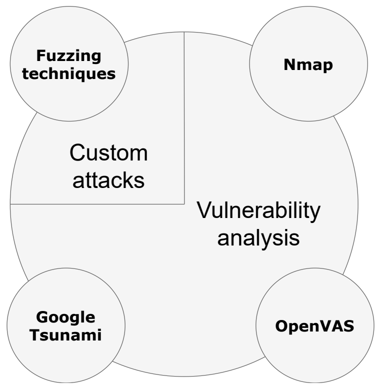


Fig. 4. Different iterations of the performed attacks.

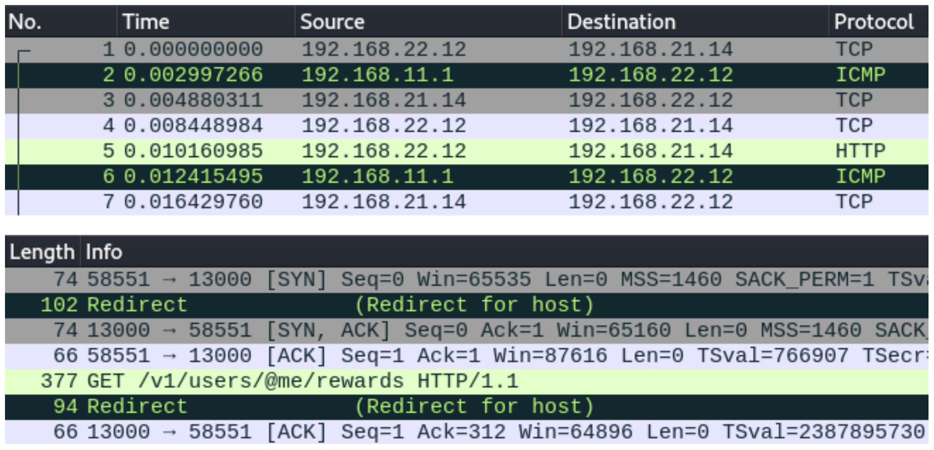
2. Start capture of packets on the *eth0* interface;
3. Use the CROSS client application to request the rewards page (`/v1/users/@me/rewards`);
4. Infer that the IP address of the server is 192.168.21.14, by inspecting the exchanged packets shown in Figure 5;
5. Conclude that the CROSS server service 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.

5.1 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 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[9]. 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

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



```
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. 5. 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.

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 4.1) and deployed network topologies (Section 4.2). UDP port 631 was detected by Nmap and is usually used for running the IPP[10] service. In the used Linux machine, the service running at port 631 was CUPS[11]. UDP port 5353 was detected by Nmap and is registered as the port to run the mDNS[12] 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

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

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

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

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.

5.2 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[13] active in the CROSS server virtual machine. This vulnerability is scored 2.6 in the base group of CVSS[14] and is therefore considered a low severity vulnerability.

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 6 represents the change on the configuration of the target system.

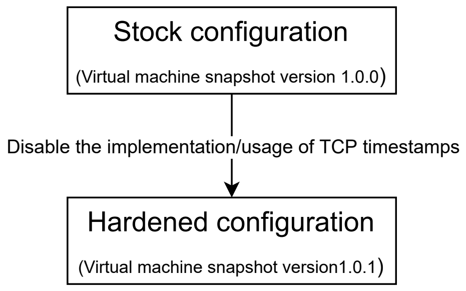

Stock configuration
(Virtual machine snapshot version 1.0.0)
Disable the implementation/usage of TCP timestamps
Hardened configuration
(Virtual machine snapshot version1.0.1)

Fig. 6. Changes made to the configuration of the CROSS server to harden it according to the changes suggested by OpenVAS.

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.

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

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

5.3 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 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.

5.4 Fuzzing Results

FFUF was used with two collections of wordlists: *SecLists*[15] and *RobotsDisallowed*[16]. 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 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

[15] `https://github.com/danielmiessler/SecLists`
[16] `https://github.com/danielmiessler/RobotsDisallowed`

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.

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.

6 Conclusion

In this paper we focused on the security of a specific location proof system but our approach can be adapted to other systems. We showed a vulnerability assessment and penetration testing techniques from an attacker's perspective. The security assessment was done with five tools, namely Nmap, AMAP, OpenVAS, Tsunami, and FFUF. Nmap and AMAP ran port scans but did not find any open ports that could be used by an attacker as an entry point. OpenVAS only detected a low severity vulnerability. FFUF used a specific fuzzing value and was able to make unauthorized writes in the database, even though it was not able to do much damage. The obtained results were used to harden the target system and reduce the attack surface.

Future work will focus on adding more tools to the used toolset, namely exploitation tools. Apart from that, we will organize a competitive tournament where multiple teams will compete against each other to perform attacks. The results from these tournaments are expected to further enrich the offensive security assessment.

Acknowledgements

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. Canlar, E.S., Conti, M., Crispo, B., Di Pietro, R.: Crepuscolo: A collusion resistant privacy preserving location verification system. In: 2013 International Conference on Risks and Security of Internet and Systems (CRiSIS). pp. 1–9. IEEE (2013)
2. Douceur, J.R.: The sybil attack. In: International workshop on peer-to-peer systems. pp. 251–260. Springer (2002)
3. Ferreira, J., Pardal, M.L.: Witness-based location proofs for mobile devices. In: 17th IEEE International Symposium on Network Computing and Applications (NCA) (Nov 2018)
4. Khan, M.A., Salah, K.: Iot security: Review, blockchain solutions, and open challenges. Future Generation Computer Systems **82**, 395–411 (2018)
5. Lee, J.H., Buehrer, R.M.: Characterization and detection of location spoofing attacks. Journal of Communications and Networks **14**(4), 396–409 (2012)
6. Maia, G.A., Claro, R.L., , Pardal, M.L.: CROSS City: Wi-Fi Location Proofs for Smart Tourism. In: The 19th International Conference on Ad Hoc Networks and Wireless (AdHoc-Now). Bari, Italy (Oct 2020)
7. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Transactions on Software Engineering **37**(3), 371–386 (2010)
8. Mónica, D., Ribeiro, C.: Wifihop-mitigating the evil twin attack through multi-hop detection. In: European Symposium on Research in Computer Security. pp. 21–39. Springer (2011)
9. Nosouhi, M.R., Sood, K., Yu, S., Grobler, M., Zhang, J.: Pasport: A secure and private location proof generation and verification framework. IEEE Transactions on Computational Social Systems **7**(2), 293–307 (April 2020). https://doi.org/10.1109/TCSS.2019.2960534
10. Saroiu, S., Wolman, A.: Enabling new mobile applications with location proofs. In: Proceedings of the 10th workshop on Mobile Computing Systems and Applications. pp. 1–6 (2009)
11. Tamilselvan, L., Sankaranarayanan, D.V.: Prevention of impersonation attack in wireless mobile ad hoc networks. International Journal of Computer Science and Network Security (IJCSNS) **7**(3), 118–123 (2007)
12. Tripathi, A., Singh, U.K.: Towards standardization of vulnerability taxonomy. In: 2010 2nd International Conference on Computer Technology and Development. pp. 379–384. IEEE (2010)
13. Tubberville, J., Vest, J.: Red Team Development and Operations: A Practical Guide. Independently Published (2020)
14. Zhu, Z., Cao, G.: Applaus: A privacy-preserving location proof updating system for location-based services. In: 2011 Proceedings IEEE INFOCOM. pp. 1889–1897. IEEE (2011)