Universiteit Leiden
The Netherlands

Opleiding Informatica

An Exploration of C2 Server Fingerprinting Capabilities with Probed HTTP Data

Hugo Hulsebosch

Supervisors:
Dr Olga Gadyatskaya & Soufian El Yadmani

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

26/01/2024

Abstract

Command and Control (C2) servers are a crucial component in coordinating cyberattacks using malware. The timely detection of these servers can guide mitigation measures that prevent cyberattacks from having a significantly destructive and costly impact. Creating a digital fingerprint of different C2 server families can aid in mutually distinguishing C2 servers. This thesis explores the automated and unique identification and fingerprinting of web-facing C2 servers by applying machine learning algorithms to cluster these servers based on probed HTTP data. The goal is to develop a general system applicable to various C2 server types by utilising an approach that involves proactive engagement with known C2 servers to collect data, offering a distinct advantage over traditional network intercepting methodologies. This approach provides the ability to inspect the original contents of encrypted messages and detect C2 servers before the threat actor initiates an attack. While the current system requires manual validation through external services, the clustering based on the Rank-Biased Overlap (RBO) distance of HTTP response headers offers a viable way to distinguish C2 families and other web services, even when placed behind proxy services.

Acknowledgements

First and foremost, I want to thank Dr Olga Gadyatskaya and Soufian El Yadmani for their support and supervision during this thesis. Furthermore, I want to thank my family and friends for their support. Last but not least, I want to thank the members of the thesis group, Hanna Straathof, Raaf van Beusekom, and Sander Honig, led by Dr Anna van der Meulen, for their support and advice on Monday mornings.

I've put in a lot of effort to ensure that this thesis is easy to read for both audible documents and screen readers. As someone with dyslexia, I understand the importance of digital accessibility. However, there might be instances where screen reader software inaccurately read aloud certain parts of this document.

Contents

1 Introduction

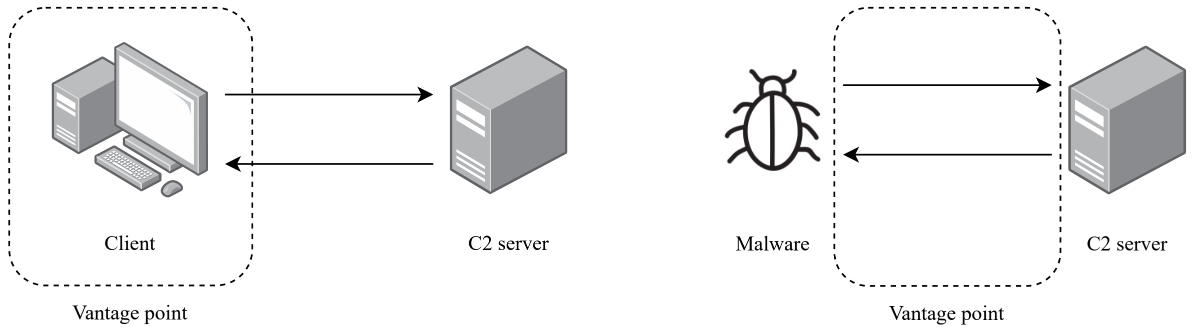
Cybersecurity is an increasingly vital practice due to the widespread use and embedding of computers across numerous parts of our society. Its significance has only grown since we started interconnecting computers to form the internet and the public release of the World Wide Web in the early 90s of the 20th century. Moreover, the need for robust cybersecurity is further underscored by the rising ownership of smartphones, devices integral to sensitive activities like online banking and equipped with numerous built-in sensors able to record sensitive information. These devices can collect personal information, which can be transmitted over high-speed cellular and home internet connections.

However, one of the downsides of digitising society is that threat actors (such as cybercriminals) can use digitisation to their advantage to perform harmful activities, which may involve creating and distributing malware. Malware, short for malicious software, refers to software specifically designed to harm or exploit computer systems, networks, and user devices. The goal of malware is to wreak mayhem, either by stealing information and resources for financial gain or intending to sabotage[1]. IBM reported in 2023 that the global average cost of a single data breach was 4.45 million USD, which is an increase of 15% in three years[2]. Even though many people think that skilled hackers create their own malicious software, cybercriminals focused on making money prefer to take an easier route. They seek stability, simplicity, support, and a guarantee, so instead of constructing their own malware, they buy pre-made packages [VG20] or misuse cracked versions of adversary simulation tools, such as Cobalt Strike[3].

As threat actors succeed in infecting devices with their malware, they establish communication channels between the infected device and a central Command and Control (C2) server, coordinating the actions of the malware residing on the infected device. To avoid detection and by blending in with regular internet traffic, threat actors may use communication channels operating on protocols associated with web traffic[4], file transfer[5], mail delivery[6], and resolving domain names to IP addresses[7]. Detecting and identifying C2 servers and their communication channels aids in preventing cyberattacks from having a significantly destructive impact, as early detection can guide mitigation measures that prevent the successful execution. The practice of creating a digital fingerprint of different C2 server families can aid in mutually distinguishing C2 servers from each other and other benign web applications.

Goal of the thesis: This thesis aims to gain insights into the possibility of automatically and uniquely distinguishing and fingerprinting C2 servers by applying machine learning algorithms to cluster C2 servers based on probed (HTTP) data. Additionally, we aspire to construct a preliminary system for automatically fingerprinting C2 servers, a general system not focused on a single C2 server

[1]https://www.cisco.com/site/us/en/learn/topics/security/what-is-malware.html
[2]https://www.ibm.com/reports/data-breach
[3]https://www.cobaltstrike.com/
[4]https://attack.mitre.org/techniques/T1071/001/
[5]https://attack.mitre.org/techniques/T1071/002/
[6]https://attack.mitre.org/techniques/T1071/003/
[7]https://attack.mitre.org/techniques/T1071/004/



(a) Client vantage point.

(b) Network vantage point.

Figure 1: Different vantage points characterise two distinct approaches for collecting C2 data. Figure 1a shows our data collection approach through an active connection to the C2 server. Contrarily, Figure 1b illustrates data collection through the interception of network traffic between the malicious software within the network and the remote C2 server.

type. Ultimately, it is utilised by security researchers and companies to detect C2 infrastructure on new domain names or IP addresses or to keep track of the landscape and arising C2 types.

Our approach proactively engages with known C2 servers to gather data about distinct types of C2 servers, a practice that can be exerted from any machine connected to the internet. This distinguishing vantage point, depicted in Figure 1a, sets our approach apart from most existing C2 detection methodologies, which typically monitor network traffic patterns from within a network, focusing on the communication generated by malware from inside the network to remote hosts, as depicted in Figure 1b. Network monitoring can fall short as it cannot inspect the original contents of encrypted messages. As our approach involves active engagement with C2 servers through a client-server connection, it can gather data even when exchanged through encrypted message traffic, specifically when the C2 server utilises the HTTPS protocol. Moreover, our approach eliminates the need for malware to be active within a network, allowing for the detection of C2 servers before the threat actor has started its attack. Finally, there is no requirement to log all traffic within a network.

1.1 Thesis Overview

This thesis contains seven chapters, including this introduction, and is structured as follows. Chapter 2 contains foundational background information essential for understanding the topic. In Chapter 3, a presentation of prior related work is given. The workings of our approach are in Chapter 4 through a methodology presentation. Chapter 5 delves into case studies focusing on distinguished malware families. Furthermore, we discuss the results of the case studies in Chapter 6, in which the limitations of our system are also critically illustrated. Finally, Chapter 7 encapsulates our findings, provides conclusive insights, and proposes further research.

2 Background

This section presents insights into the topics discussed throughout the thesis by delving into relevant subjects requiring a more detailed explanation. Section 2.1 presents the topics of botnets and C2, followed by the explanation of a C2 communication channel, namely HTTP, in Section 2.2. Section 2.3 clarifies the notion of fingerprinting.

2.1 Botnets and Infrastructure

A computer system (host) can be infected with malware by a threat actor to gain remote unauthorised control. Some malware, such as the Stuxnet worm, used to sabotage Iranian nuclear power facilities [BR17], has pre-programmed behaviour and executes commands sequentially or based on events happening on the infected host. This category includes malware with built-in behaviour that dictates its action without remote input. In practice, most types of malware programs do not have pre-programmed behaviour and rely on remote instructions from the threat actor responsible for distributing the malware[8].

Threat actors can infect a single host with malware for their exploitive operation or a network of multiple hosts. A network of infected hosts controlled by the same threat actor is called a botnet (robot network). Threat actors can exploit a single infected host or botnets consisting of an arbitrary number of infected hosts to do malicious tasks, such as performing a Distributed Denial of Service (DDoS) attack, spreading spam, stealing information from the infected host, to move further in a (local) network, and the ever-more popular activity of encrypting the infected host's data to obtain a ransom in order to unencrypt the data.

2.1.1 Command & Control

Whether the threat actor has control over a single infected host or a botnet consisting of multiple hosts, a way of communicating with the threat actor is needed to execute tasks by malware residing on an infected host. One way of establishing this is through a Command and Control[9] (C2) server, which can be set up on a machine owned or rented by the threat actor or deployed on a compromised benign server. The malicious program (beacon) on an infected host contacts the C2 server, allowing the threat actor to see how many and which machines in its botnet are online. Through the C2 server, the threat actor can send commands or instructions to its infected hosts, which then adapt their behaviour based on the response or instructions it receives.

2.2 Hypertext Transfer Protocol

As previously stated, our focus lies on C2 servers using the Hypertext Transfer Protocol (HTTP) to interact with infected hosts. The HTTP protocol is a popular choice by threat actors in facilitating communication between malware and C2 servers, as the protocol hides the traffic among benign web traffic [ZM09]. It bypasses port-based filtering firewalls, as blocking the ports used for HTTP would also make normal web usage impossible. The communication between a malware program and a C2

[8]https://bullguardreview.com/does-malware-work-without-internet/
[9]https://attack.mitre.org/versions/v14/tactics/TA0011/

```
GET / HTTP/1.1
Accept: text/html,application/xhtml+xml,
  application/xml;q=0.9,image/avif,
  image/webp,image/apng,*/*;q=0.8,
  application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9,nl;q=0.8
Cache-Control: max-age=0
Connection: keep-alive
DNT: 1
Host: liacs.leidenuniv.nl
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel
  Mac OS X 10_15_7) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/120.0.0.0
  Safari/537.36
```

(a) HTTP request.

```
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Mon, 15 Jan 2024 14:48:01 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Strict-Transport-Security: max-age=604800;
  preload

<!DOCTYPE html>
<html lang="en" data-version="1.165.00" >
<head>
...
```

(b) HTTP response.

Figure 2: An example HTTP request (Figure 2a) and response (Figure 2b) pair respectively sent to and received from a Leiden University server to obtain the LIACS homepage, `https://liacs.leidenuniv.nl/`. The HTTP response body in Figure 2b is cut short to fit on this page and only shows the first two lines of the 1352 lines of HTML code sent from the server.

server is akin to conventional web server communication. In this section, we briefly summarise the workings of HTTP.

The HTTP communication protocol is the foundation for the World Wide Web. It introduced a standardised way of communication between client and server by defining contracts such as the expected sequence of messages, the request and response structure, and conventions about handling various scenarios. [BLFF96, FGM$^+$99, BPT15]. As specified in the protocol, the client sends an HTTP request to the server in which the client specifies what resource it needs from the server or what functions it wants to perform. The server processes the request and sends an HTTP response back to the client. The client in this context can be any software program that initiates communication with a server. E.g., web browsers like Mozilla Firefox and Google Chrome are widely recognized examples of clients, but it is important to note that any program that establishes a connection with a server can be considered a client. Figure 2 shows an example of an HTTP request sent by the client (Figure 2a) and the corresponding HTTP response sent back by the web server to the client (Figure 2b). We explain these separately in the following sections.

2.2.1 HTTP Request

The HTTP request consists of three main components: the request line, the HTTP request headers, and, optionally, the request body. Figure 2a depicts an example of an HTTP request, in which we see that the first line is the request line, consisting of the HTTP request method, a Uniform Resource Locator (URL), and the HTTP version used. A single HTTP request method is set to define what action the client requests the server to perform, e.g., a GET method indicates the client wants to retrieve a resource from the server and a POST method to send data, such as the contents of a web form, to the server. The URL specifies the resource the client is referring to and consists, among other things, of the server's address and the resource's path on the server [BLMM94].

The subsequent lines of the HTTP request in Figure 2a are devoted to the HTTP request headers. The HTTP request headers list one or more key-value pairs in which the client can specify additional information about the request, such as the languages the client prefers to receive the response or the user agent is used to make the request, `Accept-Language` and `User-Agent` in Figure 2a, respectively. Finally, the HTTP request body is optional and combined with some HTTP request methods for the client to send data to the server, such as with a POST request. In Figure 2a, no request body is specified. However, if a request body were present, it would be positioned after the final request header entry and separated by a blank line.

2.2.2 HTTP Response

The HTTP response consists of three main components: a status line, the HTTP response headers, and the response body. The status line is akin to the request line of the request on the first line, which can be seen in the example response of Figure 2b. It has two parts: the HTTP version used and the response status code given by the server as a response to the client's request. The latter is divided into five classes, indicating whether the request was successful or not and if the server could execute it. In our example, the web server responds with a `200 OK`, meaning "*the resource has been fetched and transmitted in the message body*"[10]. Equivalent to the HTTP request headers, the HTTP response headers are a list of zero or more key-value pairs set by the server which give additional information about the response. An example of a response header is `Content-type: image/jpg`, which tells the client that the data sent through the response body should be interpreted as a JPG image. Lastly, a blank line separates the response body from the response headers. It contains the data or resources the client requests, such as an HTML page in our example in Figure 2b.

2.2.3 Implementational Agnostic

HTTP defines a set of rules for communication between web servers and clients, which allows interoperability but is agnostic to its implementation in software. Therefore, it can happen that different web server software families or versions may respond differently to an identical request. Moreover, sometimes different web server implementations divert from the HTTP specifications. For example, Meijian Li et al. discovered a deviation in the responses generated by three well-known HTTP server software families, Apache, Mini-Web, and Tomcat, when subjected to identical input conditions [LWX13]. We intend to use this known aspect in our approach to automatically distinguish C2 servers from each other and other web services. The variation shown by web servers

[10]https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

could enable us to identify the software a web server uses for fingerprinting by finding known instances that exhibit the same variations.

2.3 Fingerprinting

Digital fingerprinting is a technique for uniquely identifying and classifying digital entities. In this thesis, we utilise the concepts of this technique in our attempt to distinguish the C2 servers of different malware families. Like a human fingerprint, where the ridges of a finger can be used to uniquely identify a person, digital fingerprints (or sometimes footprints) can be used to identify software, machines, or digital entities. In cyberattacks and security, fingerprinting practices are conducted by adversaries to construct and carry out advanced attacks and by security professionals in their efforts to mitigate attacks. Adversaries can implement techniques in their malware to fingerprint the targeted system, which is, for example, used on first contact to determine whether a machine is vulnerable to certain kinds of attacks or to tailor a subsequent attack specific to the device. Security professionals can use fingerprinting techniques to identify malicious software and devices connected to the internet.

2.3.1 Extracting Characteristics

Fingerprinting is done by extracting critical characteristics of the target. When fingerprinting a device, the characteristics extracted can be amongst the operating system used, installed software and their respective versions, and configuration properties. To detect (malicious) software, one can use the program's binary code, known as static analysis. However, the software creator can circumvent this by simply changing the source code and recompiling the software, resulting in a completely different executable binary. This change will result in a different fingerprint signature for the recompiled software, making it impossible to connect the new version to the unchanged version based on strictly comparing the two binaries. Therefore, it is more promising to use characteristics that cannot be changed easily or are intrinsic to the workings of the software (dynamical analysis), such as behavioural patterns [SS15, Ala15]. The characteristics gathered to fingerprint the digital entity are fed to a non-cryptographic hashing algorithm, which maps the entity to a unique string called a digital signature.

3 Related Work

This section delves into the prior studies and works on fingerprinting hosts and devices using HTTP and C2 infrastructure detection methods.

Back in 2004, Shah wrote an introduction to HTTP host fingerprinting, a technique used to identify different types of HTTP servers based on their implementation differences in the HTTP protocol [Sha04]. In it, he explained simple techniques such as banner grabbing, which "*is to look at the Server field in the HTTP response header*" [Sha04] and introduced more advanced techniques that involve constructing a fingerprint by recording behavioural patterns exhibited by the server and comparing this fingerprint to known fingerprints stored in a database. Ruef contributed by developing the open-source 'httprecon' application in late 2007 [Rue07]. Like Shah's work, this tool is designed to fingerprint web servers by sending multiple HTTP requests and recording the following responses (recording behavioural patterns). The method uses a database with server-specific HTTP header pairs. Comparing these pairs to the probe response generates a similarity score, emphasising the order of response headers to identify the most probable web server software. We used Shah and Ruef's work as inspiration for our methodology in collecting HTTP response header data to distinguish servers. Their work and ours overlap as we share the same client vantage point, as illustrated in Figure 1a. Our work differs, however, as it only sends requests to a single path. Moreover, their work does not explicitly focus on C2 servers but instead on web servers in general.

Lavrenovs and Visky researched the utilisation of HTTP response headers for device classification [LV19]. Their findings indicate that, in a substantial number of instances, a single HTTP response header can confirm the particular type or, in some cases, even the model of the device. While not individually conclusive, the remaining headers can be effectively combined with other features for classification purposes. Additionally, they can serve as weak indicators to filter out devices not of primary interest to the researchers. We built upon these findings and applied them in our efforts to distinguish C2 servers. Their research was, unlike ours, not focussed on C2 servers in particular but on any internet-connected device. Our research shares the same vantage point as their research, as they also proactively connect to machines on the internet (Figure 1a).

Tang et al. introduced HTTP Header Sequence-based LSH fingerprints (HSLF), proposing a hierarchical system for classifying HTTP traffic based on HTTP headers [TWL+21]. Their approach, distinct from our research focus, involves the comprehensive classification of network traffic (network vantage point, Figure 1b) without a specific emphasis on C2 server or malware detection, which is the primary interest of our investigation. Nonetheless, the paper intrigues us due to its Local-sensitive hashing (LSH) application in the hashing of HTTP data. The authors cite LSH in its ability over traditional hashing algorithms to measure the similarity of original content within the signature's dimension. Their choice of utilising SimHash [SL07] as an LSH algorithm in HSLF inspired our utilisation of the same algorithm during the feature-encoding phase in preparation for the coarse-grained clustering phase of our methodology, as can be read in Section 4.3, where it is applied to hash the headers and body of HTTP responses.

Botnet detection using machine learning algorithms has been a subject of sustained exploration. In 2016, Miller et al. contributed an investigation reporting different machine-learning techniques and their application in botnet detection [MBE16]. This study found that unsupervised learning methods are often used for malware family detection by clustering samples with similar behaviour and supervised learning for classifying individual hosts. We utilise the foremost in our research to find patterns and similarities in C2 servers by clustering on probed HTTP data.

Nakamura and Åström developed two methodologies aimed at fingerprinting and categorising hosts to identify C2 infrastructure in their 2021 master thesis [NÅ21]. Their approaches utilised a classification machine learning algorithm, random forests, and the computation of distance scores between hosts, drawing from data obtained through network scanning (Figure 1b) and active probing (Figure 1a), inspired by Shah [Sha04] and Ruef [Rue07] and akin to our vantage point. Noteworthy distinctions between their study and this research lie in including varied data sources within their machine-learning algorithm, spanning HTTP, SSH, TLS, and DNS and their goal to classify hosts. In contrast, we focus on analysing HTTP response headers and their sequential arrangement and whether we can find patterns or clusters using this characteristic.

Other research on the classification of servers or application-server communication relies, similar to HSLF, upon analysing network traffic and its contents, as illustrated earlier in Figure 1b. Perdisci et al., for instance, undertake clustering based on behavioural HTTP traffic traces, generating generic malware signatures [PLF10]. Similarly, Zand et al. introduce an approach centred on extracting frequent strings from network traffic, each assigned a score to estimate its potential as an indicator of C2 activity [ZVYK14]. Encrypted messages do, however, limit the possibilities of C2 communication detection or classification through means of analysing the network traffic when depending on the contents of the messages sent.

Gu et al. introduced in Botsniffer the notion of categorising C2 architecture in "push" and "pull" styles. They explained push as "*the botmaster issues a command in the channel, and all the bots connected to the channel can receive it in real-time*", whereas with pull "*the botmaster simply sets the command in a file ... bots frequently connect back to read the command file*" [GZL08]. In alignment with this categorisation, our work focuses explicitly on investigating C2 servers utilising a pull-style architecture, as HTTP C2 servers instruct bots under their control by responding to an HTTP request (Figure 1a).

Bilge et al. created DISCLOSURE, which uses various features such as flow sizes, client access patterns, and temporal behaviour in NetFlow records (Figure 1b), allowing it to distinguish C2 channels from benign traffic reliably [BBR$^+$12]. We drew inspiration from the DISCLOSURE work, which criticises early attempts to perform machine learning-based over NetFlow data for fixating on non-robust features, such as specific server ports. Although this thesis focuses on HTTP data, we embraced a similar philosophy by selecting features that capture invariants in malware-C2 server communication.

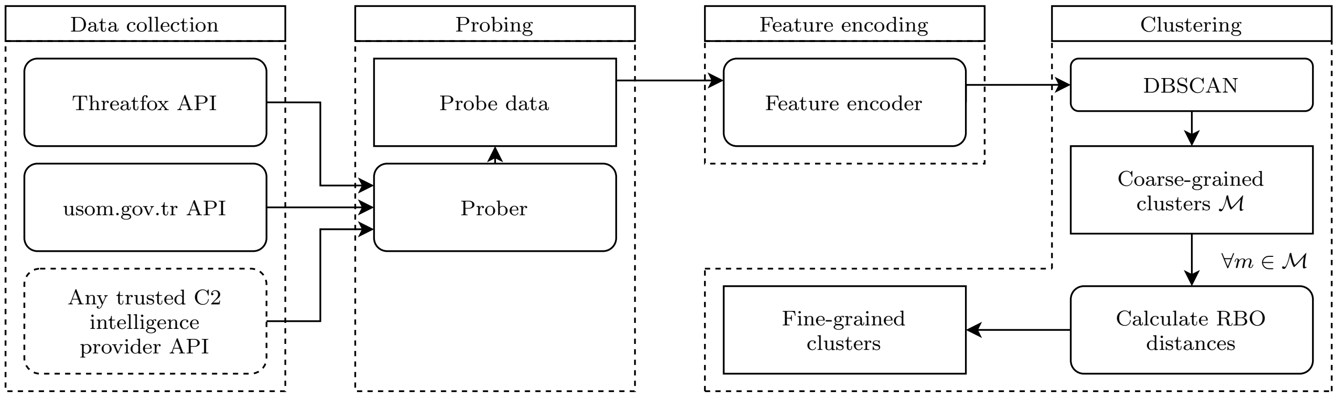


Figure 3: Overview of our proposed system to automatically fingerprint C2 server instances.

4 Methodology and Findings

This section describes our approach to automatically fingerprint C2 servers, which consists of four main phases: data collection, probing, feature encoding, and clustering, as seen in the overview in Figure 3. We obtain addresses (e.g., host/domain names or IP addresses) and other data of C2 servers known and collected by trusted intelligence providers during the data collection phase, as explained in Section 4.1. The subsequent probing phase then uses these indicators for data extraction, in which the set of unique C2 addresses is given as input and data about the characteristics of the C2 server as output. We describe this in Section 4.2. In Section 4.3, we depict the feature encoding phase, where the dataset constructed by the prober is encoded to be parsed by a machine learning algorithm. Moreover, we have divided the clustering phase (Section 4.4) into coarse-grained and fine-grained subphases. The complete data set containing all the probe data is used during coarse-grained clustering to find similarities between C2 server instances. However, these similarities need clarification and might not indicate a trustworthy relationship between the two instances. During the fine-grained clustering phase, we use the samples within a coarse-grained cluster to form more accurate fine-grained clusters in which similarities or relationships between different C2 servers should emerge. In Section 4.5, we briefly discuss the clusters that emerged using our methodology. Finally, in Chapter 5, we take a closer look at various fine-grained clusters, trying to figure out what they represent and how likely they are to point to a distinct type of C2 server.

4.1 Data Collection

To find similarities between C2 servers, we first needed a source that provided us with known C2 servers. For this, we used Threatfox[11], a platform on which security researchers share their threat intelligence, and USOM[12], a Turkish government organisation, publicly sharing cyber threat intelligence data. Notably, the selection of threat intelligence providers in the data collection phase is flexible, and the system architecture allows for integrating any threat intelligence provider equipped with an API to acquire addresses pointing to C2 servers.

[11]https://threatfox.abuse.ch/
[12]https://www.usom.gov.tr/

Threatfox hosts an open-source Indicator Of Compromise (IOC) database constructed by contributions from security researchers. Each IOC record in the database contains information such as the IOC location (e.g., a URL, domain or IP-port combination), the threat type (e.g., a botnet, malware payload), the malware family, and a confidence level expressed as a percentage. Moreover, Threatfox provides multiple API endpoints to access their data programmatically, one of which allows the retrieval of recently added IOCs.

USOM provides a publicly facing API endpoint that enables us to obtain threat intelligence about cyberattacks, (financial) fishing, and –relevant for us– malware distributions and C2 infrastructure. Moreover, their API endpoint provides functionality to query records added during a date interval, allowing them to retrieve the most recent additions. Both the Threatfox and USOM APIs are accessible over HTTPS, are configurable through URL parameters, and respond in JSON.

4.1.1 Data Filters and Constraints

From the Threatfox feed, we queried the recent IOCs labelled as threat type `botnet_cc` so that we would only retrieve C2 servers. Since this thesis relies on clustering C2 servers with similar characteristics, we only want IOCs in our dataset, for which it is most certain that it is a C2 server. Therefore, all results with a confidence level of less than 100% are removed from our set of IOCs. From the USOM feed, we queried using the values `MD` (Malware Distribution Domain), `MI` (Malware Distribution IP), `MU` (Malware Distribution URL), and `MC` (Malware Command Center) for description. Finally, from both sources, only the location is kept and combined in a single set of unique locations, then fed to the prober.

Due to threat actors shutting down or moving their C2 infrastructure once they notice their infrastructure being monitored or listed in a threat intelligence database, it is crucial to solely use recently added IOCs for leads in our dataset. Using dated IOC data may result in probing a server with no malicious activity, as threat actors often use rented server space with shared IP addresses between previous and following renters of the same server space. For this reason, it is also not practical to use the historical dataset (such as complete data dumps) of Threatfox.

4.2 Probing

To construct a dataset containing characteristic information of IOCs, we built a Python program that collects this information by actively connecting to a server marked as IOC and reading out the publicly available information. This program is called a prober, and we illustrate how it works using Figure 4, which depicts the process of probing a single IOC. The prober takes the set of unique IOC locations gathered in the data collection phase as input and makes an HTTP GET request using the address for each of these IOCs. Making an HTTP GET request requires a complete URL, including prepending protocol, to be provided. At the same time, an entry in our set of unique IOC locations can be denoted as a complete URL and IP-port combination without an explicit protocol. In the latter case, we construct a URL using HTTPS as the primary protocol and HTTP as the secondary protocol. This means that for all unresponsive IOC addresses that use the HTTPS protocol, the HTTPS protocol is replaced with the (secondary) HTTP protocol without changing any explicitly set port number, and another attempt to reach the server is made. Furthermore, the system does explicitly not validate the SSL certificate for connections over HTTPS, as the probed

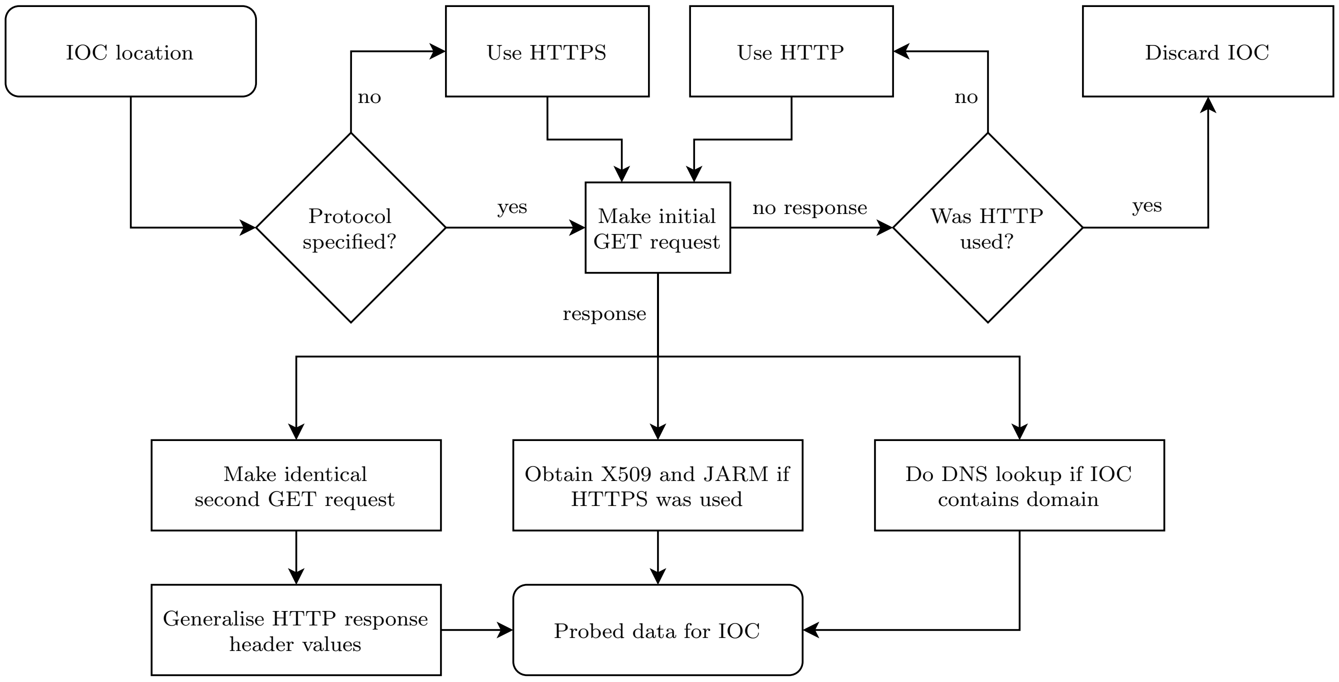

IOC location
Use HTTPS
Use HTTP
Discard IOC
no
no
Protocol specified?
yes
Make initial GET request
no response
Was HTTP used?
yes
response
Make identical second GET request
Obtain X509 and JARM if HTTPS was used
Do DNS lookup if IOC contains domain
Generalise HTTP response header values
Probed data for IOC

Figure 4: Overview of the system for probing a single IOC, starting with the IOC location and resulting in probed data for the IOC if the server responds and discards the IOC otherwise.

host might have a self-signed or other untrusted certificate that any safe consumer client would abort. When both attempts yield no result, we discard the IOC.

4.2.1 Probed Data

The HTTP response sent by a web server on our request indicates that the server we are trying to probe is online. The bottom half of Figure 4 displays the steps following this state. From the received response, we denote the HTTP status code (e.g., 200 OK), the HTTP version used by the server, the total response body content, and response headers. When the IOC location is a domain name, the process involves obtaining the IP address of the corresponding server through the execution of a Domain Name System (DNS) lookup for the specified domain. We obtained the X509 certificate and the JARM fingerprint for instances where the IOC uses the HTTPS protocol. An X509 certificate contains information about the entity it identifies, including its public key, digital signature, expiration date, and the certificate authority's signature. JARM is an active Transport Layer Security (TLS) server fingerprinting tool[13]. However, the limited scope of this thesis prevented further exploration due to time constraints. The protocol, hostname, and path of the request that resulted in the received response are also denoted.

Subsequent, the prober sends two identical requests to allow generalisability and prevent overfitting in the clustering phase, i.e. we want to prevent two samples from not being merged as a result of the variations in their HTTP response header values that are solely different because of, for example, time. With an interval of multiple seconds after receiving the initial HTTP response, the

[13] https://github.com/salesforce/jarm

prober sends an HTTP GET request identical to the initial request to the IOC. The headers of the corresponding HTTP response are compared against the response headers of the initial response to examine if any headers, such as the date or entity tag response header, have changed during the interval of the execution of the two requests. The values of the response headers that have changed over time are redacted to prevent the machine learning algorithm in the clustering phase from overfitting to specific header variations associated with individual instances. For the same reason, we redact the header values of the 'Content-Length' response header and any header whose value contains an IP address.

4.2.2 Parallel Probing

Sequentially probing all the IOCs in our set may take a long time since the offline state of a server is determined by waiting for a response to our request and timing out after a while. Shortening this timeout may incorrectly mark some servers with low response time offline and prevent them from being probed. Therefore, our prober is multithreaded and connects to multiple IOCs in parallel. A predefined number of threads each pop an item from the set of unique IOCs, which is shared amongst all threads and starts probing. After probing is completed, the information obtained by the prober is saved to the disk, and the prober pops a new IOC from the set. The system repeats this routine until the set is empty and the thread terminates.

4.3 Feature Encoding

Because the HTTP protocol data delivered by our prober is not numerical and machine learning models operate on numerical representations, we encode features in our dataset to numerical data to comprehend our dataset by machine learning algorithms. We dummy-encoded the HTTP version and response status code as we consider these features categorical. Each category gets a binary column in the dataset (e.g., for each HTTP version HTTP/1.0, HTTP/1.1, and HTTP/2, a column is added to the dataset) and denote the absence or presence of the respective category value for each sample. Each HTTP response must adhere to a single HTTP version and return a single HTTP response status code. Therefore, each sample contains only one cell representing a category marked as true to indicate the HTTP response status or HTTP version, leaving the remaining cells for the sample false, resulting in a sparse matrix.

The variability in potential values within the HTTP response headers excludes their suitability to be dummy encoded. Similarly, the HTTP response body can be versatile and host diverse file types, such as HTML, JSON, or XML. As we aim to look for similarities and to find patterns in the IOCs we probed, cryptographic hashing the response headers and body features would be impractical as these functions would generate a completely different hash for any change in the input and make it impossible to tell from the resulting hash whether the input is similar to another input. Therefore, we used the similarity hash function SimHash, introduced by Caitlin Sadowski et al. [SL07], to encode the response headers as plain text and body features separately for the coarse-grained clustering phase. SimHash is a locality-sensitive hashing algorithm designed to produce similar hash values for input values that are closely related.

4.4 Clustering

The clustering phase is separated into two subphases: coarse-grained clustering and fine-grained clustering. This separation prevents the computational cost from growing too high when the number of probe results gets larger while still creating meaningful relational clusters.

4.4.1 Coarse-grained Clustering

We employed the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm, as introduced by Martin Ester et al. [EKS+96], for coarse-grained clustering, as we required an algorithm capable of grouping data points with similar attributes without the need for predefining the number of constructed clusters. As some IOCs may not share similarities with others (e.g., when the software used is custom-made or when we mistakenly probe a benign host), we require data points not proximate to any established cluster to be labelled as noise, precluding their misattribution to a specific C2 server family.

The encoded features derived from the HTTP response headers, content body, and HTTP version are used as input variables for the DBSCAN algorithm. As the inclusion of the HTTP status code feature yielded no significant contribution to the creation of distinct clusters, we have chosen to omit this feature from the coarse-grained clustering phase.

4.4.2 Fine-grained Clustering

In the fine-grained clustering subphase of the system, we split the clusters produced in the coarse-grained clustering step into smaller groups. The coarse-grained clusters are, to this point, based on the SimHash of the plain text HTTP response headers, the SimHash of the content body, and the HTTP version of the probed IOCs in our dataset. This fine-grained clustering subphase aims to find similarities between C2 servers within the same coarse-grained cluster based on their HTTP response headers only, particularly on the order of headers.

We utilised the Rank-Biased Overlap (RBO) metric, as William Webber et al. introduced, to calculate the similarity for each pair of IOC samples within a coarse-grained cluster. RBO is a similarity measure, ranging from 0 when there is no similarity to 1 when identical, for ranked lists that may be incomplete, have different lengths, or share only a few items in their lists [WMZ10]. As the headers of an HTTP response are just an ordered set (c.q. list) of key-value pairs, we can use RBO to calculate a similarity score and analogous the distance $D_{\mathrm{RBO}}(H_a, H_b) = 1 - \mathrm{RBO}(H_a, H_b)$ between two IOC samples a and b based on their HTTP headers H_a and H_b. Using this distance measure, we fill a distance matrix to note the distance between each IOC pair in a coarse-grained cluster.

To form the fine-grained clusters, we utilised Hierarchical Agglomerative Clustering (HAC) on the RBO distance matrix of HTTP response headers. HAC is a bottom-up clustering approach where each sample begins in a separate cluster, and the closest clusters are merged repeatedly until all samples are merged into a big cluster [Mur15]. Our adaptation of HAC defines a linkage threshold that prevents clusters spaced farther apart than the threshold from merging. The threshold stops merging when the derived clusters are spaced too far apart, resulting in several clusters of similar samples.

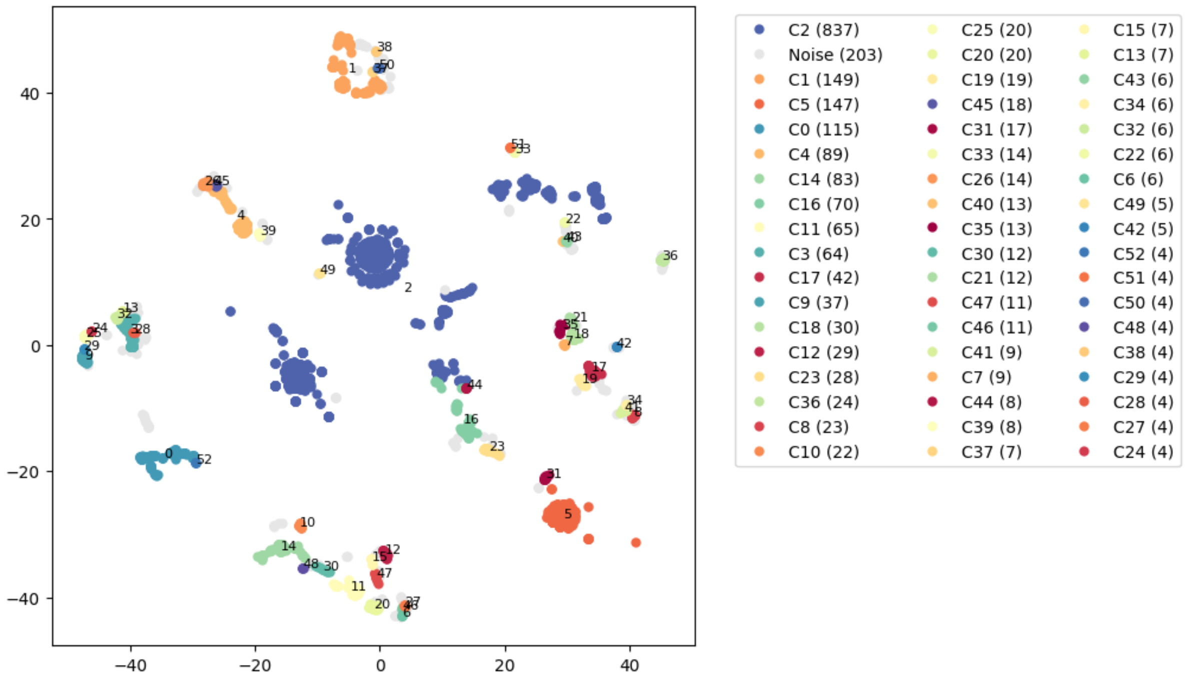


Figure 5: Projection of the coarse-grained clusters generated by the DBSCAN algorithm during the coarse-grained clustering phase, reduced to two dimensions with t-distributed stochastic neighbour embedding (t-SNE). The legend shows the coarse-grained cluster identity and cluster size in parentheses.

4.5 Findings

We briefly discuss the clusters that emerged using the above described methodology. 2382 IOCs were probed between October 10, 2023, and December 8, 2023. During the coarse-grained clustering phase, we set the DBSCAN algorithm's parameter $\epsilon = 0.1$ as the maximum distance between two samples to be considered neighbours and required a minimum of four neighbours for a sample to be considered a core point. The coarse-grained clustering phase resulted in aggregating 2382 samples into 53 distinct coarse-grained clusters, with an additional cluster designated for noise. We utilised t-distributed stochastic neighbour embedding (t-SNE), as the dataset used by DBSCAN has more than two dimensions, to visualise the clusters generated from the coarse-grained clustering phase in Figure 5. This plot shows that cluster C_2 has the most samples (837) of all coarse-grained clusters by far, followed by the dedicated cluster containing all (203) noisy samples that do not fit in any other established cluster.

Moving to the fine-grained clustering phase, we configured an RBO distance threshold of 0.25 for the HAC algorithm and utilised the 'single' linkage criterion, i.e., the HAC algorithm considers the smallest distance between all the samples in two to-be-merged clusters when comparing it against the distance threshold. The fine-grained clustering phase aggregated the 2382 samples into 458 distinct fine-grained clusters spread over the 53 coarse-grained clusters. Cluster C_2F_{36} has the most samples of all fine-grained clusters and is reported on in Section 5.1.

5 Case Studies

The goal of the case studies is to examine the effectiveness of the clustering approach by analysing the resulting clusters, expecting that a robust clustering method would yield tight and dense clusters indicative of specific types of C2 servers. Specifically, our focus is on analysing dense fine-grained clusters exhibiting heightened similarity in response headers. Due to time constraints, our analysis is concentrated on a carefully chosen subset of the densest clusters. We identify similarities observed among servers within the fine-grained cluster, ideally suggesting their association with a particular (malicious) software family and gaining insights into whether our method is a viable way to distinguish particular web-interfacing C2 servers.

In the case studies below, we explore the presence of the C2 server families Cobalt Strike, Supershell, ViperRAT, Havoc, BianLian, and other interesting findings in our clustering results. We focused on the most densely packed fine-grained clusters and attempted to categorise them into C2 types manually. The process involved examining data like response headers and body shared by samples in these clusters to look for hints indicating a specific C2 type. Further steps involved searching for samples from the fine-grained clusters using VirusTotal[14]. This tool allows us to enter URLs, providing a broad overview of verdicts by various antivirus products to determine if the reviewed item is malicious. VirusTotal also reports a crowdsourced context verdict for some items, confirming the item's association with a type of malware or security tool. Additionally, we utilised Shodan[15], an internet-connected device search engine. Shodan collects information about all devices directly connected to the internet, allowing us to search for specific C2 server types. This exploration helps us investigate how servers with a particular type of software respond, e.g., by comparing the typical response header items and composition of a particular C2 server searched by Shodan to the response header items and composition exhibited in one of our fine-grained clusters. Shodan also provides geographical and administrative information about hosts, such as the host city, country, or organisation. This information can assist in ascertaining the genuine affiliation of a server and validating the claimed organisational association. Overall, this comprehensive process establishes a robust foundation for understanding the characteristics and behaviours of the examined C2 server families, independently of the original threat-intelligence databases, Threatfox and USOM. However, this method limits us to only finding true and false positives but does not give us any insight into false negatives.

5.1 Cobalt Strike

Cobalt Strike is a commercial tool intended for red teams to simulate adversary activity[16]. In cybersecurity, a red team is a group of people without malicious intent, simulating attackers to find weak spots in an organisation's system and reporting on them, often hired by the organisation they are trying to investigate. However, threat actors misuse cracked copies of Cobalt Strike to use the tool for malicious purposes, such as launching destructive attacks[17].

We found three distinct fine-grained clusters resembling the most to Cobalt Strike, meaning

[14] https://www.virustotal.com/

[15] https://www.shodan.io/

[16] https://attack.mitre.org/software/S0154/

[17] https://duo.com/decipher/microsoft-and-partners-move-to-disrupt-use-of-cracked-cobalt-strike-copies

```
HTTP/1.1 200 OK
Date: <REDACTED>
Content-Type: application/octet-stream
Content-Length: <REDACTED>
```

(a) Associated with 200 status code.

```
HTTP/1.1 404 Not Found
Date: <REDACTED>
Content-Type: text/plain
Content-Length: <REDACTED>
```

(b) Associated with 404 status code.

Figure 6: Two variations of the HTTP response headers of most samples in cluster C_2F_{36}.

```
HTTP/1.1 404 Not Found
Date: <REDACTED>
Content-Type: text/plain
Content-Length: <REDACTED>
Connection: keep-alive
CF-Cache-Status: DYNAMIC
Report-To: <REDACTED>
NEL: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
CF-RAY: <REDACTED>
alt-svc: h3=":443"; ma=86400
```

Figure 7: HTTP response headers of two samples in cluster C_2F_{36}.

the samples in these clusters display characteristics very similar or identical to those of known Cobalt Strike servers. Firstly, cluster C_2F_{36} is the densest fine-grained cluster within the most dense coarse-grained cluster formed, consisting of 443 samples. Within this fine-grained cluster, all servers used HTTP version 1.1, and we identified two primary variations in the response headers for all but two samples. One variation had the value of the `Content-Type` header set to `application/octet-stream`, as seen in Figure 6a. The other variation had the value set to `text/plain`, as shown in Figure 6b. We observed that the status code for the former was "200 OK", while the servers that followed the latter responded with a "404 Not Found" status code. The headers featured in Figure 6 match those of numerous items displayed by Shodan when searching for Cobalt Strike servers[18]. This match suggests that the servers within this fine-grained cluster will likely be Cobalt Strike servers. VirusTotal crowdsourced context has explicitly classified the activity associated with 77 samples (17.38% of the total samples in C_2F_{36}) in this fine-grained cluster as Cobalt Strike, further supporting this finding. However, eight samples (1.81%) in this fine-grained cluster were explicitly marked as a different C2 servers or security tools. The remaining samples did not have an explicit verdict for what malware family or security tool they were, though VirusTotal marked all as malicious except for one (0.23%).

Within fine-grained cluster C_2F_{36}, two samples stand out because they possess more than three response header entries, as seen in Figure 7, a characteristic not observed in the other samples within the same cluster. We observe that the first three headers are identical to those in Figure 6b, but an additional seven entries have been appended to the response header list. The value `cloudflare` of

[18]https://www.shodan.io/search?query=product%3A%22Cobalt+Strike+Beacon%22

one of the appended headers `Server` suggests that the C2 servers corresponding to these samples use a commercial proxy service called Cloudflare[19], which could append these response headers for the functionality of their service. Upon searching the corresponding IP address on Shodan[20], we verified that the IP address is associated with a server that belongs to Cloudflare, and therefore, some headers could, in fact, be added by the Cloudflare proxy service. While the VirusTotal search corresponding to the two samples does not explicitly label the servers directly as a Cobalt Strike C2 server, we can utilise the possibility that the Cloudflare proxy service appended the additional response headers. When we remove those headers, the remaining headers are identical to the response headers, as shown in Figure 6b, supporting the finding that the two samples have highly similar response headers to the other samples in the same fine-grained cluster.

Secondly, in the same coarse-grained cluster, another fine-grained cluster C_2F_{28} consisting of 15 samples, follows the interesting pattern that all the servers in this cluster respond with a "301 Moved Permanently" status code and either upgrade to or downgrade from an HTTPS connection of the same hostname and are again all behind a Cloudflare proxy. For one of the samples (6.67%) in this fine-grained cluster, we found that VirusTotal labelled this server as a Cobalt Strike C2 server[21]. This verdict was, however, made over seven months ago at the time of writing.

Lastly, one fine-grained cluster C_2F_1 within the same coarse-grained cluster with 12 samples did not match the clusters we found before regarding similarity in response headers. Upon further inspection and looking up individual servers within the cluster on VirusTotal, we found a single match for Cobalt Strike[22]. Nevertheless, our suspicions were chiefly aroused by finding the entry `Server: NetDNA-cache/2.2` upon closer inspection of the response header values. A single Google search on "NetDNA-cache" brought us to a blob of a GitHub repository containing malleable C2 server profiles for Cobalt Strike and Empire[23]. This malleable profile aims to mimic a server's response headers in a content-delivery network that hosts the popular JavaScript library jQuery. The response headers used in the malleable profile matched the response headers we observed in this fine-grained cluster and substantiated the Cobal Strike C2 server verdict.

In this case study, we looked at three separate fine-grained clusters: C_2F_{36}, C_2F_{28}, and C_2F_1, and VirusTotal confirmed the verdict of Cobalt Strike, or we found that the server was using a Cobalt Strike malleable profile for 17.38%, 6.67%, and 100% of the samples in the fine-grained cluster, respectively. Table 1 presents a brief overview of the findings derived from the case study. These results show that clustering based on HTTP response headers can be a viable way to distinguish Cobalt Strike C2 servers from a large set of IOCs. However, C2 infrastructure behind a proxy service tampering with the response headers sent to the client remains challenging, as well as custom configurable malleable profiles.

[19]https://www.cloudflare.com/

[20]https://www.shodan.io/host/188.114.97.1

[21]https://www.virustotal.com/gui/domain/midasusme.uk

[22]https://www.virustotal.com/gui/ip-address/39.101.150.221

[23]https://github.com/BC-SECURITY/Malleable-C2-Profiles/blob/master/Normal/jquery-c2.4.2.profile

Cluster	Size	Confirmed CS	Other
C_2F_{36}	443	77 (17.35%)	8 (1.81%)
C_2F_{28}	15	1 (6.67%)	0 (0%)
C_2F_1	12	12 (100%)	0 (0%)

Table 1: A summary of the Cobalt Strike case study, highlighting the number of servers in the fine-grained clusters, those confirmed to be Cobalt Strike (CS) and those found to be something else.

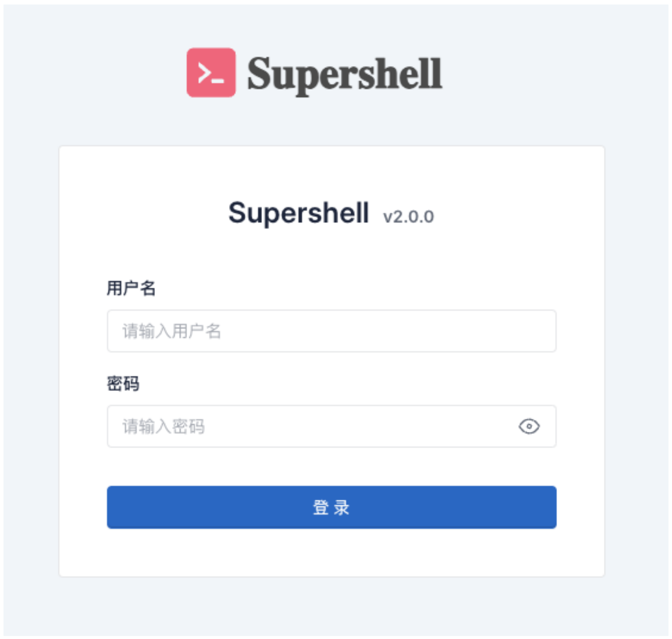


Figure 8: Render of the HTML contents of samples in fine-grained cluster C_5F_4, showing a page with a Supershell login form.

5.2 Supershell

In contrast to the commercial Cobalt Strike, Supershell is an open-source botnet, meaning anyone can use it for free without needing permission or signing up and contribute to enhancing the malware or fork the source code to create its own version[24]. It allows attackers to establish a reverse SSH tunnel, giving the attacker complete access to the victim's machine. The control panel of Supershell is containerised using Docker, meaning threat actors can set up and use the control panel with little effort[25].

For one fine-grained cluster, C_5F_4, consisting of 132 samples, we found that all response headers and HTML body contents are identical for all servers, and all show a login page for a Supershell control panel, as seen in Figure 8, occasionally with a slightly different page text in a different language, but overall remain the same HTML structure. In another fine-grained cluster within a different

[24] https://github.com/tdragon6/Supershell

[25] https : / / web . archive . org / web / 20231007081233 / https : / / socradar . io / the-future-of-open-source-botnets-and-preparedness-against-threats-supershell-botnet/

```
HTTP/1.1 200 OK
Server: nginx/1.18.0
Date: <REDACTED>
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Content-Encoding: gzip
```

(a) HTTP response headers of samples in C_5F_4.

```
HTTP/1.1 302 FOUND
Server: nginx/1.18.0
Date: <REDACTED>
Content-Type: text/html; charset=utf-8
Content-Length: <REDACTED>
Connection: keep-alive
Location: /supershell/login
```

(b) HTTP response headers of samples in C_3F_2.

Figure 9: HTTP response headers of the fine-grained clusters C_5F_4 and C_3F_2, both clusters associated with a Supershell control panel, are identical for the first three entries (excluding the status line).

Cluster	**Size**	**Confirmed SS**	**Other**
C_5F_4	132	132 (100%)	0 (0%)
C_3F_2	20	20 (100%)	0 (0%)

Table 2: A summary of the Supershell case study, highlighting the number of servers in the fine-grained clusters, those confirmed to be Supershell (SS) and those found to be something else.

coarse-grained cluster, C_3F_2, consisting of 20 samples, we found that all servers in this cluster respond with a "302 Found" status code and a response header `Location: /supershell/login` redirecting the visitor to a Supershell login page, looking similar to the Supershell control panel login page in Figure 8.

In Figure 9, we compare the response headers of C_5F_4 (Figure 9a) and C_3F_2 (Figure 9b) against each other. According to the figure, the response headers for C_5F_4 and C_3F_2 are identical in the initial three rows, excluding the status line (i.e., the line containing the HTTP version and response status code). When we manually calculate the RBO distance between the headers in Figure 9a and Figure 9b, excluding the status line, we get a distance of $D_{\text{RBO}}(H_{C_5F_4}, H_{C_3F_2}) = 0.1306$, which is less than the RBO distance threshold of 0.25 set for these case studies, and that could have resulted in these distinct fine-grained clusters being a single fine-grained cluster. However, the coarse-grained clustering phase failed to group them into the same coarse-grained cluster. The probable reason for this disparity is that the SimHash of the response body contents differs significantly, making it impossible to assign them to the same coarse-grained cluster during this phase, and therefore, the system will never calculate the RBO distance between the headers of samples of the two distinct clusters.

In this case study, we examined two fine-grained clusters, C_5F_4 and C_3F_2, finding identical response headers and Supershell login pages in the first, while the latter featured a redirect to a similar login page. We summarised the results in Table 2. Despite close header similarities, the RBO distance fell below the threshold for a merger.

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: <REDACTED>
Content-Type: text/html
Content-Length: <REDACTED>
Last-Modified: <REDACTED>
Connection: keep-alive
ETag: <REDACTED>
Accept-Ranges: bytes
```

Figure 10: HTTP response headers of all samples in cluster C_1F_9.

Cluster	**Size**	**Confirmed VR**	**Other**
C_1F_9	48	11 (22.92%)	1 (2.08%)

Table 3: A summary of the ViperRAT case study, highlighting the number of servers in the fine-grained clusters, those confirmed to be ViperRAT (VR) and those found to be something else.

5.3 ViperRAT

A Remote Access Trojan (RAT) is a type of malicious software used to access and control a victim's machine remotely [VG20]. One such RAT is ViperRAT surveillanceware, which is malware used to spy on targets, primarily targeted Israeli Defence Force personnel[26]. The information ViperRAT steals from a device can inform the attacker about the victim's location, associations with others, and incoming and outgoing messages.

In a fine-grained cluster labelled C_1F_9, comprising 48 samples, VirusTotal flagged 11 (22.92%) as ViperRAT[27] C2 servers, with one server identified as a C2 server from an unspecified malware family. The remaining samples were marked as malicious but without any specific association with a malware family or security tool. Notably, all servers in this cluster are hosted in mainland China, except for one in South Korea and three in Hong Kong. HTTPS connections are accepted on port 60000 by all servers except one server on port 60001. These servers share the same set of response headers, as depicted in Figure 10. All the server samples share a similar HTML content body, featuring the loading of a JavaScript file through a relative path. The distinguishing factor among these samples lies in the slight path variations leading to their respective JavaScript files. Notably, the page title in each HTML body is consistently set to 'VIPER', indicating these servers' affiliation with the ViperRAT malware family.
This case study shows that even on a smaller cluster, the approach is a feasible way to distinguish ViperRAT C2 servers from other web services and establish interrelations among them. Table 3 shows a summary of our findings.

[26] https://attack.mitre.org/software/S0506/, https://www.lookout.com/threat-intelligence/article/viperrat-mobile-apt

[27] https://malpedia.caad.fkie.fraunhofer.de/details/apk.viper_rat

```
HTTP/1.1 404 Not Found
Content-Type: text/html
Server: nginx
X-Havoc: true
Date: <REDACTED>
Content-Length: <REDACTED>
```

Figure 11: HTTP response headers of all samples in cluster $C_{11}F_4$.

Cluster	Size	Confirmed HA	Other
$C_{11}F_4$	44	44 (100%)	0 (0%)

Table 4: A summary of the Havoc case study, highlighting the number of servers in the fine-grained clusters, those confirmed to be Havoc (HA) and those found to be something else.

5.4 Havoc

Havoc[28] is a C2 framework and is, akin to Supershell, open source. It is post-exploitation, meaning that the activities involve maintaining control over the compromised system, extracting information, and carrying out various actions without detection. At the beginning of 2023, it was observed by the Zscaler ThreatLabz research team in an attack targeting a government organisation[29].

$C_{11}F_4$ is a fine-grained cluster containing 44 samples. In this cluster, we found that all samples contain the same response header composition and order, as shown in Figure 11. The third entry of the response headers is `X-Havoc: true`, which leads us to suspect that these servers could be running software from the Havoc framework[30]. Using this response header entry in Google search brought us to a wiki markdown file hosted on the Havoc GitHub repository[31]. This markdown file contained a section explaining how to set up a Havoc Teamserver, followed by an example listener configuration in HashiCorp Configuration Language[32] (HCL) containing the particular header. HCL is a language used to define and manage infrastructure components, such as servers. The discovered listener configuration serves as an easy way to set up a Havoc server, reinforcing the connection with cluster $C_{11}F_4$ and the Havoc framework. Moreover, we inspected the VirusTotal outcomes of each sample in this cluster. These results showed that each server is marked as malicious by one or more security vendors affiliated with VirusTotal, but we did not find a crowdsourced context verdict for any of the samples to tell us the associated malware family. However, we found a collection of IP addresses associated with Havoc C2 infrastructure, verified by VirusTotal[33]. All 44 samples in $C_{11}F_4$ use IP addresses that appear in this collection, confirming this fine-grained cluster's association with the Havoc framework. Table 4 shows a summary of our findings in this case study.

[28]https://github.com/HavocFramework/Havoc
[29]https://www.zscaler.com/blogs/security-research/havoc-across-cyberspace
[30]https://malpedia.caad.fkie.fraunhofer.de/details/win.havoc
[31]https://github.com/HavocFramework/Havoc/blob/main/WIKI.MD#listeners
[32]https://github.com/hashicorp/hcl
[33]https://www.virustotal.com/gui/collection/threatfox_win_havoc/iocs

Cluster	Size	Confirmed BL	Other
C_0F_{12}	17	17 (100%)	0 (0%)

Table 5: A summary of the BianLian case study, highlighting the number of servers in the fine-grained clusters, those confirmed to be BianLian (BL) and those found to be something else.

5.5 BianLian

BianLian is, unlike the malware families and security tools previously discussed in this thesis, a cybercriminal group that develops, deploys and extorts its victims to obtain a ransom[34]. Cybersecurity company SOCRadar reported that the group has built its own C2 infrastructure in the Go programming language, often tailored to a specific attack[35].

One intriguing example is the fine-grained cluster C_0F_{12}, consisting of 17 samples. All samples in this cluster contain the response header entry `Location: https://www.microsoft.com/` and respond with a "301 Moved Permanently" status code, which prompts the visiting web browser to redirect the client to the homepage of the Microsoft website. In addition, all servers in this particular cluster produce an empty response body. This observation raises the possibility that these domains and IP addresses may redirect visitors to the Microsoft website, potentially serving as placeholders for the organisation's web server hosting service. However, upon manually searching the IP addresses in this cluster in Shodan, we found that the Microsoft organisation hosted none of the servers in this cluster, suggesting no affiliation between the servers exhibited in this fine-grained cluster and Microsoft. To support this, we found that one or more security vendors on VirusTotal marked the subject as malicious for each sample. There were no crowdsourced context verdicts, but we came across a VirusTotal trusted and hosted collection[36] of IP addresses associated with ransomware from the BianLian group and found that it contains all the IP addresses of all samples of cluster C_0F_{12}. Table 5 exhibits a summary of this case study.

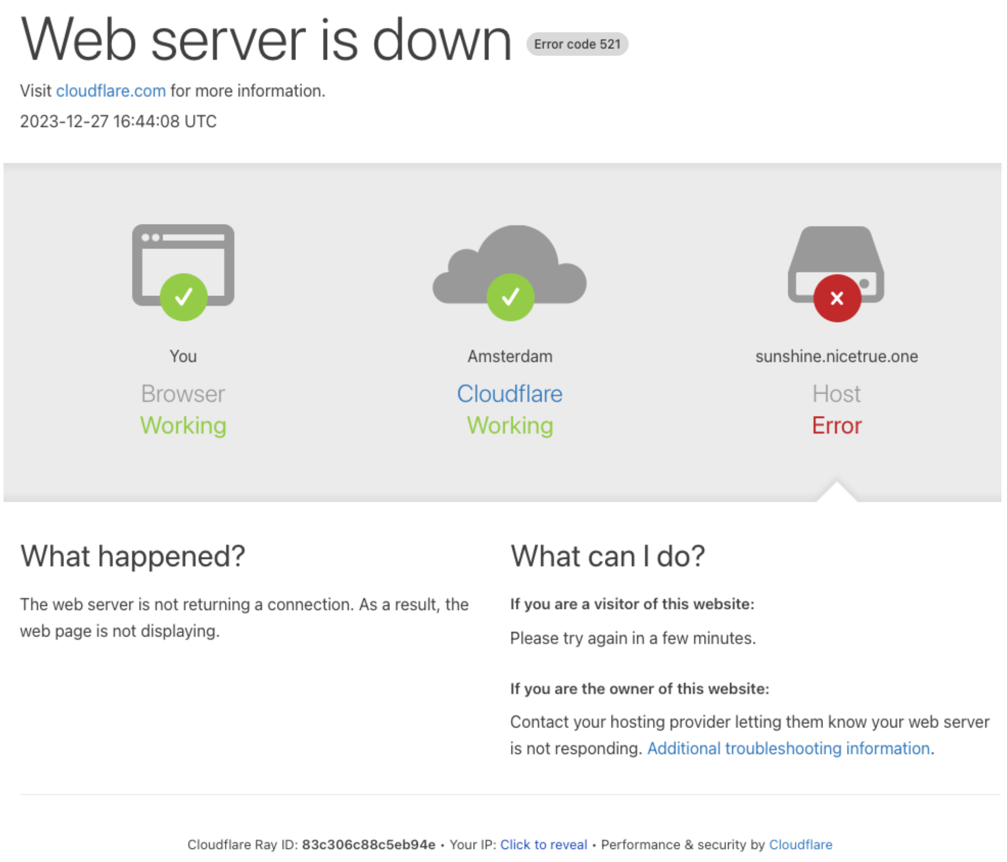
5.6 Other

An additional cluster was identified during our analysis, albeit not directly linked to a distinct malware or C2 server family. Nevertheless, we deem it relevant to acknowledge its presence as it would give us insights into our system's web service distinction abilities.

5.6.1 Cloudflare Status Page

We found a fine-grained cluster of 74 samples, characterised by response headers closely resembling those outlined in the two exempted samples illustrated in Figure 7 of Section 5.1. Our analysis revealed that these headers likely derived from a Cobalt Strike server operating behind a Cloudflare proxy. Given these observed similarities, we would have expected that most verdicts in the search results for individual samples on VirusTotal would land on Cobalt Strike. However, before consulting VirusTotal, further inspection of the response bodies associated with each sample within this cluster

[34] `https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-136a`
[35] `https://socradar.io/threat-actor-profile-bianlian-the-shape-shifting-ransomware-group/`
[36] `https://www.virustotal.com/gui/collection/threatfox_win_bianlian/iocs`



Web server is down Error code 521

Visit cloudflare.com for more information.

2023-12-27 16:44:08 UTC

What happened?

The web server is not returning a connection. As a result, the web page is not displaying.

What can I do?

If you are a visitor of this website:

Please try again in a few minutes.

If you are the owner of this website:

Contact your hosting provider letting them know your web server is not responding. Additional troubleshooting information.

Cloudflare Ray ID: **83c306c88c5eb94e** • Your IP: Click to reveal • Performance & security by Cloudflare

Figure 12: HTML rendering of the response body depicting the Cloudflare status page, evidencing the offline status of the server behind the proxy service.

revealed a distinctive revelation: Cloudflare constructed the entire HTTP response, including the headers and body, of which the HTML render of the body can be seen in Figure 12.

The figure depicts a status page presented by the Cloudflare proxy service that displays that the web server behind the proxy is offline or refuses to connect, and therefore, the proxy cannot relay the web page to the client that is trying to connect. Our observation of this status page could indicate that a threat actor is no longer actively using the server for which this proxy was put in front and may have moved its infrastructure elsewhere. It highlights that the expeditious probing of IOCs bears paramount significance, as threat actors, aware of the detection of their operations, may swiftly relocate their infrastructure. Furthermore, this fine-grained cluster shows the ability of the system to distinguish other non-malicious web applications.

6 Discussion

Our case studies reported in the previous chapters show that the clustering approach is viable, as we were able to identify several important C2 and malware-related server groups. These highlight the clustering methodology's effectiveness in consistently distinguishing specific C2 server types for the densest clusters that we have analysed. We believe this work can be extended further to fingerprint HTTP-based C2 servers effectively and utilised by security researchers and companies to detect C2 infrastructure on new domain names or IP addresses or to keep track of the landscape and arising C2 types.

6.1 Limitations

Although the approach is promising, several limitations are apparent in the methodology employed for this thesis, affecting the robustness and completeness of our approach. Firstly, the threat intelligence providers we used for this thesis, Threatfox and USOM, provided their IOCs in the form of complete URLs, including a path. During the probing phase, we probed servers with the exact paths presented by the intelligence providers. Consequently, servers within the same malware family may have been subjected to different paths during probing, potentially influencing the clustering outcomes. In hindsight, a more effective strategy would involve probing all supplied IOCs with a predefined set of paths to assess how different servers respond to identical requests, facilitating more accurate clustering based on uniform requests. In this research, however, we only probed each URL with the path the threat intelligence provider provided once, with an additional probing iteration focused solely on redacting altered headers.

Another noteworthy limitation is the loss of original probed response header value data during the redaction of changing header values, implemented to mitigate overfitting during the probing phase, which results in the retention of only generalised data. An alternative and potentially more informative approach would involve preserving the original data and applying the redaction process in a subsequent step. This modification would prevent the loss of initially probed data without redactions, offering a more comprehensive dataset for analysis. Similarly, a limitation arising from our approach is that we do not retain the data of IOCs we discard, and only the hostnames and IP addresses of the IOCs we wish to probe are retained. Due to this, we cannot provide more IOC analysis, e.g., what percentage of the probed IOCs come from which threat-intelligence provider.

In the Supershell case study, Section 5.2, we observed that the chosen threshold value for the DBSCAN algorithm might need to be revised. The algorithm failed to merge two coarse-grained clusters we would have desired to see in a single cluster, as they were both related to the Supershell control panel. Because altering the hyperparameters of the clustering algorithm of the coarse-grained phase will change the resulting clusters used as input for the fine-grained phase, the fine-grained clustering hyperparameters should also be reviewed. Therefore, more research is needed to find the correct hyperparameters for the algorithms used in the coarse-grained and fine-grained clustering phases.

A practical limitation arose from using the Python Requests library for IOC probing. During the probing phase, our lack of awareness at the time regarding default request headers added by the library and the absence of explicitly set request headers introduced uncertainties. Given that the receiving end of an HTTP request may respond differently based on the configured request headers, having precise control over this aspect is crucial. This possibility underscores the importance of thoroughly considering both the request path and headers, emphasising the need for comprehensive control over these parameters to ensure accurate and consistent results. As the library consistently added the exact request headers to each request, this limitation did not result in further problems regarding probing consistency. However, the Python Request library is too high-level for the purpose of IOC probing, losing control of the rawly constructed HTTP request. We recommend using a lower-level approach for future work, which leaves more control over how the request is sent to the receiving server.

Lastly, the approach presented in this thesis only works with C2 data provided by third-party threat intelligence providers. The work presented has yet to be experimented with a realistic, diverse dataset of HTTP requests and responses. In practice, clustering might be more noisy than the results obtained during this study. Again, more research is needed to test the capabilities of this approach with real-world data.

7 Conclusions and Further Research

In this thesis, we looked at the viability of using publicly available data, particularly HTTP response headers, to cluster Command and Control (C2) servers as a preliminary step for automatically fingerprinting C2 servers. We constructed a system consisting of four main phases: data collection, where we collected IOCs by using the APIs of trusted intelligence providers; probing, in which we obtained publicly available data by using, among other things, HTTP response headers; feature encoding, during which we prepared the data we obtained for usage by machine learning algorithms; and clustering, of which we subdivided the last phase into coarse-grained clustering with Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and fine-grained clustering by constructing a Rank-Biased Overlap (RBO) distance matrix and utilising Hierarchical Agglomerative Clustering (HAC). Utilising the first two phases of this system, we collected data between October 10, 2023, and December 8, 2023. Then, we clustered 2383 IOC samples during the clustering phase, on which we analysed the most dense fine-grained clusters. As a result of our analysis and to interpret the clustering results, we employed case studies on the results of five malware families: Cobalt Strike, Supershell, ViperRAT, Havoc, BianLian, and other notable observations. Our system currently necessitates subsequent manual validation through external services such as VirusTotal to validate the relation between samples in the cluster and a particular malware family. However, we showed that clustering based on the RBO distance of the HTTP response headers of IOCs shows a viable way to distinguish certain malware families and other web services, even placed behind a proxy service that possibly adds additional entries to the eventual response headers sent to the prober.

7.1 Further Research

In this section, we suggest two research topics for this work based on the ideas we could not realise in this thesis due to time constraints, aside from the improvements we discussed earlier in the limitations section.

First, for our case studies, we needed to define parameters used by the clustering algorithms of the coarse-grained and the fine-grained clustering subphases. In the coarse-grained subphase, we set the DBSCAN algorithm parameter that defined the maximum distance between two samples to be considered neighbours (ϵ) and the parameter defining the minimum required number of neighbours for a sample to be a core point as 0.1 and 4, respectively. In the fine-grained subphase, we set an RBO distance threshold of 0.25 for the HAC algorithm and utilised the ‘single’ linkage criterion. A potential route for further investigation lies in exploring the cluster outcomes of various parameter values, shedding light on the resulting clusters in subsequent analyses.

Lastly, we would be interested in further research on the substantive contribution of the coarse-grained clustering subphase. The case study on Supershell outlined in Section 5.2 reveals an instance where the two clusters, C_5F_4 and C_3F_2, would have merged into a single cluster during the fine-grained clustering phase if not separated during the coarse-clustering phase. It prompts an investigation into the feasibility of bypassing this particular subphase, thereby exclusively relying on the clustering performed solely on the RBA distance of the response headers of probed IOCs.

References

[Ala15] Mamoun Alazab. Profiling and classifying the behavior of malicious codes. *Journal of Systems and Software*, 100:91–102, 2015.

[BBR$^+$12] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138, 2012.

[BLFF96] Tim Berners-Lee, Roy Fielding, and Henrik Frystyk. Hypertext transfer protocol–HTTP/1.0. RFC, RFC Editor, 1996.

[BLMM94] Tim Berners-Lee, Larry Masinter, and Mark McCahill. Uniform resource locators (URL). RFC, RFC Editor, 1994.

[BPT15] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext transfer protocol version 2 (HTTP/2). RFC, RFC Editor, 2015.

[BR17] Marie Baezner and Patrice Robin. Stuxnet. Technical report, ETH Zurich, 2017.

[EKS$^+$96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996.

[FGM$^+$99] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–HTTP/1.1. RFC, RFC Editor, 1999.

[GZL08] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. 2008.

[LV19] Arturs Lavrenovs and Gabor Visky. Investigating http response headers for the classification of devices on the internet. In *2019 IEEE 7th IEEE Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, pages 1–6. IEEE, 2019.

[LWX13] Meijian Li, Yongjun Wang, and Peidai Xie. A binary analysis method for protocol deviation discovery from implementations. In *The International Conference on Information Networking 2013 (ICOIN)*, pages 670–675. IEEE, 2013.

[MBE16] Sean Miller and Curtis Busby-Earle. The role of machine learning in botnet detection. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 359–364. IEEE, 2016.

[Mur15] Fionn Murtagh. Hierarchical clustering, 2015.

[NÅ21] Yuki Nakamura and Björn Åström. Scanning and host fingerprinting methods for command and control server detection. Master's thesis, Blekinge Institute of Technology, 2021.

[PLF10] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of HTTP-based malware and signature generation using malicious network traces. In *NSDI*, volume 10, page 14, 2010.

[Rue07] Marc Ruef. HTTPrecon project, advanced web server fingerprinting. http://web.archive.org/web/20230829050250/https://www.computec.ch/projekte/httprecon/, 2007. Cached: 2023-08-29. Accessed: 2023-11-09.

[Sha04] Saumil Shah. An introduction to HTTP fingerprinting. https://net-square.com/httprint_paper.html, 2004. Accessed: 2023-11-09.

[SL07] Caitlin Sadowski and Greg Levin. Simhash: Hash-based similarity detection, 2007.

[SS15] PV Shijo and AJPCS Salim. Integrated static and dynamic analysis for malware detection. *Procedia Computer Science*, 46:804–811, 2015.

[TWL+21] Zixian Tang, Qiang Wang, Wenhao Li, Huaifeng Bao, Feng Liu, and Wen Wang. HSLF: HTTP header sequence based LSH fingerprints for application traffic classification. In *International Conference on Computational Science*, pages 41–54. Springer, 2021.

[VG20] Veronica Valeros and Sebastian Garcia. Growth and commoditization of remote access trojans. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 454–462. IEEE, 2020.

[WMZ10] William Webber, Alistair Moffat, and Justin Zobel. A similarity measure for indefinite rankings. *ACM Transactions on Information Systems (TOIS)*, 28(4):1–38, 2010.

[ZM09] Hossein Rouhani Zeidanloo and Azizah Abdul Manaf. Botnet command and control mechanisms. In *2009 Second International Conference on Computer and Electrical Engineering*, volume 1, pages 564–568. IEEE, 2009.

[ZVYK14] Ali Zand, Giovanni Vigna, Xifeng Yan, and Christopher Kruegel. Extracting probable command and control signatures for detecting botnets. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1657–1662, 2014.