UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

A reconnaissance based automatic tool for origin server discovery through misconfigured web servers

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Veciana Belmonte, Alex

Tutor: Escobar Román, Santiago

ACADEMIC YEAR: 2021/2022

Resum

Les redes de entrega de contingut (CDN) i, en particular, aquelles que integren un firewall d'aplicacions web (WAF) en ellas han experimentat un ràpid augment en el seu ús en els últims anys. Consisteixen en servidors altament distribuïts que ofereixen una entrega ràpida i confiable de contingut d'Internet. Dades els seus avantatges, al voltant del 62,3% dels 1000 principals llocs web d'escriptori utilitzen algun tipus de servei CDN per a lliurar contingut web. En aquest treball, demostrarem com es poden eludir les CDN accedint directament als llocs web a través del seu servidor d'origen, i les seves perilloses implicacions de seguretat. A més, discutirem els mètodes utilitzats per ajudar a descobrir el servidor d'origen de servidors web incorrectament configurats, aprofitant diversos mètodes de reconeixement. Recollirem alguns d'aquests mètodes i els integrarem en una eina automatitzada que desenvoluparem. A més, estudiarem com els servidors web poden configurar-se correctament per evitar exposer involuntàriament el seu servidor d'origen.

Paraules clau: CDN, WAF, descobriment de hosts, enumeració, reconeixement, hacking ètic

Resumen

Las redes de entrega de contenido (CDN) y, en particular, aquellas que integran un firewall de aplicaciones web (WAF) en ellas experimentaron un rápido aumento en su uso en los últimos años. Consisten en servidores altamente distribuidos que ofrecen una entrega rápida y confiable de contenido de Internet. Dadas estas ventajas, alrededor del 62,3 % de los 1000 principales sitios web de escritorio utilizan algún tipo de servicio CDN para entregar contenido web. En este trabajo, demostraremos cómo se pueden eludir las CDN accediendo directamente a los sitios web a través de su servidor de origen, y cuáles son sus peligrosas implicaciones de seguridad. Además, discutiremos los métodos utilizados para ayudar a descubrir el servidor de origen de servidores web incorrectamente configurados, aprovechando varios métodos de reconocimiento. Recopilaremos algunos de estos métodos y los integraremos en una herramienta automatizada que desarrollaremos. Además, estudiaremos cómo dichos servidores web pueden configurarse correctamente para evitar exponer involuntariamente su servidor de origen.

Palabras clave: CDN, WAF, descubrimiento de hosts, enumeración, reconocimiento, hácking ético

Abstract

Content Delivery Networks (CDNs) and particularly those which integrate a Web Application Firewall (WAF) on them have experienced a rapid increase in usage over the past years. They consist of highly distributed servers which provide fast and reliable delivery of web content. Given these advantages, around 62.3% of top 1000 desktop websites use some sort of CDN service to deliver content. In this work we will demonstrate how CDNs can be bypassed by accessing websites through their origin server directly, and which are its threatful security implications. Furthermore, we will discuss ways to help perform origin server discovery on misconfigured web servers by leveraging several reconnaissance methods. We will collect some of these methods and integrate them on an automated tool we will develop. Moreover, we will study how such web servers can be configured correctly to avoid unintendedly exposing its origin server.

Key words: CDN, WAF, host discovery, enumeration, reconnaissance, ethical hacking

Contents

List of Figures

List of Tables

Acronyms

AMASS Attack Surface Mapping and Asset Discovery. 20

CDN Content Delivery Network. 1

CIDR Classless Inter-Domain Routing. 29

CMS Content Manager System. 13

DNS Domain Name System. 13

DOS Denial Of Service. 39

HTML Hypertext Markup Language. 1

IDE Integrated Development Environment. 32

JS JavaScript. 13

OSINT Open Source Intelligence. 5

OWASP Open Web Application Security Project. 20

SQL Structured Query Language. 39

SSL Secure Sockets Layer. 13

WAF Web Application Firewall. iii

CHAPTER 1
Introduction

Reverse proxy utilization in website hosting has risen dramatically over the last years. They work by caching specific content from the origin's web server, replicating it and offering it to an end-user, thereby reducing the load on the web server and providing a higher level of availability of the website. Some reverse proxies consist of a Web Application Firewall (WAF), which implements different protection rules and attempts to block potential attackers from crafting specific payloads, accessing unauthorized end-points, etc.

Despite a host being vulnerable, pentesters and security researchers may struggle finding vulnerabilities due to the WAF. This raises the interest of obtaining direct access to the origin host, which in many cases allows for a broader range of vulnerabilities to be discovered. In this thesis we will discuss the creation of an automated tool which aims to help security researchers to discover an origin server behind a reverse proxy, leveraging several reconnaissance methods.

1.1 Motivation

A Content Delivery Network (CDN) consists of "geographically distributed servers to cache and efficiently deliver Internet contents, such as Hypertext Markup Language (HTML) pages, images, and videos" [5]. As the popularity and usage of the Internet has been steadily rising over the years, so has been the usage of content delivery networks. A 62.3% and a 61.1% of top 1,000 desktop and mobile websites respectively, are using a CDN [2]. Around four million websites are estimated to use CDNs. They have progressed and offer additional services, such as integrating cloud-host web application firewalls, bot management solutions, image and computing solutions and more.

As we can observe from Figure 1.1, there seems to be a direct correlation between the popularity of a website and the percentage of requests which are being managed by a CDN. One of the main reasons for this is that CDNs offer high content availability while simultaneously mitigating potential attacks, which is an attractive selling point for them. Therefore, websites tend to trust the CDNs they are using to handle attack mitigation, due to this perception of increased security. It sounds like an ideal solution, but is it?

Since content delivery networks function as a reverse proxy, they still need to have an origin server from which the content delivery network will obtain content from and cache it accordingly. Websites tend to have a significantly weaker security on their origin server, under the assumption that the CDN will manage and mitigate potential attacks. Thus, the origin server offers relevant information to pentesters and security researchers. With it, they can perform enumeration and reconnaissance methods to gather details

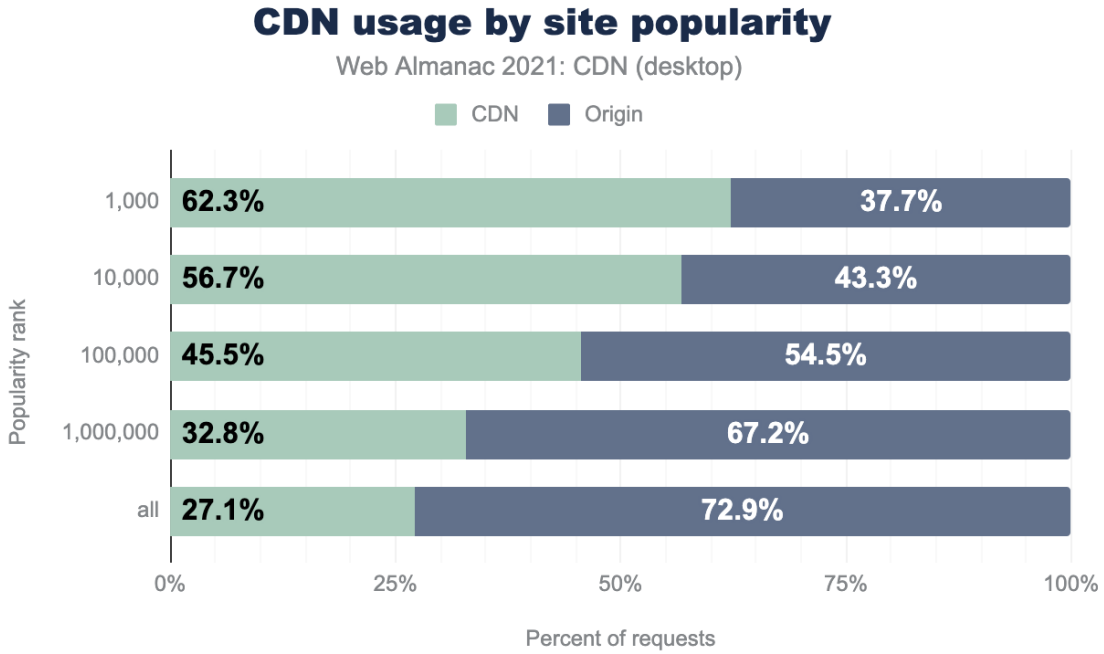


Figure 1.1: CDN usage by desktop site popularity (2021)
[2]

about which services that server is running, which ports it may have open, etc. Moreover, performing automated attacks on the target through its origin server usually becomes a trivial task and requires considerably less effort than attempting to do so via the CDN.

Taking this into consideration, it is interesting to analyze possible methods to discover origin servers behind the CDN, as it will further elucidate the importance of effectively securing and configuring web servers.

1.2 Personal Motivation

For the last two years, I started looking into web scraping and how that could help me obtain insightful information from releases of exclusive and limited items. However, many websites I was interested in scraping had some sort of reverse proxy. For the most part, this was a CDN with firewall protection, which also cached the content from the website. Since obtaining a fast and accurate response from the web server was crucial, I started looking into possible methods of skipping the cache.

After further research, I discovered one of the best ways to consistently skip cache on a website is to access it directly through its origin server, without any reverse proxy caching the requests in the way. Therefore, I gathered different methods which helped greatly in discovering the origin server behind a website. With this thesis I want to highlight the importance of effectively securing the origin of a web server and demonstrate that relying your server's security on a CDN is generally not a good idea.

1.3 Objectives

1. Develop an automated tool that will attempt to find the origin server behind a CDN, combining different reconnaissance methods.

2. Elucidate the importance of effectively securing an origin server, while simultaneously offering security researchers and pentesters further insight on origin server discovery.

1.4 Expected Impact

This work expects to impact two major parties. On one hand, security researchers and pentesters will be able to obtain more information about a specific target, and potentially help them with vulnerability discovery. Given the lack of past work on this subject, this thesis ought to motivate security researchers to investigate and share their findings on this topic, thus ultimately helping with the overall security of web servers.

On the other hand, content owners will learn about the risks of exposing their origin server and having a misconfigured web server. Moreover, this work will help them better understand how they can protect their origin server, and which measures they may take.

1.5 Methodology

The methodology on this work will be specifically focused on two aspects: research and programming the tool. Research must be considered as a key factor of this work, as there is not much previous related work about this topic. It is important to not only gather information from past resources, but moreover, to investigate and discover new findings ourselves. This is particularly relevant due to the pioneering aspect of this work.

Having done the adequate research, the tool will need to be programmed to a high standard and satisfy the specified requirements and objectives.

1.6 Structure

The work's structure will be as follows:

- **Acronyms**, a collection of acronyms, with references to their appearances on this work.

- **Chapter 1: Introduction**, in this chapter we briefly explain the reasons to make this work and what are our intended goals.

- **Chapter 2: State of the art**, we analyze the existing work on the topic of origin server discovery.

- **Chapter 3: Problem analysis**, we will discuss the requirements we expect to fulfill, and we will comment on potential solutions.

- **Chapter 4: Content Delivery Networks**, in this chapter we will discuss what are CDNs, how they operate and why they are relevant for our work.

- **Chapter 5: Reconnaissance methods**, analyzing reconnaissance and enumeration methods individually will allows us to gain a better understanding of how we pretend to reach our objectives.
- **Chapter 6: Solution design**, by discussing how the solution proposed is being designed, we aim to clearly explain how every main aspect of the software developed will work.
- **Chapter 7: Development of proposed solution**, in this chapter we will see a step-by-step procedure of how the solution was developed, including the reasons for some of the decisions taken. As well we will elaborate how the tool is to be implemented.
- **Chapter 8: Testing**, in this section we will comment how testing was performed, and which were its major influencing factors.
- **Chapter 9: Case Study: Accessing Zalando through their origin server**, in this chapter we will discuss what may be considered to be a real-world server misconfiguration, discovered with the tool developed. Details will be discussed on how it was discovered and which potential implications this may have.
- **Chapter 10: Protecting your origin server**, this section will explain how content owners can protect their origin server, in order to avoid unwanted, external users from accessing it directly.
- **Chapter 11: Conclusions**, in this chapter we will reflect on the work done, what could have been improved and what I have learnt from it.
- **Chapter 12: Future Work**, the last chapter, this includes a small discussion on what future work could include and how this work may be improved upon.

CHAPTER 2
State of the art

Throughout the last years, web servers and their configurations have changed considerably. As time passes by, previously considered safe practices start being considered unsafe and new unsafe practices appear. Therefore, in order to keep this work's relevance, it is important to research with innovative methods.

2.1 Analyzing current state of the art

There is no previous scientific or academic literature about the topic of origin server discovery, however some previous papers may resemble slightly this work.

The Development of a Reconnaissance Tool Aiming to Achieve a More Efficient Information Gathering Phase of a Penetration Test [3] written in 2021 aims to outline an approach that allows penetration testers to execute the information gathering phase of a penetration test more efficiently. Observing this thesis, we may notice that this tool's focus is placed on port discovery, through analyzing IP-ranges. This reconnaissance method is not particularly useful in our scenario, as port discovery is outside the scope of our objectives. Moreover, the tool developed does not check if the hosts being discovered are part of a CDN or not, it simply checks the services running on the ports discovered. Nevertheless, it may be a relevant thesis to analyze as key points of tool development can be extracted, which will help maintain a better structure on this work.

Another brief document interesting to analyze is *Automation of Recon Process for Ethical Hackers* presented on the 2022 International Conference for Advancement in Technology [13]. This document is only six pages long; however, it displays a summarized overview of passive and active reconnaissance hacking methods, placing special focus on the latter. The only reconnaissance method which can be extracted from this document and used on this work is subdomain enumeration. However, we will be leveraging public Open Source Intelligence (OSINT) tools which are specifically designed for this purpose for our work. Considering there is substantially more previous work on the subject of subdomain enumeration, scarce value may be added onto this thesis by developing a specific module for it on this work. Similar to the previously discussed thesis, this document facilitates keeping an accurate and precise structure on writing a detailed explanation for a new tool.

Despite this being the first academic work in the topic of origin server discovery, other non-scientific resources do exist which discuss origin server discovery in the form of blogs. A special mention must be done to the article *Finding The Origin IP Behind CDNs* published under the website InfoSec Write-Ups by the username *HolyBugx* [6]. On this article, the writer features several methods for origin server discovery, including

both active and passive reconnaissance methods. Unlike previously discussed papers, this article does not detail development for an automated tool for the purpose of origin server discovery. However, we may obtain some key takeaways from it and the article helps better understand the topic of discussion.

2.2 Proposed solution

Having discussed the current state of the art, we considered it would be fruitful to make specific, unprecedented work on this topic. The tool developed will leverage passive reconnaissance methods for host discovery, targeted towards origin server discovery. Furthermore, the tool will implement a state-of-the-art method for testing content similarity between content responses. All in all, this work will help other security researchers and ethical hackers to gain more knowledge on this topic, and possibly incentivize further literature to be written about it.

CHAPTER 3
Problem analysis

So as to discuss our problem, we will attempt to tackle our objectives previously defined, while explaining the thought process behind. Therefore, we can subdivide this work into two main topics, explaining the problem in-depth and discussing the development for an automated tool to solve the problem. Starting with the former, we considered to include relevant figures and explanations of each separate key topic, such as reconnaissance and CDN. Moving onto the latter, we decided to craft a list of functional and non-functional requirements for the tool, which must be fulfilled in order to consider the tool to be successfully developed.

3.1 Requirements

The aim of this section is to present and detail how the requirements were organized. We will subdivide requirements between functional and non-functional requirements. Functional requirements define how the system must work, whereas non-functional requirements will define how the system must perform and which quality constraints it must satisfy. Requirements are redacted in Table 3.1.

3.2 Ethical considerations

Nowadays, most tools used for penetration testing and cyber security research are public. However, we can consider these tools can also be used by black-hat hackers for malicious purposes. This may be the case for this newly developed tool, however at the time of publishing this work, this tool is not open-source and available for anyone to use. Nevertheless, even if this tool does eventually reach public domain, its intended users are ethical hackers and penetration testers. Moreover, the information discussed on this work can help content owners to emphasize the importance of having securely configured servers.

3.3 Potential solutions

As one of the core aspects of the proposed solution is the discovery of new, related hosts, it is interesting to observe the existence of numerous host discovery methods which can be implemented onto a new solution in order to obtain more results. Considering time limitations for this work, emphasis was placed on those specific methods which from personal experience yield more success in the process of origin server discovery, which

Functional Requirements	Non-Functional Requirements
The system shall have a dictionary to insert and get hostname results.	The system shall only insert hostnames which do not belong to CDN subnets.
The system shall handle network requests to external APIs with credential authorization.	The system shall report on standard output which requests it is performing and their results.
The hostname dictionary shall be written to an external file.	The user shall optionally define an external file path to which to dump to the hostname dictionary.
The system shall load hostnames from a provided file path.	The user shall optionally define an external file path to which to read hostnames from.
The system shall perform HTTP GET requests to a provided target URL.	The system shall allow for insecure requests to be performed, e.g: a request to a target with an invalid SSL certificate.
The system shall spoof real browser network requests.	The user shall optionally define with a flag input, if they want to perform browser-like network requests.
The system shall distribute its dictionary's hostnames to workers.	The user shall optionally define with a flag input, the number of workers the system will run.
Workers shall algorithmically compare content from network responses.	The user shall optionally define with a flag input, the threshold value the system will decide content similarity upon.
Workers shall share their results to the main running process.	The system shall report all results obtained from the workers into the standard output, including the content similarity and if it detected a match or not.

Table 3.1: Functional and non-functional requirements

are all passive reconnaissance methods. Nevertheless, other solutions may exist, which can include further enumeration methods, such as using active reconnaissance methods.

Therefore, the number of potential solutions is very extensive, and this work marks a precedent in this topic which may help further solutions to be developed on this matter.

CHAPTER 4
Content Delivery Networks

Content delivery networks utilize network resources more efficiently, thus improving the quality of experience when delivering digital contents to end users. A CDN generally caches digital contents in locations geographically nearby end users, routes their content requests to these locations, and ultimately transfers said content to the respective users.

End users, content owners and CDN providers form the different parties involved in the content delivery process. End users consume contents through electronic devices, with the use of laptops, smart phones, tables. Content owners are customers of a CDN and own digital contents which are meant to be shared. CDN providers manage and operate the CDN infrastructure.

4.1 Content Delivery Process Overview

Once the end user has made the decision to use a CDN to deliver their content, there is a separate content delivery process which is in place, and it consists of several main steps.

Initially, content owners place digital contents in their origin servers. The selected CDN distributes and replicates the contents from the origin server across its plentiful edge servers (ranging from hundreds to thousands of servers). Such edge servers are geographically distributed to provide low access times to end users.

As we can observe from Figure 4.1, once the edge servers have obtained contents from origin servers, there are different actions that are taken to provide these contents to an end user.

1. End users request the contents they are interested in consuming to the CDN directly, acting as a man in the middle.

2. CDNs use request routing mechanisms to select and redirect the request to one of its numerous edge servers.

3. Edge servers perform security checks and will decide whether the content requested is to be delivered to the end user from cache. These checks depend on the level of security and protection required by content owners which are generally able to configure them through the CDN.

4. In the case that there is a cache miss, edge servers will first obtain information from the origin server, update their contents accordingly and deliver them to the end user.

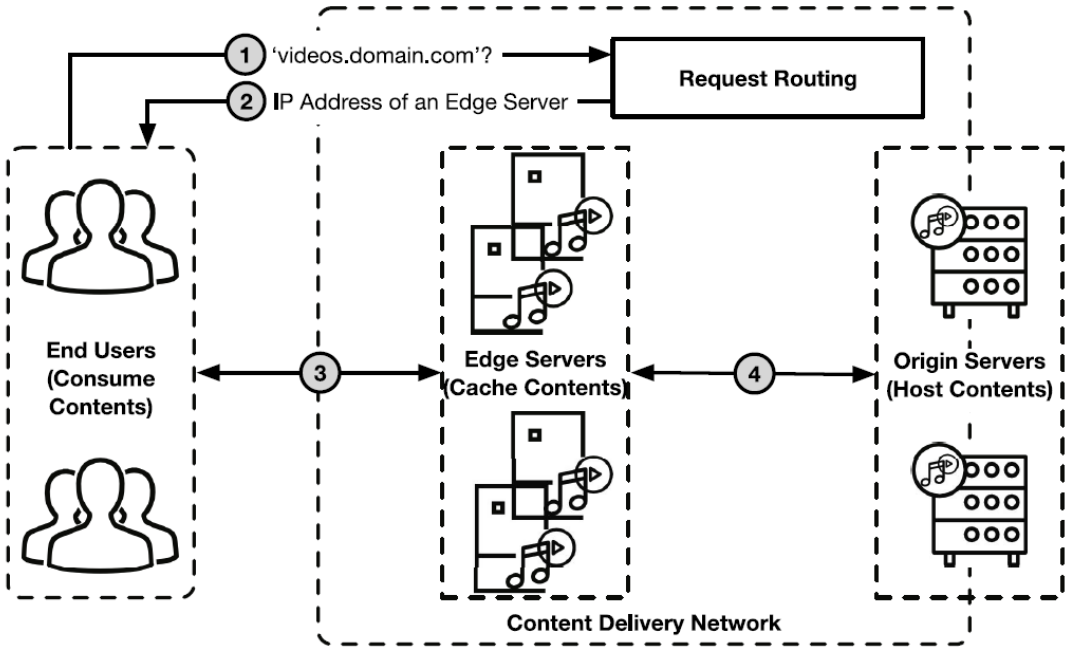


Figure 4.1: Content Delivery Network Infrastructure. The request routing component redirects end user requests to edge servers that cache and serve contents. On cache misses, edge servers forward requests to origin servers that host contents. Origin servers are operated by either CDN providers or content owners.
[5]

4.2 Analyzing CDN effectiveness

As we have observed, the CDN model offers high availability, while also maintaining a relatively low load on the origin server thus offering fast content access.

Figure 4.2 depicts how content delivery is performed when users access digital contents from the origin server directly. Allowing any user to access and download content from your origin server increases the load on your server and may put a significant toll in performance, which is a relevant factor for content owners.

The remarkable difference in origin server load between Figure 4.2 and Figure 4.3, explains why content owners may be interested in delivering content through a CDN.

Figure 4.2: Content delivery without using a CDN
[6]

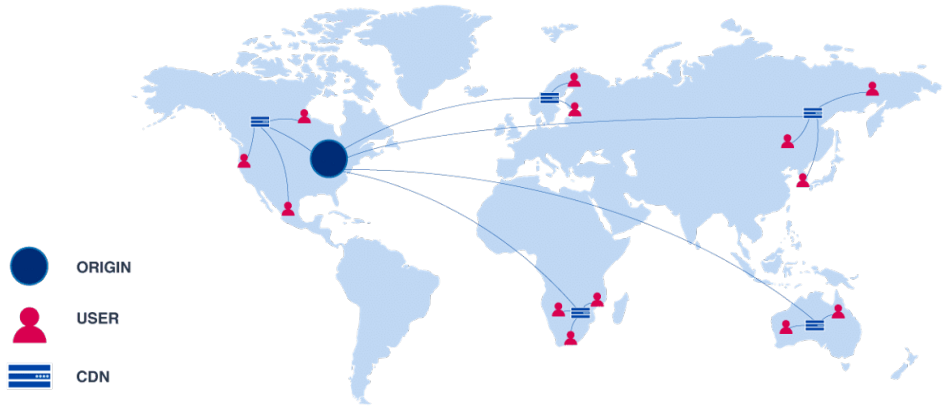


Figure 4.3: Content delivery with a CDN
[6]

Performance of the origin server can be preserved, while simultaneously offering an improved user's quality of experience.

4.3 WAF integrations within CDNs

CDN providers nowadays offer a wide variety of services and products which can be embedded into CDNs to enhance the security of content delivery.

Akamai, one of the most relevant content delivery network providers in the market right now, offers products such as its bot manager, depicted in Figure 4.4 to mitigate bot attacks. CDN providers offer web application firewalls and similar products whose end goal is to secure content delivery and reduce the number of attacks that can be performed on content owner's servers.

Furthermore, setup of CDNs with web application firewalls integrated is a trivial task for content owners. Taking this into consideration, many content owners using a CDN will integrate its web application firewall and may ultimately take for granted the performance and security of their website.

Nevertheless, once attackers gain direct access to origin servers, they may pose a significant threat to their security and performance. Figure 4.5 depicts a simple network diagram of content delivery from an origin server to an attacker, which in the general case is unintended.

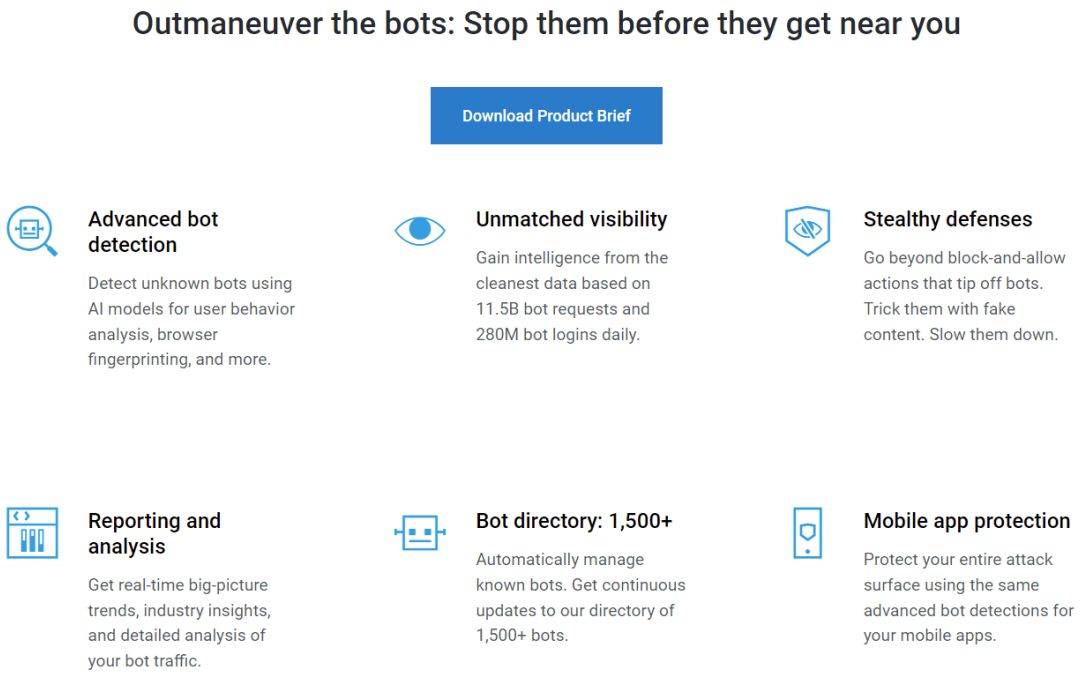


Figure 4.4: Akamai Bot Manager
[14]

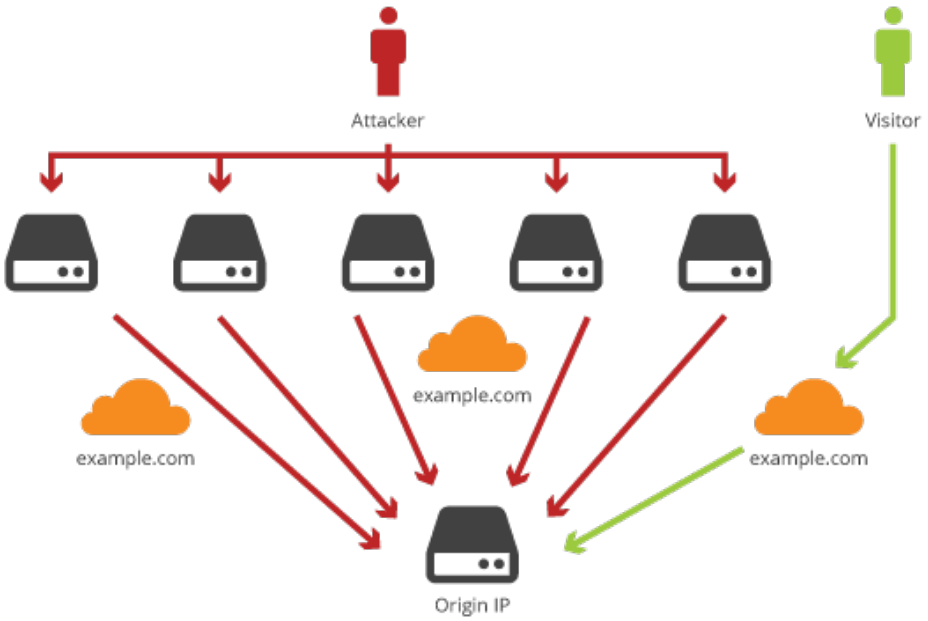


Figure 4.5: Attacker accessing a server through its direct origin IP address
[6]

CHAPTER 5
Reconnaissance methods

Hackers have diverse ways to help them discover more information on a target and potentially obtain their origin server. We call this phase reconnaissance. "Reconnaissance is a set of processes and techniques (Footprinting, Scanning & Enumeration) used to covertly discover and collect information about a target system" [11]. We can subdivide reconnaissance into passive and active methods.

Passive reconnaissance involves gathering information about a target without interacting with the target directly. Therefore, no request is being sent to the target server, thus it has no way of knowing you are collecting information on them. Often, public resources are used to gather information, this is OSINT. With it, attackers can discover IP addresses, host names, domain names, historic Domain Name System (DNS) records and which services they are running.

Active reconnaissance involves directly interacting with the target to gain information about it. In the scope of origin server discovery, there are several methods which can be used, such as scanning the content delivered by the target for IP addresses and hostnames that may appear inside HTML or JavaScript (JS) files for example. Other methods involve making use of debug headers, using WordPress XLM-RPC pingback API if the target's Content Manager System (CMS) is WordPress or even searching for favicon hashes.

On Table 5.1 we can observe some of the most common discovery methods for origin IP searching. For this thesis we will focus on some of the most used and effective passive recon methods, such as Secure Sockets Layer (SSL) certificate searching, looking through historical DNS records, or finding subdomains and IP addresses related to the target.

Active	Passive
Favicon Hashes	DNS Records
XML/RPC Pingbacks	MX Records and Methods
SSRF Trackbacks	SSL Certificates
Misconfigured Subdomains	Shodan, Censys, Zoomeye
Verbose Errors	Vhosts
Debug Headers	IP Range/CIDR

Table 5.1: Active and Passive reconnaissance methods

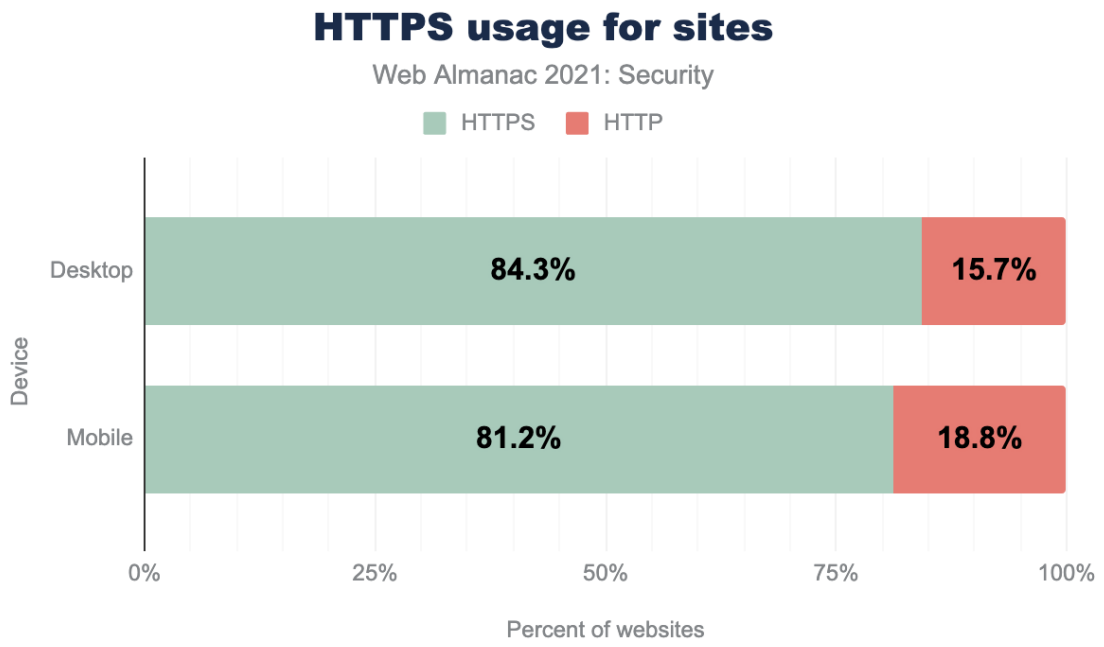


Figure 5.1: HTTPS usage for websites
[2]

5.1 Search matching SSL certificates

An SSL certificate is a digital certificate that authenticates a website's identity and enables an encrypted connection. "SSL stands for Secure Sockets Layer, a security protocol that creates an encrypted link between a web server and a web browser" [8].

Observing Figure 5.1 we may assess that in general terms, websites prefer to use SSL/TLS to encrypt their connection between the end user and the server.

With the number of certificates being issued increasing rapidly, in 2013 a solution was proposed to keep track of them by Ben Laurie and Adam Langley, which was named *Certificate Transparency*. The core idea behind *Certificate Transparency* is the public, verifiable, append-only log. "Creating a log of all certificates issued that does not need to be trusted because it is cryptographically verifiable allows clients to check that certificates are in the log, and servers can monitor the log for mis-issued certificates" [9]. Certificate transparency allows to match a specific SSL/TLS certificate to a SHA-1 or SHA-256 hash, which can therefore be cross-referenced and matched across different domains and websites.

As it is portrayed in Figure 5.2 the same SSL certificate can be used across different domains and subdomains. When a content owner sets their website on the Internet with the CDN, it may expose the SSL certificate it is using on their origin IP. Scanning the full range of possible IP addresses 0.0.0.0/0, it would be possible to discover the origin IP behind a website using a CDN, as their SSL certificate hashes will match.

Since scanning the entire range of IP addresses is impractical, there exist services such as Censys which allow you to obtain the SSL/TLS fingerprints of websites serving with HTTPS.

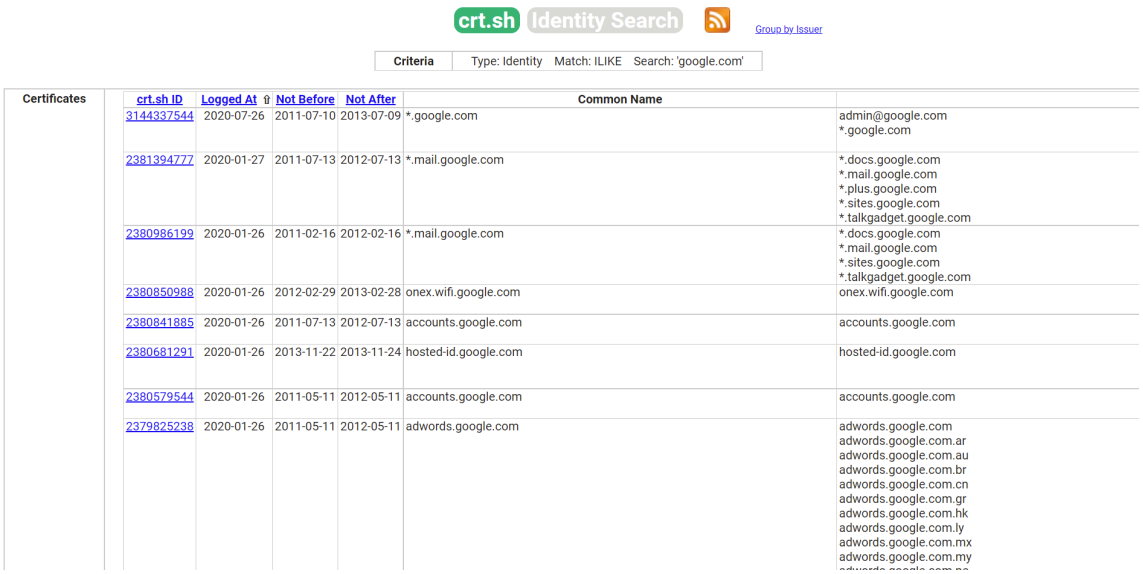


Figure 5.2: Certificate Transparency Logs on *google.com*, obtained from crt.sh

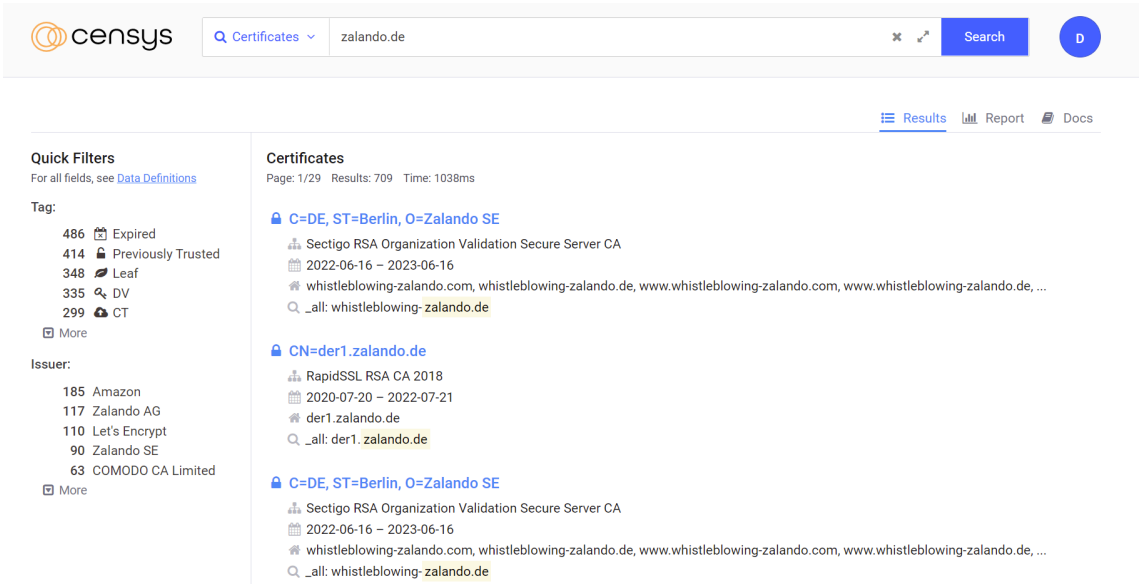


Figure 5.3: Searching for certificates on *zalando.de* with Censys

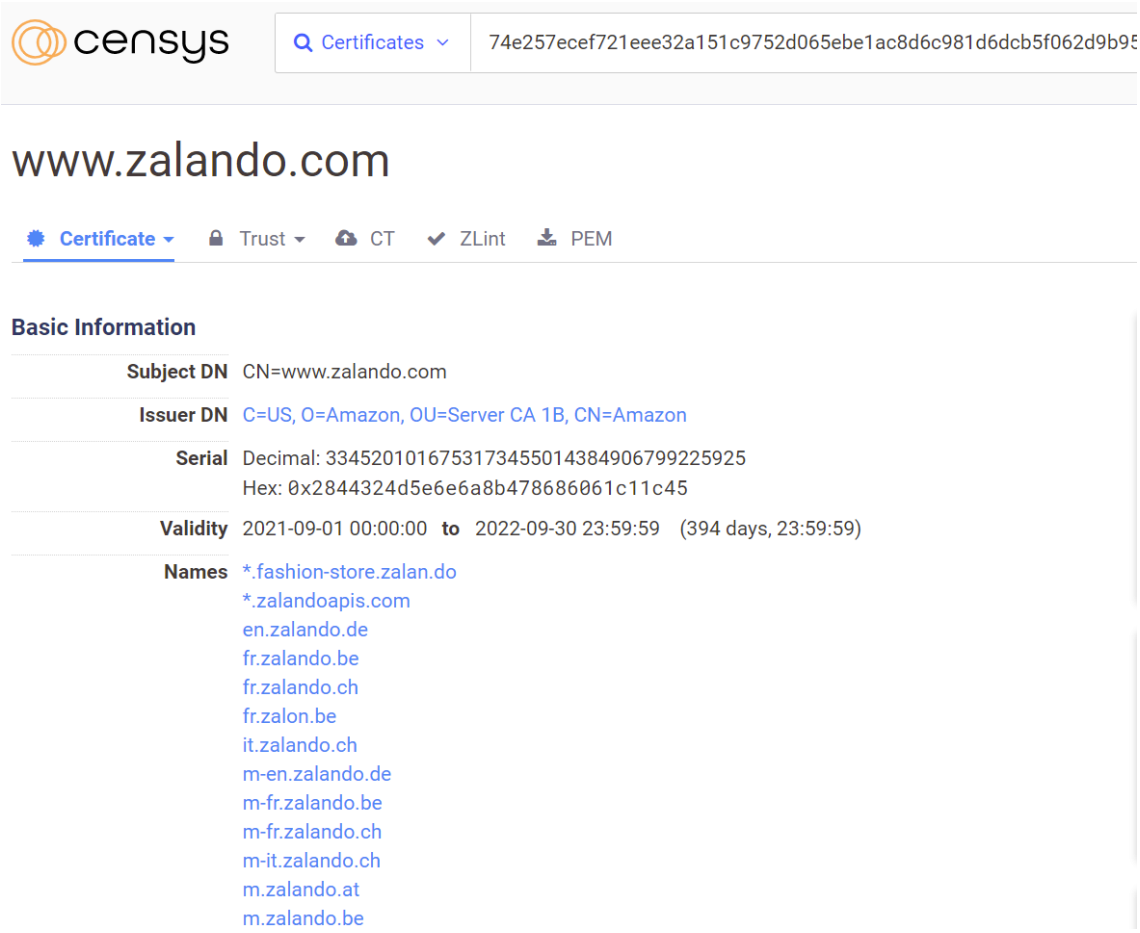


Figure 5.4: Example certificate of *zalando.de*

As we may observe from Figure 5.3, we obtain several SSL certificates which are issued for specific hosts matching a Subject Distinguished Name [1]. We may obtain further information regarding each certificate such as its SHA-1 and SHA-256 fingerprints, as can be observed on Figure 5.4.

Censys, as well as other similar providers, offer specific OSINT about the target, in such way that we may find which IPs are serving that same SSL/TLS certificate fingerprint. By performing a reverse search on the certificate fingerprint, it is therefore possible to obtain which other IP addresses or host names are using the same HTTPS encryption.

Judging from Figure 5.5, searching for a specific certificate fingerprint has returned previously undiscovered hosts which share the same fingerprint as our original target delivering content through their CDN. They may be potentially tested to attempt a direct connection to the origin server.

5.2 Enumerating IP addresses with DNS historical records

"DNS stands for Domain Name System, and it is considered the phonebook of the internet" and has been essential for the functionality of the Internet since 1985". Humans access information online through domain names, such as *google.com* or *upv.es*. Web browsers and digital devices interact through IP addresses. DNS translates domain names to IP addresses so browsers can load Internet resources [15].

[1]Subject Distinguished Name is a term which describes the owner or requestor of a certificate

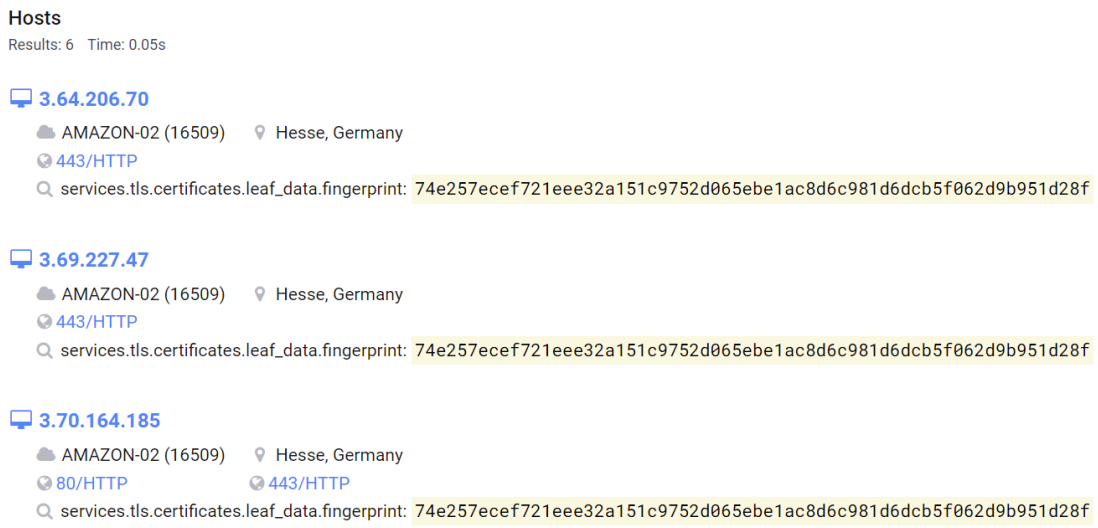


Figure 5.5: Searching for hosts provided a certificate fingerprint on Censys

Each device connected to the Internet has associated a unique IP address which other machines use to find the device. Therefore, DNS servers eliminate the need for humans to memorize IP addresses such as 127.0.0.1 in the case of IPv4 or more alphanumeric IPv6 addresses such as 2400:cb00:2048:1::c629:d7a2.

DNS plays a vital role in content delivery networks, as we previously portrayed its impact resolving IP addresses geographically nearby the end user in Figure 4.3. The main advantage of using DNS in content delivery networks, is that it can deliver fast and reliable content to end users across a wide range of geographical locations.

One of the key parts of the structure of a DNS is its domain name space, which consists of a tree data structure, portrayed in Figure 5.6. In it, each node or leaf contains several resource records, which hold information associated with the domain name. Each record has a type, an expiration time, a class, and type-specific data. The number of different resource records has been expanding over the years, and the information they hold varies from IP address records to SSH Public Key fingerprints. Since one of the key elements of reconnaissance is to collect IP addresses that belong to the target, we will be focusing on resource records with type A, MX and TXT [17].

- The function of A resource records is to return 32-bit IPv4 addresses. It is most commonly used to map hostnames to an IP address of the host, but is also used for storing subnet masks in RFC 1101, DSNBLs, etc.
- MX records map domain names to a list of message transfer agents for that domain. Assuming the target shares the same IP for its webhosting than for its mail servers, analyzing these records provide a useful method of searching potential origin servers.
- TXT was originally planned for human-readable text in a DNS record; however, it often contains machine-readable data. On it, other potentially relevant IP addresses may appear, which may easily be obtained with a regular expression search.

Nowadays, there exists numerous DNS lookup tools which retrieve and return resource records from DNS. Most modern Linux systems include dig (Domain Information Groper), a command used to gather DNS information. An example usage may be found below in Figure 5.7.

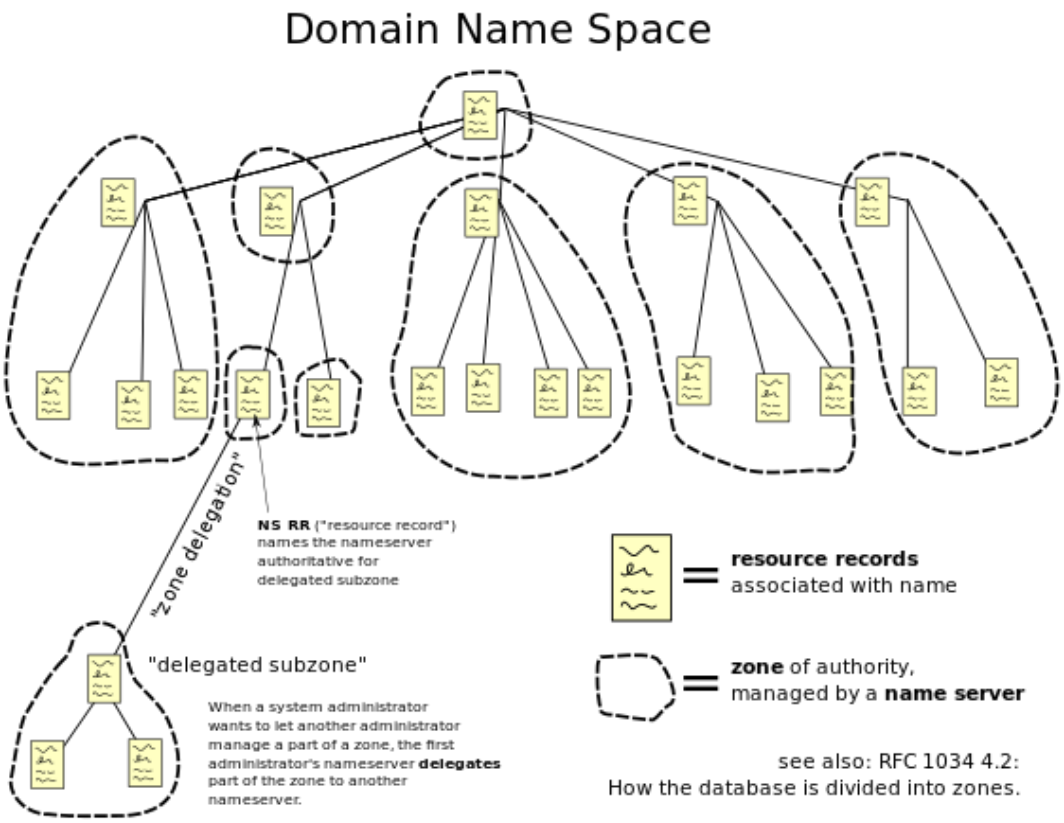
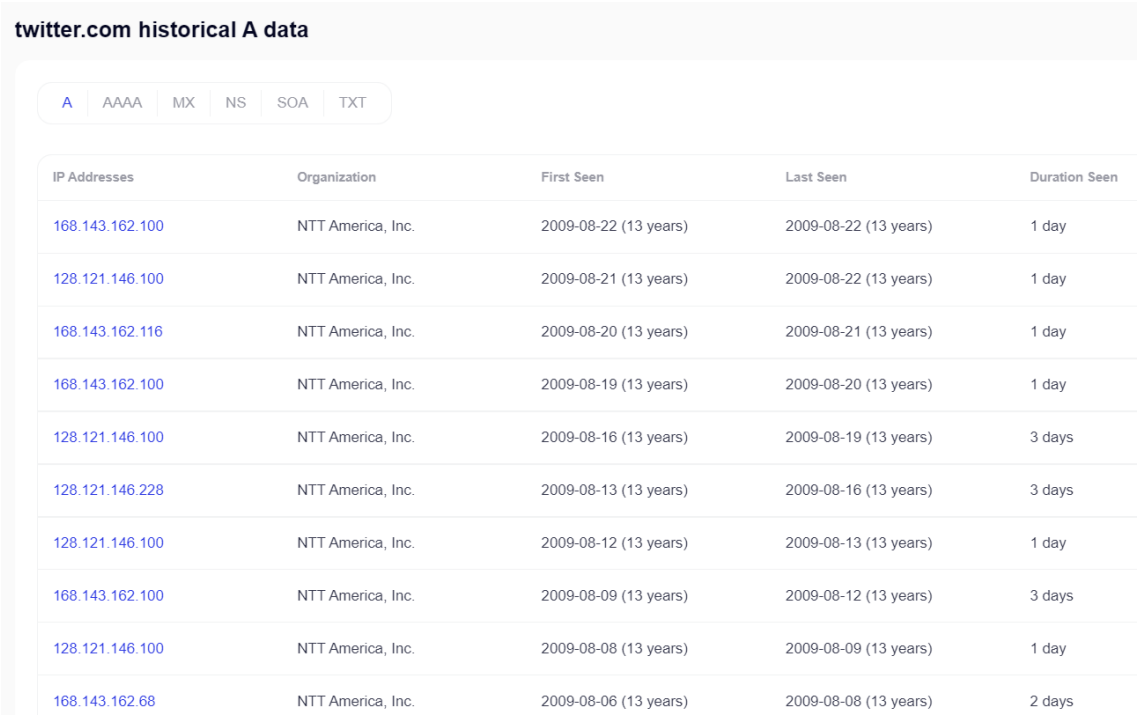


Figure 5.6: Domain Name Space structure
[15]

```
❯ dig google.com

; <<>> DiG 9.16.27-Debian <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19022
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; MBZ: 0x0005, udp: 4000
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             5       IN      A       142.250.201.78

;; Query time: 33 msec
;; SERVER: 192.168.163.2#53(192.168.163.2)
;; WHEN: Wed Jun 22 23:32:23 CEST 2022
;; MSG SIZE  rcvd: 55
```

Figure 5.7: Dig usage on *google.com*



twitter.com historical A data

A | AAAA | MX | NS | SOA | TXT

IP Addresses	Organization	First Seen	Last Seen	Duration Seen
168.143.162.100	NTT America, Inc.	2009-08-22 (13 years)	2009-08-22 (13 years)	1 day
128.121.146.100	NTT America, Inc.	2009-08-21 (13 years)	2009-08-22 (13 years)	1 day
168.143.162.116	NTT America, Inc.	2009-08-20 (13 years)	2009-08-21 (13 years)	1 day
168.143.162.100	NTT America, Inc.	2009-08-19 (13 years)	2009-08-20 (13 years)	1 day
128.121.146.100	NTT America, Inc.	2009-08-16 (13 years)	2009-08-19 (13 years)	3 days
128.121.146.228	NTT America, Inc.	2009-08-13 (13 years)	2009-08-16 (13 years)	3 days
128.121.146.100	NTT America, Inc.	2009-08-12 (13 years)	2009-08-13 (13 years)	1 day
168.143.162.100	NTT America, Inc.	2009-08-09 (13 years)	2009-08-12 (13 years)	3 days
128.121.146.100	NTT America, Inc.	2009-08-08 (13 years)	2009-08-09 (13 years)	1 day
168.143.162.68	NTT America, Inc.	2009-08-06 (13 years)	2009-08-08 (13 years)	2 days

Figure 5.8: Historical *A* DNS records for *twitter.com*

Figure 5.9: Subdomain example
[18]

We may leverage this command to collect IP addresses from the current DNS information about a specific hostname. Nevertheless, it is even more fruitful to obtain and analyze historical DNS records, in such way that we may obtain past records thus expanding the number of IP addresses we can gather.

There exist many services that provide DNS historical records, for example Security Trails, which we will be using in the development of our automation tool.

Figure 5.8 displays the historical DNS records for a specific website, from it we may gather more IP addresses, which could lead us to finding a potential direct origin server access.

5.3 Subdomain and IP enumeration

In the Domain Name System hierarchy, a subdomain is a domain which is part of another main domain [18]. For example, a website could be served under example.com, and said website could offer an online shop under shop.example.com. A visual example can be found in Figure 5.9

Subdomains are generally used by internet service providers supplying web services. They allocate subdomains to their clients, thus allowing them to independently adminis-

```
C:\Users\alexv\Documentos\Pentest Tools\amass_windows_amd64>amass enum -config config.ini -ip -d google.com
google-proxy-66-102-6-210.google.com 66.102.6.210
rate-limited-proxy-74-125-217-120.google.com 74.125.217.120
google-proxy-74-125-209-210.google.com 74.125.209.210
google-proxy-74-125-212-100.google.com 74.125.212.100
google-proxy-64-233-172-156.google.com 64.233.172.156
forcesafesearch.google.com 216.239.38.120
rate-limited-proxy-72-14-199-20.google.com 72.14.199.20
google-proxy-64-233-172-99.google.com 64.233.172.99
dns.google.com 8.8.4.4,8.8.8.8
rate-limited-proxy-74-125-217-0.google.com 74.125.217.0
ns1.google.com 216.239.32.10
google-proxy-66-102-7-200.google.com 66.102.7.200
adwords.google.com 64.233.185.139,64.233.185.102,64.233.185.101,64.233.185.100,64.233.185.113,64.233.185.138
google-proxy-74-125-209-110.google.com 74.125.209.110
time1.google.com 2001:4860:4806::,216.239.35.0
profiles.google.com 172.217.18.142
google-proxy-64-233-172-104.google.com 64.233.172.104
google-proxy-64-233-172-136.google.com 64.233.172.136
216-239-45-63.google.com 216.239.45.63
mail-oi1-f200.google.com 209.85.167.200
google-proxy-64-233-173-71.google.com 64.233.173.71
rate-limited-proxy-72-14-199-100.google.com 72.14.199.100
google-proxy-64-233-172-170.google.com 64.233.172.170
google-proxy-64-233-172-54.google.com 64.233.172.54
google-proxy-66-102-7-121.google.com 66.102.7.121
google-proxy-64-233-173-87.google.com 64.233.173.87
currents.google.com 172.217.16.238
google-proxy-66-102-6-4.google.com 66.102.6.4
google-proxy-64-233-172-26.google.com 64.233.172.26
```

Figure 5.10: Subdomain enumeration performed with Amass on *google.com*

ter them. Furthermore, subdomains are used by organizations which wish to subdivide a particular service, or department within their organization.

A key aspect of the reconnaissance phase is enumerating IP addresses and subdomains. Given the case where a target hosts two or more different subdomains under the same server, it may be possible to access them within each other. Under the scenario that our target subdomain is being hosted under a CDN, while another subdomain we are interested in targeting is not, we could potentially obtain direct origin server access while sending requests to the subdomain without a CDN setup.

Nowadays, there exists many tools and resources which allow for subdomain and IP address enumeration. For our automation tool we will be using Open Web Application Security Project (OWASP) Attack Surface Mapping and Asset Discovery (AMASS) tool.

OWASP is a nonprofit foundation that works to improve the security of software. OWASP Foundation is the source for developers and technologists to secure the web through community-led open-source software projects. Through their Amass tool it is possible to perform external asset discovery leveraging OSINT and active reconnaissance techniques.

While Amass offers five main subcommands: intel, enum, viz, track and db; we will be focusing on the "enum" subcommand, which is used to perform DNS enumeration and network mapping of systems exposed to the Internet [10].

This subcommand leverages different API services, both free and paid, to gather subdomains and their associated IP addresses. There exist numerous flags available to execute the enum command, some of which are most relevant to our use case are: "-active" to perform active recon methods, "-brute" to perform brute force subdomain enumeration, "-d" to select the target domain to obtain information from and "-ip" to display the associated IP addresses. An illustrative example may be found in Figure 5.10

Having collected the associated subdomains and IP addresses to our target, we may have more information which help find direct origin server access.

```
Host = uri-host [ ":" port ]
```

Figure 5.11: Host header structure
[12]

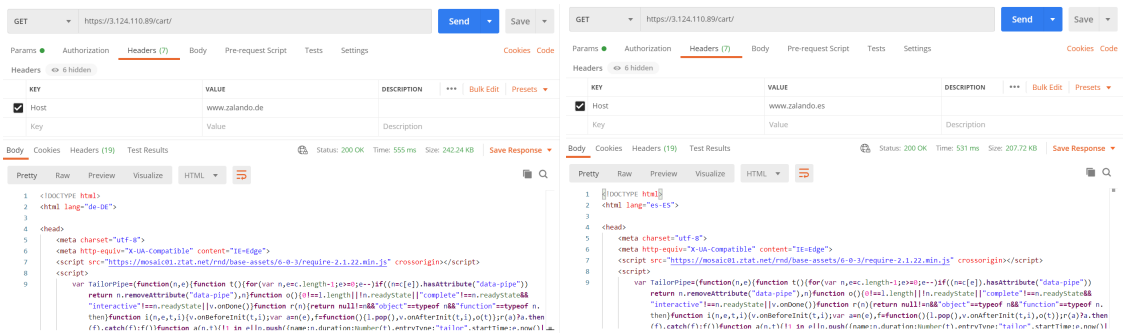


Figure 5.12: Host header spoofing to route internal requests

5.4 Comparing content similarity

According to RFC 9110, the "Host" header field in a request provides the host and port information from the target URI, enabling the origin server to distinguish among resources while servicing requests for multiple host names [12]. "In HTTP/2 [HTTP/2] and HTTP/3 [HTTP/3], the Host header field is, in some cases, supplanted by the ":authority" pseudo-header field of a request's control data".

The Host header structure is clearly depicted in Figure 5.11. The target URI's authority information is critical for handling a request. A user agent MUST generate a Host header field in a request unless it sends that information as an ":authority" pseudo-header field. A user agent that sends Host should send it as the first field in the header section of a request.

As host and port information act as an application-level routing mechanism, it may be possible to redirect requests to internal servers, without first verifying that the connection is targeting a valid IP address for the provided host.

Abusing possible server misconfigurations, it is therefore possible to route a request targeting a host, to deliver content from an associated subdomain or host.

Analyzing Figure 5.12 we may realize that supplying a different Host header to the provided IP address yields two different results. On the left side of the figure, we can observe from the html lang tag, how we were able to access the German content from the Zalando online shop, while on the right side we obtained its Spanish counterpart.

An improperly misconfigured server may deliver content for a different subdomain than the target domain if an arbitrary Host header is supplied.

We may, therefore, use the IP addresses and subdomains previously enumerated to perform requests, spoofing the Host header thus potentially obtaining content from the target origin's server.

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise,} \end{cases}$$

Figure 5.13: Levenshtein distance function
[16]

		m	e	i	l	e	n	s	t	e	i	n
	0	1	2	3	4	5	6	7	8	9	10	11
l	1	1	2	3	3	4	5	6	7	8	9	10
e	2	2	1	2	3	3	4	5	6	7	8	9
v	3	3	2	2	3	4	4	5	6	7	8	9
e	4	4	3	3	3	3	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

Figure 5.14: Levenshtein distance visualized
[4]

Nevertheless, the content we receive should be compared against the original content from the target serving through a CDN, in order to allow us to verify the host we targeted is in fact the origin server.

A popular computer science metric used to measure the difference between two strings is the Levenshtein distance. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other [16].

The Levenshtein distance between two strings a, b (of length $|a|$ and $|b|$ respectively) is given by $lev(a, b)$ where the *tail* of some string x is a string of all but the first character of x, and $x[n]$ is the nth character of the string x, counting from 0. We may observe this from the function definition Figure 5.13.

A visualization of Levenshtein's edit distance may be found in Figure 5.14

Using Levenshtein's distance to compare the difference between the original content and the latest content obtained by spoofing the Host header request, we have a precise metric to decide upon the similarity of the content. This metric is relevant as it does not depend on subjective evaluation and can be integrated on programming languages and automation software.

Maintaining a low processing rate while performing Levenshtein's distance algorithm with extensive strings, is a challenging task. According to HTTP Archive, the median desktop page had 29 KB of HTML. Considering every character of text has a size of one Byte, this translates into a non-negligible total number of characters on the page in question. Comparing the similarity between multiple pages with respect to median or extensive HTML pages can result in a tedious and prolonged task.

A solution is proposed to reduce the comparison time and ultimately find the similarity between two texts, absolute content distance. Despite it is not a valid edit distance algorithm, it may be in some cases an adequate solution to discover content similarities between two pages, by comparing their length differences. Absolute content distance has a time complexity of $O(1)$, while Levenshtein distance has a time complexity of $O(nm)$ being n and m the texts to be compared. We may conclude that absolute content distance might be a preferred solution for extensive page content comparison if at the same time, achieving results with speed is relevant.

CHAPTER 6
Solution design

This provided solution considers the provided reconnaissance methods and incorporates them with the goal of obtaining IP addresses and relevant associated hosts to a provided host target, this constitutes the first phase of the tool. Subsequently, the second phase comprises of performing requests to the previously collected IP addresses and hosts, while spoofing the Host header and their responses are respectively compared to the original response using Levenshtein's distance or alternatively absolute content distance when applicable.

6.1 System architecture

This solution encompasses several features which distinguishes it considerably from other existing projects. Key features of the project are depicted on table 6.1.

S.No	Features of the project
1	Time taken for both reconnaissance phase and content similarity comparison is considerately low, as the tool is fully written in Golang. Golang provides built-in concurrency support and handles multiple network requests with ease.
2	Provides support to obtain content from specific paths or directories, which may yield better results than accessing the raw / path.
3	Effective support to spoof requests as if they were being made through a Chrome browser, by matching exactly its headers and TLS fingerprint.
4	Allows for external hosts or IP addresses to be appended directly onto the total collection of hosts, by reading from a provided file path.
5	With CDNs having already known and existing CIDRs, this solution allows for a simple way of blacklisting IP addresses to be included on the host collection.

Table 6.1: Key features of the project

6.2 Workflow of the proposed model

1. Reads and validates several input flags from the user as a command line argument to the tool.

2. Extracts the host and path from the provided URL, then initializes a host collection dictionary.

3. If an optional flag is provided to load hosts from an external file, it will attempt to read it line by line and store each host onto the previously initiated dictionary.

4. Performs an initial network request to the URL provided by the user and stores its string response on a variable.

5. Enumerates SSL fingerprints associated with the host previously extracted, through Censys API.

6. Iterates through every SSL fingerprint and collects IP addresses which have matching SSL fingerprints, once again through Censys API. The IP addresses discovered are then added onto the existing host dictionary.

7. Obtains historical A, MX and TX DNS records from the original host, using the API offered by Security Trails.

8. Scans every record to obtain IP addresses from them and adds them onto the host dictionary.

9. Initializes an optionally specified number of workers which are arbitrarily sent IP addresses and hosts from the host dictionary, one by one.

10. Each worker will perform four unique requests to the parsed host, two requests with the provided host, HTTP scheme and the path initially extracted, and two other requests with the same host, HTTPS scheme and the same path extracted. Moreover, one of each respective HTTP and HTTPS request will have no SSL certificate verification, thus allowing insecure hosts to be analyzed.

11. Upon obtaining each response, its content is then analyzed using Levenshtein's distance or absolute content distance if the content length of the original response is above 65536 characters.

12. Workers then submit the similarity result to the main thread which will be displayed in the terminal.

6.3 Flow diagram of the proposed model

The step-by-step process of the model is portrayed in Figure 6.1 in the form of a flowchart. It clearly depicts how each module interacts with each other and displays its working flow. Each main task like SSL certificate fingerprint search, DNS historical record search are termed as a module as they contribute to the final output of the model.

6.4 Detailed design

Having considered a brief overview of the proposed model for the tool, it is important to classify and discuss in-depth its main individual modules.

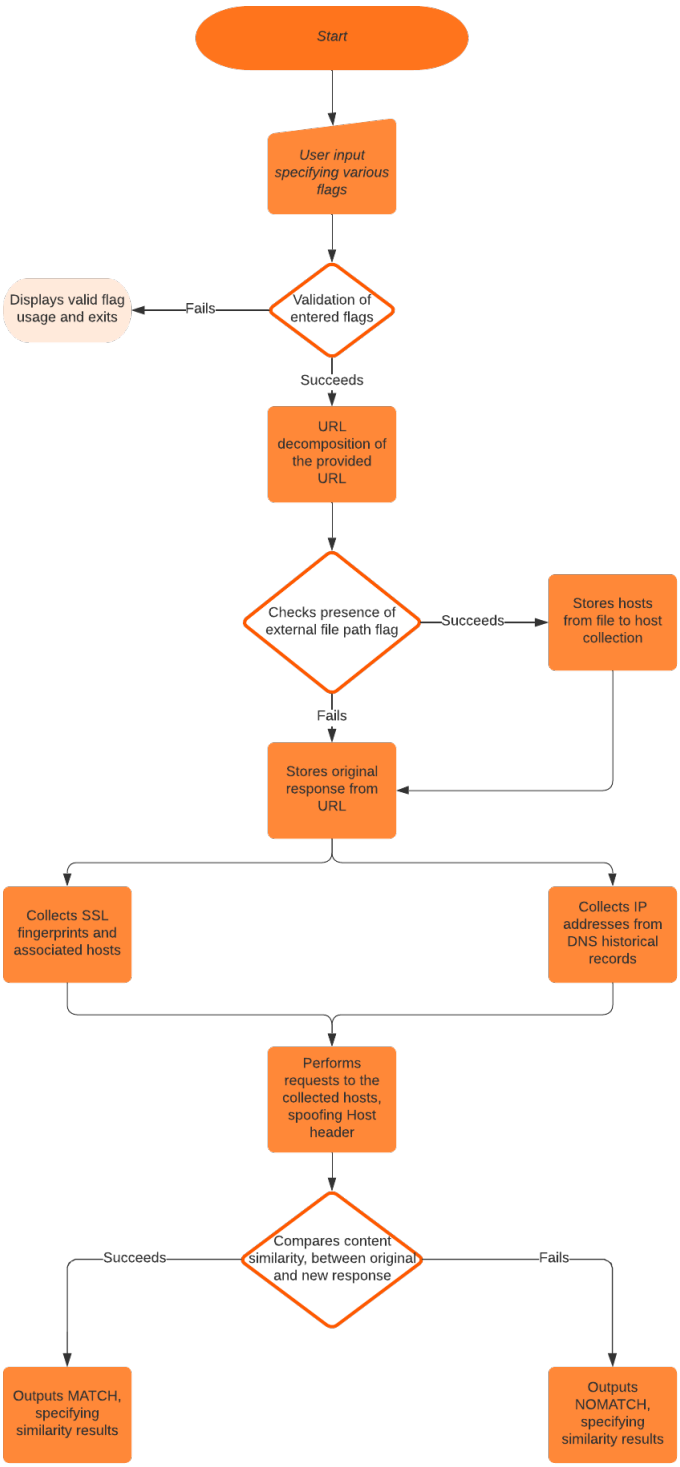


Figure 6.1: Flow diagram of proposed model

6.4.1. Flag Validation

It is important to start by discussing the expected input flags by a user from the command line. The tool reads command line arguments from the standard input and automatically parses them, storing their respective values. Table 6.2 presents a detailed explanation of every input flag.

Flag Input	Explanation
Threshold	Uses the prefix "$-l$" and has a default value of 5. This integer input defines the Levenshtein threshold which represents the maximum number of characters that may differ between the response from the spoofed Host header requests and the original response. A higher threshold value implies the tool will require less character difference in order to display a match.
Workers	Uses the prefix "$-t$" and has a default value of 32. This integer input defines the number of workers which will be crafting requests to the recently obtained hosts from the host collection dictionary. Workers may be used to fasten execution times, but it may also lead to rate-limits or IP address blacklisting from certain servers.
Censys API ID	Uses the prefix "$-censys-api-id$" and has no default value. This string input defines the Censys API ID, which is part of the API credentials required to utilize Censys services.
Censys API Secret	Uses the prefix "$-censys-api-secret$" and has no default value. This string input defines the Censys API secret, which is part of the API credentials required to utilize Censys services.
Security Trails API Key	Uses the prefix "$-sec-api-key$" and has no default value. This string input defines the Security Trails API key, which will be used in order to obtain DNS historical records from Security Trails databases.
Host File	Uses the prefix "$-h$" and has no default value. This string input defines a file path containing a list of hostnames which will be added onto the hosts collection dictionary, which will later be utilised to generate spoofed requests.
Output File	Uses the prefix "$-o$" and has no default value. This string input defines a file path to which the tool will try to write all the hosts discovered.
Fingerprint	Uses the prefix "$-f$" and has a default value of false. This boolean input defines whether the tool will try to emulate Chrome's browser in order to load content responses. It may be useful for websites with a more advanced bot protection.
Input URL	Uses the prefix "$-u$" and has no default value. This string input defines the URL of the website we want to obtain an original response from.

Table 6.2: Flag input explanation

6.4.2. Host Addition

The tool has a previously loaded list of known CDN Classless Inter-Domain Routing (CIDR) [1], which will be used to discard IP addresses belonging to those subnets, thus reducing the number of unwanted hosts collected.

Therefore, the module will read the provided hostname, parse it as an IP address if possible, and either add it to the existing hosts collection dictionary or discard it and finish the function execution.

6.4.3. External Host File Parsing

Considering the user entered a flag to load hostnames from an external file path, this module will attempt to open the file provided, and read it line by line. Each line will be considered as a hostname and thus directed to the host addition module.

6.4.4. Loading original response

With the URL previously by the user, the tool will craft a request to the target hostname. It utilizes headers which attempt to match real-like browser headers, in order to increase the chances of obtaining a successful response. Moreover, if the fingerprint flag has been provided, it will spoof Chrome's TLS fingerprint, thus making the request crafted seemingly more legitimate.

Once the request has been crafted, it is executed and the response obtained is stored onto a new variable, if the request fails to execute correctly, the tool will terminate its execution and exit with an unsuccessful status code.

6.4.5. Leveraging Censys API for host discovery

Censys is a business which provides internet asset discovery as well as attack surface management solutions. They provide a useful API which allows for distinct types of queries, ranging from SSL certificate searches to IP address discovery. Our tool leverages two main API queries, certificate searching and host name searching.

Figure 6.2 depicts the API request URL needed in order to make a search on certificates, as well as it portrays the request body required.

The module uses the query *parsed.names: hostname AND tags.raw: trusted AND NOT parsed.names: cloudflaressl.com*, where *hostname* represents the hostname of the original URL provided. It is important to discuss the two different filters being applied. The first filter *tags.raw: trusted* ensures only certificates from trusted certificate authorities will be collected. The second filter, *AND NOT parsed.names: cloudflaressl.com* ensures SSL certificates belonging to Cloudflare, a popular CDN provider are not included. These filters limit the number of results, while maintaining relevant results.

Moreover, we only want to obtain the SHA 256 fingerprint generated from every SSL certificate collected, thus we will select as fields *parsed.names* and *parsed.fingerprint_sha256*.

These requests are repeated, with an increasing order number of the *page* parameter, yielding all the available SSL certificates for a specified hostname.

Having collected the fingerprints for all potential SSL certificates which match the hostname provided, the tool leverages the host search functionality of the API. It will

[1] CIDR is a way of organizing IP addresses which are allocated on the same range

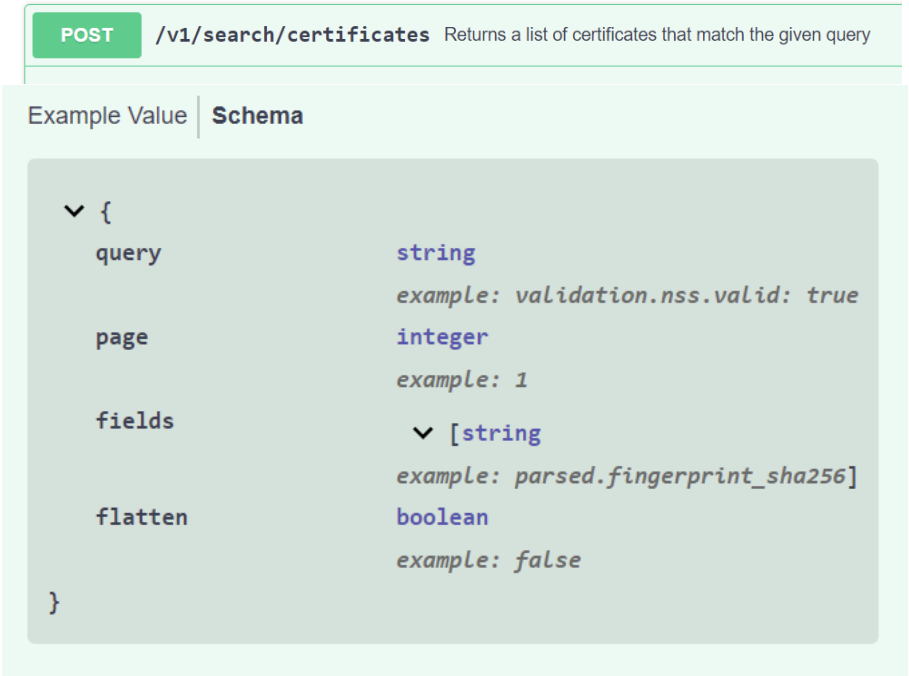


Figure 6.2: Censys API search certificate URL and schema

perform requests for every fingerprint obtained and store the hostnames which share the same SSL certificate to the original hostname. Once the execution for every fingerprint has finished, all the hosts are then shared and added to the hosts collection dictionary, using the host addition module.

6.4.6. Leveraging Security Trails API for host discovery

Security Trails is a company which provides data security, threat hunting and attack surface management tools. Alike Censys, they expose an API for everyone to use, which amongst other things, provides historical DNS record searching.

As we previously explained in section 5.2, it is of our interest to obtain historical DNS records for A, MX and TXT resource records. Figure 6.3 displays the schema for the API query.

This module will perform searches for every resource record for the original host name provided. It will then parse every host and IP address collected from the respective responses. Once full module execution has concluded, every hostname is added onto the hosts collection dictionary.

6.4.7. Performing requests to hostnames gathered

Having inserted hosts obtained from different sources into the hosts collection dictionary, we may assert the enumeration and reconnaissance sections of the tool have concluded. It is relevant to finish analyzing them one by one, and thus attempting to obtain the same content from the original host name provided.

Workers previously initialized will be delivered host names at an arbitrary order, thus parallelizing their execution. Each one of them will perform four different requests, as we previously detailed on the section 6.1. The main difference between these requests and a regular request, is that the *Host* header of the request will be spoofed and replaced

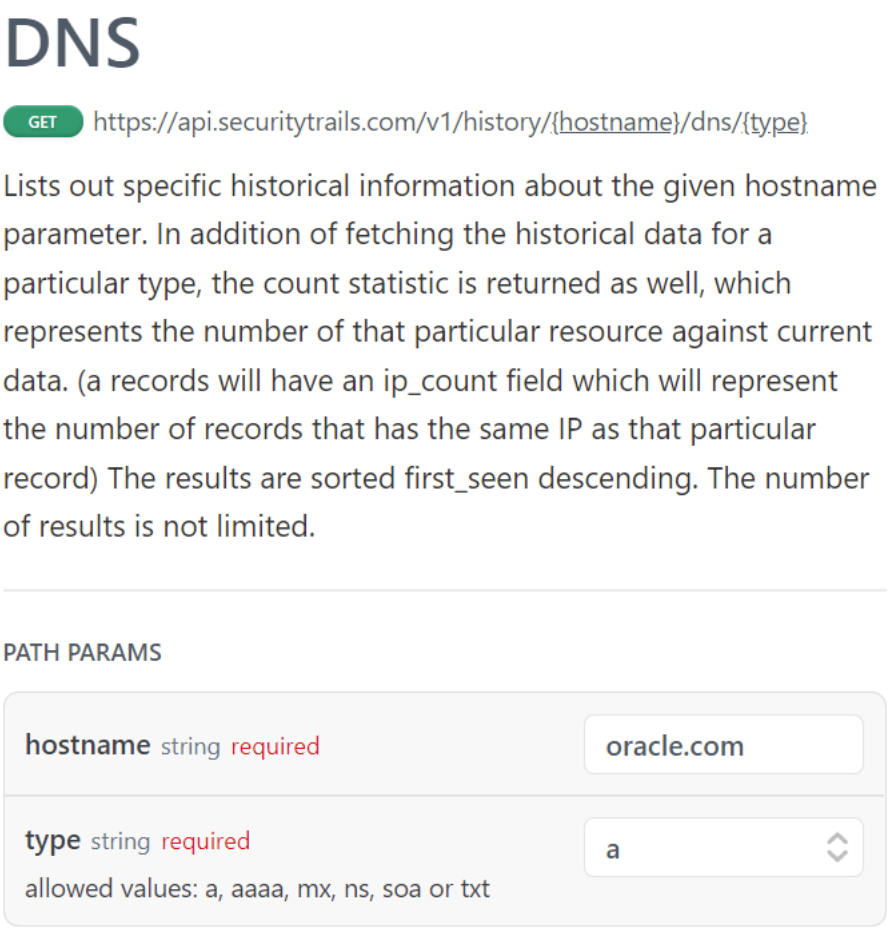


Figure 6.3: Security Trails API DNS historical record search schema

by the hostname obtained from the original URL provided. This is a key aspect of this module, as in many cases this makes the difference between obtaining direct origin server responses or not doing so. Each response will be analyzed individually according to the Levenshtein distance or content distance when preferred and the similarity result obtained will be printed on the standard output.

6.5 Technology used

As concurrency is a strong advantage for the proposed solution, the tool was coded in Golang. Golang is becoming an increasingly popular programming language due to its extremely good performance in concurrency. Moreover, it natively supports low level network handling, thus making it significantly easier to perform more advanced requests which will try to emulate a real browser. Furthermore, it is a language which I consider to be relatively easy to learn and there is a considerable number of open-source packages and libraries which may be used. The tool was coded using GoLand, which in my opinion is an excellent Integrated Development Environment (IDE) with many useful features such as syntax highlighting and debugging.

Two main external libraries were used for the tool. I used a library which allowed accurate Chrome's TLS fingerprinting, which may be used to spoof browser requests precisely. Another external package is used to calculate Levenshtein's distance, which offers an adequate performance. The main reason to use libraries for these tasks and not a manual implementation for this feature was for time efficiency, as it would not influence the result.

Two main external libraries were used for the tool. I used a library which allowed accurate Chrome's TLS fingerprinting, which may be used to spoof browser requests precisely. Another external package is used to calculate Levenshtein's distance, which offers an adequate performance. The main reason to use libraries for these tasks and to not develop a manual implementation for them was for time efficiency, as it would not influence the final tool had I developed them myself.

Furthermore, we selected both Censys and Security Trails API services as they are simple to use, easy to learn and offer a small free plan which is an attractive point for other people to make use of the tool. Moreover, they provide information on what our tool requires, and they have extensive databases which deliver a considerable number of results. Nevertheless, there are many other existing API providers which offer similar services, so a similar tool can be developed with different providers.

Another advantage about this tool is that it can be compiled, built, and executed on Linux, Windows and Mac operating systems, thus offering full operating system interoperability. This is especially relevant for the cybersecurity sector, as Linux is a popular operating system to use for research, and some languages such as C# are harder to be built and compiled on Linux OS.

CHAPTER 7
Development of proposed solution

Initially, we started by developing flag input validation. This was a trivial task as simply checking if a value was empty or not would be enough to decide if that flag was correctly entered. Furthermore, Golang's native library for flag handling, offers support for in-built data types such as integers, strings and Boolean, thus making this task even easier.

Then simple modules such as host file parsing and host addition were implemented onto the tool. They are not complex, as they perform basic read and write operations, and use simple programming objects. However, after performing several executions of the tool once it was finished, we noticed the number of hosts collected was too extensive. Therefore, we decided to store a simple list of subnets which will be blacklisted as they belong to CDN providers, which we are not interested in targeting.

Continuing with module programming, we coded the module to perform a HTTP request to the original URL provided. This module would consider if the fingerprint flag was enabled, and would decide based on a simple conditional, how the request was going to be crafted. As there were many examples of HTTP requests on the Internet for Golang, this task was not hard to perform.

Moving on, we can discuss what we consider to be the core of the tool, which is enumeration and reconnaissance performed leveraging our selected API providers. The extensive documentation from both providers, facilitated programming the reconnaissance modules. However, it is important to note that they return API responses with paging. Therefore, we had to parse the last available page and enclose the API requests on a loop until all the available results were collected.

```
1  // GetAllPagesHistoricalDNS
2  // Performs historical DNS records search for every resource record (A, MX, TXT
       ).
3  // Moreover, it will read page by page every response obtained, until results
4  // from the last page are obtained.
5  func GetAllPagesHistoricalDNS(client *nhttp.Client, domain, apiKey string)
       error {
6    // Iterates through a previously defined list of resource records, stored in
         the variable lookupTypes
7    for _, lookupType := range lookupTypes {
8      response, err := getResponse(client, apiKey, fmt.Sprintf("%s/history/%s/dns
         /%s", secTrailsApiHost, domain, lookupType))
9      if err != nil {
10       return err
11     }
12     var secTrailsResp = SecurityTrailsResponse{}
13     // Parses the response obtained from the API.
14     err = json.Unmarshal(response, &secTrailsResp)
15     if err != nil {
16       return err
```

```
17      }
18      // Obtains the maximum page from the parsed response object.
19      maxPage := secTrailsResp.Pages
20      // Adds the obtained hosts to the host dictionary.
21      addSecTrailsHosts(&secTrailsResp, lookupType)
22      // Iterates through every page until the maximum page.
23      for i := 2; i <= maxPage; i++ {
24        response, err = getResponse(client, apiKey, fmt.Sprintf("%s/history/%s/
              dns/%s/?page=%s", secTrailsApiHost, domain, lookupType, strconv.Itoa(
              i)))
25        if err != nil {
26          return err
27        }
28        err = json.Unmarshal(response, &secTrailsResp)
29        if err != nil {
30          return err
31        }
32        addSecTrailsHosts(&secTrailsResp, lookupType)
33      }
34    }
35    return nil
36  }
```

Above is the code for the function responsible for collecting hostnames from the historical DNS records provided by Security Trails. As it can be observed, it iterates through the total number of pages available, by using the variable *maxPage*.

After, I programmed the module responsible for performing spoofed requests to the new hostnames discovered. This module was similar to the one which is used to obtain the original response from the user provided URL, except for one minor adjustment which was modifying the *Host* header's value. Once this was finished, we implemented a simple conditional which would compare the content length of the obtained response in order to decide whether to use Levenshtein's distance or absolute content distance as metrics to define the similarity between responses. This similarity result would include the response HTTP status code, the Levenshtein distance or absolute content distance and whether there was a close enough similarity to be considered as a match.

Finally, we leveraged Golang's native sync WaitGroups in order to provide concurrency to the tool. The tool initializes a pool of workers and uses three different channels to allow communication between them and the main process. On one channel, hosts from the hosts collection dictionary are loaded and thus read by the worker pool, which arbitrarily assigns hostnames to the available workers. The worker uses another channel to push the similarity result to the main process and display it onto the standard output. The last channel is used by the workers to indicate when their job is done, and when all workers have finished their execution, the tool terminates its execution with a successful status code.

7.1 Implementation

As this tool is a self-contained application, its implementation can be briefly described as performing a successful execution when using it. Building and compiling an application with Go is a relatively simple task, as it only requires for a simple *go build* command call. Figure 7.1 depicts the building process of the application, which generates a new executable *Origin_Finder.exe*.

In Figure 7.2 we may observe a sample execution session for the tool. As the figure depicts, every successful response from the hostnames discovered shows its HTTP re-

```
C:\Users\alexv\GolandProjects\Origin Finder>go build .

C:\Users\alexv\GolandProjects\Origin Finder>
```

Figure 7.1: Building the tool

```
C:\Users\alexv\GolandProjects\Origin Finder>Origin_Finder.exe -censys-api-id
[redacted] -censys-api-secret [redacted]
[redacted] -sec-api-key [redacted] -u https://en.za
lando.de/robots.txt
2022/06/22 23:23:05 [+] Loaded original response with status code: 200
2022/06/22 23:23:06 [+] Loaded 19 SSL certificates from Censys
2022/06/22 23:23:06 [+] Found 6 potential hosts from given SSL certificates
2022/06/22 23:23:06 [+] Loaded hosts from Censys
2022/06/22 23:23:13 [+] Loaded hosts from Security Trails
NOMATCH http://3.70.164.185 Status 308 Levenshtein Distance 264
NOMATCH http://3.124.110.89 Status 308 Levenshtein Distance 264
NOMATCH http://3.71.164.216 Status 308 Levenshtein Distance 264
NOMATCH http://3.70.164.185 Status 308 Levenshtein Distance 264
NOMATCH http://3.71.164.216 Status 308 Levenshtein Distance 264
NOMATCH http://3.124.110.89 Status 308 Levenshtein Distance 264
MATCH https://3.71.164.216 Status 200 Levenshtein Distance 0
MATCH https://3.70.164.185 Status 200 Levenshtein Distance 0
MATCH https://3.124.110.89 Status 200 Levenshtein Distance 0
NOMATCH https://3.64.206.70 Status 403 Levenshtein Distance 233
NOMATCH https://3.69.227.47 Status 403 Levenshtein Distance 233
NOMATCH https://3.125.96.115 Status 403 Levenshtein Distance 233
```

Figure 7.2: Sample execution session on https://en.zalando.de/robots.txt

sponse status code, and its similarity result. In this specific execution we may observe we have actually found three potential IP addresses which could provide direct origin server access to the content from the target URL provided.

CHAPTER 8
Testing

Testing for the tool can be subdivided into input testing and module testing.

8.1 Input Testing

As the user may only input some specific flags and they are all validated, simple tests were performed in order to verify that the input flags were not empty.

Parameter	Mandatory
Threshold	No
Workers	No
Censys API ID	Yes
Censys API Secret	Yes
Security Trails API Key	Yes
Host File	No
Output File	No
Fingerprint	No
Input URL	Yes

Table 8.1: Overview of flag parameters which are mandatory

In table 8.1 we can observe an overview of the flag parameters which are mandatory and thus need a non-empty value to be introduced. Tests were performed on each parameter, ensuring every mandatory parameter must be filled in, in order to proceed with program execution and those optional parameters can be omitted.

8.2 Module Testing

Module testing comprises of verifying previously discussed modules operate as they should. As Go does not have try catch clauses, but it relies on handling errors individually, the number of unhandled errors or exception is minimal. Execution tests have been performed to ensure the intended behavior for each module appearing in table 8.2 is guaranteed.

Module	Intended Behaviour
Host Addition	The specified input hostname to the module is added if applicable, to the hosts collection dictionary.
External Host File Parsing	The specified input file path, if has the right read access permissions, will be read line by line and extract all its hostnames.
Loading Original Response	Given a URL, this module will craft and perform a request, which if successful will store its response content.
Censys API handling	Given valid Censys credentials, this module will load every SSL certificate fingerprint available, and its respective hosts.
Security Trails API handling	Given valid Security Trails credentials, this module will load every historical DNS record for the specified resource records defined and extract hostnames from them.
Loading Response On New Hosts	Given the original hostname and a new hostname, this module will craft a request to the new hostname spoofing its *Host* header as if the request were intended to the original hostname. Moreover, this module shall successfully compare the content responses between both and obtain a conclusive similarity result.

Table 8.2: Overview of modules and their intended behaviours

CHAPTER 9

Case Study: Accessing Zalando through their origin server

Zalando, an online sneaker and clothing retailer with more than 48 million active customers across 23 markets [7] uses Akamai to deliver their content to end users. Akamai, as we have previously examined offers bot management solutions and a powerful web application firewall which mitigates potential threats.

We may assume that Zalando's intended content delivery process to end users is that it should be handled through Akamai CDNs and not through their origin servers directly.

As it may be observed from the response status code 403 on Figure 9.1, the CDN has successfully blocked this basic Structured Query Language (SQL) injection attack. Moreover, we can observe from the Server header from the response, AkamaiNetStorage, that the CDN provider being used by Zalando is Akamai. Its web application firewall detects and blocks potentially threatful payloads to protect the content owner's server. This generally poses a problem to security researchers and pentesters as they will have to craft more thoughtful payloads to successfully execute an attack on targets behind a CDN with a web application firewall. Moreover, since they usually implement admission control, it makes it significantly harder for hackers to perform a successful Denial Of Service (DOS) on their origin server. Let us observe what happens when we make this same request through one of the origin servers behind these edge servers.

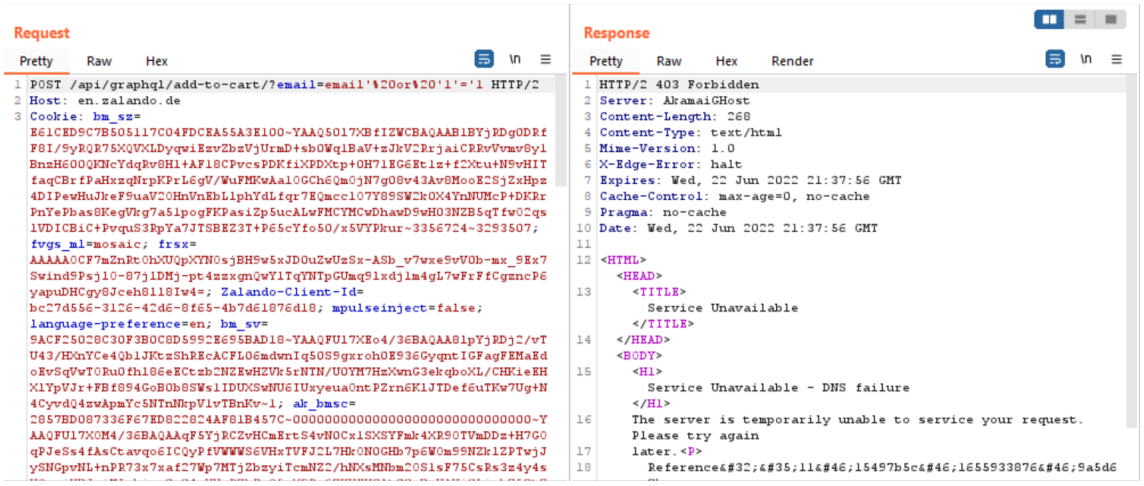


Figure 9.1: Basic SQL injection attack on the target through Akamai's CDN

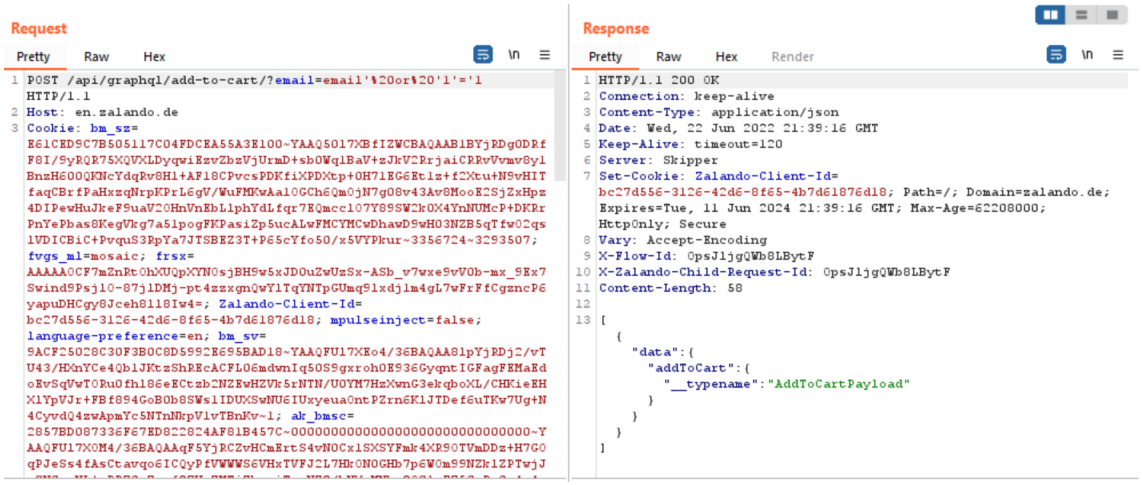


Figure 9.2: Basic SQL injection attack on the target through one of its origin servers

Judging from Figure 9.2 we can determine the attack has been successfully executed, and as it can be observed from the Server response header, we have performed this request directly passing through Akamai's edge servers. Considering performing simple SQL injections attack such as *' or '1'='1* is allowed on the target, opens the range of possible attacks which may be executed by hackers. Furthermore, since the content delivery from the target is no longer distributed across hundreds or thousands of edge servers, performing a DoS attack on it becomes a trivial task.

This case study further reinforces the point that content owners should at the very least implement their own firewall on the origin server, as even large and recognized companies fall into the trap of assuming they have a secure content delivery system in place just because they are using a CDN with WAF.

This is usually not considered as a vulnerability by many companies, but it can be considered as a major server misconfiguration. It may pose a serious security threat as it certainly facilitates new vulnerabilities to be found and exploited and performing DDoS or DoS attacks may be considerably easier. As per Figure 7.2, these IP addresses and direct origin server accesses have been found using the tool developed in this thesis, which demonstrates it's real-life utility in finding real origin servers from target websites.

CHAPTER 10
Protecting your origin server

In the scenario that we may have found out our origin server IP address has been exposed to the public, we might need to know how to protect it, to prevent unwanted users to access it directly. We may consider that a server misconfiguration is happening if the content owner does not intend to expose its content directly through its origin server. Below are explained some of the most common ways for protecting an origin server IP address, and thus correctly configuring the origin server.

10.1 Preventing external connections

Some CDN providers such as Cloudflare, propose content owners to exclusively whitelist traffic from IP addresses belonging to Cloudflare IP addresses or the IP addresses of trusted partners, vendors or applications [1].

Another way of preventing external connections is to use a Tunnel such as Cloudflare Tunnel which will encrypt all traffic between the origin server and prevent inbound connections.

Preventing external connections through any of the above methods is generally considered to be the best practice, no traffic can possibly come from unwanted IP addresses if servers are correctly configured.

10.2 Changing IP address

Another potential protection method to consider changing the IP address of the origin server. This method should probably be combined with some other protection methods, as it may still be possible to discover the new IP address and obtain direct origin access through it.

10.3 Avoiding generic subdomain names

We have previously seen tools such as AMASS which help enumerate subdomains from a provided domain. One of the methods it may use is a brute-force search from a predefined list of keywords. If generic subdomain names are avoided, it may be harder for discover some subdomains via brute-force searching.

10.4 Avoid leaving a trace in DNS records

As historical DNS records may be exploited to obtain new hostnames, it is important to avoid leaving trace in them. Migrating your email service to a different server may help reduce the information which can be obtained from MX records for example. It is generally recommended to check your current DNS records and verify we are not exposing more information than required.

10.5 Reducing sensitive data

Publicly accessible system and server logs on a web server (e.g., phpinfo), may also expose sensitive information from them, such as their IP address. It is recommended to check the services running on a website and require authorization for paths and endpoints that expose sensitive information.

CHAPTER 11
Conclusions

This thesis aimed to provide further insight onto origin server discovery and elucidate its importance nowadays. Therefore, protection mechanisms to avoid exposing an origin server were outlined. An automated tool was specifically developed for origin server discovery, which combined several passive reconnaissance methods. Based on the real-world obtained results from the tool, we may conclude it has successfully accomplished its intended objective. Nevertheless, it is important to note the developed tool still has room for significant improvements, as further reconnaissance methods may be implemented onto it.

While comprehensively understanding every topic discussed in this thesis was initially a difficult task, it has greatly improved my overall knowledge on ethical hacking and web security. It is worth mentioning that topics such as SSL certificate identification or CDN infrastructures posed a remarkable challenge to be effectively understood. Due to its high-performance, Golang was learnt to develop the automated tool mentioned in this thesis. This serves as a boost both for personal and professional experience, as it is a programming language which is becoming to be more on-demand.

11.1 Relating work developed to studies coursed

First of all, it is important to mention that an important part of my knowledge with networks and application software comes from working professionally as a freelance software developer for almost three years on these topics. Nevertheless, the work developed on this thesis is also closely related to the contents of several subjects I have taken along the degree.

Starting off, subject *Redes de computadores* has greatly helped with the work developed in this thesis. In this subject we studied how network traffic is handled, under the different layers available. With the knowledge obtained from this subject, I better understood how things such as DNS operate, as well as helping to understand how HTTP requests are handled internally. Furthermore, another topic introduced on this subject on the application layer unit, was about REST APIs. They are briefly explained and it helped me get introduced into them in order to understand how to handle them, as they are a key part of the work I developed.

Moreover, another closely related subject to the work developed is *Tecnología de sistemas de información en la red*, especially its practical aspect. On the practical side of the subject, we learnt to code in JavaScript, language which is very similar in many things to Golang, thus helping to learn this new language. Since my tool is developed in Golang, I

consider this subject was relevant and very helpful as it may have taken me considerably more time to understand how to code in Golang without this knowledge.

In addition, I consider *Hacking ético* a subject which made me be more motivated on cyber security and web vulnerabilities. Because of it, I discovered a completely new field of informatics I had previously never explored, but which I personally find fascinating and motivated me to investigate about the topic of the work I developed.

Finally, I consider *Aprendizaje permanente* to be a very relevant transversal competence for this work developed. Despite some of the topics discussed on this work have been previously introduced by the subjects I coursed; these topics did not go into enough depth. Therefore, by continuously reading other related work, and investigating by myself, I have been able to gain a much deeper understanding of the topics on this work: DNS, SSL/TLS certificates, CDNs, etc. Another relevant competence, which I have gained through my studies is *Conocimiento de problemas contemporáneos*, as the work developed in this thesis discusses topics and fields of study which have been scarcely explored before.

CHAPTER 12
Future Work

Given the time provided for this work, it was difficult to include more enumeration and reconnaissance methods onto this tool. Nevertheless, on future work, active reconnaissance methods can be leveraged, in order to obtain potential hostnames. We have already discussed some active recon methods, such as scanning files from the website for hostnames.

Another potential way of obtaining origin server hostnames may be by performing requests to the target server when its CDN is having network performance outages, as they have sometimes happened. In those specific scenarios the origin server hostname may be exposed directly. We did not consider this for my work as it is impractical for the most part, as many CDN providers tend to have excellent uptime statistics.

From my point of view, future work could benefit from this thesis in order to provide for more versatile and complete vulnerability scanners. As we have observed some of the implications of exposing an origin server directly, vulnerability discovery becoming a considerably easier task is an interesting take for security researchers.

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Considero que ningún ODS se relaciona con mi trabajo ya que mi trabajo es muy específico en el sector de la ciber seguridad. En menor grado, podría relacionarse con el ODS 9, puesto que no hay apenas trabajo previo de este tema específico considero que aporto a la innovación de la ciber seguridad. Es posible que gracias a este trabajo, la seguridad web en general se vea reforzada, y que ganen experiencia y conocimiento, tanto los investigadores de seguridad, como los usuarios que deseen ofrecer contenido a través de Internet.

Bibliography

[1] Cloudflare. Protect your origin servers. https://developers.cloudflare.com/fundamentals/get-started/task-guides/origin-health/free/. [Online; accessed 20-June-2022].

[2] Web Almanac Contributors. 2021 web almanac. https://almanac.httparchive.org/static/pdfs/web_almanac_2021_en.pdf, 2021. [Online; accessed 14-June-2022].

[3] Madelen Berg Dalseth. The development of a reconnaissance tool aiming to achieve a more efficient information gathering phase of a penetration test, 2021.

[4] Exorbyte. The levenshtein-algorithm. http://www.levenshtein.net/. [Online; accessed 14-June-2022].

[5] Milad Ghaznavi, Elaheh Jalalpour, Mohammad A. Salahuddin, Raouf Boutaba, Daniel Migault, and Stere Preda. Content delivery network security: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2166–2190, 2021.

[6] HolyBugx. Finding the origin ip behind cdns. https://infosecwriteups.com/finding-the-origin-ip-behind-cdns-37cd18d5275, December 2020. [Online; accessed 14-June-2022].

[7] Patrick Kofler. Zalando full year results. https://corporate.zalando.com/en/investor-relations/news-stories/zalando-full-year-results-2021, 2021. [Online; accessed 14-June-2022].

[8] AO Kaspersky Lab. What is an ssl certificate? https://www.kaspersky.com/resource-center/definitions/what-is-a-ssl-certificate. [Online; accessed 14-June-2022].

[9] Ben Laurie. Certificate transparency. *Commun. ACM*, 57(10):40–46, sep 2014.

[10] OWASP contributors. Owasp amass - users' guide. https://github.com/OWASP/Amass/blob/master/doc/user_guide.md. [Online; accessed 14-June-2022].

[11] Tutorials Point. Ethical hacking - reconnaissance. https://www.tutorialspoint.com/ethical_hacking/ethical_hacking_reconnaissance.htm. [Online; accessed 14-June-2022].

[12] RFC contributors. Rfc9110 - host and :authority. https://www.rfc-editor.org/rfc/rfc9110.html#name-host-and-authority. [Online; accessed 14-June-2022].

[13] Vijaya R Saraswathi, Iftequar Sk Ahmed, Sriveda M Reddy, S Akshay, Vrushik M Reddy, and Sanjana M Reddy. Automation of recon process for ethical hackers. In *2022 International Conference for Advancement in Technology (ICONAT)*, pages 1–6, 2022.

[14] Akamai Technologies. Bot manager. https://www.akamai.com/en/products/bot-manager, 2022. [Online; accessed 14-June-2022].

[15] Wikipedia contributors. Domain name system — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Domain_Name_System&oldid=1092781489, 2022. [Online; accessed 14-June-2022].

[16] Wikipedia contributors. Levenshtein distance — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Levenshtein_distance&oldid=1082661551, 2022. [Online; accessed 14-June-2022].

[17] Wikipedia contributors. List of dns record types — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=List_of_DNS_record_types&oldid=1090675362, 2022. [Online; accessed 14-June-2022].

[18] Wikipedia contributors. Subdomain — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Subdomain&oldid=1084975153, 2022. [Online; accessed 14-June-2022].