UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc «Cybersecurity and Data Science»

ΠΜΣ «Κυβερνοασφάλεια και Επιστήμη Δεδομένων»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**TLS Fingerprinting for Analysis of Encrypted Malware Traffic** Ανάλυση Υποδομών Κακόβουλου Λογισμικού μέσω TLS Fingerprinting
Student's name-surname: Ονοματεπώνυμο φοιτητή:	**Vasiliki Stathopoulou** Βασιλική Σταθοπούλου
Father's name: Πατρώνυμο:	**Christos** Χρήστος
Student's ID No: Αριθμός Μητρώου:	ΜΠΚΕΔ21049
Supervisor: Επιβλέπων:	**Konstantinos Patsakis, Professor** Κωνσταντίνος Πατσάκης, Καθηγητής

March 2026/ Μάρτιος 2026

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Constantinos Patsakis **Professor** Κωνσταντίνος Πατσάκης Καθηγητής	**Panayiotis Kotzanikolaou** **Professor** Παναγιώτης Κοτζανικολάου Καθηγητής	**Efthimios Alepis** **Professor** Ευθύμιος Αλέπης Καθηγητής

Περίληψη

Η διαδεδομένη χρήση του πρωτοκόλλου TLS έχει ως αποτέλεσμα την μείωση της ορατότητας στην διαδικτυακή επικοινωνία, με αποτέλεσμα η ανίχνευση κακόβουλης δραστηριότητας να είναι πιο δύσκολη. Η τεχνική TLS fingerprinting παρέχει έναν εναλλακτικό τρόπο ανίχνευσης, αναλύοντας χαρακτηριστικά του TLS handshake χωρίς την απαίτηση αποκρυπτογράφησης. Η παρούσα μελέτη ερευνά την χρήση του JARM fingerprint με σκοπό την ανάλυση υποδομών σχετικές με κακόβουλα λογισμικά, χρησιμοποιώντας δημόσια διαδικτυακά δεδομένα καθώς και ελεγχόμενη εκτέλεση κακόβουλων λογισμικών σε απομονωμένο περιβάλλον. Επιπλέον, μελετήθηκαν τα JARM fingerprints διαφόρων Command-and-Control (C2) frameworks, όπου οι ρυθμίσεις παραμέτρων τους μεταβλήθηκαν για να εξεταστεί η επιρροή των αλλαγών στα JARM fingerprints. Τα αποτελέσματα δείχνουν πως τα TLS fingerprints αποκαλύπτουν μοτίβα επαναχρησιμοποίησης υποδομών, καθώς και μεταβλητότητα στην συμπεριφορά των διακομιστών. Επιπλέον, παρατηρήθηκε ότι διαφορετικοί τύποι κακόβουλου λογισμικού μπορεί να χαρακτηρίζονται από παρόμοια TLS χαρακτηριστικά, και πως αλλαγές στις παραμέτρους της εκάστοτε υποδομής έχουν ως αποτέλεσμα την σημαντική αλλαγή του JARM fingerprint.

Abstract

The widespread use of Transport Layer Security (TLS) has reduced visibility into network traffic, making the detection of malicious activity more challenging. TLS fingerprinting offers an alternative approach by analyzing observable handshake characteristics without decryption. This study examines the use of JARM fingerprints to analyze malware-related infrastructure using both public network data and controlled malware execution in an isolated environment. In addition, JARM fingerprints were examined in command-and-control (C2) frameworks, where infrastructure configurations were modified to evaluate their impact on fingerprint values. The results show that TLS fingerprints reveal patterns of infrastructure reuse while also demonstrating variability in server behavior. It was observed that different malware types may share similar TLS characteristics, and that changes in infrastructure configuration can significantly affect JARM fingerprints.

Contents

1 Introduction

The growing digitization of communication has significantly increased the use of encryption protocols, making TLS the de facto standard for securing Internet traffic. TLS provides confidentiality, integrity, and authentication of data transferred over the network. Hence, TLS is extensively utilized across a range of applications, such as web browsing, cloud services, and enterprise communication systems.

. However, this widespread use of encryption enhances privacy and security but presents considerable obstacles to network monitoring and threat detection.

According to conventional network security models, deep packet inspection techniques require access to the packet payload to identify malicious activities. With the increasing use of TLS, such methods are now ineffective since the contents of the communication are not accessible unless decrypted [2]. In this way, malicious activities are carried out behind encrypted channels for data exfiltration, malware communication, and C2 operations.

Modern malware uses TLS-based communications to contact remote infrastructures. Such communications are often forged to resemble legitimate encrypted traffic, making them very difficult to distinguish from benign activity. Security analysts are thus faced with identifying malicious activities without the content of network traffic.

Several alternative methods based on observable metadata have thus been proposed. One is TLS fingerprinting, which is based on analyzing the characteristics of the TLS handshake, including protocol versions, cipher suites, and extensions. These features are visible during the encrypted session and provide informative insight into the behavior of the client and server.

Several techniques, such as JA3, JA3S, and JARM, have been proposed to capture these characteristics and generate fingerprints representing TLS implementations [3][4]. JA3 and JA3S provide passive fingerprints for client and server behavior, respectively, while JARM actively probes the server to produce a composite fingerprint. These methods have demonstrated potential in detecting patterns of encrypted traffic and clustering-like infrastructure.

However, several vital issues arise with the TLS fingerprinting techniques. Fingerprints are not unique; thus, different systems may share one fingerprint with similar TLS libraries or configurations. Also, modern network infrastructures are commonly composed of intermediaries, such as reverse proxies, load balancers, and CDNs, which modify TLS behavior and further obfuscate the relationship between observed fingerprints and underlying systems.

Existing literature mainly uses TLS fingerprints in detection and classification tasks, mainly machine learning tasks. Although these improve the detection rate, they do not give much information on how TLS fingerprints reflect infrastructure behavior and how these vary with changing configurations.

Therefore, this thesis explores how TLS fingerprints reflect malware-related infrastructure and how they behave when controlled changes are made. It especially considers the stability and variability of server-side fingerprints, such as JARM, in command-and-control environments.

1.1 Objectives

This thesis uses TLS fingerprinting techniques to study malware-related infrastructures over encrypted networks. The observable features of the TLS handshake are discussed in how they infer server behavior patterns and find similarities between different infrastructures involved in malicious activities.

This work is both a passive and active analysis approach. TLS fingerprints are extracted and analyzed from publicly available network datasets, giving insights into real-world traffic and known malicious infrastructures. Subsequently, controlled experiments are conducted in which samples of malware are executed in a strictly isolated laboratory environment. Thus, empirical observation and experimental verification of TLS behavior are allowed.

Further research explores C2 frameworks as a significant element in the lifecycle of modern malware. C2 infrastructures will be deployed and analyzed in a controlled environment to examine how TLS fingerprints behave with different configurations. The focus is mainly on how changes to infrastructure parameters—such as server configuration and the addition of intermediary components, e.g., reverse proxies—affect the fingerprint values.

This thesis aims to:

- Analyze the TLS fingerprint data (JARM) to understand the patterns of infrastructure behavior
- Examine the degree that TLS fingerprinting can be used reliably for analyzing encrypted network traffic.

To that end, this thesis aims to understand the way TLS fingerprinting can be utilized to detect and analyze infrastructure behavior in encrypted environments.

1.2 Previous work

As adoption of encrypted traffic by threat actors and malware has become more widespread, extensive research has been conducted in order to discover how network behavior can be analyzed without payload inspection. With the previous constraint in mind, TLS fingerprinting can be utilized as it is based on the TLS handshake's visible metadata.

Althouse's [3] paper introduced JA3 and JA3s, which fingerprint the TLS ClientHello and ServerHello parameters respectively. The above have been applied academically as well as operationally with the goal of clustering systems having similar TLS configurations. However, there are several known limitations, namely: non-unique serialization and system configuration sharing, that make this approach vulnerable.

The first studies on using TLS in malware communications are by Anderson et al. [5]. In their work, they demonstrated that patterns of malicious traffic could be discovered by using TLS metadata without the need for its decryption. Their study clearly showed the relevance of incoming features derived from the handshake and further stimulated interest in research into analyzing encrypted traffic.

Several other studies show how to use TLS fingerprints in classifying traffic. For example, Anderson and McGrew [6] proposed machine learning techniques using TLS features with flow-level data to classify malicious traffic. Related research in encrypted traffic classification shows that TLS metadata can also serve as input features to a supervised learning framework to achieve high detection performance [7].

Most of the proposed methods work in performance-centric mode and do not consider interpretability. There is almost no insight into the infrastructure that generated those patterns and, even more, into the reasons why.

In order to overcome the shortcomings of passive fingerprinting, JARM was introduced as an active fingerprinting technique that probes servers with multiple crafted TLS handshakes [4]. In this way, it creates a composite fingerprint representing a wider spectrum of server behavior, which is very helpful when identifying similarities at the infrastructure level.

More recent work has also dealt with the stability of TLS fingerprints over time. Evidence indicates that TLS fingerprints may change due to software updates, configuration changes, and evolution within the protocol, especially the adoption of TLS 1.3 [8]. Other works have also

demonstrated that applying techniques such as TLS extension randomization significantly impacts the effectiveness of static fingerprinting methods [9].

Other studies have also considered the influence of network infrastructure on TLS behavior. Reverse proxies, load balancers, and CDNs can change handshake characteristics so much that attributing fingerprints to specific backend systems is difficult [10].

In spite of these developments, the way TLS fingerprints behave under controlled experimental conditions, such as malware environments, has not yet been adequately investigated and documented. Most studies are based on passive observation or large-scale classification, with little interest in controlled experimentation.

There has also been little research on the variability of fingerprints such as JARM across different command-and-control frameworks and infrastructural configurations. Therefore, this thesis combines real-world data analysis with controlled experimentation to assess fingerprint variability and interpretability.

2 Theoretical Background

This chapter presents the theoretical foundations required to understand TLS fingerprinting and its application in the analysis of encrypted network traffic. It introduces the Transport Layer Security (TLS) protocol, describes the TLS handshake process, and explains how observable handshake characteristics can be used to derive fingerprints such as JA3, JA3S, and JARM.

2.1 Transport Layer Security (TLS)

Transport Layer Security (TLS) is a protocol that is used to protect communication between two systems over a network. In simple terms, TLS makes sure that when data is sent from one computer to another, it cannot be read, changed, or faked by anyone else. This is extremely important in modern networks, where sensitive information such as passwords, financial data, and personal messages are transmitted every second.

TLS provides three main types of protection: **confidentiality, integrity, and authentication** [1]. Confidentiality means that the data is encrypted so that even if someone intercepts it, they cannot understand it. Integrity means that the data cannot be modified during transmission without being detected. Authentication ensures that the client is communicating with the correct server and not with a malicious system pretending to be legitimate.

From a technical perspective, TLS operates above the transport layer, typically on top of the Transmission Control Protocol (TCP). Before any secure data can be exchanged, TLS establishes a secure channel between a client and a server. This is done through a process called **cryptographic negotiation**, where both sides agree on how they will encrypt and protect the communication.

An important part of this process is **key exchange**, where both the client and the server agree on shared secret keys. These keys are then used to encrypt and decrypt the data. Without these keys, the communication cannot be understood.

The latest version of the protocol, **TLS 1.3**, improves both performance and security. It reduces the number of steps required to establish a secure connection, which makes communication faster. It also removes older and less secure cryptographic methods. However, even though TLS 1.3 improves security, it does not hide everything. Some parts of the communication process, especially during the initial setup, remain visible.

This is important because it allows analysts to study TLS behavior without decrypting the data. In other words, even though the content of the communication is hidden, the way the communication is established can still be observed. This forms the basis for techniques such as TLS fingerprinting, which are used in this thesis.

2.2 TLS Handshake Process

An SSL handshake is a process that begins a communication session. The two parties acknowledge one another, determine how they will protect information, verify one another's security protocols, and set session keys.

The SSL handshake steps result from those agreements, and they can vary depending on what the two sides want.

In general, an SSL handshake proceeds via these steps:

1. **Contact:** A browser sends a "client hello" message to the server. The message includes critical details, such as the SSL version the client uses, cipher settings (more on that in a minute), and session-specific information.

2. **First response:** The server sends back proof of security (via certificates), the server's cipher settings, and session-specific data.
3. **Authentication:** The browser verifies the security certificate to ensure it made contact with the right authority.
4. **Key exchange:** The browser and the server exchange keys, validating the security of their exchange.
5. **Wrap up:** Both the server and the browser confirm that the work is complete, and the handshake is finished.

While the payload is encrypted, many handshake characteristics remain visible and thus extractable from network traffic such as cipher suite lists, TLS version negotiation, extension ordering, supported cryptographic groups, thus the aforementioned characteristics elements are considered fundamental to TLS Fingerprinting

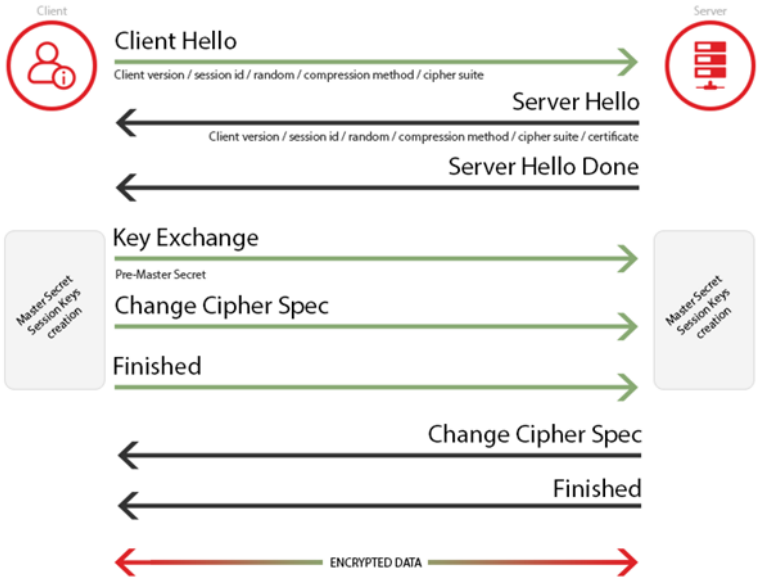

Figure Theoretical Background.1 TLS Handshake

2.3 TLS Fingerprinting

TLS fingerprinting identifies a system by its TLS handshake behavior. Instead of examining the encrypted data, fingerprinting focuses on attributes visible during the connection establishment.

The critical idea behind TLS fingerprinting is the different behaviors of various software implementations. For instance, Chrome, a command-line tool, and malware will all use TLS, but the cipher suites, extensions, and even versions of the protocol may be different. Such differences form patterns unique to each, which can be observed and recorded.

Such patterns are called fingerprints. A fingerprint is how a system behaves during the TLS handshake. Similar fingerprint comparisons can cluster systems or identify anomalous behavior.

This can be done in two ways:

Passive fingerprinting: This is based on observing live network traffic without any active intervention on the system. It is non-intrusive and can be applied to live traffic.

Active fingerprinting: This is where specially crafted requests are sent to a server, and the responses are analyzed. In this way, more control can be exercised, and more details of the system can be uncovered.

This type of TLS fingerprinting is very useful when dealing with encrypted traffic. Since traditional inspection methods cannot see the content, fingerprinting is a technique that looks at communication from the perspective of behavior instead of data.

2.3.1 JA3 Fingerprinting (Client-side)

JA3 is a passive TLS fingerprinting technique that focuses on the client side of communications. It creates a fingerprint by extracting values from the ClientHello message into one value.

The JA3 fields are as follows:

- TLS version
- Cipher suites
- Extensions
- Elliptic curves
- Elliptic curve formats

These are joined into a string, hashed, and yield a unique identifier: a JA3 fingerprint.

In other words, JA3 captures how a client behaves when starting a TLS connection. This "signature" can be used to identify or cluster similar clients.

JA3 is helpful in identifying patterns and anomalies. For instance, if the normal network comprises traffic from typical web browsers, the emergence of a new, unusual JA3 fingerprint indicates suspicious activity.

However, JA3 has some limitations. Different applications can share TLS libraries and produce the same fingerprint. Thus, JA3 is not always unique to an application. Fingerprints may also change over time due to software updates or reconfiguration.

2.3.2 JA3S Fingerprinting (Server-side)

This is the server-side counterpart of JA3: instead of analyzing the ClientHello message, JA3S analyzes the ServerHello message.

The JA3S fingerprint depends on the following:

- TLS version selected by the server
- Selected cipher suite
- Selected extensions

In that respect, JA3S provides information about the server response in the handshake process, which may help identify typical behavior patterns of the infrastructure servers.

However, JA3S is more limited than JA3 since it only reflects one handshake outcome. The server's answer depends on what the client asks for, so one client may have a different JA3S fingerprint for one server than another.

Thus, JA3S is not to be used by itself; it has greater value when combined with other techniques, such as JARM, to paint a more holistic picture of server behavior. JARM is an active fingerprinting technique for more profound server analysis. Unlike JA3S, which depends on a single observed handshake, JARM actively probes the server.

2.3.3 JARM Fingerprinting (Active Server Profiling)

JARM sends a series of ClientHello messages with slight parameter variations to the server. Each time, the server sends back a message, which is collected and analyzed.

This involves:

Sending several probes with different TLS configurations

Recording the server's responses

Combining the responses into one fingerprint

Such a method gives a much better view of the server's behavior under various conditions. Hence, it can be used to identify infrastructure patterns and cluster similar servers.

However, JARM is not flawless. A change in the server configuration will change the fingerprint:

- Changing cipher suites
- Updating the TLS version
- Introducing reverse proxies or load balancers

Such changes will manifest in different behaviors and fingerprints.

3 Methodology

This chapter describes the methodology followed in this study for the analysis of TLS fingerprints in malware-related infrastructure. The approach combines analysis of publicly available network data with controlled experimentation in an isolated laboratory environment. This dual methodology enables both the observation of real-world behavior and the evaluation of TLS fingerprint variability under controlled conditions.

3.1 Data Sources

The primary dataset used in this study consists of publicly available packet capture (PCAP) files containing network traffic associated with malware activity. These datasets provide realistic examples of encrypted communications between infected hosts and command-and-control (C2) infrastructure.

From the PCAP files, relevant TLS connections were identified and extracted. Each connection was analyzed to obtain metadata related to the TLS handshake, including client and server parameters. In addition, domains and IP addresses associated with the connections were recorded and linked to known malware families where possible.

This dataset forms the basis for the passive analysis performed in this study.

Specifically, the PCAP files were sourced from the following domains:

- Malware Traffic Analysis (malware-traffic-analysis.net)
- Triage (tria.ge)
- ANY.RUN (app.any.run)

The extracted file is comprised of the columns Destination IP, Destination Port, Domain Name, JA3S and JARM.

3.2 TLS Fingerprint Extraction

For each PCAP file the TLS fingerprints were extracted by using multiple analysis tools. More specifically, JA3S fingerprints were derived from TLS ServerHello messages and represent server-side responses and were gathered using ZUI, a tool to aid with analyzing and viewing datasets. Specifically, as shown on the Figure 2, a Zui query was used in order to pull the SSL connections and extract from them the destination IP Address, Destination Port, Server Name and JA3S.

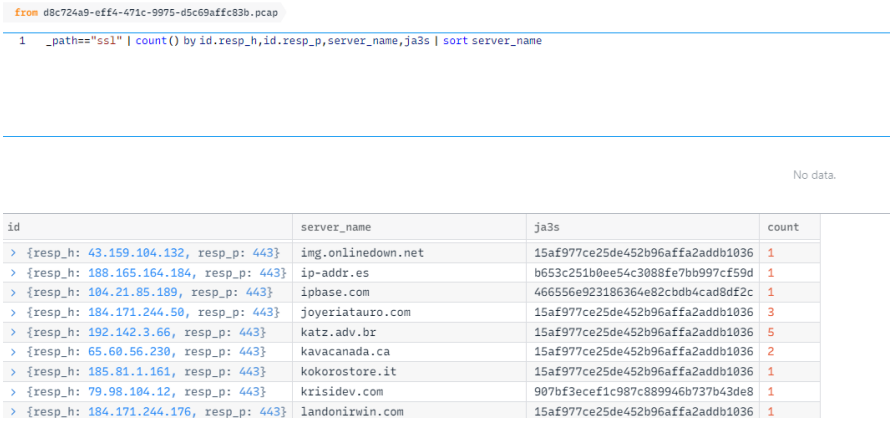

Figure Methodology.2 Zui Query for information extraction from PCAP files

Afterwards, the domains that were associated with legitimate traffic, such as Google, Microsoft etc., were filtered out.

The next step was to obtain the JARM fingerprints by actively probing identified server endpoints, using the script “jarm.py” which was created by Salesforce. To do so, the input provided to a PowerShell script that exports the online domains to a csv file, this step was done in order to eliminate scanning the domains that are no longer online.

dns-resolve.ps1

```powershell
Get-Content to_scan.txt | ForEach-Object {
    $domain = $_.Trim()
    if ([string]::IsNullOrWhiteSpace($domain)) { return }

    try {
        $result = Resolve-DnsName -Name $domain -ErrorAction Stop
        [PSCustomObject]@{ Domain = $domain; Status = "UP"; IP = $result[0].IPAddress }
    }
    catch {
        [PSCustomObject]@{ Domain = $domain; Status = "DOWN"; IP = $null }
    }
} | Export-Csv -Path results.csv -NoTypeInformation
```

Figure Methodology.3 PowerShell Script that checks the availability of the domains

The aforementioned online domains were scanned by Salesforce’s jarm.py in order to retrieve the JARM Fingerprint.

```
PS                                                > python .\jarm-master\jarm.py -i .\to_scan.txt
Domain: 3xp3cts1aim.sbs
Resolved IP: 40.91.108.115
JARM: 2ad2ad0002ad2ad22c2ad2ad2ad2adc2ddcfd203d071c45b4b0ffe3d7b4b89
Domain: 7070-ppxcx-a1-3gg5ufwp666ee644-1300076834.tcb.qcloud.la
Resolved IP: 122.192.127.53
JARM: 29d29d00029d29d21c42d42d000000307ee0eb468e9fdb5cfcd698a80a67ef
Domain: book.rollingvideogames.com
Resolved IP: 23.235.202.121
JARM: 20d19d20d21d20d00042d43d0000004c02e61e96dd0a7ac239d8564cd3eaff
```

Figure Methodology.4 JARM extraction

3.3 Experimental Environment

To complement the analysis of public datasets, controlled experiments were conducted within an isolated laboratory environment. The environment was designed to safely execute malware samples and analyze their network behavior without risk to external systems.

The setup consisted of:

- A Virtual machine running a Windows operating system (Windows 10)
- A separate system used to simulate command-and-control (C2) infrastructure (Kali Linux)
- Network monitoring tools for capturing and analyzing traffic as well the state of the “Victim” virtual machine (Wireshark, System Informer)

3.3.1 Malware Execution Procedure

Malware samples obtained from publicly available repositories were executed within the controlled environment. Each sample was run under monitored conditions to generate network traffic.

During execution:

- System Informer was used to monitor the state of execution and whether the malware proceeded with its intended infection chain (e.g. did not exit due to being run in an analysis VM system)
- Network traffic was captured using Wireshark
- TLS connections were identified and extracted, focusing on potentially malware-related ones.

The following steps are described in TLS Fingerprint Extraction.

3.3.2 Command-and-Control (C2) Frameworks

To understand the TLS fingerprint behavior better, C2 frameworks, which are the underlying mechanism that malware and threat actors utilize to establish communications with remote servers, were placed within the lab environment.

C2 frameworks were initially deployed on the Kali Linux machine using Cobalt Strike and Sliver. Then, an HTTPS listener was configured on each framework to set up an encrypted communication channel between the C2 server and the victim host. A Windows executable beacon was created using the configured listener. In order to ensure the successful execution of the beacon, a folder exclusion was added on Windows Defender of the victim's machine. The beacon was transferred to the victim machine. During the execution of the beacon, the C2 communication was captured via Wireshark and saved to a pcap file for later analysis. The JARM fingerprint of the C2 server was then extracted using the Salesforce JARM Python script. After each run, the victim machine was restored to its original clean state.

Different configurations of the C2 infrastructure were implemented, including default framework configurations, usage of intermediary components such as reverse proxies and modification of the TLS settings on the reverse proxies TLS fingerprints, particularly JARM, were collected for each configuration to evaluate how infrastructure changes affect server-side behavior.

3.3.3 Infrastructure Configuration Variations

A key aspect of this study is the evaluation of how TLS fingerprints change when infrastructure configurations are modified.

To that end, we proceeded with introducing a reverse proxy layer in front of the C2 listener, that forwarded the C2 traffic internally to the actual server. To achieve this, we altered the listener's port to 8443, the beacon's destination port to 443 and set up the reverse proxy Nginx to intercept the traffic on port 443 and then forward it internally to port 8443.

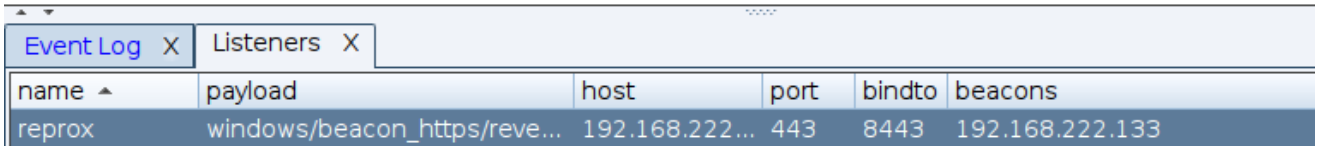

Figure Methodology.5 Example of the Cobalt Strike's listener

With this variation, the Nginx TLS configuration was exposed, instead of the actual C2 server's one. Hence, Figure 5 illustrates the configuration profile of the nginx.

```
  GNU nano 8.7.1                                c2pro
server {
    listen 443 ssl;


    ssl_certificate    /home/kali/Desktop/cert.pem;
    ssl_certificate_key /home/kali/Desktop/key.pem;


    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:TLS_AES_128_GCM_SHA256';
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass https://127.0.0.1:8443;  # Listener port
        proxy_ssl_verify off;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }
}
```

Figure Methodology.6 Initial nginx configuration

Additionally, this provided the capability of altering the profile configuration of Nginx to extract different JARM fingerprints. With that in mind, two other modifications in TLS configuration parameters by modifying Nginx's configuration profile and altering the fields "ssl_ciphers" and "ssl_protocols". These variations were intentionally introduced to assess the sensitivity of TLS fingerprinting techniques, particularly JARM, to changes in server behavior.

```
  GNU nano 8.7.1                                c2pro
server {
    listen 443 ssl;


    ssl_certificate    /home/kali/Desktop/cert.pem;
    ssl_certificate_key /home/kali/Desktop/key.pem;

    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers 'AES256-SHA:AES128-SHA:ECDHE-RSA-AES256-SHA';
    ssl_prefer_server_ciphers on;

    location / {
        proxy_pass https://127.0.0.1:8443;  # CS teamserver port
        proxy_ssl_verify off;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $remote_addr;
    }
}
```

Figure Methodology.7 First variation of Nginx configuration

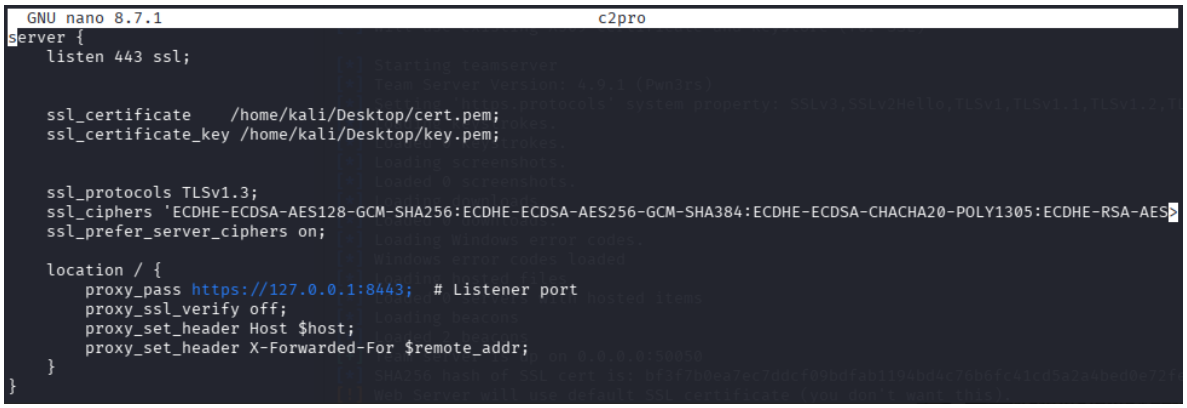

Figure Methodology.8 Second variation of Nginx configuration

Every mentioned nginx configuration was implemented in Cobalt strike and Sliver and the JARM fingerprints were identified respectively.

3.4 Data Analysis and Clustering

Afterward, all the JARM fingerprints were compared in an attempt to understand their behaviors, similarities, and relations. A rule-based clustering approach was applied to enable structured interpretation of the collected data.

The first step was to group all the entries based on their JARM fingerprint, thus each unique JARM is a separate cluster, specifically if more than one endpoint shares the same JARM with another is placed in the same cluster. To gain more value, the JA3S was incorporated in order to examine further each cluster, as it helps identify differences between the same group by exhibiting variations on TLS handshakes responses. The clusters that were used are the following:

- Stable clusters: they share multiple JARM fingerprints but only one JA3S value, meaning that the there is a consistency in the TLS behavior across all samples in the group
- Variable clusters: clusters that contains the same JARM fingerprint but more than one JA3S value, meaning that the infrastructure appears to be similar but the TLS differs
- Unique clusters: clusters that contains only one sample, probably isolated configurations

By applying these rules, it is easier to understand whether a cluster represents a stable infrastructure or a more dynamic behavior.

Data from experiments with C2 frameworks, Cobalt Strike and Sliver, were excluded from the clustering and used as a form of validation to further interpret the results.

These experiments were designed to evaluate how changes in infrastructure affect JARM fingerprints. Specifically:

- Initial executions of the C2 frameworks produced distinct JARM fingerprints
- The introduction of a reverse proxy (Nginx) resulted in identical JARM fingerprints across different frameworks
- Further modifications to TLS configuration (e.g., protocols and cipher suites) led to new fingerprint values but also same across the c2 frameworks.

In short, the C2 experiments reinforce the premise that the JARM fingerprint is TLS configuration-dependent; thus, any change in a component or parameter in the infrastructure can cause it to change. Hence, the C2 experiments were not included in the clustering but aided in explaining the clusters' behavior. Notably, this allows a controlled explanation for the phenomenon

in which a cluster includes several endpoints with the same JARM fingerprint, which can be explained by shared elements of infrastructure, not necessarily identical backend systems.

Overall, this methodology aims for a more accurate and grounded interpretation of TLS fingerprinting results and also focuses on the importance of utilizing dataset-driven patterns as well as experimentally validated behavior

4 Results and Discussion

4.1 Dataset Overview

As stated above this dataset contains JARM fingerprints, JA3S fingerprints, IP addresses, ports, and server-related identifiers that are combined by publicly available data and malware executions. In total, 20 PCAP files were used, resulting in 115 domains or IP addresses. From this data, 47 unique JARM values were identified, meaning 47 different clusters. For the analysis, the focus was placed on the 10 most common JARM clusters. The samples were associated with malware families such as Vidar, Lumma, Havoc, Specula C2, StealC, AsyncRAT, and Remcos.

At the initial exploration of the dataset both repetition and diversity were revealed TLS configurations appear multiple times across different endpoints, while others only once. This indicates a mixture of reused infrastructure and unique deployments, which forms the basis for clustering and further analysis. The total amount of samples

The first stage of the analysis involves grouping samples based on JARM fingerprints. Each unique JARM value represents a cluster.

The distribution of these clusters is highly uneven. A small number of clusters contain a large proportion of the dataset, while the majority contain only a few samples.

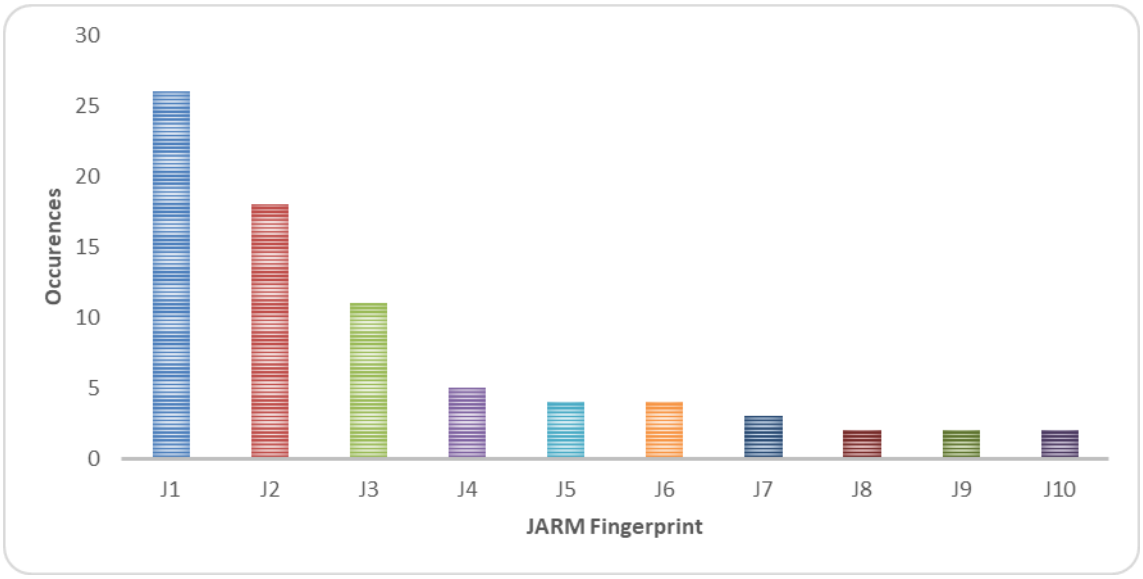

Figure Results and Discussion.9 Distribution of JARM Clusters (Top Clusters by Frequency)

This pattern is illustrated in Figure 4.1, where “dominant” clusters have significantly more samples compared to the rest. This suggests that specific TLS configurations are widely reused across multiple endpoints. Such reuse is likely due to shared infrastructure, standardized deployment practices, or the repeated use of similar frameworks.

On the other hand, smaller clusters represent less common or unique configurations, potentially corresponding to custom deployments or short-lived infrastructure.

This finding highlights a key characteristic of malicious ecosystems: infrastructure reuse. Rather than deploying entirely unique configurations, attackers often rely on repeated setups, which can be identified through clustering.

4.2 Cluster Classification Rules

To provide a structured interpretation of the clusters, a rule-based classification approach was applied using two key metrics:

- The total number of samples per JARM cluster
- The number of distinct JA3S fingerprints within each cluster

The classification rules are defined as follows:

		Rules	
Stable Cluster	Total Samples per JARM cluster >1	Unique JA3S =1	Repeatable server behavior
Variable Cluster	Total Samples per JARM cluster>1	Unique JA3S>1	Similar infrastructure with variability
Unique Cluster	Total Samples per JARM cluster=1		Rare or one-off configurations.

Additionally, Figure 4.2 illustrates the aforementioned Classification. Stable clusters indicate infrastructure reuse, while variable clusters exhibit that similar environments can still produce different TLS responses. Unique clusters reflect the diversity of configurations within the dataset. This reinforces that TLS fingerprints should be interpreted as behavioral indicators rather than unique identifiers.

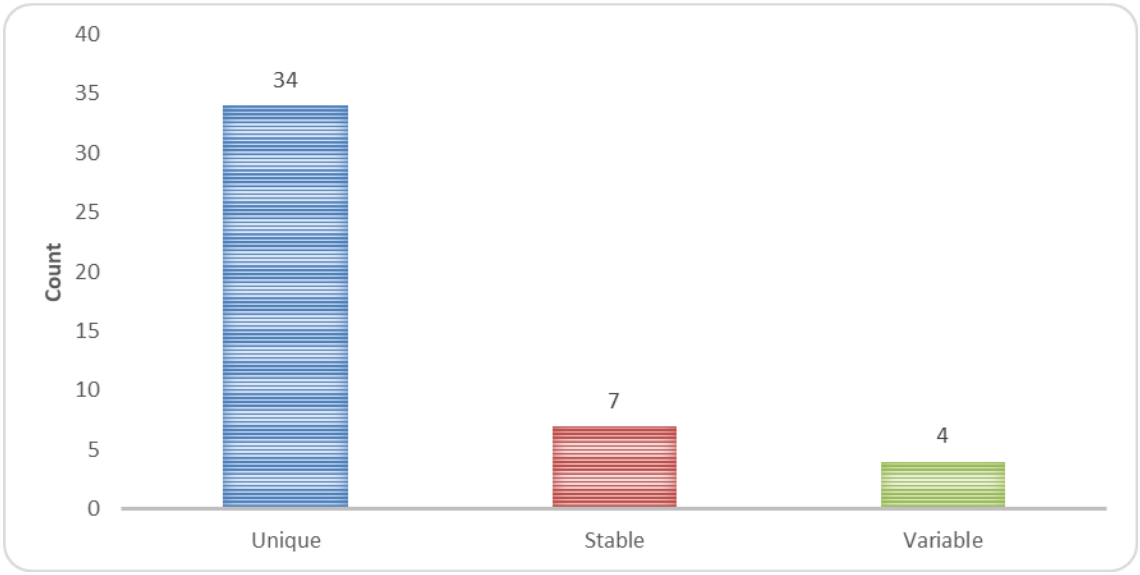

Figure Results and Discussion.10 Cluster Types

For further investigation of the cluster behavior, an analysis was performed regarding the diversity of JA3S fingerprints within each JARM cluster. It was identified that some clusters contain only a single JA3S value, indicating highly stable behavior, while others contain multiple values, reflecting variability. This demonstrates that JARM provides high-level grouping, while JA3S reveals internal differences. Variability may arise from configuration changes or load balancing etc.

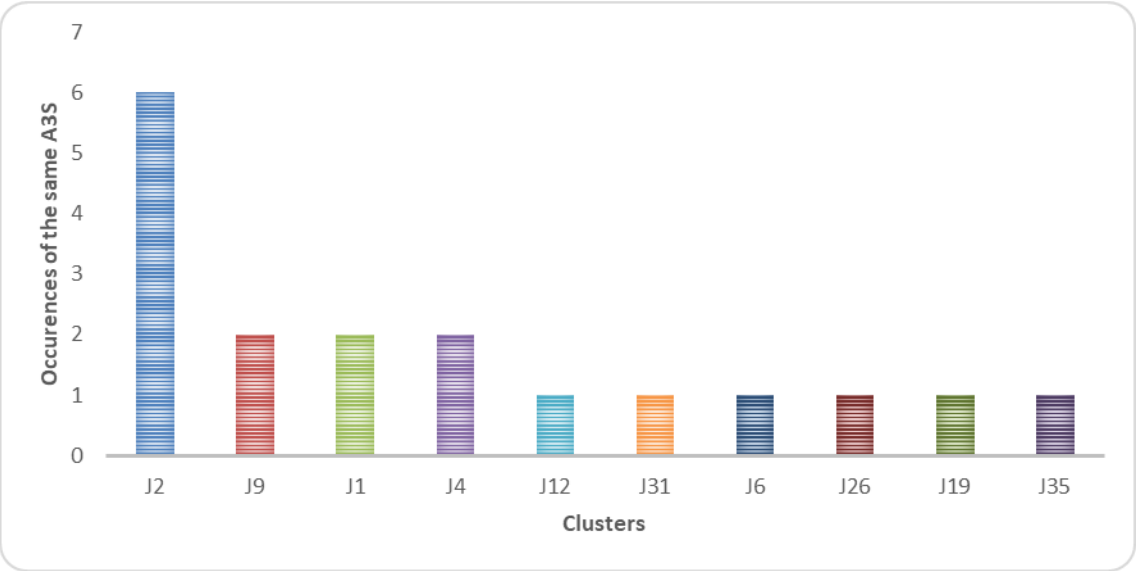

Figure Results and Discussion.11 JA3S Diversity

Moreover, the dataset was divided into the top five most frequent JARM clusters and the remaining clusters. The results show that the top five clusters have 55.65% of the dataset, while the remaining clusters account for 44.35%. This indicates that although a subset of TLS configurations appears more frequently, the dataset is not overwhelmingly dominated by a small number of clusters.

The relatively high percentage of the top clusters suggests the presence of infrastructure reuse, where similar TLS configurations are deployed across multiple endpoints, which may be associated with standardized setups, shared hosting environments, or the repeated use of common frameworks. At the same time, the remaining 44.35% demonstrates diversity in TLS behavior, indicating that many endpoints operate with distinct or less common configurations, which pinpoint to the fact that malicious infrastructure is not uniform, but instead consists of both widely reused and unique deployments.

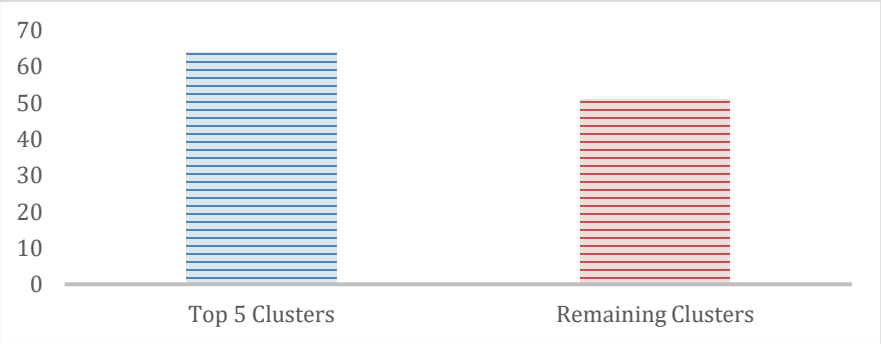

Figure Results and Discussion.12 Distribution of JARM Clusters (Top 5 vs Remaining Clusters)

4.3 Experimental Evaluation of C2 Infrastructure

To further investigate the behavior of TLS fingerprints under controlled conditions, additional experiments were conducted using command-and-control (C2) frameworks. Specifically, Cobalt Strike and Sliver were deployed in a controlled laboratory environment, and their corresponding JARM fingerprints were captured under different infrastructure configurations.

Initially, both frameworks were executed without any changes to their configuration. Under these conditions, each framework produced a distinct JARM fingerprint, indicating that JARM is capable of differentiating between server-side implementations and configurations.

Furthermore, a reverse proxy (Nginx) was introduced in front of both C2 servers. After this modification, both frameworks produced an identical JARM fingerprint, which differed from their original values, which indicates that the observed fingerprint reflect to Nginx and not on the underlying C2 framework.

The above finding indicates that JARM captures the externally exposed TLS behavior rather than the internal backend infrastructure, thus in environments that use reverse proxies or load balancers the fingerprint exhibits the intermediary layer.

Further changes performed on the TLS configuration of the Nginx proxy, specifically by changing twice the supported protocol versions and cipher suites. As a result, a new JARM fingerprints were produced respectively. The fingerprints remained identical across both frameworks but were different from the previous ones.

	Cobalt Strike	**Sliver**
Default	2ad2ad16d2ad2ad00042d42d00042ddb04deff a1705e2edc44cae1ed24a4da	00000000000000000043d43d00043de2a97ea bb398317329f027c66e4c1b01
Nginx	29d29d00029d29d00042d42d00000000f78d2dc0ce6e5bbc5b8149a4872356	
Cipher &TLS	08d08d00008d08d08c08d08d08d08d36e918f9500670cf2180a0ac86cc7648	
Cipher &TLS	00000000000000000042d42d0000009535d5979f591ae8e547c5e5743e5b64	

This further shows that JARM is very sensitive to TLS configuration changes and that minor changes could drastically change fingerprint values. Therefore, JARM reflects the behavior of the exposed endpoint, not the application itself.

This rationale in the controlled explanation clarifies why several endpoints in the dataset have the same JARM fingerprint. Experimental results show that such similarities may be due to shared infrastructure components.

.

4.4 Key Findings

Accordingly, TLS fingerprint behavior concerning underlying infrastructure should not be considered a static identifier but rather as a deployable feature depending on deployment practices, configuration options, and intermediate components.

Hence, the most meaningful result derived from the data analysis is partial infrastructure reuse. While some JARM clusters are more frequent and make up a sensible part of the samples, they are not the majority in the dataset. Indeed, the top clusters take a little over half of the observed data, with a considerable portion distributed amongst less frequent clusters. This means that standard configurations are undoubtedly reused in malicious infrastructure, but diversity is present as well. This shows that attackers want to be efficient in reusing known-good configurations but, at the same time, want to be diverse for evasion or operational purposes.

Another crucial point is "internal variability within clusters." Indeed, the JA3S analysis showed that an endpoint with a JARM value does not always respond the same way to TLS. More often than not, one JARM cluster consists of several JA3S fingerprints. This means that high-level TLS configurations can lead to different server-side behaviors. This internal variability may be due to various software implementations, configuration parameters, and intermediate infrastructure components. JARM provides a coarse-grained view of infrastructure, while JA3S gives more profound insights into its behavior.

A big takeaway of this work is how infrastructure components affect TLS fingerprints, especially through testing command-and-control frameworks. The results showed that when a reverse proxy is placed in front of different C2 servers, they produce the same JARM fingerprint, hiding the differences between them. In simple terms, TLS fingerprinting may reflect the behavior of the intermediate layer (like the proxy) instead of the actual backend systems.

Further experiments included an analysis where configuration parameters [such as supported protocols and cipher suites) were modified, and new JARM fingerprints were obtained. More interestingly, fingerprints became identical across different C2 frameworks when the configuration was set to match one type of framework, regardless of which C2 framework; thus, it reiterates that JARM represents behavior influenced by system configuration rather than system identity. This is significant since attackers/administrators may spoof fingerprint values based solely on changing certain configurations.

Since the above-mentioned evidence relies on matching publicly available data collected and experimental traffic, uncertainty about the above-mentioned claims would instead refer precisely to this. The match means that the lab can reproduce realistic TLS behavior in the wild. This also means that controlled experiments can replace field studies.

Therefore, such findings generally call for caution in interpreting TLS fingerprinting. First, such features are sensitive to environmental changes. Second, they can reveal information about infrastructure similarities and reuse. In this respect, TLS fingerprinting will be better considered a behavioral analysis tool, not a systems identification tool.

5 Conclusion and Future Work

5.1 Conclusion

This thesis thus explores the use of TLS fingerprinting techniques, JARM and JA3S, in analyzing malware-related infrastructure. It was thus based on combining publicly available network data with controlled execution of malware in an isolated environment to try to better understand how TLS behavior reflects infrastructure characteristics.

The study shows that TLS fingerprinting can provide useful information about encrypted communication even without decrypting the data. For example, analysis of JARM clusters showed that a subset of TLS configurations is widely shared across many endpoints, implying infrastructure reuse. In contrast, much of the dataset exhibits diverse TLS behavior, reflecting unique or infrequently used configurations.

Rule-based clustering allows for structured interpretation of such patterns. JARM was used as a grouping mechanism, while JA3S was used as a refinement layer to distinguish consistent behaviors from variable behaviors within a given cluster. Indeed, this multi-layer approach proved highly effective in showing both similarities and differences in TLS behaviors across the dataset.

Another vital contribution of this research is the experimental validation of TLS fingerprint behavior within C2 frameworks. The results suggest that when a reverse proxy is placed in front of different C2 servers, the same JARM fingerprint is generated, effectively hiding the underlying infrastructure. Any modification in the TLS configuration produces entirely different JARM values, thus showing that fingerprinting is highly sensitive to changes in the configuration.

This, therefore, validates that TLS fingerprints should not be taken as an identifier for a system but rather as an externally observable behavior indicator. While they can indicate patterns of infrastructure reuse and similarity, deployment choices, intermediary components, and configuration options influence them.

Overall, the present study found that TLS fingerprinting is a powerful but highly context-dependent technique. If used correctly, it will open up much more visibility into encrypted traffic and help analyze malicious infrastructure. However, the practitioner must be aware of the limitations to avoid incorrect interpretation.

5.2 Future Work

One potential extension is the use of larger and more diverse datasets, and longer observation periods. This would allow for a more comprehensive analysis of TLS behavior and improve the robustness of the findings.

Another area for future work is the integration of machine learning techniques. While this study focused on rule-based analysis, machine learning models could be used to automate clustering, classification, and anomaly detection.

Further research could also explore the impact of different command-and-control frameworks and infrastructure configurations on TLS fingerprints. This includes examining how changes in deployment, server software, or network architecture influence fingerprint values.

In addition, longitudinal analysis could be conducted to study how TLS fingerprints evolve over time. This would provide insight into the stability of fingerprints and the dynamics of infrastructure changes.

Finally, combining server-side and client-side fingerprinting techniques could provide a more comprehensive understanding of encrypted traffic, enabling more accurate and informative analysis.

6 Bibliography

[1] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, Aug. 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446

[2] B. Anderson and D. McGrew, “Identifying Encrypted Malware Traffic with Contextual Flow Data,” in Proc. ACM Workshop on Artificial Intelligence and Security (AISec), Vienna, Austria, 2016
.

[3] J. Althouse, “JA3: TLS Client Fingerprinting for Malware Detection,” Salesforce Engineering, 2017. [Online]. Available: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967

[4] Salesforce, “JARM: Active TLS Server Fingerprinting,” Salesforce Engineering, 2020. [Online]. Available: https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a

[5] B. Anderson, S. Paul, and D. McGrew, “Deciphering Malware’s Use of TLS (without Decryption),” Journal of Computer Virology and Hacking Techniques, vol. 14, no. 3, pp. 195–211, 2018.

[6] B. Anderson and D. McGrew, “Machine Learning for Encrypted Malware Traffic Classification,” in Proc. ACM Workshop on Artificial Intelligence and Security (AISec), 2016.

[7] F. Holland, N. Schmitt, and F. Guillemin, “TLS Traffic Classification Using Machine Learning,” in Proc. IEEE Conference on Communications (ICC), 2020.

[8] E. Rescorla, “TLS 1.3 Overview and Deployment Considerations,” IETF Draft, 2019.

[9] D. McGrew et al., “Randomization of TLS Extensions and Its Impact on Fingerprinting,” in Proc. NDSS Workshop, 2017.

[10] M. Durumeric et al., “The Security Impact of HTTPS Interception,” in Proc. Network and Distributed System Security Symposium (NDSS), 2017.