博士論文

Identification of Encrypted C2 Communication Used by Malware

Atsushi KANDA
神田 敦

情報セキュリティ大学院大学
情報セキュリティ研究科
情報セキュリティ専攻

2026 年 3 月

Identification of Encrypted C2 Communication Used by Malware

Atsushi KANDA

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy in Informatics
at the Graduate School of Information Security

INSTITUTE of INFORMATION SECURITY, Japan

March 2026

List of Publications

This thesis is based on the below publications.

Peer-Reviewed Journal Articles

[1]Atsushi Kanda, Masaki Hashimoto, and Takao Okubo. “Can We Create a TLS Lie Detector?” In: *Journal of Information Processing* 32 (2024), pp. 1114–1124.

[2]Atsushi Kanda, Masaki Hashimoto, and Takao Okubo. “An Analysis of TLS Parameter Variation in Malware C2 Communication”. In: *Journal of Information Processing* 33 (2025), pp. 1119–1127.

Peer-Reviewed Papers in Proceedings of International Conference

[3]Atsushi Kanda and Masaki Hashimoto. “Identification of TLS Communications Using Randomness Testing”. In: *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2021, pp. 1099–1106.

Non-Peer-Reviewed Papers

[4]Atsushi Kanda and Masaki Hashimoto. “乱数性を用いた TLS 通信の識別”. In: **コンピュータセキュリティシンポジウム** *2019* **論文集** (2019), pp. 683–690. URL: https://cir.nii.ac.jp/crid/1050011097135380736.

[5]Atsushi Kanda and Masaki Hashimoto. “TLS 嘘発見器は作れるか”. In: **コンピュータセキュリティシンポジウム** *2023* **論文集** (2023), pp. 1365–1372. URL: https://cir.nii.ac.jp/crid/1050297969507812864.

[6]Atsushi Kanda and Masaki Hashimoto. “プロトコル構造を保持した特徴表現を用いた TLS パラメータの変異解析”. In: **コンピュータセキュリティシンポジウム** *2025* **論文集** (2025).

Abstract

In today's Internet, where encryption is the default option for communication, identifying malicious Command and Control (C2) communications has become a critical challenge. While the widespread adoption of protocols like HTTPS has improved privacy and security for legitimate users, it has also enabled attackers to conceal their activities. Recent industry reports show that the majority of malware now leverages TLS, making traditional security solutions that rely on payload inspection less effective.

This thesis addresses two major technical obstacles in identifying encrypted C2 communication: "Parameter Variation" and "Parameter Spoofing."

The first challenge regarding "Parameter Variation" is that natural changes in handshake metadata undermine the reliability of traditional hash-based TLS fingerprinting methods like JA3 and JA4. By leveraging an unprecedented long-term dataset spanning approximately 11.5 years, this thesis defines and empirically demonstrates two distinct phenomena: "Parameter Fluctuation," which refers to short-term, session-level changes, and "Parameter Drift," which describes long-term evolution driven by software updates. The analysis reveals that these variations are prevalent across major malware families such as Trickbot and Cobalt Strike, leading to significant divergence in existing fingerprints even with minor parameter differences.

To overcome the challenge of Parameter Variation, the thesis introduces two structural techniques: "Compacted Protocol Representation (CPR)" and "Structural Edit Distance (SED)." CPR provides a structured, compact way to represent TLS handshake parameters, preserving their hierarchical relationships and enabling precise comparison. SED quantifies the difference between two CPR objects by counting the minimal number of structural edits needed to transform one into the other. These methods allow for robust identification of encrypted communications, even when parameters fluctuate or drift over time. This thesis evaluates several identification strategies using a reproducible dataset designed to induce parameter fluctuations. The experiments compare methods such as Exact Match, Nearest Neighbor, and a novel strategy called "Exact Match after De-fluctuation

(EM-D)," which selectively filters out known fluctuating parameters before matching. The results demonstrate that the EM-D strategy achieves superior performance and robustness, particularly in environments with sparse databases of known fingerprints, by effectively isolating the client identity's invariant signal from parameter noise.

The second challenge, Parameter Spoofing, involves attackers deliberately manipulating or fabricating handshake parameters (also known as "FakeTLS"). To address this, the thesis proposes a payload-centric detection framework that analyzes the statistical randomness of encrypted application data using entropy and features from the Monobit and Runs tests (NIST SP 800-22). The "TLS Lie Detector" is trained only on benign TLS payloads and can identify spoofed communications by detecting deviations in payload randomness. Experiments show that this method is highly effective at detecting FakeTLS using weak ciphers (like XOR).

The proposed techniques are designed for practical integration into security operation workflows, enhancing SIEM/IDS alerts and supporting threat hunting. By combining structural fingerprinting and payload-randomness-based features with other contextual information, organizations can improve detection accuracy and reduce alert fatigue, even as encrypted traffic becomes dominant without relying on decryption.

Acknowledgment

First and foremost, I express my heartfelt gratitude to Associate Professor Masaki Hashimoto for his unwavering support throughout this research. Since enrolling at the Institute of Information Security (IISEC) in October 2017, he has patiently guided me for over 8.5 years. During my leave of absence due to work circumstances, he maintained regular contact and provided sincere counsel. After moving from IISEC to Kagawa University, he continued to offer advice. Without his thoughtful support, I doubt I would have been able to complete my doctoral program.

I am also deeply grateful to Professor Takao Okubo, who took over as my supervisor after April 2024. Although taking on the role of supervisor may have been an additional burden, he graciously accepted and provided invaluable guidance.

Additionally, I appreciate the members of my doctoral thesis examination committee, Professor Atsuhiro Goto and Professor Seiko Arita, for their insightful advice. Their incisive comments from diverse expertise led me to develop a grounded, substantial research study.

I also thank the anonymous reviewers who provided valuable feedback on my papers.

I extend my gratitude to my colleagues at the Institute of Information Security. During daily lab life, we shared struggles, encouraged each other, and had meaningful discussions. I was fortunate to have Mr. Ryu Sato, whose research theme was related to mine, for intensive discussions during my final research stages.

I am thankful to NTT DOCOMO BUSINESS, Inc. (formerly NTT Communications Corporation) for supporting me as a working professional doctoral student. Balancing research with full-time work was challenging. I was lucky to have understanding colleagues and superiors, who helped me complete my doctoral program. I am especially indebted to my former supervisor, Dr. Mitsuhiro Hatada, and my current supervisor, Mr. Shingo Kashima, for their daily consideration regarding my work responsibilities. I could not have achieved this without their guidance during times I nearly gave up on continuing my doctoral program.

Lastly, I extend my deepest gratitude to my family: my wife Yukino, my son Ryuto, my daughter Konoha, and my dogs Daifuku and Anko. Not only did they show understanding of my research endeavors and support my daily life, but they also gave me a sanctuary by providing a place to return to. They have always been my greatest emotional support.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Cyber Threat in the Era of Default-encrypted Communication

Driven by growing awareness of privacy and security, the adoption of encrypted Internet communication, most notably HTTPS, has accelerated dramatically in recent years. According to Google's Transparency Report [1], as of December 2025, more than 98% of web pages loaded in Chrome on Windows and macOS are delivered over HTTPS. Similarly, statistics published by Let's Encrypt [2] indicate that the proportion of web pages loaded via HTTPS in Firefox exceeds 85% globally. Encryption has become a default expectation rather than an exception, marking the advent of an era of default encryption.

However, the benefits of encrypted communication are not limited to legitimate users. Threat actors, including cybercriminals and state-sponsored attackers, also leverage encryption to conceal malicious activities. Sophos reported that the percentage of malware using TLS for outbound communication doubled from 23% in 2020 to 46% in 2021 [3]. Palo Alto Networks observed that 12.91% of traffic generated by malware during sandbox analysis in 2022 was encrypted using SSL, with projections indicating continued growth [4]. Furthermore, Zscaler reported that between October 2023 and September 2024, it blocked 32.1 billion malicious communications over TLS, accounting for 87.2% of all threats detected [5]. Encrypted traffic is inherently more challenging to analyze than plain-text communication, giving attackers a significant advantage in evading detection. Thus, while the widespread adoption of encryption enhances overall security, it simultaneously creates opportunities for adversaries to exploit.

One promising approach to identifying encrypted communication is to analyze the TLS handshake that precedes the establishment of an encrypted session. Unlike the encrypted payload, handshake messages are transmitted in plain-text and contain a variety of parameters—such as cipher suites, extensions, and version information—that reflect the implementation details of the communicating endpoints. These parameters can serve

as valuable fingerprints for distinguishing legitimate clients from malware. Prior research has leveraged TLS fingerprinting techniques such as JA3 [6] and JA4 [7] to detect malicious TLS traffic, demonstrating their effectiveness in large-scale monitoring environments. However, these approaches face significant limitations when confronted with natural variability and evasive strategies in handshake parameters.

Specifically, two critical challenges arise:

Parameter Variation

Parameter variation refers to natural inconsistencies in TLS handshake parameters that occur across sessions or over time, regardless of the malware author's intent. These variations can manifest in two forms: short-term changes between consecutive sessions and long-term drift caused by software updates, library changes, or evolving configurations. Importantly, this phenomenon is not limited to malicious software; it also occurs in benign applications. Such variability significantly reduces the reliability of static fingerprints.

Parameter Spoofing

Parameter spoofing refers to deliberate techniques where malware manipulates or fabricates TLS handshake parameters to evade detection. Although such behavior is not widely reported in malware communications, one documented example is FakeTLS [8], where the handshake headers mimic TLS while the actual payload does not conform to the TLS protocol. This extreme approach breaks protocol assumptions and renders traditional fingerprinting ineffective.

1.2 Contribution

The primary contribution of this thesis is the development of novel identification technologies that overcome the limitations of existing methods for identifying encrypted Command and Control (C2/C&C) communications used by malware. This thesis directly addresses the two critical challenges that render traditional fingerprinting techniques ineffective: **Parameter Variation** and **Parameter Spoofing**. The specific contributions are summarized in the following three points:

Clarification of TLS parameter variation characteristics through an unprecedented longitudinal analysis of real-world malware C2

This thesis highlights fundamental challenges regarding "**Parameter Variation**" by analyzing a large-scale dataset (the MTA dataset [9]) spanning approximately 11.5 years, a duration unprecedented in prior research. Specifically, the study defines and demonstrates two distinct phenomena: "**Parameter Fluctuation**," in which parameters change between sessions, and "**Parameter Drift**," in which parameters evolve over long periods due to software updates. By analyzing major malware families such as Trickbot, IcedID, and Cobalt Strike, this research provides empirical evidence for why traditional static fingerprinting methods (e.g., JA4) fail to maintain reliability over time.

Development of robust structural fingerprinting identification techniques

This thesis establishes a more precise and robust identification algorithm and evaluation methodology against the "**Parameter Variation**." In this thesis, "robustness" is defined as "the resilience to accurately identify clients despite the presence of unknown parameter variations not represented in the knowledge base, while minimizing false positives." This thesis introduces the "**Compacted Protocol Representation (CPR)**" for the expression of structural parameters and the "**Structural Edit Distance (SED)**" for quantifying differences between parameters, enabling the precise capture and quantification of even minor variations in protocol parameters. This thesis proposed the "**Exact Match after De-fluctuation**" strategy, which removes known fluctuating parameters before matching, and showed that it provides the highest identification performance in en-

vironments with many unknown fingerprints. Furthermore, it established a reproducible method for generating a dataset to intentionally induce parameter fluctuations, enabling the reliable verification of the robustness of identification methods.

Establishment of a TLS communication identification methodology based on the statistical randomness of encrypted payloads

This thesis establishes a novel framework for characterizing encrypted communications by analyzing the statistical randomness of application data, directly addressing the challenge of **Parameter Spoofing**, in which protocol metadata is intentionally manipulated. By evaluating randomness tests defined in NIST SP 800-22 [10], this study identified that statistics from the Monobit Test and the Runs Test are uniquely effective for identifying TLS configurations without relying on the handshake parameters. This thesis developed a novelty detection model that unmasks FakeTLS by identifying deviations in payload randomness. This method is particularly effective at detecting low-randomness ciphers (e.g., XOR) used by threat actors such as Lazarus to spoof TLS [11, 12, 13]. Unlike existing methods that require TLS-specific features, this payload-randomness-based approach itself can identify spoofed communications even when the TLS handshake is omitted entirely.

1.3 Structure of the Thesis

Chapter 2 provides the background of this research, covering threat actors who leverage encrypted communication, traditional security solutions, and the concept of active defense through threat hunting.

Chapter 3 presents the preliminaries needed to understand the thesis's technical foundations. It details the history and protocol design of SSL/TLS and introduces the mathematical concepts of randomness in cryptography, specifically focusing on Shannon entropy and the NIST SP800-22 statistical test suite.

Chapter 4 reviews related work in the field of malicious encrypted traffic identification. It categorizes existing approaches into signature-based and machine-learning-based identification and surveys trends in malware TLS communication.

Chapter 5 addresses the first primary challenge: Parameter Variation. By analyzing a large-scale dataset spanning approximately 11.5 years, this chapter clarifies the long-term

characteristics of TLS parameters in real-world malware. It defines two new concepts, "Parameter Fluctuation" and "Parameter Drift," and discusses the limitations of existing fingerprinting technologies in handling these phenomena. To overcome these issues, this chapter introduces "Compacted Protocol Representation (CPR)" for the expression of structural parameters and "Structural Edit Distance (SED)" for quantifying differences between parameters. Furthermore, it presents evaluation experiments on a more precise and robust identification algorithm, including the "Exact Match after De-fluctuation" strategy and a reproducible method for generating datasets that intentionally induce parameter fluctuations to verify the robustness of identification methods.

Chapter 6 addresses the second primary challenge: Parameter Spoofing. It establishes an identification methodology based on the statistical randomness of encrypted payloads to unmask "FakeTLS". This chapter proposes the "TLS Lie Detector," a novelty-detection model that uses entropy and features derived from the Monobit and Runs tests to identify spoofed communications, even when handshake parameters are manipulated or omitted.

Chapter 7 provides a comprehensive discussion of the findings, synthesizing insights from analyses of parameter variations and spoofing countermeasures.

Chapter 8 concludes the thesis by summarizing the primary contributions and outlining potential directions for future work in the development of robust encrypted communication identification technologies.

Chapter 2
Background

2.1 Threat Actors using Encrypted Communication

The adoption of encrypted communication by threat actors has been increasingly reported since the late 2010s. The upward trend is corroborated by various security companies. Cisco reported that the volume of malware utilizing encrypted network communications approximately tripled between 2016 and 2017 [14]. Sophos documented that the proportion of malware using TLS for outbound communication doubled from 23% in 2020 to 46% in 2021 [3]. Palo Alto Networks observed that 12.91% of traffic generated by malware during sandbox analysis in 2022 was encrypted using SSL, with projections indicating continued growth in the following years [4].

The trend has remained prevalent in recent years. Zscaler reported that between October 2023 and September 2024, it blocked 32.1 billion malicious communications over TLS, accounting for 87.2% of all threats detected [5]. WatchGuard's recent 2025 Q2 report showed that 70% of all malware is now delivered via encrypted (TLS) connections [15].

This trend highlights attackers' growing reliance on encrypted communication to obfuscate their activities and remain stealthy. In particular, TLS remains a popular encryption technique for threat actors, not only because it can be easily concealed within large volumes of legitimate TLS traffic, but also because open-source libraries widely support its use.

2.2 Traditional Security Solutions against Encrypted Communication

In the default-encrypted era, where more than 98% of web traffic is delivered over HTTPS, there is a significant challenge to middlebox-based security solutions, such as firewalls, Intrusion Detection Systems (IDS), Intrusion Prevention Systems (IPS), and Deep Packet Inspection (DPI). Many of the security functions provided by these middle-

boxes are predicated on the ability to inspect the contents of application-layer payloads. However, when dealing with encrypted traffic, these solutions cannot fully leverage their capabilities because the communication remains opaque. Specifically, in the context of web application security, filtering based on HTTP headers, URLs, and payload content becomes impossible. Furthermore, behavior-based detection and blocking mechanisms are rendered ineffective as they cannot capture the granular details of the underlying communication.

To mitigate these issues, TLS inspection (also known as TLS interception) is often employed as a countermeasure. TLS inspection involves terminating the TLS session at the middlebox to decrypt the traffic, allowing traditional security functions to operate on the plaintext data before re-encrypting it for the final destination.

However, "TLS inspection is not a cure-all," as the National Security Agency (NSA) claimed in its advisory [16]. TLS inspection faces several critical limitations:

- **Performance constraints**: The processes of decryption and re-encryption impose substantial computational overhead. In environments with active TLS inspection, network performance inevitably degrades for middleboxes with identical hardware specifications. Consequently, it is difficult to apply to latency-sensitive applications. Achieving the required throughput often necessitates upgrading to higher-grade, significantly more expensive hardware.
- **Privacy concerns**: By decrypting confidential communications, TLS inspection risks exposing private or highly sensitive information. This creates significant legal and ethical barriers, making the approach inapplicable to environments used by third parties or the general public where no explicit contractual agreement regarding privacy exists.
- **Conflicts with existing security features**: TLS inspection fundamentally operates as a trusted man-in-the-middle (MITM). This design conflicts with security technologies designed to ensure end-to-end session integrity. A prominent example is certificate pinning, a feature that restricts an application to establishing TLS sessions only with specific, trusted certificates or public keys. In applications that use certificate pinning, TLS inspection causes session establishment to fail, requiring these applications to be excluded from inspection.

- **Client incompatibility**: Implementing TLS inspection requires specific client-side configurations, such as installing custom root certificates. However, many environments, most notably IoT devices and network appliances, lack the interface or capability to support such configurations.

While TLS inspection remains a vital strategy for maintaining security levels in the era of ubiquitous encryption, it is evident that a significant portion of encrypted traffic inevitably bypasses these measures. This underscores the ongoing and critical need for novel identification technologies that can distinguish malicious activity without relying on decryption, essentially identifying threats while maintaining the integrity of the encrypted flow.

2.3 Active Defense with Threat Hunting

As cyber attacks grow increasingly sophisticated, traditional defensive measures that rely solely on automated detection have reached their functional limits. To address this challenge, defenders must consider more advanced and evolving tactics within a strategic framework, such as Lee's "Sliding Scale of Cyber Security" [17].

While "Passive Defense"—systems added to the architecture to provide consistent protection without constant human interaction—remains a fundamental component, it can be bypassed by determined, well-resourced adversaries. Consequently, organizations must shift their focus toward "Active Defense". Active Defense is defined as "the process of analysts monitoring for, responding to, learning from, and applying their knowledge to threats internal to the network." In "Sliding Scale of Cyber Security," direct actions against attackers are classified as "Offensive" and are clearly distinguished from Active Defense.

A key operational tactic of this Active Defense approach is "Threat Hunting". It is defined as "the process of proactively and iteratively searching through networks to detect and isolate advanced threats that evade existing security solutions" [18]. The ultimate goal of these advanced defensive tactics is not merely to react to alerts, but to evolve the security posture over time by internalizing lessons learned from encounters with adversaries. From this perspective, Threat Hunting is often discussed in conjunction with

"Detection Engineering," a process of improving automated detection capability.

It is important to note that the Active Defenses are most effective and resource-efficient when built upon a solid architecture and robust Passive Defenses.

Chapter 3
Preliminaries

3.1 SSL/TLS

3.1.1 History of SSL/TLS

SSL (Secure Sockets Layer) was originally developed by Netscape as a secure protocol positioned in the session layer of the OSI reference model. Its primary purpose is to secure application communications, and it is utilized across various applications, including web (HTTP), email (SMTP, POP, IMAP), and domain name resolution (DNS). While the protocol was developed up to SSL 3.0 [19], the IETF (Internet Engineering Task Force) subsequently standardized its successor and renamed it to TLS (Transport Layer Security), leading to the standardization of versions 1.0 [20], 1.1 [21], and 1.2 [22]. In 2018, TLS 1.3 was standardized [23], marking a significant evolution from its predecessors.

The sequential deprecation of outdated and vulnerable versions is also underway. Following the publication of the RFC document deprecating SSL 3.0 in 2015 [24], TLS 1.0 and 1.1 were deprecated in 2021 [25]. In response to these recommendations, major browsers have already disabled versions ranging from SSL 3.0 to TLS 1.1. The National Institute of Standards and Technology (NIST) published a guideline (SP800-52 Rev2) [26] in 2019 for federal systems to transition to TLS 1.2 and 1.3. In Japan, aligned with these global trends, the Information-technology Promotion Agency (IPA) published the "TLS Cryptographic Setting Guidelines" [27] to promote the transition to TLS 1.2 and 1.3.

As of 2025, TLS 1.2 and 1.3 have become the mainstream protocols for encrypted communication on the Internet. According to SSL Pulse [28], statistics for the top 150,000 HTTPS-enabled websites showed support rates of 100% for TLS 1.2 and 75.3% for TLS 1.3 in June 2025. They were 97.4% and 30.6%, respectively, in June 2020. This demonstrates how rapidly TLS 1.3 is gaining widespread adoption.

3.1.2 TLS 1.3

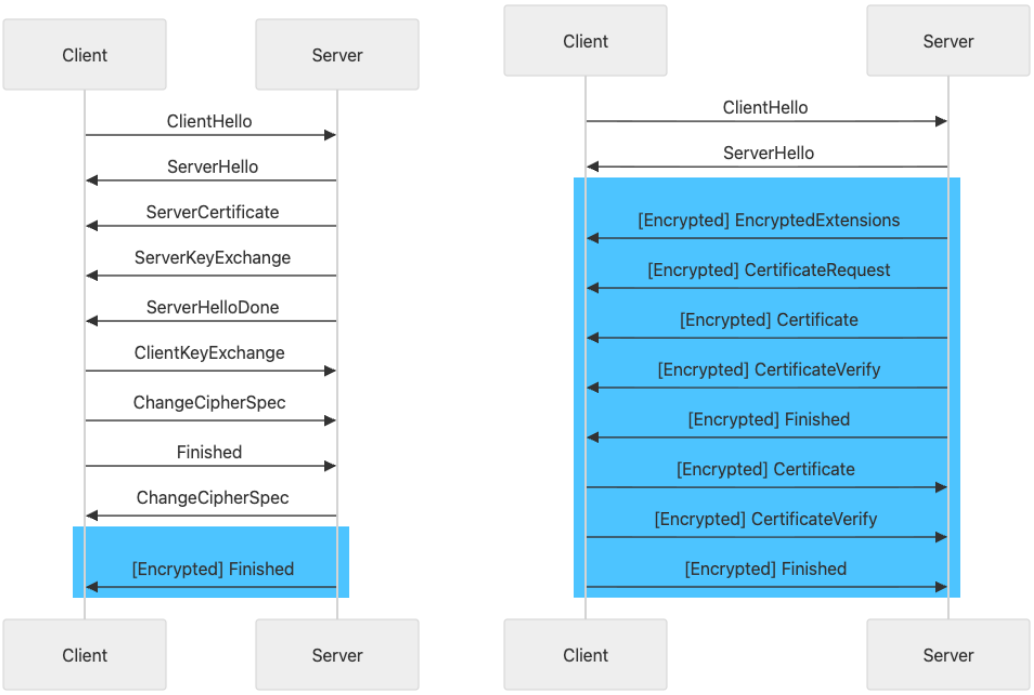
Unlike previous version updates that primarily addressed specific vulnerabilities or adjusted cipher strengths, TLS 1.3 prioritized security over backward compatibility. This version fundamentally overhauled the protocol design by abolishing compromised cryptographic techniques and involving cryptography researchers from the design phase to theoretically verify its security. The key changes introduced in TLS 1.3 compared to TLS 1.2 include:

- Earlier encryption in the handshake
- Exclusive use of AEAD (Authenticated Encryption with Associated Data) algorithms
- Deprecation of all key exchange mechanisms that do not provide Forward Secrecy
- Abolishment of compression functions within the handshake
- Introduction of HKDF (HMAC-based Extract-and-Expand Key Derivation Function) for key derivation

From the threat-detection and threat-hunting perspectives, the most critical change in TLS 1.3 is its early encryption in the handshake. In TLS 1.3, the encryption of handshake messages begins earlier than in previous versions; consequently, only the ClientHello and ServerHello messages are transmitted in plaintext (Figure 3.1). While the contents of server certificates often serve as effective features for malicious communication detection, unlike TLS 1.2, the server certificate in TLS 1.3 is encrypted and does not flow in plaintext. To identify malicious TLS communications, it is essential to extract as many characteristics as possible from the ClientHello and ServerHello messages, which are the only parts of the TLS 1.3 handshake that are readable.

Looking more closely at the ClientHello and ServerHello messages, while TLS 1.3 maintains the same message format as TLS 1.2, several specific fields have changed significantly [22, 23] (Figure 3.2 and Figure 3.3). Since the message fields of the ClientHello and ServerHello are substantially identical, the following sections will focus exclusively on the ClientHello for the sake of brevity.

In TLS 1.2 and earlier, the `client_version` field at the beginning of the ClientHello message specifies the client's desired TLS protocol version for the session. In TLS 1.3, however, this field has been renamed to `legacy_version`, and its value is now fixed at



(a) TLS 1.2 **(b)** TLS 1.3

Figure 3.1. TLS Handshake sequence

0x0303 (the TLS 1.2 version number). The actual protocol version is now communicated through the `supported_versions` extension.

The `session_id` field, which in TLS 1.2 is used by the client to indicate a session ID for session resumption, has been renamed to `legacy_session_id` in TLS 1.3. Although it may still contain a non-empty value, this data is no longer utilized for the handshake process. This is because the session resumption functionality in TLS 1.3 has been merged into the pre-shared key mechanism.

Similarly, the `compression_methods` field, which conveys supported compression algorithms in TLS 1.2, has been renamed to `legacy_compression_methods` in TLS 1.3. Since compression functionality was removed in the new protocol design, this field is now statically set to exactly one byte "0," indicating "null" compression.

As described above, TLS 1.3 retains certain fields solely for compatibility, many of which now carry only fixed values. Numerous identification and fingerprinting techniques

designed prior to TLS 1.3 rely on these field values as key features. Consequently, as TLS 1.3 becomes the dominant standard, there is a significant concern that the effectiveness of these legacy methods will deteriorate due to the loss of discriminative information in these fields.

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<2..2^16−2>;
    CompressionMethod compression_methods<1..2^8−1>;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16−1>;
    };
} ClientHello;
```

(a) TLS 1.2

```
struct {
    ProtocolVersion legacy_version = 0x0303; /* TLS v1.2 */
    Random random;
    opaque legacy_session_id<0..32>;
    CipherSuite cipher_suites<2..2^16−2>;
    opaque legacy_compression_methods<1..2^8−1>;
    Extension extensions<8..2^16−1>;
} ClientHello;
```

(b) TLS 1.3

Figure 3.2. TLS ClientHello message structure in RFC's presentation language

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    select (extensions_present) {
        case false:
            struct {};
        case true:
            Extension extensions<0..2^16−1>;
    };
} ServerHello;
```

(a) TLS 1.2

```
struct {
    ProtocolVersion legacy_version = 0x0303; /* TLS v1.2 */
    Random random;
    opaque legacy_session_id_echo<0..32>;
    CipherSuite cipher_suite;
    uint8 legacy_compression_method = 0;
    Extension extensions<6..2^16−1>;
} ServerHello;
```

(b) TLS 1.3

Figure 3.3. TLS ServerHello message structure in RFC's presentation language

Table 3.1. A subset of TLS Extensions used in both TLS 1.2 and 1.3

Type value	Name	Reference
0	server_name	RFC 6066 [29]
5	status_request	RFC 6066 [29]
10	supported_groups	RFC 8422 [30]
13	signature_algorithms	RFC 5246 [22], RFC 8446 [23]
16	application_layer_protocol_negotiation	RFC 7301 [31]
21	padding	RFC 7685 [32]
35	session_ticket	RFC 5077 [33]
2570, and others	(GREASE)	RFC 8701 [34]

3.1.3 TLS Extensions in ClientHello/ServerHello

In TLS, many of its functions are implemented through Extensions. This section focuses on the Extensions and parameters that are particularly relevant to this thesis.

Extensions used in both TLS 1.2 and 1.3

The Extensions in Table 3.1 are the Extensions focused on in this thesis and are used in both TLS 1.2 and 1.3.

- **server_name**: The "server_name (SNI)" extension [29] allows a client to specify the hostname it wants to connect to, enabling servers to present the correct certificate for multi-hosted environments. Its data contains a list of server names.
- **status_request**: The "status_request" extension [29] enables a client to request certificate status information (such as OCSP stapling) from the server during the TLS handshake. Its data contains a combination of status type, a responder ID list, and request extensions. This mechanism allows the server to provide certificate revocation status directly, improving efficiency and privacy.
- **supported_groups**: The "supported_groups" extension [30] allows a client to indicate which elliptic curves or finite field groups it supports for key exchange during the TLS handshake. Its data contains a list of identifiers representing the supported

groups.

- **signature_algorithms**: The "signature_algorithms" extension [22, 23] allows a client to specify which signature and hash algorithm pairs it supports for digital signatures during the TLS handshake. Its data contains a list of identifiers, each representing a supported signature algorithm.
- **application_layer_protocol_negotiation**:
  The "application_layer_protocol_negotiation (ALPN) " extension [31] allows a client and server to negotiate which application protocol (such as HTTP/2 or HTTP/1.1) will be used over the TLS connection. Its data contains a list of protocol names. This enables efficient protocol selection during the TLS handshake without additional round-trip.
- **padding**: The "padding" extension [32] allows a client to add arbitrary bytes to the ClientHello message to adjust its length, helping to meet compatibility in some environments. Its data contains a sequence of padding bytes (all zeros) of the specified length.
- **session_ticket**: The "session_ticket" extension [33] enables clients and servers to use stateless session resumption in TLS by exchanging encrypted session tickets. Its data structure is opaque, and the server is responsible for interpreting its contents. This mechanism allows clients to resume previous TLS sessions efficiently without storing session state on the server.
- **GREASE**:
  The "Generate Random Extensions And Sustain Extensibility (GREASE)" [34] is not a functional extension, but it ensures TLS protocol extensibility by inserting reserved, dummy values into handshake messages, preventing implementations from making incorrect assumptions about valid extension types. For extension fields, the GREASE value uses the same format as regular extension fields, but the extension type is set to one of the reserved GREASE values, such as 2570 (0x0a0a) and 6682 (0x1a1a). The extension data is typically empty or contains dummy data. GREASE is also used in the Cipher suites list, supported_groups extension list, and signature_algorithms extension list.

Table 3.2. A subset of TLS Extensions used only in TLS 1.2

Type value	Name	Reference
11	ec_point_formats	RFC 8422 [35]
24	token_binding	RFC 8472 [36]

Extensions used only in TLS 1.2

The Extensions in Table 3.2 are the Extensions focused on in this thesis and are used only in TLS 1.2.

- **ec_point_formats**: The "ec_point_formats" extension [35] allows a client to indicate which elliptic curve point formats (such as uncompressed or compressed) it supports for key exchange. Its data contains a list of format identifiers. It is only used in TLS 1.2, as TLS 1.3 requires the uncompressed format exclusively.
- **token_binding**: The **token_binding** extension [36] allows a client and server to negotiate the use of Token Binding, which cryptographically ties security tokens to the TLS session to prevent token theft and replay attacks. Its data includes a Token Binding protocol version and a list of key parameters. As of writing, there is only a standard for TLS 1.2 and earlier, and there is no standard for TLS 1.3.

Extensions used only in TLS 1.3

The Extensions in Table 3.3 are the Extensions focused on in this thesis and are used only in TLS 1.3.

- **delegated_credential**: The "delegated_credential" extension [37] allows a server to delegate its authentication authority to short-lived credentials, enabling secure and flexible certificate management without exposing the main private key. Its data structure includes the delegated credential and a signature.
- **pre_shared_key**: The "pre_shared_key" extension [23] enables clients and servers to resume sessions or establish new connections using previously shared keys, allowing faster handshakes and 0-RTT data in TLS 1.3. Its data structure includes a list of PSK identities, obfuscated ticket ages, and binders for key confirmation.
- **early_data**: The "early_data" extension [23] enables a client to send application

Table 3.3. A subset of TLS Extensions used only in TLS 1.3

Type value	Name	Reference
34	delegated_credential	RFC 9435 [37]
41	pre_shared_key	RFC 8446 [23]
42	early_data	RFC 8446 [23]
43	supported_versions	RFC 8446 [23]
44	cookie	RFC 8446 [23]
45	psk_key_exchange_modes	RFC 8446 [23]
49	post_handshake_auth	RFC 8446 [23]
(17153)	application_settings	draft-vvv-tls-alps-01 [38]
65037	encrypted_client_hello	draft-ietf-tls-esni-25 [39]

data to the server immediately after the first handshake message, enabling 0-RTT (zero-round-trip time) communication in TLS 1.3. In ClientHello, its data is empty, only indicating the client's intent to send 0-RTT data.

- **supported_versions**: The "supported_versions" extension [23] allows a client to indicate all TLS protocol versions it supports, enabling the server to select the highest mutually supported version during the handshake. Its data contains a list of version identifiers. This extension is required for TLS 1.3 and ensures precise version negotiation between the client and the server.
- **cookie**: The "cookie" extension [23] allows a server to send a stateless cookie to the client. Its data consists of a variable-length opaque byte sequence representing the cookie value. The primary purposes of this extension are offload and DoS protection.
- **psk_key_exchange_modes**: The "psk_key_exchange_modes" extension [23] allows a client to indicate which key exchange modes it supports when using pre-shared keys. Its data contains a list of mode identifiers.
- **application_settings**: The "application_settings (ALPS)" extension [38] allows a client and server to exchange application-layer protocol settings, such as HTTP/3 configuration, during the TLS 1.3 handshake. Its data contains a sequence of opaque bytes representing the application-specific settings data. This extension is not a standard. It appears to be implemented within Chromium and was once submitted as an IETF Internet Draft, but it is currently in an expired status.

- **encrypted_client_hello**: The "encrypted_client_hello (ECH)" extension [39] enables a client to encrypt sensitive parts of the ClientHello message, such as the server name (SNI), to protect user privacy during the TLS 1.3 handshake. Its data structure contains a configuration identifier, an ephemeral public key, and an encrypted payload representing the inner ClientHello. This extension is not a standard, but an active Internet Draft.

3.2 Cryptography and Randomness

Encrypted data should ultimately appear random to a third party. In this thesis, we refer to the property of random numbers as "Randomness." Examples of randomness include equiprobability (the property that values appear with equal frequency) and irregularity (the property that the occurrence of a value is independent of its past values and there is no rule). Randomness is a metric for measuring cryptographic performance, and cryptographic systems that generate high-quality randomness are essential for strong cryptography.

3.2.1 Shannon Entropy (Average Information Content)

Shannon entropy (hereafter, simply “entropy”) is also known as the average information content. In the field of information theory (or communication theory), it is used as a measure of uncertainty or the improbability of events.

The entropy $H(X)$ of a random variable X, where the probability of $X = i$ is P_i, is defined as shown in Equation 3.1:

$$H(X) = -\sum_i P_i \log_2 P_i \tag{3.1}$$

As a simple example, we calculate the entropy per byte for the byte sequences shown in Figure 3.4. Each sequence is 256 bytes long, and each block represents one byte, taking values from 0 to 255.

For sequence (a), all values are zero, so the probability of 0 is 1, and all other values have probability 0. Substituting into Equation 3.1, we obtain Equation 3.2, and the entropy H_a of sequence (a) is 0. This means that sequence (a) carries an average of 0 bits of information per byte.

$$H_a = -1 \cdot \log_2 1 - 0 \cdot \log_2 0 - 0 \cdot \log_2 0 \cdots - 0 \cdot \log_2 0 = 0 \tag{3.2}$$

Sequence (b) alternates between 0 and 1, so the probabilities of 0 and 1 are $\frac{1}{2}$, and all other values have probability 0. Substituting into Equation 3.1, the entropy H_b is 1 (Equation 3.3).

1	2	3	···	254	255	256
0	0	0	···	0	0	0

(a) All zeros

1	2	3	···	254	255	256
0	1	0	···	1	0	1

(b) Alternating 0 and 1

1	2	3	···	254	255	256
0	1	2	···	253	254	255

(c) Increment by 1

Figure 3.4. Entropy calculation examples (byte sequences)

$$H_b = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} - 0\cdot\log_2 0\cdots - 0\cdot\log_2 0 = 1 \tag{3.3}$$

Similarly, for sequence (c), where each value from 0 to 255 appears exactly once, the entropy H_c is 8 (Equation 3.4).

$$H_c = -256\cdot\left(\frac{1}{256}\log_2\frac{1}{256}\right) = 8 \tag{3.4}$$

Entropy takes its lowest value when all values are identical (H_a) and its highest value when all values occur with equal probability (H_c). Equal probability (uniform distribution) is one of the properties of random numbers, and the closer a sequence is to randomness, the higher its entropy.

3.2.2 Random Number Testing and NIST SP 800-22

Random number testing determines whether a random number generator has passed or failed based on the randomness of the data it generates. Typical methods for random number testing include NIST SP 800-22 [40], Dieharder [41], and TestU01 [42]. These methods combine several statistical tests to comprehensively assess randomness.

NIST SP 800-22 specifies methods for testing random numbers, and the tools implementing the methods are also available. In NIST SP 800-22, a total of 15 different statistical testing methods for bit sequences are defined. Each test calculates the p-value, the probability that the bit sequence is generated by an ideal true random source. If the value exceeds the level of significance (1%), it is deemed to be random. However, because the random numbers each test focuses on differ in nature, it is recommended to combine several tests and run them on more than 1000 samples to make an overall judgment.

The following are detailed explanations of the seven statistical tests from NIST SP 800-22 (Rev. 1a) that are referenced in this thesis.

Frequency (Monobit) Test

The Frequency (Monobit) Test evaluates the randomness of a given bit sequence based on the frequency of occurrence of 0s and 1s. For a bit sequence of length n, denoted as $\epsilon = \epsilon_1, \epsilon_2, \cdots, \epsilon_n$, the p-value is computed using the following steps:

Step 1 Convert the bit sequence ϵ from (0,1) to a numeric sequence X of (-1,1), and compute the sum S_n of X_i ($X_i = 2\epsilon_i - 1 = \pm 1$):

$$S_n = X_1 + X_2 + \cdots + X_n$$

Step 2 Compute the test statistic S_{obs} from S_n:

$$S_{obs} = \frac{|S_n|}{\sqrt{n}}$$

Step 3 Calculate the p-value:

$$P\text{-value} = \boldsymbol{erfc}\left(\frac{S_{obs}}{\sqrt{2}}\right)$$

where $\boldsymbol{erfc}$ denotes the complementary error function:

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-u^2}\, du$$

Step 4 If the p-value is less than 1% (0.01), the sequence is considered non-random; otherwise, it is considered random.

Frequency Test within a Block

The Frequency Test within a Block evaluates the randomness of a given bit sequence by dividing it into N non-overlapping blocks of length M bits and analyzing the proportion of 1s in each block. For a bit sequence of length n, denoted as $\epsilon = \epsilon_1, \epsilon_2, \cdots, \epsilon_n$, the p-value is computed using the following steps:

Step 1 Divide the sequence into $N = \lfloor \frac{n}{M} \rfloor$ non-overlapping blocks. Any remaining bits are discarded and not used in subsequent calculations.

Step 2 For each block, compute the proportion of 1s, denoted as π_i $(1 \leq i \leq N)$:

$$\pi_i = \frac{\sum_{j=1}^{M} \epsilon_{(i-1)M+j}}{M}$$

Step 3 Compute the test statistic χ^2:

$$\chi^2(obs) = 4M \sum_{i=1}^{N} \left(\pi_i - \frac{1}{2} \right)^2$$

Step 4 Calculate the p-value:

$$P\text{-value} = \boldsymbol{igamc}\left(\frac{N}{2}, \frac{\chi^2(obs)}{2} \right)$$

where $\boldsymbol{igamc}$ denotes the incomplete gamma function:

$$igamc(a, x) = \frac{1}{\Gamma(a)} \int_x^{\infty} e^{-t} t^{a-1} \, dt$$

Step 5 If the p-value is less than 1% (0.01), the sequence is considered non-random; otherwise, it is considered random.

Runs Test

The Runs Test evaluates the randomness of a given bit sequence based on the number of "runs," where a "run" is defined as a consecutive sequence of identical bits. A run of length k refers to k consecutive identical bits, with different values immediately before and after the run. Note that the Runs Test is performed only if the sequence passes the Frequency Test beforehand.

For a bit sequence of length n, denoted as $\epsilon = \epsilon_1, \epsilon_2, \cdots, \epsilon_n$, the p-value is computed using the following steps:

Step 1 Compute the proportion of 1s in the sequence, denoted as π:

$$\pi = \frac{\sum_{j=1}^{n} \epsilon_j}{n}$$

Step 2 Verify the condition for passing the Frequency Test:

$$\text{Pass condition: } |\pi - \frac{1}{2}| < \tau, \quad \tau = \frac{2}{\sqrt{n}}$$

If this condition is not satisfied, the p-value is set to 0.0000 and the subsequent steps are skipped.

Step 3 Compute the test statistic $V_n(obs)$:

$$V_n(obs) = \left\{ \sum_{k=1}^{n-1} r(k) \right\} + 1$$

where $r(k)$ is defined as:

$$r(k) = \begin{cases} 0, & \epsilon_k = \epsilon_{k+1} \\ 1, & \epsilon_k \neq \epsilon_{k+1} \end{cases}$$

$V_n(obs)$ counts the boundaries between runs (i.e., transitions between 0 and 1), representing the total number of runs.

Step 4 Calculate the p-value:

$$P\text{-value} = \boldsymbol{erfc}\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)} \right)$$

Step 5 If the p-value is less than 1% (0.01), the sequence is considered non-random; otherwise, it is considered random.

Test for the Longest Run of Ones in a Block

The Test for the Longest Run of Ones in a Block evaluates the randomness of a given bit sequence by dividing it into N non-overlapping blocks of length M bits and analyzing the longest run of 1s within each block. For a bit sequence of length n, the p-value is computed using the following steps:

Step 1 Divide the sequence into $N = \lfloor \frac{n}{M} \rfloor$ non-overlapping blocks. Any remaining bits are discarded and not used in subsequent calculations. The block length M is determined based on n:

$$M = \begin{cases} 8, & 128 \leq n < 6272 \\ 128, & 6272 \leq n < 750000 \\ 10^4, & 750000 \leq n \end{cases}$$

Step 2 For each block, compute the longest run of 1s. Classify the longest-run values according to Table 3.4 and count the frequencies in each class.

Table 3.4. Classification of Longest Runs

		M		
		8	128	10^4
ν_i	ν_0	≤ 1	≤ 4	≤ 10
	ν_1	2	5	11
	ν_2	3	6	12
	ν_3	≥ 4	7	13
	ν_4		8	14
	ν_5		≥ 9	15
	ν_6			≥ 16

Step 3 Compute the test statistic χ^2 using K and π_i defined in Table 3.5:

$$\chi^2(obs) = \sum_{i=0}^{K} \frac{(\nu_i - N\pi_i)^2}{N\pi_i}$$

Step 4 Calculate the p-value:

$$P\text{-value} = \boldsymbol{igamc}\left(\frac{K}{2}, \frac{\chi^2(obs)}{2}\right)$$

Table 3.5. Parameters K and π_i for χ^2 Calculation

$M = 8$ $(K = 3)$	
Class	π_i
ν_0	$\pi_0 = 0.2148$
ν_1	$\pi_1 = 0.3672$
ν_2	$\pi_2 = 0.2305$
ν_3	$\pi_3 = 0.1875$

$M = 128$ $(K = 5)$	
Class	π_i
ν_0	$\pi_0 = 0.1174$
ν_1	$\pi_1 = 0.2430$
ν_2	$\pi_2 = 0.2493$
ν_3	$\pi_3 = 0.1752$
ν_4	$\pi_4 = 0.1027$
ν_5	$\pi_5 = 0.1124$

$M = 10^4$ $(K = 6)$	
Class	π_i
ν_0	$\pi_0 = 0.0882$
ν_1	$\pi_1 = 0.2092$
ν_2	$\pi_2 = 0.2483$
ν_3	$\pi_3 = 0.1933$
ν_4	$\pi_4 = 0.1208$
ν_5	$\pi_5 = 0.0675$
ν_6	$\pi_6 = 0.0727$

where ***igamc*** denotes the (regularized) incomplete gamma function:

$$igamc(a, x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t}\, t^{a-1}\, dt$$

Step 5 If the p-value is less than 1% (0.01), the sequence is considered non-random; otherwise, it is considered random.

Serial Test

The Serial Test evaluates the randomness of a given bit sequence based on the frequencies of all possible m-bit patterns. For a sequence of length n, denoted as ϵ, the p-values are computed as follows:

Step 1 Append the first $(m-1)$ bits of ϵ to its end to form ϵ'.

Step 2 Count the occurrences $\nu_{i_1 \cdots i_m}$ of all possible m-bit patterns $i_1, \ldots, i_m$ in ϵ'. For example, when $m = 3$, the patterns are $\nu_{000}, \nu_{001}, \nu_{010}, \nu_{011}, \nu_{100}, \nu_{101}, \nu_{110}, \nu_{111}$. Similarly, count the occurrences for $(m-1)$-bit and $(m-2)$-bit patterns, denoted by $\nu_{i_1 \cdots i_{m-1}}$ and $\nu_{i_1 \cdots i_{m-2}}$, respectively.

Step 3 Compute $\psi_m^2, \psi_{m-1}^2, \psi_{m-2}^2$:

$$\psi_m^2 = \frac{2^m}{n} \sum_{i_1 \cdots i_m} \left(\nu_{i_1 \cdots i_m} - \frac{n}{2^m} \right)^2 = \frac{2^m}{n} \sum_{i_1 \cdots i_m} \nu_{i_1 \cdots i_m}^2 - n$$

$$\psi_{m-1}^2 = \frac{2^{m-1}}{n} \sum_{i_1 \cdots i_{m-1}} \left(\nu_{i_1 \cdots i_{m-1}} - \frac{n}{2^{m-1}} \right)^2 = \frac{2^{m-1}}{n} \sum_{i_1 \cdots i_{m-1}} \nu_{i_1 \cdots i_{m-1}}^2 - n$$

$$\psi_{m-2}^2 = \frac{2^{m-2}}{n} \sum_{i_1 \cdots i_{m-2}} \left(\nu_{i_1 \cdots i_{m-2}} - \frac{n}{2^{m-2}} \right)^2 = \frac{2^{m-2}}{n} \sum_{i_1 \cdots i_{m-2}} \nu_{i_1 \cdots i_{m-2}}^2 - n$$

Step 4 Compute the first and second differences:

$$\nabla \psi_m^2 = \psi_m^2 - \psi_{m-1}^2, \qquad \nabla^2 \psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2.$$

Step 5 Compute the two p-values:

$$P\text{-value1} = \boldsymbol{igamc}\left(2^{m-2},\, \nabla \psi_m^2\right), \qquad P\text{-value2} = \boldsymbol{igamc}\left(2^{m-3},\, \nabla^2 \psi_m^2\right).$$

Here, ***igamc*** denotes the (regularized) incomplete gamma function:

$$igamc(a, x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t}\, t^{a-1}\, dt.$$

Step 6 If either p-value is less than 1% (0.01), the sequence is considered non-random; otherwise, it is considered random.

Approximate Entropy Test

The Approximate Entropy Test evaluates the randomness of a given bit sequence based on the frequencies of all possible m-bit patterns. For a sequence of length n, denoted as ϵ, the p-value is computed as follows:

Step 1 Append the first $(m-1)$ bits of ϵ to its end to form the extended sequence ϵ'.

Step 2 For all possible m-bit patterns i (expressed in binary), compute the relative frequencies

$$C_i^m = \frac{\#i}{n},$$

where $\#i$ denotes the number of occurrences of pattern i in ϵ'. For example, when $m = 3$, the set is $C_{000}^3, C_{001}^3, C_{010}^3, C_{011}^3, C_{100}^3, C_{101}^3, C_{110}^3, C_{111}^3$.

Step 3 Compute $\varphi^{(m)}$:

$$\varphi^{(m)} = \sum_{i=0}^{2^m-1} C_i^m \log C_i^m.$$

Step 4 Repeat Steps 1–3 with $m+1$ (i.e., replace m with $m+1$) to compute $\varphi^{(m+1)}$.

Step 5 Compute the test statistic χ^2:

$$\chi^2(obs) = 2n\left[\log 2 - ApEn(m)\right],$$

where

$$ApEn(m) = \varphi^{(m)} - \varphi^{(m+1)}.$$

Step 6 Calculate the p-value:

$$P\text{-value} = \boldsymbol{igamc}\left(2^{m-1},\ \frac{\chi^2(obs)}{2}\right),$$

where ***igamc*** denotes the (regularized) incomplete gamma function:

$$igamc(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty e^{-t}\, t^{a-1}\, dt.$$

Step 7 If the p-value is less than 1% (0.01), the sequence is considered non-random; otherwise, it is considered random.

Cumulative Sums (Cusum) Test

The Cumulative Sums (Cusum) Test evaluates the randomness of a given bit sequence by converting it into a numeric sequence of $(-1, 1)$ and computing the maximum absolute value of cumulative sums from the forward or backward direction. For a bit sequence of length n, denoted as $\epsilon = \epsilon_1, \epsilon_2, \cdots, \epsilon_n$, the p-value is computed using the following steps:

Step 1 Convert the bit sequence ϵ from $(0, 1)$ to a numeric sequence X of $(-1, 1)$ and define the cumulative sums S_i ($X_i = 2\epsilon_i - 1 = \pm 1$):

$$S_i = \begin{cases} S_{i-1} + X_i = \sum_{j=1}^{i} X_j, & \text{mode} = 0 \text{ (forward)} \\ S_{i-1} + X_{n-i+1} = \sum_{j=n-i+1}^{n} X_j, & \text{mode} = 1 \text{ (backward)} \end{cases}$$

Step 2 Compute the maximum absolute value of the cumulative sums:

$$z = \max_{1 \le k \le n} |S_k|.$$

Step 3 Calculate the p-value:

$$\begin{aligned} P\text{-value} = 1 &- \sum_{k=\frac{\frac{-n}{z}+1}{4}}^{\frac{\frac{n}{z}-1}{4}} \left[\Phi\left(\frac{(4k+1)z}{\sqrt{n}} \right) - \Phi\left(\frac{(4k-1)z}{\sqrt{n}} \right) \right] \\ &+ \sum_{k=\frac{\frac{-n}{z}-3}{4}}^{\frac{\frac{n}{z}-1}{4}} \left[\Phi\left(\frac{(4k+3)z}{\sqrt{n}} \right) - \Phi\left(\frac{(4k+1)z}{\sqrt{n}} \right) \right], \end{aligned}$$

where Φ denotes the standard normal cumulative distribution function:

$$\Phi(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-\frac{u^2}{2}} \, du.$$

Step 4 Compute the p-values for both modes (forward and backward). If either p-value is less than 1% (0.01), the sequence is considered non-random; otherwise, it is considered random.

Chapter 4
Related Work

This chapter systematically summarizes related work in the field of malicious encrypted traffic identification. From an identification approach perspective, we can categorize these studies into two types: **Signature-based** and **Machine-Learning-based**.

4.1 Signature-based Identification

Signature-based methods identify traffic by matching it with predefined signatures. Such signatures consist of features extracted from previously known activities.

One of the long-established research fields in characterizing encrypted traffic is TLS fingerprinting. Based on the methods used to collect source information, TLS fingerprinting is classified into two types: **passive fingerprinting** and **active fingerprinting**.

4.1.1 Passive Fingerprinting

Passive TLS fingerprinting is a technique that passively collects data from flowing TLS traffic to generate a fingerprint without actively initiating communication. While this technique is constrained by the requirement that the sensor must be operational at the time of communication and along the communication route, there is much less risk of being detected by the target. It is also applicable to both servers and clients.

Sslhaf [43], developed by Ivan Ristic of Qualys SSL Labs, was designed for estimating HTTP clients. It utilized SSL/TLS version, cipher suite, and HTTP UserAgent as features. Marek integrated SSL/TLS fingerprinting into the passive fingerprinting tool p0f [44]. This approach, targeting OS/application estimation, used SSL/TLS version, cipher suite, TLS extension parameters, and several heuristic flag features. FingerprinTLS [45], also for application estimation, collected compression methods and elliptic curve cryptography parameters in addition to the above features. JA3/JA3S [6], developed by Althouse et al. at Salesforce, aimed to identify malware and attacker environments. While it used features similar to FingerprinTLS, JA3/JA3S became popular by integrating with major

tools and services, including VirusTotal [46], Shodan [47], Wireshark [48], and Suricata [49]. The JA3 values of malware communications were once publicly shared at SSLBL [50]. JA3 development is now discontinued, with its successor being the fingerprinting suite JA4+ [7]. Frolov et al. proposed a TLS fingerprinting method to circumvent censorship [51]. Their fingerprint used the TLS version, cipher suite, selected extension parameters, and extension field values. Joy [52, 53], developed by Anderson et al. at Cisco, estimated client processes across protocols such as TLS, DNS, and HTTP. Joy's fingerprint, called Network Protocol Fingerprint (NPF), is distinct for normalizing and including GREASE [34] values. Joy's development was later taken over by Mercury [54, 55], which added even more extension parameters.

4.1.2 Active Fingerprinting

Active TLS Fingerprinting is a technique that actively initiates TLS communication with a target host and generates fingerprints from collected responses. While we can collect responses to any request, both normal and abnormal, at any given time, there is a risk that the target will perceive it as an attempt to collect information. Furthermore, since the sensor acts as the client, it cannot be used for client fingerprinting.

TLS Prober [56] mainly performs abnormal SSL/TLS handshakes against the server and infers the server's SSL/TLS library implementation from its responses. JARM [57] encodes responses from 10 different TLS handshakes, combining normal and abnormal requests.

4.1.3 Fingerprint Collision

A known problem in TLS fingerprinting is collision, where different applications yield the same fingerprint. Anderson et al. noted that 59 JA3 fingerprints attributed to malware were also associated with benign processes such as Internet Explorer, Python, and Java [58]. To compensate for TLS fingerprinting's shortcomings while still using it, studies have emerged that combine TLS fingerprints with other contextual information. Examples include the research by Anderson et al. [58] and MVDet [59].

4.2 Machine-Learning-based Identification

Machine Learning-based methods construct classifiers by training with features extracted from the traffic. Later, they are used to classify traffic. Based on Wang's categorization [60], we grouped the features into three types: protocol-agnostic, protocol-specific, and raw.

4.2.1 Protocol-Agnostic Features

Protocol-agnostic features are numerical features statistically calculated or extracted from the traffic [61, 62]. These features have two granularities: packet-level and session-level (flow-level). Packet-level features can be extracted per packet, such as packet size, payload size, and inter-arrival time. Session-level features can be extracted per session flow, such as session duration, total bytes sent/received for each direction, and total number of packets sent/received for each direction.

The distance-based method proposed by Liu [61] utilizes four types of protocol-agnostic features: TCP/IP header-based features, time-based features, length-related features, and packet variation features. The study by Cui [62] extracts statistical features and uses them to cluster malicious traffic channels.

4.2.2 Protocol-Specific Features

Protocol-specific features are extracted from protocol-specific metadata, such as protocol header fields. In the case of TLS, protocol version, cipher suite information, and extension information are commonly used [63, 64, 65].

Although their origins are closely related to those of signature-based identification methods, their applications differ. Signature-based identification methods use protocol-specific information sets to construct signatures that represent communication (endpoints), whereas machine learning approaches use them as input to train a classifier.

MalDetect [63] collects both protocol-agnostic and TLS-specific features and adopts Online Random Forest as a classifier. Gomez extracts 90 features from TLS-specific fields in traffic and uses them to detect and cluster malicious TLS flows [65].

4.2.3 Raw Features

Raw features are used in conjunction with deep learning methods [64, 66, 67, 68]. In this approach, a subset of the traffic data is used as-is or converted to a numerical sequence. Feature extraction is incorporated into machine learning rather than relying on expert-designed heuristic feature sets.

TLS2Vec [64] extracts TLS metadata from a TLS session and applies Word2Vec techniques to its decimal representation as a corpus. TLARNN [67] and DETD [68] extract the first N bytes of the first M packets from the encrypted session stream and train a detection model using deep neural networks.

Chapter 5
TLS Parameter Variation

5.1 Introduction

The widespread use of TLS in recent years has made identifying and detecting malicious activities increasingly challenging. With encrypted communication, tools that rely on payload information no longer work, forcing defenders to judge malicious (or benign) activity using limited observable data.

In this context, TLS fingerprinting effectively serves as a means of identifying encrypted communication. Passive TLS Fingerprinting remains one of the most widely used methods. It extracts parameter information from the readable fields of the TLS Handshake Client Hello and Server Hello messages for characterization.

Most existing Passive TLS fingerprint algorithms are designed based on experts' heuristics and experience. However, systematic verification of the validity of parameter selection and data processing methods has been largely insufficient. For instance, prior studies have reported cases in which multiple fingerprints are generated by the same browser application depending on the context [51]. Conversely, cases in which the same fingerprint is associated with multiple applications have also been reported [58].

Furthermore, one weakness of existing well-known TLS fingerprinting methods, such as JA4 [7], is their vulnerability to parameter changes. Many techniques specialize in exact matching and use cryptographic hash functions. While this has the advantage of representing features as fixed-length values, it also means that a slight change in the TLS parameters can lead to a significant change in the final value. From an attacker's perspective, it is possible to intentionally evade detection by slightly changing the parameters. Akamai reported observing TLS tampering (randomizing the order of TLS cipher suites) in 2019 [69]. They concluded that it is a tactic by attackers to evade detection, and they named this technique Cipher Stunting. In addition to attackers' intentional changes, similar issues may arise from software modifications. Existing TLS fingerprinting cannot

keep up with such parameter changes.

In this chapter, based on the hypothesis that lesser-known TLS parameter changes exist in malware communications, we investigate in detail how TLS characteristics change, particularly within individual malware families. We introduce two new concepts related to parameter changes, "**Parameter Fluctuation**" and "**Parameter Drift**." By analyzing an unprecedentedly large-scale dataset (the MTA dataset [9]) spanning approximately 11.5 years, we provide specific examples of these phenomena. While these variations were initially observed in malware C2 traffic, they can, in principle, also occur in legitimate applications.

This research aims to develop a TLS fingerprinting method that is resilient to parameter fluctuations. To achieve this goal, We quantitatively analyze distributions, transitions, and variations within TLS Client Hello messages using an existing large-scale dataset. We propose the "**Compacted Protocol Representation (CPR)**," a novel format for analyzing communication protocol parameters, and the "**Structural Edit Distance (SED)**," a metric for precisely calculating parameter differences. These two serve as foundational, highly reproducible tools for structurally analyzing and precisely quantifying differences in TLS parameters. By utilizing proposed techniques, this study reveals the actual state of parameter variation in both benign and malicious TLS Client Hellos. Next, we generate a dataset suitable for evaluating Parameter Fluctuation resilience. Through experimentation, we develop an effective identification strategy for classifying communication based on TLS fingerprints. Specifically, the "**Exact Match after De-fluctuation**" strategy achieves the best identification performance in environments with sparse known fingerprints.

5.2 Related Work

In this section, we discuss research on fingerprint proximity and past surveys on the use of TLS in malware communication.

5.2.1 Research on Fingerprint Proximity

A few studies have discussed the use of fingerprint proximity [51, 58]. Frolov [51] discussed the distribution of TLS parameters in legitimate communications by clustering

them based on the Levenshtein distance of the parameter sets. Anderson [58] introduced a method for searching for approximate fingerprints using the Levenshtein distance when no exact-matched fingerprint is available to handle unknown endpoints. In both studies, the main focus remains exact matching, and fingerprint proximity calculation is used only as a supplement. Also, while they use the Levenshtein distance as a metric of proximity, this is merely the adoption of a popular method, and there is room to consider other distance metrics.

5.2.2 Survey of Malware TLS C2 Communication

Anderson [70] investigated the TLS usage of 18 malware families in 2016 and discussed the differences in parameters between benign and malicious communications and the characteristics of parameters for each family. However, the target TLS version was TLS 1.2, and TLS 1.3 was not included. Barradas et al. [71] analyzed the differences in TLS certificate sizes and TLS record data sizes between TLS 1.2 C2 and TLS 1.3 C2 in public datasets and proposed a classifier based on their findings.

In this study, we investigate the variations of TLS parameters across malware families using a dataset spanning a longer period than that used in prior research. In particular, we examined individual infection cases within the same malware family and identified detailed parameter variations by comparing parameters within and between cases.

5.3 Definition

In this study, we introduce two definitions for TLS parameter variations: **Parameter Fluctuation** and **Parameter Drift**.

Parameter Fluctuation

Parameter Fluctuation is a phenomenon where TLS handshake parameters between the same communication endpoints fluctuate with each communication. Applying this to malware TLS communications means that the parameters vary from session to session across communications between the same malware and C2 server pair. Examples include parameters such as session IDs and encryption key information. Existing TLS fingerprints deal with such fluctuations by excluding these parameters from the calculation. However,

in this study, we demonstrate the presence of parameter fluctuations that have not been considered in prior research.

Parameter Drift

Parameter Drift is a phenomenon in which TLS handshake parameters for the same software family change over a long period of time. Generally, this corresponds to parameter changes due to software upgrades. While it is unsurprising that upgrades can occur in malware families, this study examines specific examples of such changes and their extent.

5.4 Dataset Collection for Real-world Analysis

We investigated the characteristics of malware TLS communications, particularly the long-term transitions of TLS parameters in specific malware families, over an extended period including before and after the spread of TLS 1.3. Due to the nature of the dataset, we could only collect C2 server responses to specific client requests at the past point in time. Thus, we focused our investigation on the characteristics of the TLS client side, which can be expected to exhibit stable behavior regardless of the server-side settings.

5.4.1 Malware Traffic Analysis Dataset

We used the Malware Traffic Analysis (MTA) dataset [9] for our study. MTA is a service operated by Brad Duncan, an analyst at Palo Alto Networks Unit 42, that publicly shares data on malware traffic. It is a well-known dataset frequently used in existing research on malware TLS communications [72, 73, 74, 71]. Because of the nature of the service, the MTA dataset may contain biases and lack comprehensiveness due to the ease of collecting malware samples and the analyst's personal preferences. However, it is also a rare open dataset compiled over more than a decade, with data analyzed promptly in line with trends. We adopted MTA for this study because our objective is to track specific malware families over a long period and to reveal transitions in their TLS parameters. It should be noted that the MTA dataset was collected in a sandbox environment, and the resulting PCAP files may include legitimate communications unrelated to the infection.

The data collection period covers approximately 11.5 years, from June 2013 to December 2024. We collected 155,047 TLS Client Hello packets from 1,436 PCAP files that contained

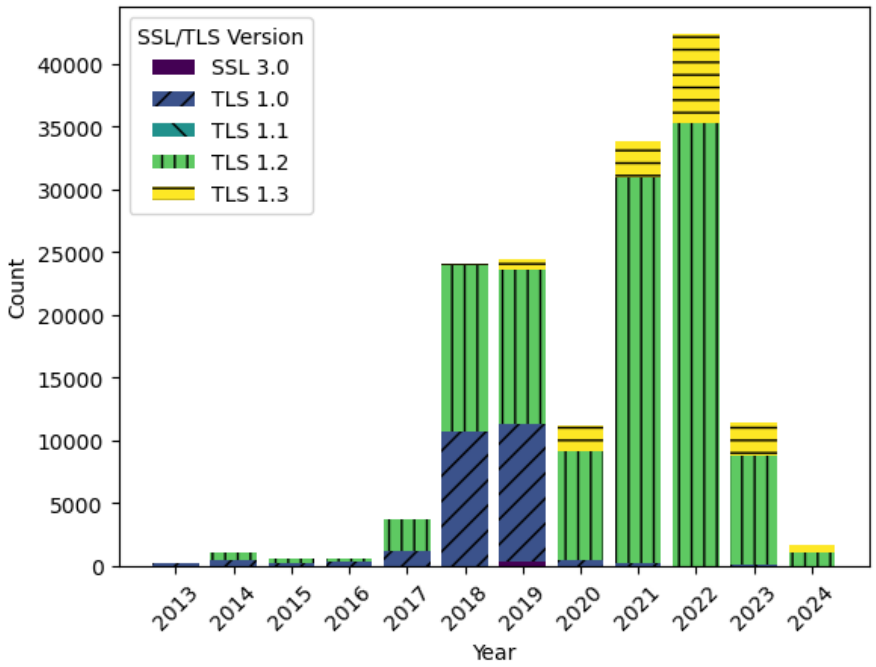


Figure 5.1. SSL/TLS packet count by year (MTA Dataset)

them.

To our knowledge, no other investigation has collected and analyzed malware communications over a period exceeding 10 years, and we are the first to handle MTA data spanning such an extended period.

5.4.2 Trends in TLS Parameters

As mentioned earlier, while the MTA dataset has biases and is not suitable for observing trends across the entire internet, it is data from the periods when each malware and its corresponding C2 server were active, allowing us to capture the rough trends in malware communications.

Figure 5.1 shows the number of packets for each SSL/TLS version. There is a sharp increase in TLS communications from 2017 to 2018. Cisco also reported an increase in encrypted communications by malware during the same period [14], suggesting that the use of TLS by malware began to increase in earnest around this time. Looking at the year-by-year proportion (Figure 5.2), the proportion of TLS 1.0 decreased from 2019 through 2020, and use of TLS 1.3 started to increase. Since TLS 1.3 became an RFC standard

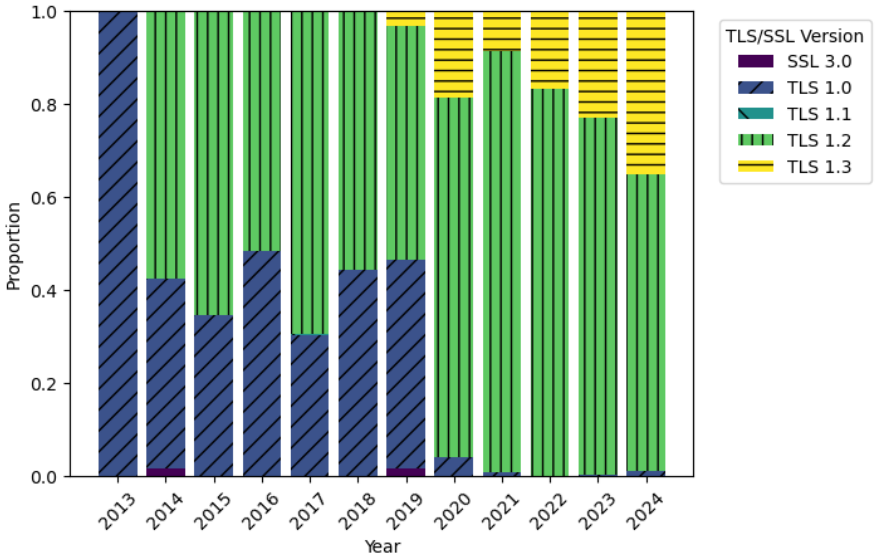


Figure 5.2. SSL/TLS packet proportion by year (MTA Dataset)

in 2018 [23], it seems that the adoption of TLS 1.3 has also progressed in malware C2 communications in line with the spread of the TLS 1.3 protocol implementation. However, as of 2024, TLS 1.2 is still the most widely used. According to SSL Pulse [75], supported TLS versions in popular websites in the Alexa ranking as of May 2024 are 99.9% for TLS 1.2 and 70.1% for TLS 1.3, indicating that the transition to TLS 1.3 is still in progress and TLS 1.2 will likely remain mainstream for a while.

To gain a detailed understanding of parameter changes introduced by TLS 1.3, we split the dataset into two parts based on an acute change in the proportion of TLS versions, the period up to 2019 (named “before TLS 1.3”) and the period from 2020 (named “after TLS 1.3”).

First, we focused on variations in cipher suites and TLS extensions across the two periods. Table 5.1 shows distinct numbers of cipher suites and TLS extensions. Regarding cipher suites, we observe a decrease in their variations. This is likely due to TLS 1.3 eliminating vulnerable cipher suites and restricting the available cipher suites. In contrast, TLS extensions have seen nearly three times the variation. Many new extension types introduced in TLS 1.3 are considered a contributing factor.

Table 5.2 shows average numbers of parameter elements included in a single Client Hello. Regarding cipher suites, the average number of cipher suites in a single handshake

Table 5.1. Distinct number of parameter sets in MTA Dataset

	2013-2019	2020-2024
Cipher suites	174	158
Extensions	566	1775

Table 5.2. Average number of elements included in a single Client Hello (MTA Dataset)

	2013-2019	2020-2024
Cipher suites	23.20	19.46
Extensions	13.75	17.18

decreased after TLS 1.3 emerged. This is presumed to be due to a reduction in the number of cipher suite choices, similar to the decrease in the number of distinct parameter sets. Regarding TLS extensions, on the other hand, the parameters contained within a single Client Hello message increased after the adoption of TLS 1.3. This is thought to be partly due to the handshake being shortened, resulting in more information being packed into the Client Hello.

Figure 5.3 shows histograms of cipher suite sets and TLS extension sets in a single Client Hello. The distribution of TLS Fingerprints is known to be long-tailed and this trend is also observable for cipher suites and TLS extensions.

Looking at the distribution of cipher suite sets requested in Client Hello, the cumulative relative frequency is steeper for “after TLS 1.3” than for “before TLS 1.3”. The top seven cipher suite sets accounted for more than 90% of all Client Hello in “before TLS 1.3”, while the top five sets accounted for “after TLS 1.3”. As mentioned earlier, this is thought to be an effect of parameter sets becoming concentrated on specific combinations due to the limited cipher suites available in TLS 1.3.

Unlike cipher suites, the cumulative relative frequency of extension sets becomes more shallow. The top ten cipher suite sets accounted for more than 90% of all Client Hello in “before TLS 1.3”, while the top thirteen sets accounted for over 90% in“after TLS 1.3”.

Detailed statistics for the top 20 parameters are shown in Tables 5.3 to 5.9.

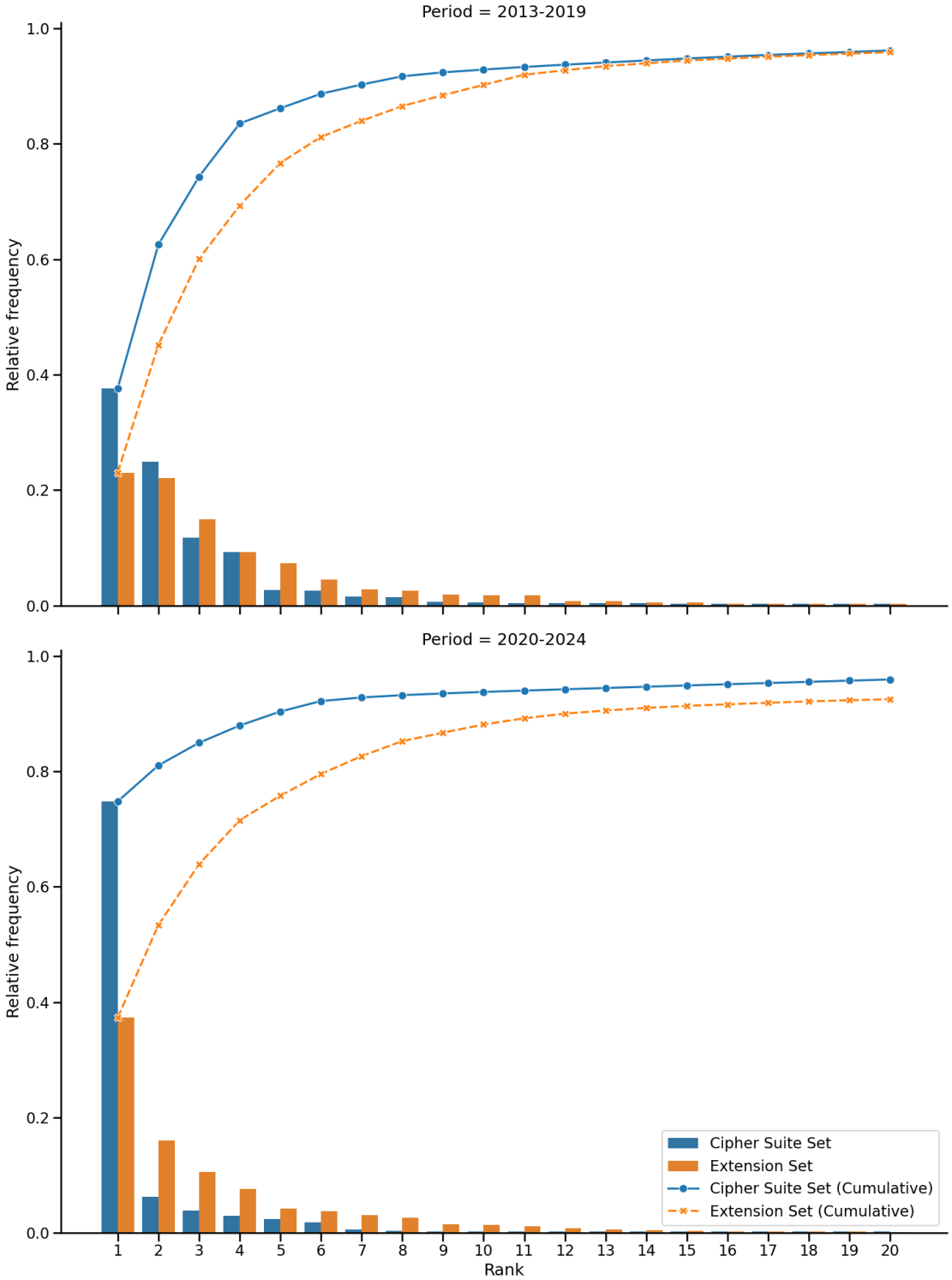


Figure 5.3. Histogram of parameter set in MTA dataset (Top 20)

Table 5.3. Cipher suite parameters in MTA dataset (Top 10, 2013-2019)

Rank	Cipher Suites	Count	Relative frequency[%]
1	47-53-5-10-49171-49172-49161-49162-50-56-19-4	20493	37.65
2	60-47-61-53-5-10-49191-49171-49172-49195-49187-49196 -49188-49161-49162-64-50-106-56-19-4	13540	24.88
3	49195-49199-52393-52392-49196-49200-49162-49161-49171 -49172-51-57-47-53-10	6414	11.78
4	49196-49200-159-52393-52392-52394-49195-49199-158-49188 -49192-107-49187-49191-103-49162-49172-57-49161-49171 -51-157-156-61-60-53-47-255	5015	9.21
5	51-47-53-10-5-4-255	1443	2.65
6	49162-49172-136-135-57-56-49167-49157-132-53-49159-49161 -49169-49171-69-68-51-50-49164-49166-49154-49156-150 -65-4-5-47-49160-49170-22-19-49165-49155-65279-10-255	1364	2.51
7	49195-49199-49162-49161-49171-49172-49170-49159-49169 -51-50-69-57-56-136-22-47-65-53-132-10-5-4-255	867	1.59
8	49199-49195-49200-49196-158-162-163-159-49191-49187 -49171-49161-49192-49188-49172-49162-103-51-64-107-56 -57-156-157-164-160-63-62-50-49-48-49201-49197-49193 -49189-49166-49156-60-47-165-161-106-105-104-55-54-49202 -49198-49194-49190-49167-49157-61-53-136-135-134-133 -132-69-68-67-66-65-255	784	1.44
9	5-10-19-4-255	367	0.67
10	4866-4867-4865-49199-49195-49200-49196-158-49191-103 -49192-107-163-159-52393-52392-52394-49327-49325-49315 -49311-49245-49249-49239-49235-162-49326-49324-49314 -49310-49244-49248-49238-49234-49188-106-49187-64-49162 -49172-57-56-49161-49171-51-50-157-49313-49309-49233 -156-49312-49308-49232-61-60-53-47-255	264	0.49

Table 5.4. Cipher suite parameters in MTA dataset (Rank 11-15, 2013-2019)

Rank	Cipher Suites	Count	Relative frequency[%]
11	49200-49196-49192-49188-49172-49162-165-163-161-159 -107-106-105-104-57-56-55-54-136-135-134-133-49202-49198 -49194-49190-49167-49157-157-61-53-132-49199-49195 -49191-49187-49171-49161-164-162-160-158-103-64-63-62-51 -50-49-48-154-153-152-151-69-68-67-66-49201-49197-49193 -49189-49166-49156-156-60-47-150-65-7-49170-49160-22 -19-16-13-49165-49155-10-255	249	0.46
12	52393-52392-52244-52243-49195-49199-49196-49200-49161 -49171-49162-49172-156-157-47-53-10	211	0.39
13	49200-49196-49192-49188-49172-49162-165-163-161-159 -107-106-105-104-57-56-55-54-136-135-134-133-49202-49198 -49194-49190-49167-49157-157-61-53-132-49199-49195 -49191-49187-49171-49161-164-162-160-158-103-64-63-62-51 -50-49-48-154-153-152-151-69-68-67-66-49201-49197-49193 -49189-49166-49156-156-60-47-150-65-49170-49160-22-19 -16-13-49165-49155-10-7-49169-49159-49164-49154-5-4-255	202	0.37
14	49192-49191-49172-49171-159-158-157-156-49196-49195 -49188-49187-49162-49161-61-60-53-47-106-64-56-50-10 -19-5-4	199	0.37
15	49192-49191-49172-49171-159-158-157-156-61-60-53-47 -49196-49195-49188-49187-49162-49161-106-64-56-50-10 -19-5-4	184	0.34

Table 5.5. Cipher suite parameters in MTA dataset (Rank 16-20, 2013-2019)

Rank	Cipher Suites	Count	Relative frequency[%]
16	49172-49171-49162-49161-53-47-56-50-10-19-5-4	176	0.32
17	49195-49196-49199-49200-158-159-49161-49162-49171-49172 -51-57-49159-49169-156-157-47-53-5-255	159	0.29
18	49200-49196-49192-49188-49172-49162-165-163-161-159 -107-106-105-104-57-56-55-54-49202-49198-49194-49190 -49167-49157-157-61-53-49199-49195-49191-49187-49171 -49161-164-162-160-158-103-64-63-62-51-50-49-48-154-153 -152-151-49201-49197-49193-49189-49166-49156-156-60-47 -150-49170-49160-22-19-16-13-49165-49155-10-255	144	0.26
19	49192-49191-49172-49171-159-158-157-156-49196-49195 -49188-49187-49162-49161-61-60-53-47-106-64-56-50-10 -19	140	0.26
20	52393-52392-52244-52243-49195-49199-49162-49172-49161 -49171-156-53-47-10	138	0.25

Table 5.6. Extensions parameters in MTA dataset (Top 20, 2013-2019)

Rank	Extensions	Count	Relative frequency[%]
1	65281-0-10-11-13	12421	23.03
2	65281-10-11	11927	22.11
3	65281-0-10-11	8043	14.91
4	11-10-35-13-22-23	4996	9.26
5	0-23-65281-10-11-35-16-5-13	3978	7.38
6	21-0-23-65281-10-11-35-16-5-13	2430	4.51
7	0-35	1511	2.80
8	0-11-10-13-15	1376	2.55
9	0-11-10-35-13-15-21	1014	1.88
10	65281-10-11-13	967	1.79
11	0-11-10-35-13-15	950	1.76
12	0-10-11-13-23-65281	414	0.77
13	0-11-10-13-15-13172-16-21	387	0.72
14	0-11-10-35-22-23-13-43-45-51	264	0.49
15	65281-5-10-11	263	0.49
16	65281-0-5-10-11	179	0.33
17	65281-0-5-10-11-13	176	0.33
18	65281-0-23-35-13-5-18-16-30032-11-10-24	149	0.28
19	10-11-13-23-65281	141	0.26
20	10-11	128	0.24

Table 5.7. Cipher suite parameters in MTA dataset (Top 10, 2020-2024)

Rank	Cipher Suites	Count	Relative frequency[%]
1	49196-49195-49200-49199-49188-49187-49192-49191-49162 -49161-49172-49171-157-156-61-60-53-47-10	75245	74.78
2	4866-4865-49196-49195-49200-49199-49188-49187-49192 -49191-49162-49161-49172-49171-157-156-61-60-53-47	6273	6.23
3	49196-49200-159-52393-52392-52394-49195-49199-158-49188 -49192-107-49187-49191-103-49162-49172-57-49161-49171 -51-157-156-61-60-53-47-255	3941	3.92
4	49199-49200-49195-49196-52392-52393-49171-49161-49172 -49162-156-157-47-53-49170-10-4865-4867-4866	3007	2.99
5	49196-49195-49200-49199-159-158-49188-49187-49192-49191 -49162-49161-49172-49171-157-156-61-60-53-47-10	2449	2.43
6	49196-49195-49200-49199-49188-49187-49192-49191-49162 -49161-49172-49171-157-156-61-60-53-47	1827	1.82
7	47-53-5-10-49171-49172-49161-49162-50-56-19-4	623	0.62
8	4865-4867-4866-49195-49199-52393-52392-49196-49200 -49162-49161-49171-49172-51-57-47-53-10	405	0.40
9	49200-49199-49192-49191-49172-49171-159-158-157-156 -61-60-53-47-49196-49195-49188-49187-49162-49161-106 -64-56-50-10-19	294	0.29
10	60-47-61-53-5-10-49191-49171-49172-49195-49187-49196 -49188-49161-49162-64-50-106-56-19-4	271	0.27

Table 5.8. Cipher suite parameters in MTA dataset (Rank 11-20, 2020-2024)

Rank	Cipher Suites	Count	Relative frequency[%]
11	6682-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	230	0.23
12	31354-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	229	0.23
13	64250-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	223	0.22
14	60138-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	222	0.22
15	27242-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	221	0.22
16	23130-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	213	0.21
17	10794-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	211	0.21
18	35466-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	210	0.21
19	39578-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	207	0.21
20	56026-4865-4866-4867-49195-49199-49196-49200-52393 -52392-49171-49172-156-157-47-53	206	0.20

Table 5.9. Extension parameters in MTA dataset (Top 20, 2020-2024)

Rank	Extensions	Count	Relative frequency[%]
1	0-10-11-13-35-23-65281	37524	37.29
2	10-11-13-35-23-65281	16151	16.05
3	0-5-10-11-13-35-23-65281	10604	10.54
4	0-5-10-11-13-35-16-23-65281	7675	7.63
5	0-5-43-13-35-10-51-49-23-65281-45-41	4286	4.26
6	11-10-35-13-22-23	3777	3.75
7	5-10-11-13-35-23-65281	3124	3.10
8	0-5-10-11-13-65281-18-43-51	2613	2.60
9	0-10-11-13-35-16-23-24-65281	1464	1.46
10	0-10-11-13-35-16-23-65281	1431	1.42
11	10-11-13-35-23-24-65281	1094	1.09
12	0-5-43-13-35-10-51-49-23-65281-45	808	0.80
13	5-10-11-13-65281-18-43-51	531	0.53
14	65281-10-11	474	0.47
15	43-13-35-10-51-49-23-65281-45-41	343	0.34
16	0-5-43-13-35-10-16-51-49-23-65281-45	278	0.28
17	5-10-11-13-35-23-24-65281	262	0.26
18	0-10-11-13-35-23-24-65281	242	0.24
19	21-0-23-65281-10-11-35-16-5-40-43-65283-13-45	208	0.21
20	65281-0-10-11-13	172	0.17

Table 5.10. Malware families that use TLS and had been observed over a long period in MTA dataset

	First seen	Last Seen	Case count
Emotet	2016-12-01	2023-03-17	219
Trickbot	2017-06-12	2021-09-01	175
IcedID	2018-06-18	2023-11-27	152
Cobalt Strike	2019-07-03	2024-04-18	128

Note that Emotet uses TLS only after 2021-11-15.

5.5 In-Depth Analysis

5.5.1 Methodology

This study focuses on examining the dynamics of TLS Client Hello parameters within specific malware families over an extended period, utilizing a long-term dataset. Our primary objective is to reveal their characteristics with a particular focus on Parameter Fluctuation and Parameter Drift. From this perspective, we extracted family names and dates from the PCAP filenames in our dataset and chose malware families that had been observed over a long period, specifically to capture Parameter Drift which had occurred in many cases, and used TLS for C2 communications (Table 5.10). Here, we considered one PCAP file as one case. We finally selected three malware families, Trickbot, IcedID, and Cobalt Strike, as our targets for investigation, since they had more than 100 cases, and their period from first seen to last seen exceeded 1,500 days, and they used TLS. Although Emotet also met the requirements, we excluded it because it only started using TLS for C2 communications after its revival following the takedown in 2021, making its period of TLS usage short.

For the target malware families, we collected from the Client Hello packets JA4 fingerprints [7], Mercury's NPF [54], and raw parameter sets. Collecting raw parameter sets enabled detailed analysis of the specific changes in Parameter Fluctuation and Parameter Drift, which existing fingerprints might not fully capture. Starting with the most frequently appearing fingerprint, we manually checked the original PCAP files to confirm

whether these were indeed fingerprints of malware communications. We then examined whether other fingerprints appeared in communications with the same source and destination IP address pair, i.e., whether Parameter Fluctuation occurred. Additionally, we sorted the malware C2 fingerprints we collected by observation period to check for the occurrence of Parameter Drift. Ultimately, we collected nine JA4 fingerprints for Trickbot, nine for IcedID, and fourteen for Cobalt Strike (Table 5.11).

The methodology in this section was designed to reveal and characterize Parameter Fluctuation and Parameter Drift, focusing on their specific details often missed by existing TLS fingerprints. Collecting a long-term dataset allowed us to observe Parameter Drift. By gathering raw parameter sets in addition to JA4 and NPF, we were able to analyze the specific nature of variations, including those existing fingerprints that might obscure, providing a basis to discuss their impact on detection.

5.5.2 Parameter Variations within a Malware Family

Parameter Fluctuation

To specifically identify and characterize the patterns of Parameter Fluctuation, we first extracted multiple communication sessions observed between the same malware sample and C2 server. From the raw parameter sets of Client Hello packets within these sessions, we performed a detailed comparative analysis. This analysis considered their temporal relationships and communication phases. Through this comparative analysis, we repeatedly observed patterns where parameter variations were consistently associated with specific conditions, behaviors, or malware communication specifications. By organizing and classifying these observed patterns based on their underlying conditions and principles, we identified three distinct types of Parameter Fluctuation: fluctuations related to connection destination specification, fluctuations related to session continuation, and fluctuations suspected of multiple modules.

In fluctuations related to connection destination specification, the parameter set changes depending on whether a domain name or IP address is specified for connection. In most cases, the only difference is the presence or absence of `server_name` extension, also known as Server Name Indication (SNI). Trickbot was designed to connect to C2 servers using IP addresses, but it also accessed IP geolocation services using domain names during the

initial infection phase to obtain the global IP address of the infected host. Cobalt Strike was confirmed to frequently access C2 servers using both IP addresses and domain names.

In fluctuations related to session continuation, we confirmed cases where specific extensions were either present or absent only during the initial connection. These situations arise from their function in carrying over session-specific states like security tokens, encryption keys, and the validity of the TLS certificate across multiple TLS sessions. IcedID was confirmed to set `status_request` only during the initial C2 communication and not in subsequent communications in multiple cases. Cobalt Strike was confirmed to have cases where combinations of `status_request`, `token_binding`, and `pre_shared_key` fluctuate.

Fluctuations suspected of multiple modules are cases where multiple communication modules are suspected of being involved in TLS communication. IcedID was confirmed to offer multiple different specific cipher suite lists even with the same malware sample and the same C2 server pair. Though further investigation is needed, this suggests the possibility of different modules operating concurrently within a single malware infection event. Parameters between different modules appear to be independent and should be considered separately from other fluctuations.

How would these Parameter Fluctuations appear in existing TLS fingerprints? Table 5.12 shows examples of JA3 and JA4 fingerprints of the Client Hello combinations for each type of Parameter Fluctuations.

The smallest differences are the presence or absence of one to three parameter elements. Since JA3 takes the MD5 value at the end, the fingerprint value changes significantly even if there is only a difference in the presence or absence of one parameter element. JA4 also significantly changes the value of the corresponding SHA256-based section when cipher suites or extensions vary. However, since server_name is encoded in a section that does not use hashing, fluctuations due to connection destination specification methods result in only two character differences in the JA4 fingerprints.

If only one fingerprint from these fluctuating fingerprint groups were known and the rest were unknown, traditional matching-based identification would, in the worst case, miss approximately half of the malicious communications. This problem could be solved by a function to search for fingerprints with the closest edit distance as is the case with Mercury. This is discussed in detail later in this paper.

Table 5.11. JA4 Fingerprints of malware families

Trickbot	t10d120400_d94e65cdb899_f8ec56bc740a
	t10i120300_d94e65cdb899_f8ec56bc740a
	t10i120400_d94e65cdb899_368bf2d64253
	t12d190800_d83cc789557e_7af1ed941c26
	t12d1909h2_d83cc789557e_16bbda4055b2
	t12i190700_d83cc789557e_16bbda4055b2
	t12i190700_d83cc789557e_7af1ed941c26
	t12i1908h2_d83cc789557e_16bbda4055b2
	ts3i050000_d30d14220708_e3b0c44298fc
IcedID	t10d120400_d94e65cdb899_f8ec56bc740a
	t12d190800_d83cc789557e_16bbda4055b2
	t12d190800_d83cc789557e_7af1ed941c26
	t12d210700_76e208dd3e22_2dae41c691ec
	t12d210700_76e208dd3e22_f28add8e7af0
	t12d210800_76e208dd3e22_16bbda4055b2
	t12d230700_7ffbe1e0fba8_f28add8e7af0
	t12d260800_abdbad740de4_16bbda4055b2
	t13d201100_2b729b4bf6f3_9e7b989ebec8
Cobalt Strike	t12d180700_4b22cbed5bed_2dae41c691ec
	t12d190700_d83cc789557e_2dae41c691ec
	t12d190700_d83cc789557e_f28add8e7af0
	t12d190800_d83cc789557e_16bbda4055b2
	t12d190800_d83cc789557e_47617cf24602
	t12d190800_d83cc789557e_7af1ed941c26
	t12d190900_d83cc789557e_c2986708b002
	t12i180600_4b22cbed5bed_2dae41c691ec
	t12i190600_d83cc789557e_2dae41c691ec
	t12i190600_d83cc789557e_f28add8e7af0
	t12i190700_d83cc789557e_47617cf24602
	t12i190800_d83cc789557e_c2986708b002
	t13d201100_2b729b4bf6f3_9e7b989ebec8
	t13d201200_2b729b4bf6f3_2e4f304f1f45

Table 5.12. Examples of TLS Parameter Fluctuation

Fluctuation type	Malware family	JA3	JA4	Parameter difference
Connection destination specification	Trickbot	6734f37431670b3ab4292b8f60f29984 1d095e68489d3c535297cd8dffb06cb9	t10i120300_d94e65cdb899_f8ec56bc740a t10d120400_d94e65cdb899_f8ec56bc740a	server_name
	Cobalt Strike	51c64c77e60f3980eea90869b68c58a8 37f463bf4616ecd445d4a1937da06e19	t12i190600_d83cc789557e_2dae41c691ec t12d190700_d83cc789557e_2dae41c691ec	server_name
Session continuation	IcedID	3b5074b1b5d032e5620f69f9f700ff0e ce5f3254611a8c095a3d821d44539877	t12d210700_76e208dd3e22_f28add8e7af0 t12d210800_76e208dd3e22_16bbda4055b2	status_request
	Cobalt Strike	51c64c77e60f3980eea90869b68c58a8 37f463bf4616ecd445d4a1937da06e19 7dd50e112cd23734a310b90f6f44a7cd 57f3642b4e37e28f5cbe3020c9331b4c	t12i190600_d83cc789557e_2dae41c691ec t12d190700_d83cc789557e_2dae41c691ec t12i190700_d83cc789557e_47617cf24602 t12d190800_d83cc789557e_47617cf24602	server_name, status_request, token_binding
Suspected of multiple modules	IcedID	3b5074b1b5d032e5620f69f9f700ff0e a0e9f5d64349fb13191bc781f81f42e1	t12d210700_76e208dd3e22_2dae41c691ec t12d190800_d83cc789557e_7af1ed941c26	2 cipher suites, status_request
	IcedID	648432770b162235911a5150d4b08679 3248817f54a6ae4a9114c97a29f8ab9b	t12d210700_76e208dd3e22_2dae41c691ec t12d190800_d83cc789557e_7af1ed941c26	2 cipher suites, status_request

Parameter Drift

We confirmed changes in parameter sets that appear to be major version upgrades of modules in all three malware families targeted in this study. Trickbot had been generating TLS 1.0 C2 communications from 2017 until around the beginning of 2020. However, in 2020, versions that generated TLS 1.2 C2 communications appeared, and versions with slightly changed parameter sets have also been subsequently confirmed. IcedID and Cobalt Strike had been observed to communicate using TLS 1.2 over a wide period, but around 2022 to 2023, TLS 1.3 C2 communications began to be observed.

Compared to Parameter Fluctuation, Parameter Drift involves a larger amount of parameter changes. A primary driver of Parameter Drift is the evolution of TLS versions. When the TLS version changes (e.g., from TLS 1.2 to TLS 1.3), it fundamentally alters the set of supported cipher suites and extensions, resulting in drastic and comprehensive changes to the entire parameter set. This is because newer TLS versions often eliminate obsolete cryptographic algorithms and features, and introduce new features or requirements, necessitating a substantial overhaul of the handshake parameters. For example, when Trickbot upgraded from TLS 1.0 to TLS 1.2, almost all cipher suites were replaced in addition to the TLS version value, and the number of TLS extensions increased from three to eight.

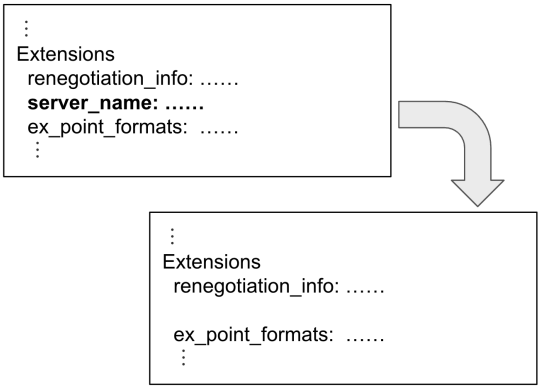
To uncover the true nature of parameter drift, it is necessary to track the same software family over a sufficiently long period. Therefore, the findings on parameter drift in this study are the result of tracking a dataset spanning over ten years from the perspective of malware families.

5.5.3 Discussion on the Impact of Parameter Variations

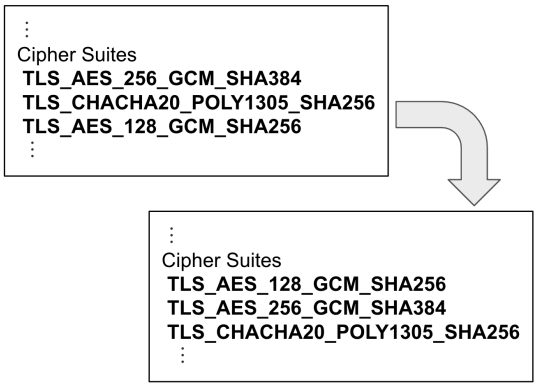
Based on our findings and previous related work, we have classified the Parameter Fluctuations that should be considered when designing fingerprints into three categories: Value Fluctuation, Existence Fluctuation, and Order Randomization (Figure 5.4).

Value Fluctuation refers to changes in the values that parameters take. Examples include session IDs and encryption keys. The basic strategy to address Value Fluctuation is to exclude these values. This approach is taken in both JA4 and Mercury's NPF, with measures to ensure that values are not incorporated into fingerprints. However, the only difference is the way they handle GREASE (RFC 8701) [34]. GREASE is a technology

(a) Value Fluctuation



(b) Existence Fluctuation



(c) Order Randomization

Figure 5.4. Categories of Parameter Fluctuation

that inserts information expected to be ignored for backward compatibility, and some TLS parameters may include GREASE values. JA4 excludes GREASE values, while NPF normalizes them by converting them to specific values. Since none of the malware families selected in this study utilized GREASE, the presence of GREASE values themselves could be a differentiator between legitimate software and malware, suggesting that normalization might be more desirable for malware identification purposes.

Existence Fluctuation refers to the fluctuation in the presence of parameter elements themselves. Among the fluctuations revealed in this paper, examples include the presence or absence of `server_name` and `status_request`. Existing TLS fingerprintings do not consider most of the Existence Fluctuations. One of the exceptions is GREASE as a TLS extension type. JA4 and NPF exclude it while generating fingerprints. Similarly, excluding potentially fluctuating extensions (Fluctuating Extensions) is also a possible remedy. However, as the number of excluded parameters increases, the parameter space becomes narrower, and the probability of false positives might increase.

Order Randomization refers to randomizing the order of parameters that take list values. One example is Cipher Stunting, as reported by Akamai [69]. In 2022, extension permutation was implemented in Chrome [76], which rearranges the order of extensions in Client Hello as a countermeasure against potential vulnerabilities, and followed by Mozilla NSS [77]. As a remedy to such randomization, JA4 sorts cipher suites and extensions before generating fingerprints. NPF also sorts extensions in lexicographic order in new types of their fingerprints, tls/1 and tls/2. While we did not confirm cases using order randomization in the malware families we focused on, we have also confirmed that communications appear to be legitimate browsers using extension permutation after 2023 in MTA.

The nature of Parameter Fluctuation and Parameter Drift confirmed in this study is not inherently limited to malware communications; it can also occur in benign communications. For example, Existence Fluctuation of the `server_name` extension, dependent on the method of specifying the communication destination, can fundamentally occur in legitimate software. The Parameter Drift associated with TLS version updates, observed in malware C2 communications following the spread of TLS 1.3, also occurs with major version upgrades of legitimate software. Indeed, JA4 itself states that its fingerprints "will change as application TLS libraries are updated, about once a year" [7]. Further-

more, the implementation of extension order randomization in major browsers since 2022 demonstrates that parameter variations can also occur in legitimate software.

Existing TLS fingerprinting technologies, particularly methods relying on exact matching and cryptographic hash functions like JA3 [6] and JA4 [7], are vulnerable to even slight changes in parameters. As revealed in this study, even minor differences, such as the presence or absence of just 1 to 3 parameter elements in TLS parameters, cause significant changes in JA3's MD5 value and substantial alterations in JA4's SHA256-based section values.

This impact is also likely to extend to machine learning-based methods that incorporate TLS parameters as one of their features. For example, MVDet [59] utilizes JA3 fingerprint as one of its various feature dimensions. ETA-GNN [78] uses JA3 fingerprint in the graph representation of TLS traffic. JA3, by its nature, is vulnerable to parameter fluctuations, but the phenomena have not been adequately considered by these methods.

Proximity search focusing on TLS parameter set proximity, as in Anderson et al.'s work [58], could serve as a countermeasure for Parameter Fluctuation. However, as our preliminary experiment results indicate, approaches that exclude fluctuating parameters or calculate proximity using simple Levenshtein distance may lead to unacceptable false positives, suggesting the need for a more careful fingerprint design.

This study also revealed Parameter Drift, a phenomenon where the parameter distribution of malware families changes over long periods. This implies that previously collected fingerprint databases become obsolete over time, reducing the effectiveness of existing detection rules and databases. For machine learning-based detection methods that utilize TLS parameters as features, this suggests that long-term changes in feature distributions necessitate retraining or model updates. Barradas et al. [71] focused on certificate sizes and payload sizes rather than TLS parameters as a robust approach against TLS version updates. While their method aims at C2/non-C2 classification, which differs in granularity from our study's objective, they have identified features that might complement the shortcomings of TLS fingerprinting.

5.6 Proposed Methods

We introduced two methods, Compacted Protocol Representation (CPR) and Structural Edit Distance (SED), to measure parameter changes more precisely.

5.6.1 Compacted Protocol Representation (CPR)

Compacted Protocol Representation (CPR) is a data representation format for structurally analyzing and utilizing communication protocol parameters. The key idea is to directly represent the data structure of communication protocols using Concise Binary Object Representation (CBOR) [79]. CBOR is a data format that can be considered the binary version of JSON, standardized as RFC 8949, "whose design goals include the possibility of extremely small code size, fairly small message size, and extensibility without the need for version negotiation". Due to its flexibility and extensibility, CBOR is used in various applications, such as CBOR Web Token (CWT) [80], a lightweight security token, and C-DNS [81], a lightweight DNS packet capture format.

Similar to the related technology Mercury NPF [54], CPR is conceived with the concept of generality, supporting various communication protocols, but it has characteristics that differ from NPF in the following respects:

Ease of implementation through the use of standardized technology

CPR utilizes CBOR, which is itself an RFC-standardized technology, and can be easily introduced by leveraging existing libraries implemented in various programming languages, including JavaScript, PHP, Python, Ruby, and C. In contrast, NPF is defined in a proprietary format using hexadecimal notation separated by parentheses, and its reference implementation is limited to the open-source tool published by Cisco, creating a high barrier to adoption. Furthermore, CBOR is designed for JSON compatibility supports for mutual conversion, meaning CPR data can also be treated as JSON. This facilitates easy data processing and analysis, as data can be represented in JSON and translated to and from CPR. Additionally, data validation for CPR is easier to implement, as Concise Data Definition Language (CDDL) [82], a descriptive language for defining CBOR data structures, is standardized, and CDDL-based data validation library implementations exist.

Lightweight design by separating readability requirement

Since CBOR is a binary representation format oriented towards size reduction, CPR can efficiently represent information from a data size perspective. NPF, on the other hand, is designed to represent binary data entirely in human-readable strings using hexadecimal notation. Consequently, CPR generally produces a smaller footprint than NPF for the same amount of information, except when the original information volume is extremely small. While the binary nature of CPR might be a portability bottleneck for data sharing, it can be encoded as a human-readable string using binary-to-text encoding such as Base64 if needed.

Expressibility equivalent to JSON

Since CPR is compatible with JSON, CPR can represent complex data structures combining arrays and key-value pairs, and it can be decoded to JSON even without a data schema. NPF represents tree-structured data by combining strings enclosed in parentheses, but it cannot express key-value pairs directly.

5.6.2 Compacted TLS Client Hello (CTLS-CH)

In this study, we analyzed data using the Compacted TLS Client Hello (CTLS-CH), which represents the Client Hello message—the primary source of client TLS fingerprints—in CPR format. As an initial implementation of CTLS-CH, we collected the same parameters as the existing technique NPF tls/2 [54], applied the same preprocessing, and encoded it as CPR.

The format of CTLS-CH is as follows:

```
[ legacy_version,
  normalized_cipher_suites,
  sorted_normalized_extensions ]
```

CTLS-CH consists of three elements. The ordering of these elements corresponds to the field order in the Client Hello message, except for random-like fields (such as Random and Session ID) that do not represent client characteristics.

legacy_version is the value of the TLS version field. Before TLS 1.2, this parameter represented the maximum version number supported by the client, but in TLS 1.3, it is fixed to 0x0303 (the TLS 1.2 version number).

normalized_cipher_suites is a list of cipher suites with GREASE [34] values "normalized". "Normalization" is a preprocessing step adopted in the NPF that replaces GREASE values with a fixed value (0x0a0a).

sorted_normalized_extensions is a key-value pair of extension parameters where GREASE values are normalized and sorted by the extension parameter type value. The reason for sorting is to suppress the negative effect of extension permutation. The sorting process is also adopted in recent methods such as JA4 and NPF tls/2.

Similar to NPF, since some extension parameter field values change per session, the field values are incorporated only for specific extension parameters, and other values are replaced with nulls (Table 5.13 for details).

A key difference from NPF is that CTLS-CH stores the single-value field values in binary format and the list-like field values in arrays. This allows for a more precise analysis of field value changes.

5.6.3 Structural Edit Distance (SED)

We introduce the Structural Edit Distance (SED), a metric that focuses on the structural differences between elements. While existing studies [51], [58] have defined the distance between two fingerprints using the Levenshtein distance, their concrete calculation methods are not clearly disclosed, leaving the calculation unclear. In this study, we define SED to resolve this ambiguity and enhance the reproducibility of our methodology.

SED is defined as the "number of edit operations (add / remove / replace / move) required to transform CBOR data A into CBOR data B." The edit operations referred to here are specifically JSON Patch [83] operations used with JSON data. Since JSON Patch represents a transformation as a sequence of edit operations, we apply this concept to CBOR and treat the number of operations as the distance. Because the "move" operation is permitted, SED is closer to the concept of extending the edit distance proposed by Cormode et al. [84] to structured data, rather than the original Levenshtein distance.

While existing research primarily focused on increases or decreases in cipher suites or extension parameters, measuring the SED value of CTLS-CH enables us to capture fine differences even in list-formatted extension parameter field values. For example, if the only difference between two CTLS-CH instances is the presence or absence of the padding extension parameter, the SED value is 1. Similarly, if the only difference in the supported

Table 5.13. TLS Extensions that include field values in CTLS-CH

Extension type value	Extension name
0x0001	max_fragment_length
0x0005	status_request
0x0007	client_authz
0x0008	server_authz
0x0009	cert_type
0x000a	supported_groups
0x000b	ec_point_formats
0x000d	signature_algorithms
0x000f	heartbeat
0x0010	application_layer_ protocol_negotiation
0x0011	status_request_v2
0x0018	token_binding
0x001b	compress_certificate
0x001c	record_size_limit
0x002b	supported_versions
0x002d	psk_key_exchange_modes
0x0032	signature_algorithms_cert
0x5500	channel_id

versions list specified in the supported version extension parameter is the presence or absence of both TLS 1.1 (0x0302) and TLS 1.0 (0x0301), the SED value is 2; however, how NPF evaluates this kind of difference is not explicitly indicated.

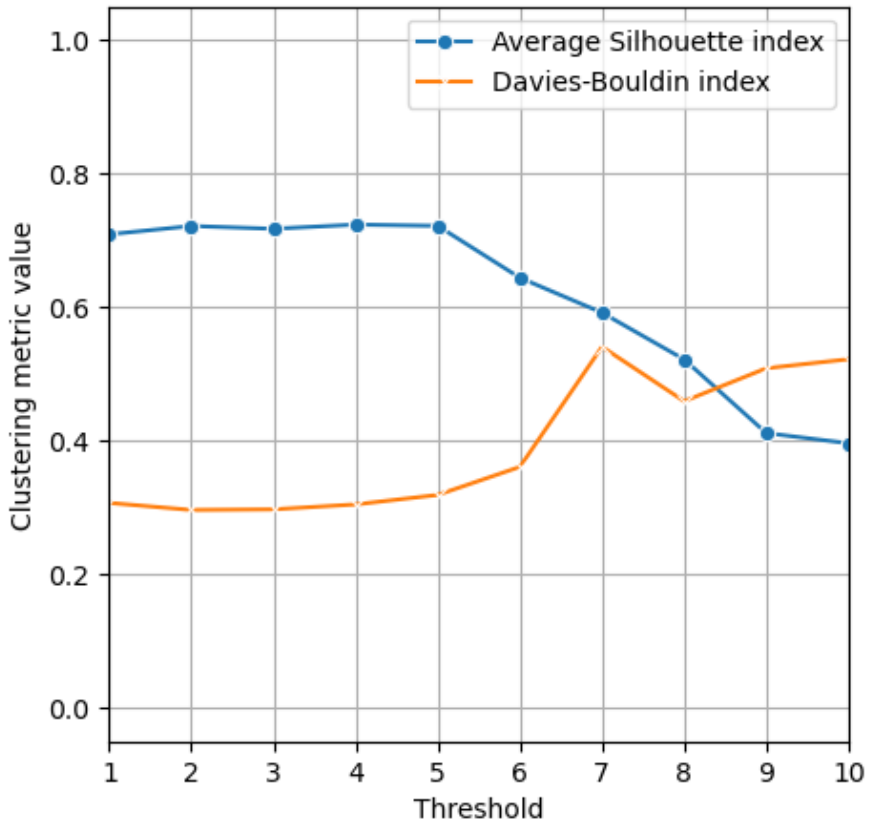
5.7 Additional Analysis

From the MTA dataset, we collected 371 distinct CTLS-CH across the entire period, comprising 202 for TLS 1.2, 125 for TLS 1.3, and 44 for other TLS versions. With these 371 CTLS-CH instances, we conduct an additional analysis to gain deeper insights into parameter changes.

Following prior research, we considered the optimal clustering threshold for grouping TLS parameters based on the TLS parameter edit distance (SED). From the perspective of clustering performance, we calculated the Silhouette and Davies-Bouldin indices (Figure 5.5). Both metrics clearly showed that performance deteriorated when the threshold exceeded 6. A distance of 6 is the minimum distance at which CTLS-CH from TLS 1.2 and TLS 1.3 begin to merge into the same cluster. Within the range of threshold 5 or less, the best values for the Silhouette index and the Davies-Bouldin index were 0.724 at threshold 4 and 0.296 at threshold 2, respectively. Furthermore, when we limited the data to 2020 onwards, when TLS 1.3 began to be widely adopted, the best values for both indices were obtained at threshold 2.

Next, we investigated how the cluster characteristics of TLS 1.2 and TLS 1.3 changed over time (Figure 5.6, Figure 5.7). Comparing TLS 1.2 and TLS 1.3, TLS 1.3 has had more clusters and a smaller average cluster size since 2022. Although the number of clusters decreased for both TLS 1.2 and TLS 1.3 between 2022 and 2024, the number of TLS packets themselves also decreased during this period, suggesting a possible influence that requires further investigation. Conversely, even amidst the decrease in TLS packets, the average cluster size for TLS 1.3 in 2024 reached its highest value ever, suggesting the possibility of greater variation within TLS 1.3 clusters in the future.

We then investigated the co-occurrence rate of variations based on the SED (Figure 5.8). The co-occurrence rate is defined as the measure of how often a pair of distinct CTLS-CH instances with a specific edit distance originates from the same network capture (PCAP file). Specifically, this rate is calculated as the ratio of observed pairs within the same PCAP file to the total number of pairs sharing that same edit distance. The co-occurrence rate is a quantitative indicator that indirectly suggests the presence of Parameter Fluctuation. In the range closer than SED value 6, the co-occurrence rate increases as the distance



Note: Higher values indicate better Silhouette Index performance,
while lower values indicate better Davies-Bouldin Index performance.

Figure 5.5. Clustering metrics in MTA dataset

decreases. Even in the range of SED value 6 or greater, the co-occurrence rate does not exceed 0.4. This suggests that, as proximity decreases, parameter fluctuations originating from the same communication endpoint become more dominant than the incidental observation of close values in neighboring regions.

Finally, we further investigated the specific parameter differences within the same cluster. To derive future implications from the trends of the recently dominant TLS 1.2 and TLS 1.3, we focused our investigation on 2020 onwards and listed the fluctuating parameters in detail (Table 5.14). Our results show that, in addition to the fluctuating parameters observed in prior analysis (such as `server_name`, `status_request`, and `token_binding`), numerous other extensions also fluctuate with context. Regarding version differences, our results also showed that TLS 1.3 exhibits greater variation than TLS 1.2. This in-

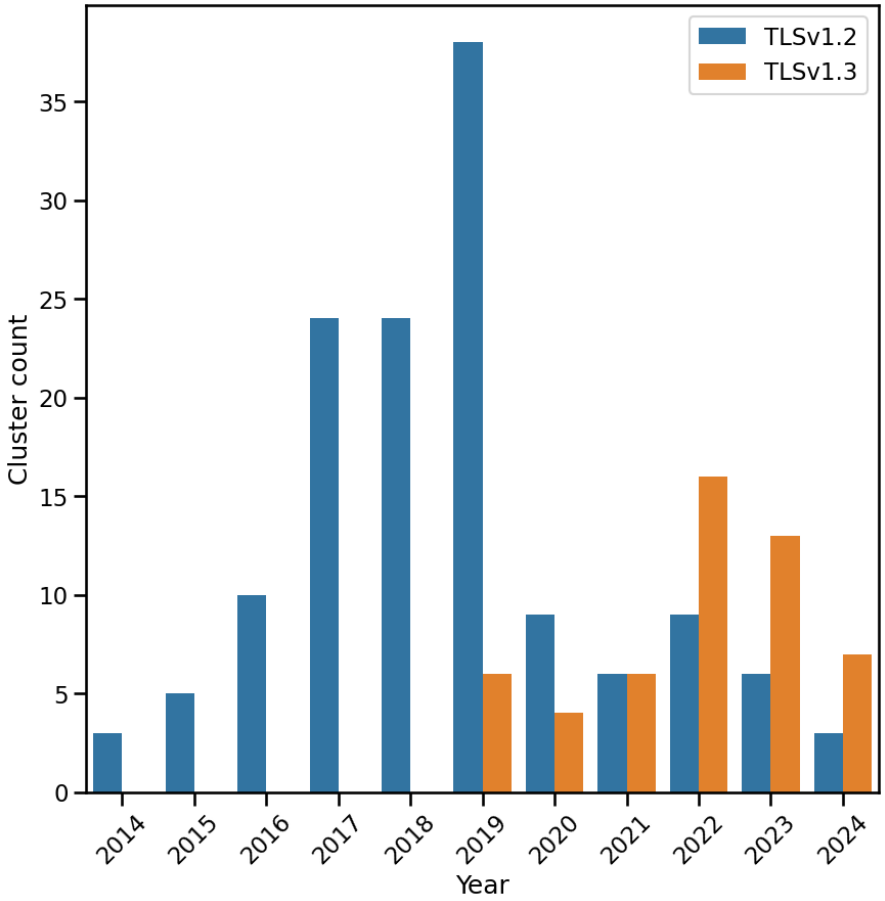


Figure 5.6. Number of cluster in MTA dataset

cludes extension parameters implemented since TLS 1.3, such as `pre_shared_key` and `early_data`, as well as extension parameters standardized in recent years or currently under standardization, such as `encrypted_client_hello` and `delegated_credentials`. Potentially, TLS 1.3 is poised for even greater fluctuations in the future.

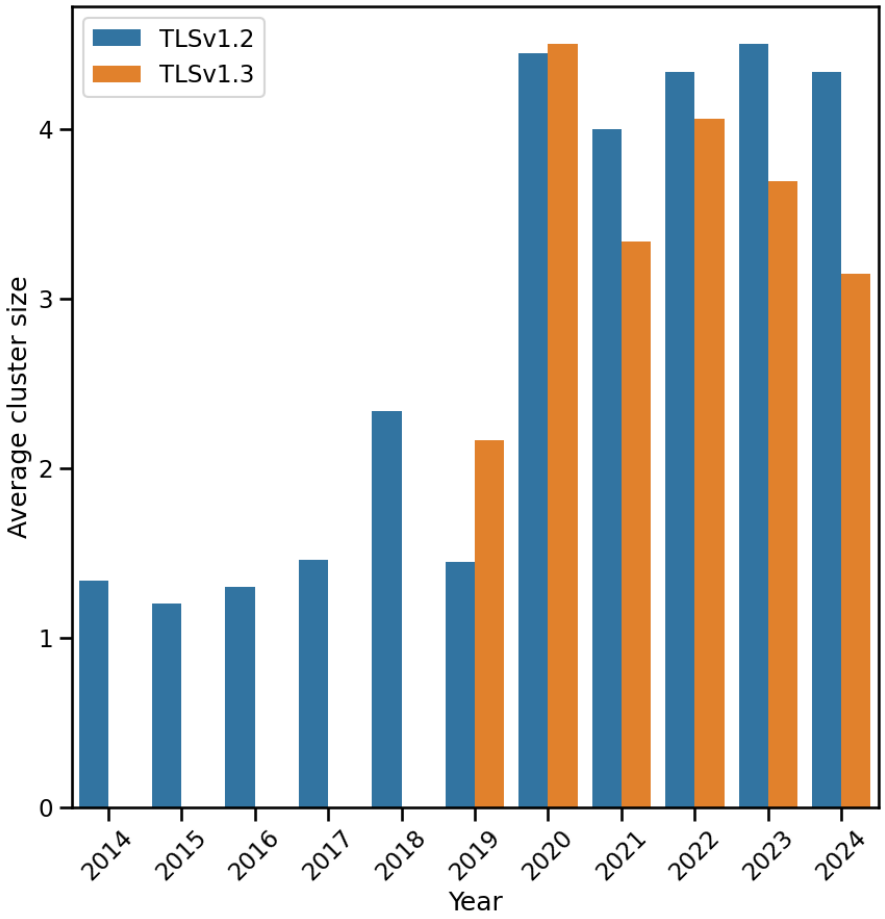


Figure 5.7. Average cluster size in MTA dataset

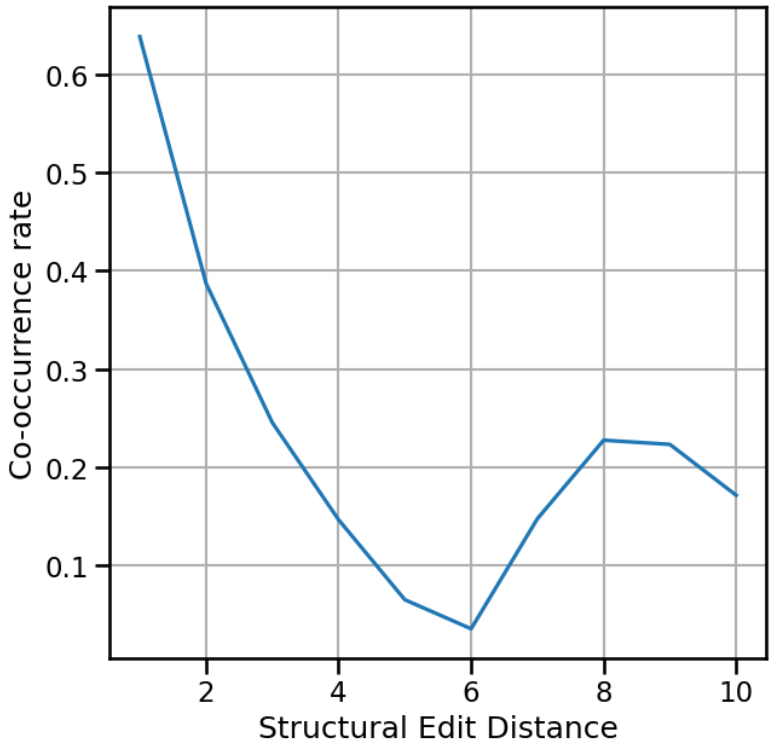


Figure 5.8. Co-occurrence rate in MTA dataset

Table 5.14. Parameter differences within SED 2 (2020 onwards)

TLS parameter difference	TLS1.2	TLS1.3
status_request	✓	✓
application_layer_protocol_negotiation	✓	✓
server_name	✓	✓
token_binding	✓	
cipher_suites (list)	✓	✓
padding	✓	✓
application_layer_protocol_negotiation (list)	✓	✓
session_ticket	✓	✓
signature_algorithms (list)	✓	✓
pre_shared_key		✓
encrypted_client_hello		✓
application_settings		✓
supported_groups (list)		✓
supported_versions (list)		✓
ec_point_formats		✓
delegated_credentials		✓
psk_key_exchange_modes		✓
post_handshake_auth		✓
early_data		✓
cookie		✓

Note: Parameters labeled “list” indicate that the parameter exists in both, but the lists of field values differ.

5.8 Dataset Construction

5.8.1 Requirements

Based on our analysis, we first focused on generating a dataset to effectively evaluate countermeasures against Parameter Fluctuation. This approach was necessary because existing public datasets did not meet the requirements for our analysis.

To evaluate identification performance in a parameter-fluctuating environment, the requirements for the experimental dataset are as follows:

- Sufficient Fluctuations: The dataset must contain an adequate level of parameter fluctuations.
- Clear Session-Level Labeling: The dataset must be clearly labeled by the client at the session level.
- Recent Traffic Data: The dataset should include recent traffic data.

Since this study aims to evaluate robustness against parameter fluctuations, it is essential that the dataset naturally includes sufficient parameter variation.

Furthermore, accurate performance evaluation requires that the ground truth be known at the session level, meaning per fingerprint unit. Given the applicability to TLS 1.3 and the fact that malware began to widely adopt it around 2020 (according to prior research), data from at least 2020 onward is desirable.

Existing datasets, however, fail to meet these requirements. While Stratosphere Lab's Malware Capture Facility Project datasets [85] labeled traffic data of both malicious and benign, they contain data only prior to 2020, making the evaluation of TLS 1.3 difficult. MTA [9] contains abundant fluctuations, as some PCAP files span several days of communication data. However, it is difficult to strictly label the data at the session level because multiple benign and malicious applications are often mixed within the same capture. The recent CipherSprctrum TLS 1.3 dataset [86] provides PCAP data on HTTPS traffic to 40 domains, with client applications (Chrome/Firefox) clearly labeled. However, the dataset shows minimal fluctuations, with only three CTLS-CH instances per browser across specific cipher suites.

Table 5.15. Software versions in dataset generation environment

Software	Version
Selenium Grid	4.36.0
Chrome	91.0 to 141.0 (12 versions)
Edge	93.0 to 141.0 (12 versions)
Firefox	93.0 to 143.0 (12 versions)

5.8.2 Dataset Generation Methods

To generate a reproducible dataset that meets the requirements outlined in Section 5.8.2, we designed a custom traffic-collection environment. Using docker-selenium [87], we treat each browser instance as a container and attach tcpdump [88] to each container individually. This setup allowed us to collect and record traffic independently while operating the respective browser instances in parallel.

The versions of the Selenium Grid and browsers used for data collection are described in Table 5.15. To maximize the potential for parameter variation, browser versions were selected at approximately equal intervals, starting with the oldest functional version and ending with the latest. Traffic was collected across 12 versions per browser.

To ensure the generated communication closely resembles real-world traffic, each browser instance was directed to perform HTTPS access to the Top 30 domains from the Tranco list [89] (generated on 02 November 2025). Crucially, to induce Parameter Fluctuation, during each access attempt, the browser accessed the website once and was immediately forced to reload the screen, triggering a second access. This sequence was repeated 5 times for each domain, resulting in a total of 150 browser accesses per browser instance.

Ultimately, we collected 110,381 TLS Client Hellos (4,440 for TLS 1.2 and 105,941 for TLS 1.3). From this data, we derived 61 distinct CTLS-CH instances.

To support research reproducibility, we have published the configuration of this environment and the collected dataset on GitHub [90].

5.9 Experiment

5.9.1 Experimental Setup

We evaluated the robustness of identification methods against parameter fluctuations using the 61 unique CTLS-CH instances and their corresponding client-label information collected in Section 5.8.2.

We first create a known fingerprint group by randomly sampling a certain percentage of the 61 unique fingerprints. Based on this known fingerprint group, we compared the performance of four strategies for classifying the labels (browser type with version) across all 61 distinct CTLS-CH instances.

The four identification strategies are defined as follows:

- Exact Match (EM): The conventional TLS fingerprinting method. A label is assigned only if a complete match is found within the known fingerprint group; otherwise, the instance is classified as "Unknown".
- Nearest Neighbor (NN): The method adopted by Mercury. If no exact match is found, the instance is classified using the label of the closest matching fingerprint in the known fingerprint group.
- Neighbor Aggregation (NA): A method designed based on the implication derived in Section 5.7. This strategy aggregates the labels of all fingerprints within a certain threshold and assigns a collective decision. We selected the optimal cluster threshold, SED distance 2, from the preliminary analysis.
- Exact Match after De-fluctuation (EM-D): This strategy involves a preprocessing step in which known fluctuating parameters are removed from the fingerprints (de-fluctuated), followed by Exact Match classification. The target parameters for removal were those marked without a "list" label in Table 5.14.

The experiment varied the sampling rate of the known fingerprint group from 10% to 90% in 10% increments. For each sampling rate, 1,000 random samples were drawn, and the four strategies were used to identify clients based on the resulting known fingerprint group.

Given the nature of TLS fingerprint, where multiple different clients can share the same value, TLS fingerprinting is inherently a multi-label classification problem. Indeed, our dataset contains many instances where multiple browser versions share the same CTLS-CH. Therefore, we adopted evaluation metrics suitable for multi-label classification:

- Hamming Loss: Calculated by averaging the Hamming distance of the label vectors for each identification attempt. A lower value indicates fewer misclassifications.
- Average Jaccard Index: Calculated by averaging the ratio of the number of elements in the intersection and union of the two label sets (predicted and ground truth). A higher value indicates greater similarity between the identification result and the ground truth.
- Weighted Average F1-Score: Calculated as the weighted average of the F1-scores per label, weighted by the frequency of label occurrence. A higher value indicates better-balanced performance in terms of precision and recall.

5.9.2 Results

The Hamming Loss results (Figure 5.9) show that, except for the Neighbor Aggregation (NA) strategy, identification performance generally improves as the sampling rate (i.e., the number of known parameter sets) increases. Overall, the Nearest Neighbor (NN) strategy exhibited better performance than the simple Exact Match (EM). The Exact Match after De-fluctuation (EM-D) strategy demonstrated the best performance at low sampling rates (30% or less). Conversely, when the sampling rate exceeded 60%, EM-D resulted in more misclassifications than EM. NA had the worst misclassification performance among all strategies.

The results for the Average Jaccard Index (Figure 5.10) generally followed the same trends observed in the Hamming Loss analysis. NN again showed better overall scores, but EM-D achieved the best value at sampling rates of 30% or less. NA performed better than EM at sampling rates below 40%, but its performance deteriorated as the sampling rate increased.

For the Weighted Average F1-Score (Figure 5.11), the range of sampling rates where EM-D achieved the best value expanded. EM-D outperformed NN in regions with a sampling rate of approximately 50% or less. The tendency for the performance of NA and

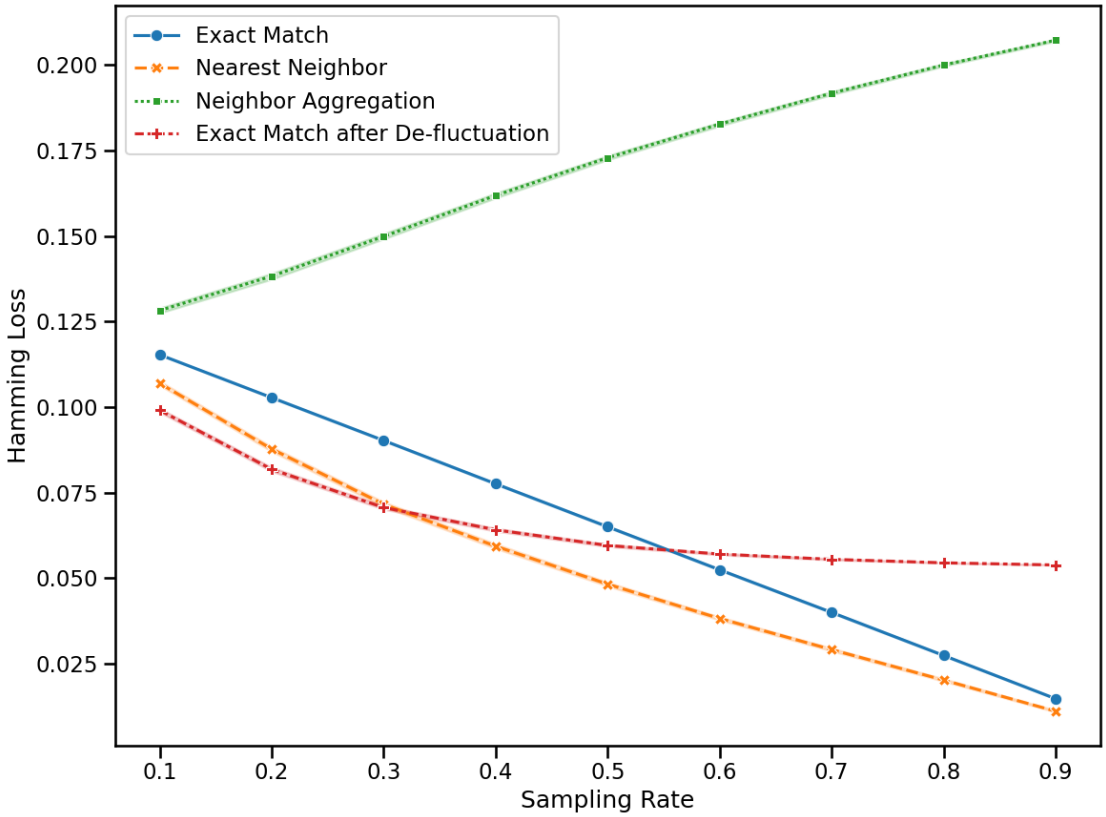


Figure 5.9. Experiment result (Hamming Loss)

EM to reverse around the 30% sampling rate was consistent with the trend observed in the Average Jaccard Index.

5.10 Discussion

5.10.1 Interpretation of Results

The experimental results from Section 5.9.2 provide several key insights regarding the robustness of fingerprinting methods against parameter fluctuations.

The Exact Match after De-fluctuation (EM-D) strategy consistently demonstrated the best performance at low sampling rates (up to 30-50%) across all metrics. This strongly suggests that when the known fingerprint group is sparse, the negative impact of parameter fluctuation is dominant. Removing the fluctuating parameters allows the sys-

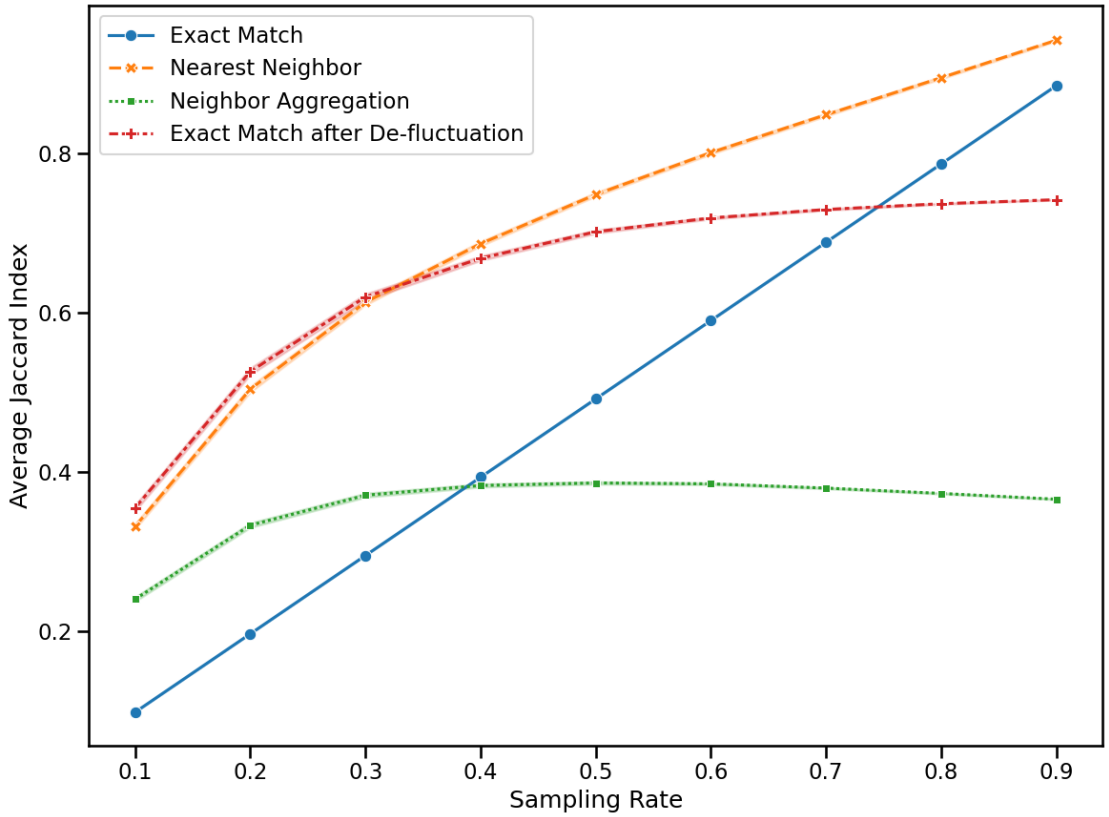


Figure 5.10. Experiment result (Average Jaccard Index)

tem to correctly identify the core client characteristics, making EM-D the most effective countermeasure in complex traffic environments with a high proportion of clients whose characteristics are unknown. The decline in EM-D performance at high sampling rates indicates that the benefit of noise removal is outweighed by the loss of discriminative power. Specifically, removing fluctuating parameters narrows the feature space, leading to increased collisions when structurally similar but distinct clients are incorrectly identified as identical when the known fingerprint set is rich.

Although conducted in a limited experimental setting, the numerical results demonstrate the high utility of the Exact Match after De-fluctuation (EM-D) strategy. Specifically, within the 30% or less sampling rate range, the Weighted Average Precision increased by 43.3% to 56.5% compared to the baseline Exact Match (EM) strategy. This significant improvement indicates strong resilience against the risk of false negatives (missed detections) caused by parameter fluctuations. In contrast, the Weighted Average Recall

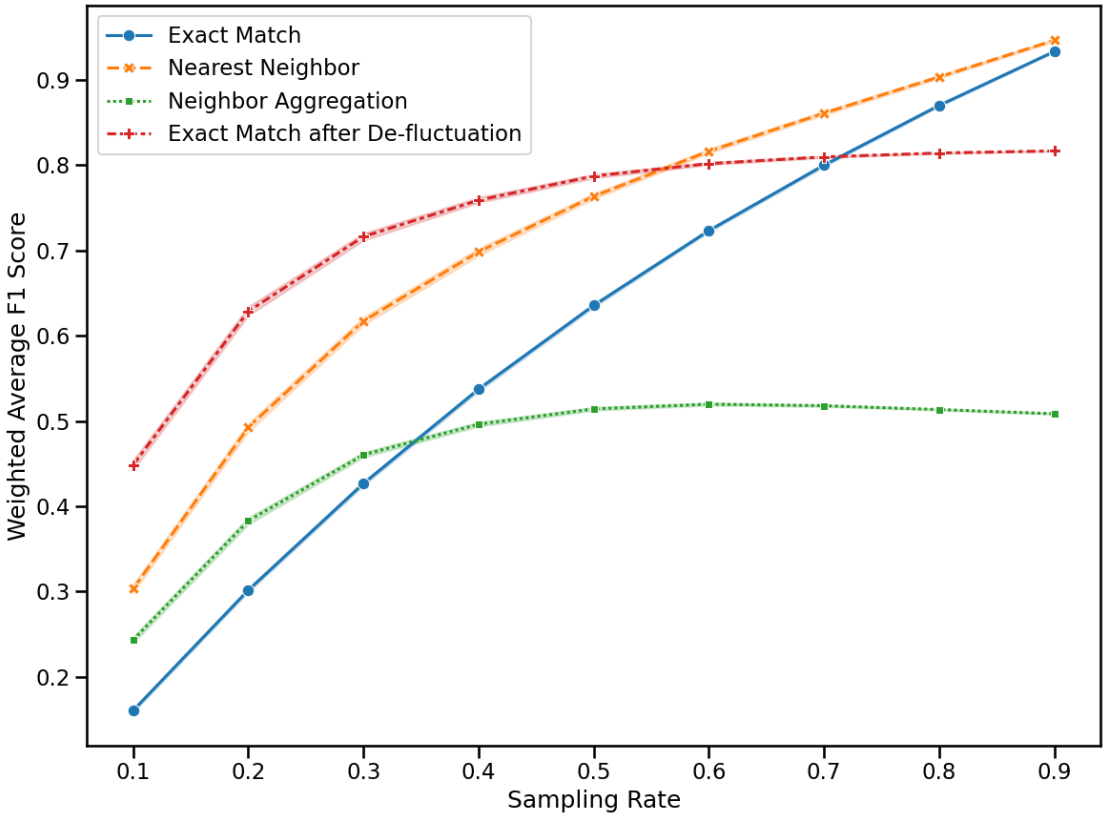


Figure 5.11. Experiment result (Weighted Average F1-Score)

decreased by 8.0% to 24.6% compared to EM, a reduction likely attributable to collisions arising from the narrower feature space, suggesting a risk of false positives (over detection). Nevertheless, the Weighted Average F1-Score improved by 28.9% to 32.7%, showing an overall higher potential than existing methods.

The Nearest Neighbor (NN) strategy generally performed better overall than the conventional Exact Match (EM). This confirms the prior work's finding [58] that, even when an exact match is missing due to fluctuations, searching the nearest neighbor effectively compensates for minor variations and significantly reduces the number of unidentifiable instances compared to EM.

The Neighbor Aggregation (NA) strategy performed the worst overall in terms of misclassification (Hamming Loss) and showed performance degradation as the sampling rate increased. This outcome contradicts the implication from Section 5.7 that fluctuations are dominant within close proximity.

The most likely reason for this failure is that aggregating all labels within a small threshold (SED ≤ 2) leads to an excessive number of false positives (misclassifications). While parameter fluctuation creates multiple unique CTLS-CH from the same client, the low threshold likely still captures different clients that happen to be structurally similar, especially as the known fingerprint group grows. This results in label pollution, where the ground-truth label is diluted by many incorrect labels from close neighbors, leading to poor performance despite fluctuations.

5.10.2 Applicable Use Cases

The optimal TLS fingerprinting strategy varies depending on use cases. Here, we organize the analysis around two crucial factors: tolerance for False Negatives (FN) and False Positives (FP), and the volume of known fingerprints.

In use cases such as threat detection with IDS/IPS or threat hunting with SIEM, a single missed detection (False Negative, FN) can cause severe damage, making these scenarios more sensitive to FN than to False Positives (FP). Moreover, in enterprise networks or large-scale cloud environments, the high diversity of clients makes it impractical to grasp all fingerprints, resulting in a low ratio of known fingerprints. Under these conditions, a strategy that maximizes discriminative power by removing noise, the De-fluctuation strategy (i.e., Exact Match after De-fluctuation, EM-D), is suitable. This is because it drastically reduces the FN risk associated with identifying novel or fluctuating threats. However, since excessive FPs lead to increased alert fatigue, it is more practical to use a multi-context detection logic that combines the TLS fingerprinting result with other contextual information, rather than relying solely on the fingerprinting output.

In contrast, the Nearest Neighbor (NN) strategy, which is highly versatile and maintains a good balance of FN/FP tolerance, is the primary candidate for filtering use cases in commercial service environments where the negative impact of misblocking (FP) is significant, or in organizational situations where suppressing alert fatigue is prioritized due to limited operational resources. For cases that are particularly sensitive to FP, the even stricter Exact Match (EM) strategy may be required.

Similarly, in controlled communication environments where the proportion of known fingerprints is high, NN or EM are preferred, as the need for highly accurate noise removal in EM-D diminishes.

5.10.3 Limitations

Our study focused on analyzing parameter fluctuations using a controlled dataset generated by specific versions of three major web browsers (Chrome, Edge, Firefox) accessing the Tranco top 30 domains. While this methodology allowed us to precisely control the environment and induce specific fluctuations, the findings have limitations in terms of versatility. Although traffic was generated to simulate real-world conditions, the complexity and dynamic nature of real network traffic were not fully replicated. Specifically, further validation is required for applicability to malicious programs such as malware.

At this stage, our research does not include feasibility considerations regarding computational cost or processing time within its scope. Implementing the Nearest Neighbor (NN) strategy requires proximity searches, which are generally computationally expensive operations. Challenges such as database structure and search optimization must be addressed for practical implementation.

Our work focused on mitigating natural parameter fluctuations, not deliberate evasion (anti-fingerprinting) techniques. Adversaries may employ dynamic parameter randomization or fingerprint spoofing, which could render current fluctuation-removal strategies less effective.

Also, the fluctuating parameters identified in this study may not remain constant. Continuous monitoring is required to update the De-fluctuation parameters to maintain the performance of the EM-D strategy.

5.11 Conclusion

In this chapter, we focus on the evolution of TLS Client Hello parameters in malware C2 communications and examine their characteristics before and after the spread of TLS 1.3, using a long-term dataset. We define two phenomena within Parameter Variation: Parameter Fluctuation and Parameter Drift. Using an unprecedented dataset spanning over a decade, we demonstrate for the first time that these phenomena occur in real-world malware TLS communications. We demonstrate, through concrete examples, that existing TLS fingerprinting techniques do not adequately account for these Parameter Variations and are vulnerable to parameter fluctuations. Moreover, we reveal that the Parameter

Variations we confirmed are not, by their nature, limited to malware communications and can also occur in legitimate software.

We also address the critical challenge of Parameter Fluctuation in TLS fingerprinting.

We introduce the Compacted Protocol Representation (CPR) and the Structural Edit Distance (SED) to precisely quantify and analyze parameter differences. Through preliminary analysis, we provide insights into the optimal granularity for clustering TLS parameter variations. We also introduce a reproducible methodology for generating a dataset exhibiting TLS Parameter Fluctuations. Using the dataset, we organize and evaluate effective identification logic based on TLS fingerprints. The Exact Match after De-fluctuation strategy proves to be the most effective in environments with fewer known fingerprints.

To transition the insights from this study into a practical, resilient TLS fingerprinting system, it is essential to expand the scale of the experiments and the diversity of client fingerprints beyond the browser traffic assessed here. Specifically, constructing effective malware fingerprint datasets is a critical challenge.

Performance also must be evaluated under real-world conditions. This includes conducting assessments based on the actual fingerprint occurrence frequency in operational environments to provide a more accurate metric of operational performance.

In a real-world environment, we need to consider computational cost and processing efficiency and establish a practical implementation.

The failure of the Neighbor Aggregation strategy indicates that there is still room for improvement in the aggregation logic. The label pollution issue can potentially be resolved by adopting more complex strategies than simply incorporating all information from proximate fingerprints.

Chapter 6
TLS Parameter Spoofing

6.1 Introduction

Identification techniques such as TLS fingerprinting rely on the premise that communications are properly adhering to the TLS protocol. However, in a world where TLS is widespread, attackers may employ tactics that deliberately introduce communications that appear to be legitimate TLS. As this technique is commonly known as FakeTLS, we adopt this term throughout the thesis.

Generally, FakeTLS first performs the TLS handshake. It then pretends that the TLS session is continuing, when in fact it is exchanging encrypted data using a different cipher than the one negotiated in the handshake. Some threat actors have been identified to further develop this technique to skip the TLS handshake and generate encrypted C2 communications with packet structures that spoof TLS communications [91]. In this study, we define FakeTLS as a method of spoofing TLS communication regardless of the presence of a TLS handshake. This relatively new type of threat began to be reported around 2016. It poses a challenge that we believe is difficult to distinguish because it mimics benign TLS communication with the clear intention of bypassing existing security solutions such as Deep Packet Inspection (DPI).

To date, multiple threat actors have been confirmed to be using FakeTLS. US government agencies have reported several cases of the North Korean threat group Lazarus (aka HIDDEN COBRA) using FakeTLS for malware C2 communications. Specifically, the following malware BADCALL [11], TAINTEDSCRIBE [12], and PEBBLEDASH [13] combined FakeTLS with XOR/AND operations, the Linear Feedback Shift Register (LFSR) algorithm, and RC4, respectively, for C2 communication. Zscaler has reported that a threat group named Higaisa used FakeTLS and AES encryption [92]. Palo Alto Networks has reported that a threat group named Scarlet Mimic started using FakeTLS in the process of improving the FakeM malware [91]. Some FakeM variants have been identified

that do not perform TLS handshakes.

The use of FakeTLS is not limited to threat actors. Telegram, the instant messaging service, has implemented FakeTLS in its official proxy tool MTProxy to avoid blocking Telegram communications [93].

Having considered these factors, we set the following as our research question. **Is it possible to detect spoofed TLS communications from encrypted exchanges over the communication path, under conditions where a TLS handshake does not exist, or even if it does exist, its contents cannot be trusted?**

Against such stealthy threats, which are difficult to detect and block with a single detection mechanism, the approach named "Threat Hunting" is used. In this paper, we adopt Sqrrl's definition of Threat Hunting [94], which is "the process of proactively and iteratively searching through networks to detect and isolate advanced threats that evade existing security solutions". The goal of our research is to clarify the possibility of identifying FakeTLS from the encrypted data stream and its usefulness as new context information in Threat Hunting.

In this study, we propose a classifier called TLS Lie Detector, which determines FakeTLS based on the randomness of the TLS application data. We conducted experiments on a dataset with normal TLS and FakeTLS.

The results demonstrate that our detector can detect FakeTLS with an F0.5 score of 0.88, an F1 score of 0.78, an F2 score of 0.70 and a Matthews correlation coefficient of 0.74.

6.2 Related Work

Since our research focuses on the randomness of encrypted traffic, we briefly review related research using data randomness, especially regarding Shannon entropy and NIST SP800-22, for traffic identification.

Olivain *et al.* observed the entropy value at the beginning of a session, focusing on the fact that the entropy value of transmitted and received data during normal communication such as SSH and SSL transitions on a unique curve. They proposed Net-Entropy, a mechanism that detects anomalies based on deviations from predefined entropy transitions, and demonstrated that it can detect attacks that exploit vulnerabilities in SSH and

SSL [95]. When the entire session is regarded as a single chunk of information, entropy is calculated using a portion of that data. For this reason, Olivain *et al.* named this entropy "sampling entropy". White *et al.* used the entropy of data sequences sampled from packets as one of the features in distinguishing between opaque communications, where encryption or compression is used, and other transparent communications from large traffic [96]. Zhang *et al.* proposed a method to distinguish between high-entropy and low-entropy communications using cumulative entropy and per-packet entropy as an initial phase for detecting botnet encrypted communications [97]. Luo *et al.* proposed a method to measure a sampling entropy while sliding over data with a specific window size. They used the entropy values to create fingerprints and applied them to anomaly detection, showing a certain effectiveness in detecting botnets that carry out encrypted C2 communications [98].

Zhao *et al.* proposed an index named differentiability for the identification of encrypted data using NIST SP800-22. They evaluated seven types of encrypted and compressed data formats and concluded that the cumulative sums (cusum) test was the most effective test [99]. Sengupta *et al.* investigated features using NIST SP800-22 for the purpose of identifying communication in mobile applications. They conducted experiments on 11 types of encryption algorithms. Based on their validation that the p-value of the discrete fourier transform (spectral) test is most suitable for identification, they proposed a feature based on frequency transformation of packet data [100].

6.3 Preliminary Experiments

6.3.1 The Design of The Experiments

In order to examine the features applicable for TLS cipher suite identification, we collected traffic data of TLS communications and conducted a series of research experiments. In this section we describe the design of those experiments.

6.3.1.1 Dataset

Our experiments are targeted at HTTPS, one of the most popular applications, and in particular TLS v1.2 and TLS v1.3, which are expected to be in continued use in the

Table 6.1. Preliminary experimental environment

	OS	SSL/TLS Library	Web Server/ Client
Server	Alpine Linux (3.9)	OpenSSL (1.1.1b)	Nginx (1.17.0)
Client A	Alpine Linux (3.9)	OpenSSL (1.1.1b)	Curl (7.64.0)
Client B	Alpine Linux (3.9)	wolfSSL (4.0.0)	Curl (7.65.1-DEV)

Note: Version information is written in parentheses.

future. To collect our dataset, we used Docker to run an HTTPS server, HTTPS clients and traffic capture. All instances use Alpine Linux x86 (64-bit) as the operating system. In order to verify the differences between SSL/TLS libraries, we created two types of HTTPS clients with different SSL/TLS libraries. The configuration of each environment is described in Table 6.1.

With the environment described in Table 6.1, we collected traffic data (PCAP files) for a total of 13 cipher suites for which HTTPS access was established, and repeated 1000 times by each HTTPS client. The cipher suites for the datasets collected by each client are shown in Table 6.2.

6.3.1.2 The Design of Features

The candidate features used to identify the cryptographic techniques were selected on the basis of NIST SP 800-22. These include the p-value of each test, the statistics used in the course of the test, and the conversion of these statistics into percentages according to the bit length. Each of the NIST SP 800-22 tests has a recommended value for the input bit length. Since it was expected that the data size of a number of communication sessions might not meet the recommended values for some tests, we decided to use the seven tests from the NIST SP 800-22, with short recommended values for bit length.

We extracted only the TLS application data from the collected dataset (PCAP file) and generated binary data through combining them by session. In our experiments, we did not set an upper limit on the data size, and used the whole range of extracted data

Table 6.2. Preliminary experimental dataset

TLS Ver.	Cipher suite				Client	
	Key exchange	Authentication	Encryption (Key length)	MAC	A	B
1.2	DH	RSA	AES(128)	SHA256	✓	✓
1.2	DH	RSA	AES(256)	SHA256	✓	✓
1.2	DH	RSA	AESGCM(128)	AEAD		✓
1.2	DH	RSA	AESGCM(256)	AEAD	✓	✓
1.2	DH	RSA	CHACHA20/ POLY1305(256)	AEAD	✓	✓
1.2	ECDH	RSA	AES(128)	SHA256	✓	✓
1.2	ECDH	RSA	AES(256)	SHA384	✓	✓
1.2	ECDH	RSA	AESGCM(128)	AEAD	✓	✓
1.2	ECDH	RSA	AESGCM(256)	AEAD	✓	✓
1.2	ECDH	RSA	CHACHA20/ POLY1305(256)	AEAD	✓	✓
1.3	(Negotiate)	(AEAD)	AESGCM(128)	AEAD	✓	✓
1.3	(Negotiate)	(AEAD)	AESGCM(256)	AEAD	✓	✓
1.3	(Negotiate)	(AEAD)	CHACHA20/ POLY1305(256)	AEAD	✓	✓

until the end of the session. However, in the Test for the Longest Run of Ones in a Block, there are three different sets of parameters depending on the bit length. For this reason, we used only the first 6271 bits for data that exceeds this range, so that all data would fit into the smallest parameter set, in order to evaluate with aligned parameter sets.

The selected tests and the defined features, as well as the minimum, average, maximum, and standard deviation values for each of the features calculated using the dataset, are shown in Table 6.3.

Each of the features are described in the following paragraphs. In common with all tests, the "p-value" represents the p-value of the test defined in NIST SP 800-22 and n represents the bit length of the data.

For the features based on the Monobit Test, we used the p-value as well as S_{0n}/n,

S_{1n}/n, $S_{0n}/\sqrt{n}$, and $S_{1n}/\sqrt{n}$, where S_{0n} and S_{1n} are the total number of 0s and 1s in a bit sequence, respectively. Also, S_{obs} is a statistic that is also used in the test and is defined as follows.

$$S_{obs} = \frac{|S_{0n} - S_{1n}|}{\sqrt{n}}$$

The Frequency Test within a Block and the Test for the Longest Run of Ones in a Block split the dataset into blocks. Hence, for the features based on these tests, we adopted the chi-square value $\chi^2(obs)$ used in these tests, as well as the p-value.

For the features based on the Runs Test, we used the p-value and $V_n(obs)/\sqrt{n}$, where $V_n(obs)$ is the number of runs.

In the Serial Test, we set the pattern size m to 4 bits. Since two different p-values are used in this test, we referred to them as p-value1 and p-value 2. In addition to these, we used ψ^2_m, ψ^2_{m-1}, ψ^2_{m-2}, $\nabla\psi^2_m$, and $\nabla^2\psi^2_m$, which are computed during the course of the test. The definition is the same as the one in NIST SP 800-22, so please refer to that for more information.

For the features based on the Approximate Entropy Test, with the pattern size m fixed at 3 bits, we used the p-value, $\varphi^{(m)}$, $\varphi^{(m+1)}$, and the chi-square value χ^2, and the approximate entropy value $ApEn(m)$. The definition here is also the same as the one in NIST SP 800-22.

The Cumulative Sums (Cusum) Test has $mode = 0$ (forward addition) and $mode = 1$ (backward addition) according to the direction of addition, and p-values are calculated for each direction. In addition to these p-values, we used $z/\sqrt{n}$, where z is the maximum value at addition.

6.3.2 Research Experiments

We used the above selected features to verify the effective features (combinations of features). For the classification algorithms, we used decision trees and random forests because their decision-making logic is very clear. We also used 10-fold cross-validation to estimate the combination of the TLS version and the cipher suite. Finally, we evaluate the features (combination of features) by using the average of the accuracy.

In decision trees and random forests, which are based on the decrease of the Gini coefficient, the degree of contribution (importance) of each feature in the classification

can be calculated. Therefore, we also evaluated each of the features from the perspective of its importance.

We conducted evaluations based on the following four points.

- The classification performance of a single feature
- The classification performance of dual features
- The importance of each feature when all features are used
- The classification performance when combining features of the Frequency (Monobit) Test and the Runs Test

The Classification Performance of a Single Feature

When only a single feature from the designed features was used, the Frequency (Monobit) Test and Runs Test features occupied the top 10 positions in classification performance. The results are shown in Table 6.4.

The Classification Performance of Dual Features

When two features from the designed features were combined, the Frequency (Monobit) Test features occupied the top 10 positions in classification performance. The results are shown in Table 6.5.

Table 6.3. The defined features and their statistics in preliminary experiment

NIST SP 800-22 Test	Features	Min.	Avg.	Max.	Std Dev
Frequency (Monobit) Test	p-value	0.000	0.496	1.000	0.288
	S_{0n}/n	0.479	0.500	0.522	0.005
	S_{1n}/n	0.478	0.500	0.521	0.005
	$S_{0n}/\sqrt{n}$	42.15	50.37	69.79	9.823
	$S_{1n}/\sqrt{n}$	42.15	50.37	69.79	9.823
	S_{obs}	0.000	0.803	3.938	0.601
Frequency Test within a Block	p-value	0.000	0.499	1.000	0.287
	$\chi^2(obs)$	53.21	99.05	164.6	13.94
Runs Test	p-value	0.000	0.500	1.000	0.289
	$V_n(obs)/\sqrt{n}$	42.05	50.38	69.92	9.837
Test for the Longest Run of	p-value	0.000	0.497	0.993	0.284
Ones in a Block	$\chi^2(obs)$	0.084	2.990	31.45	2.385
Serial Test	p-value1	0.000	0.498	1.000	0.288
($m = 4$)	p-value2	0.000	0.498	1.000	0.289
	ψ^2_m	0.586	15.04	91.97	8.647
	ψ^2_{m-1}	0.033	7.025	61.76	5.495
	ψ^2_{m-2}	0.000	3.020	36.89	3.017
	$\nabla\psi^2_m$	0.260	8.023	41.12	4.019
	$\nabla^2\psi^2_m$	0.015	4.019	24.36	2.839
Approximate Entropy Test	p-value	0.000	0.498	1.000	0.288
($m = 3$)	$\varphi^{(m)}$	-4.382	-4.382	-4.379	0.000
	$\varphi^{(m+1)}$	-5.075	-5.074	-5.070	0.000
	χ^2	0.2594	8.028	41.842	4.021
	$ApEn(m)$	0.6910	0.6927	0.693	0.000
Cumulative Sums (Cusum) Test	p-value[a]	0.000	0.500	1.000	0.289
	p-value[b]	0.000	0.501	1.000	0.289
	$z/\sqrt{n}$ [a]	0.331	1.253	4.373	0.510
	$z/\sqrt{n}$ [b]	0.345	1.252	3.983	0.511

[a]mode=0, [b]mode=1

Table 6.4. Classification performance of a single feature in preliminary experiment (Top 10)

Feature	Algorithm	Accuracy [%]
S_{obs}	DecisionTree	47.15
p-value (Monobit)	DecisionTree	47.14
$S_{1n}/\sqrt{n}$	DecisionTree	47.02
$S_{0n}/\sqrt{n}$	DecisionTree	47.02
S_{obs}	RandomForest	46.83
p-value (Monobit)	RandomForest	46.82
$V_n(obs)/\sqrt{n}$	DecisionTree	46.77
$S_{1n}/\sqrt{n}$	RandomForest	46.77
$S_{0n}/\sqrt{n}$	RandomForest	46.77
$V_n(obs)/\sqrt{n}$	RandomForest	46.56

Table 6.5. Classification performance of dual features in preliminary experiment (Top 10)

Feature 1	Feature 2	Algorithm	Accuracy [%]
S_{1n}/n	$S_{1n}/\sqrt{n}$	DecisionTree	48.10
S_{1n}/n	$S_{0n}/\sqrt{n}$	DecisionTree	48.10
$S_{1n}/\sqrt{n}$	S_{0n}/n	DecisionTree	48.10
S_{1n}/n	$S_{0n}/\sqrt{n}$	RandomForest	48.08
S_{1n}/n	$S_{1n}/\sqrt{n}$	RandomForest	48.08
S_{0n}/n	$S_{0n}/\sqrt{n}$	DecisionTree	48.07
$S_{1n}/\sqrt{n}$	S_{0n}/n	RandomForest	48.05
S_{0n}/n	$S_{0n}/\sqrt{n}$	RandomForest	48.03
p-value (Monobit)	$S_{0n}/\sqrt{n}$	DecisionTree	48.02
p-value (Monobit)	$S_{1n}/\sqrt{n}$	DecisionTree	48.02

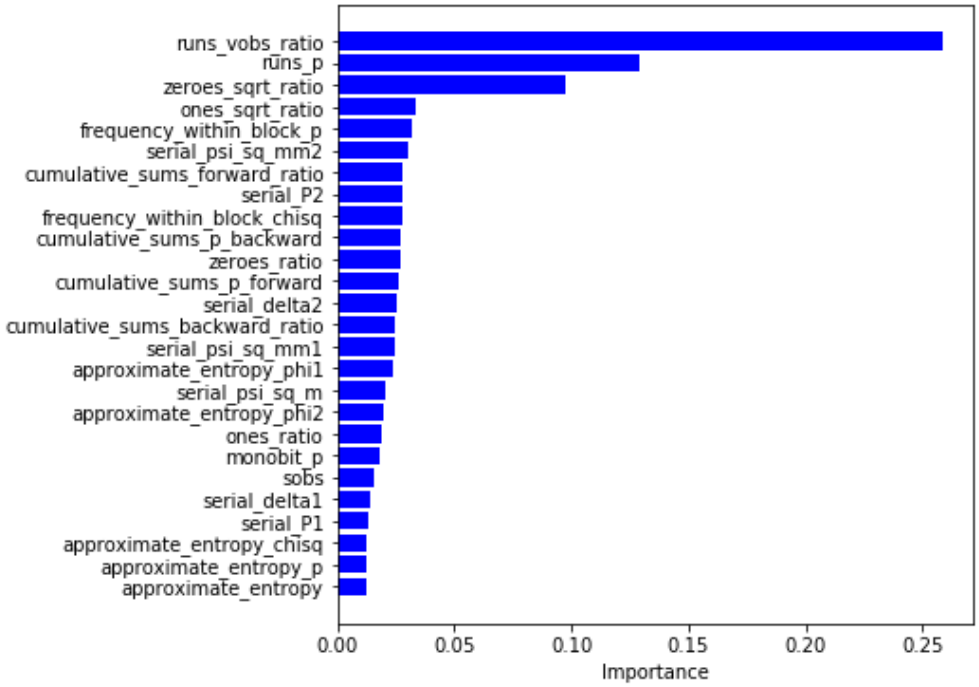


Figure 6.1. Importance of each feature in preliminary experiment (Decision Tree)

The Importance of Each Feature When All Features Are Used

Figures 6.1 and 6.2 show the results of the calculation of the importance of each feature when all of the designed features were used. In terms of importance, the p-value and $\frac{V_n(obs)}{\sqrt{n}}$ of the Runs Test and $\frac{S_{0n}}{\sqrt{n}}$ of the Frequency (Monobit) Test stand out in decision trees. Similarly, in random forests, $\frac{S_{0n}}{\sqrt{n}}$ and $\frac{S_{1n}}{\sqrt{n}}$ of the Frequency (Monobit) Test, and $\frac{V_n(obs)}{\sqrt{n}}$ of the Runs Test are outstanding.

The Classification Performance When Combining Features of the Frequency (Monobit) Test and the Runs Test

The experiments so far have shown that the features of the Frequency (Monobit) Test and the Runs Test have high importance in classification. In order to find the best combination of the features, we examined the four patterns specified in Table 6.6.

Pattern 1 is a set of features included in the top 10 of the classification performance of a single feature. Pattern 2 is a set of features included in the top 10 of the classification performance of dual features. Pattern 3 consists of the four features with particularly high importance. Pattern 4 consists of all the features of the Frequency (Monobit) Test

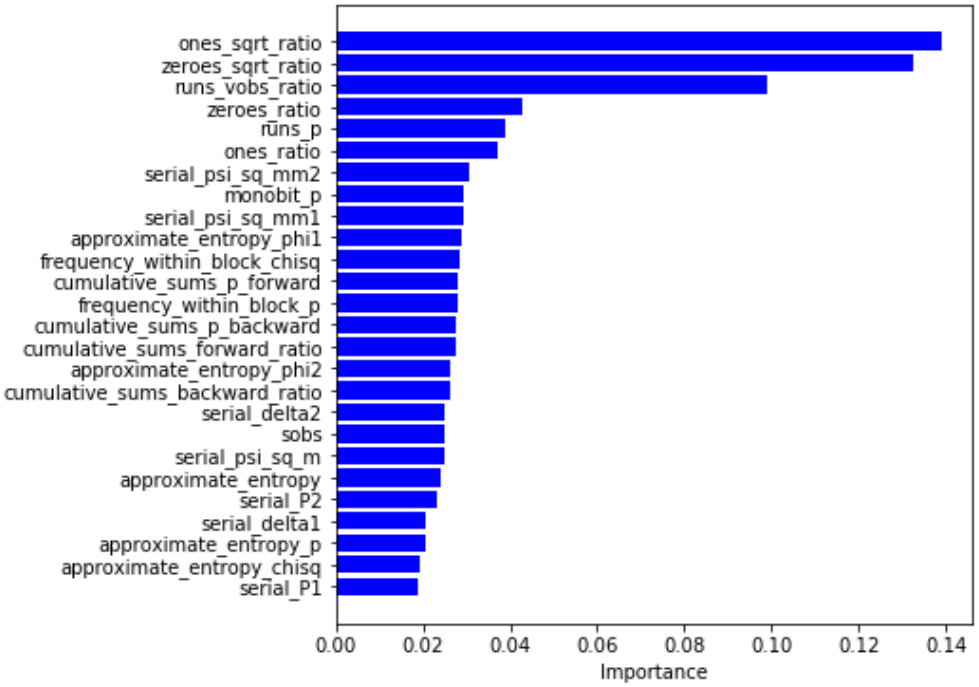


Figure 6.2. Importance of each feature in preliminary experiment (Random Forest)

and the Runs Test defined in this experiment.

For each pattern, we used decision trees and random forests to classify cryptographic techniques, and evaluated their performance using 10-fold cross-validation. The results are shown in Figure 6.3.

All of these patterns achieve higher accuracy than random guessing (7.70%), among which Pattern 2 achieves the highest accuracy for the decision tree and Pattern 4 for the random forest. These results suggest that Patterns 2 and 4 are the most effective feature combinations.

Note that this experiment was constructed using traffic collected from the same OS environment. High accuracy may have been achieved due to confounding factors, such as insufficient entropy in the random number seed. Therefore, the generality and universality of the results have not been proven. Further experiments are necessary to verify universality, such as testing the results under entirely different environmental conditions.

Table 6.6. Feature combination of Frequency (Monobit) Test and Runs Test in preliminary experiment

	Frequency (Monobit)						Runs	
	$\frac{S_{0n}}{n}$	$\frac{S_{1n}}{n}$	$\frac{S_{0n}}{\sqrt{n}}$	$\frac{S_{1n}}{\sqrt{n}}$	S_{obs}	p-value	$\frac{V_n(obs)}{\sqrt{n}}$	p-value
1			✓	✓	✓	✓	✓	
2	✓	✓	✓	✓		✓		
3			✓	✓			✓	✓
4	✓	✓	✓	✓	✓	✓	✓	✓

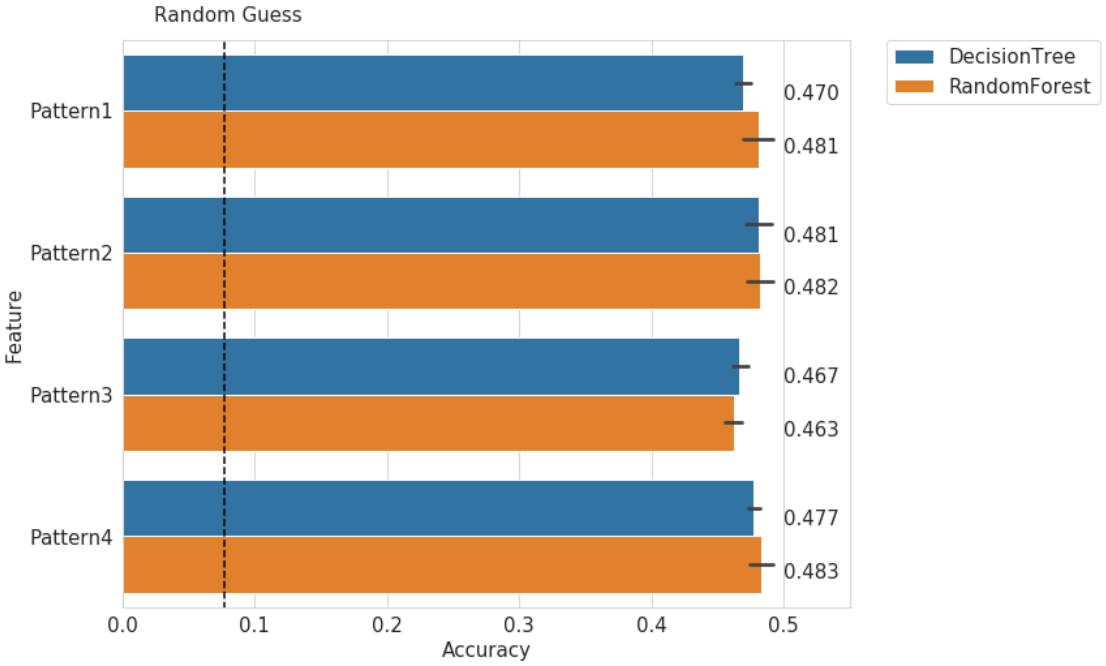


Figure 6.3. Classification performance using feature combinations of the Frequency (Monobit) Test and the Runs Test in preliminary experiment

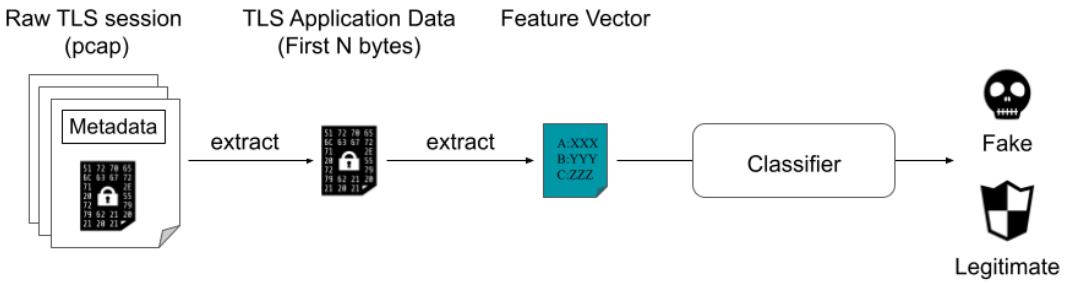


Figure 6.4. Process Overview of TLS Lie Detector

Table 6.7. Features used in TLS Lie Detector

Features	Description
H	Entropy
$\bar{S_n}$	Mean difference between the number of 0s and 1s
$V_n(\bar{obs})$	Average number of runs

6.4 The Design of the TLS Lie Detector

In this section, we demonstrate our proposed method, TLS Lie Detector. Figure 6.4 shows an overview of the process of our method.

6.4.1 Feature Vector

We selected three types of features related to data randomness. One is entropy. We calculate the entropy of the extracted byte sequence. The other two are the values related to NIST SP800-22 tests. Based on our preliminary experiment, the features derived from the Monobit test and the Runs test perform well in estimating TLS parameters. Therefore, we chose the statistics related to the Monobit test and the Runs test. Table 6.7 is a list of the features we used in our method.

H is 1 octet (8 bits) entropy.

$\bar{S_n}$ is a feature derived from the Monobit test. $\bar{S_n}$ is the average value of the difference S_n between the number of 0s and 1s divided by the octet length n of the data sequence, where the data sequence is evaluated as a bit sequence. When the number of 0s in the

data sequence is S_{n0} and the number of 1s is S_{n1}, it is calculated as the formula (6.1).

$$\bar{S_n} = \frac{S_{n0} - S_{n1}}{n} \tag{6.1}$$

$V_n(\bar{obs})$ is a feature derived from the Runs test. $V_n(\bar{obs})$ is the average of the number of runs divided by the octet length n of the data sequence when the data sequence is regarded as a bit sequence. It is expressed by the following formula (6.2).

$$V_n(\bar{obs}) = \frac{\{\sum_{k=1}^{n-1} r(k)\} + 1}{n} \tag{6.2}$$

$r(k)$ is expressed by the formula (6.3), where ϵ_i is the ith octet value of the data sequence.

$$r(k) = \begin{cases} 0, & \epsilon_k = \epsilon_{k+1} \\ 1, & \epsilon_k \neq \epsilon_{k+1} \end{cases} \tag{6.3}$$

In FakeTLS, the parameters exchanged in the TLS handshake are forged and cannot be trusted. Therefore, we only use the application data part of the TLS packet sent after the handshake. Specifically, each feature is calculated using the data sequence that combines the application data part of the TLS session as input.

Also, we set an upper limit on the length of data used for feature calculations to shorten processing time and detect the FakeTLS in the early phase of the communication. The longer the data, the closer the features are to true random numbers, so the aim is also to obtain effective features by separating it at lengths that do not increase its randomness. NIST SP800-22 (Revision 1a) recommends data size for the Monobit test and the Runs test to be 100 bytes or longer. The optimal upper limit is still an open question, but since NIST SP800-22 (Revision 1a) recommends that the data size for the Monobit and the Run tests be at least 100 bytes, we assume that around 100 bytes may be suitable.

6.4.2 Classifier

We cannot know in advance what encryption method an attacker will use to implement FakeTLS. Therefore, using FakeTLS data to train a detector is not practical. We therefore adopt the novelty detection method, which only trains on legitimate TLS communication data and detects unknown novelty values (FakeTLS).

In this research, we compared and evaluated the results using the following methods: One-Class SVM (or Single Class SVM) [101], a variant of Support Vector Machine (SVM), Isolation Forest [102], an ensemble method using decision trees, and Local Outlier Factor (LOF) [103], a method focusing on nearby local density in the feature space. In order to reduce the influence of the size of each feature, we normalized the features as preprocessing in One-Class SVM.

6.4.3 Comparison with Related Work

First, we clarify the positioning of our proposed method. Within machine learning-based identification approaches, our study verifies the effectiveness of randomness features in protocol-agnostic features, especially focusing on FakeTLS, a relatively new threat in the era of default encryption. To our knowledge, there are no publicly available datasets of malicious communication that include FakeTLS, and there are no studies that have set FakeTLS as their research scope. Research on communication identification using random features has been conducted for a long time. However, FakeTLS, which intentionally mimics benign TLS, is a different type of threat from before, and the effectiveness of randomness features in such an area is unverified.

In the problem domain of FakeTLS, it is also important to note that some of the features used in existing methods cannot be used. For example, some variants of malware FakeM [91] implement FakeTLS that omit the TLS handshake, which means methods that depend on the TLS handshake cannot be applied. Methods using TLS-specific features [63, 64, 65] mainly extract features from the TLS handshake. Therefore they are inherently inapplicable to the identification of FakeTLS. Our proposed method uses only the payload of TLS so that it can handle this problem.

Existing studies using protocol-agnostic features [61, 62] attempt to identify malware communication from communication characteristics such as packet arrival intervals and communication frequency. These methods are features related to malware specifications and are complementary to ours. It is conceivable that existing related methods can acquire detection capabilities for FakeTLS by combining the features we propose, but it remains an open question.

Raw-feature methods such as TLARNN [67] and DETD [68], which extract features using deep learning by incorporating payloads as-is, may be able to learn features leading

Table 6.8. Source data for experimental dataset for TLS Lie Detector

Commands	Data size[B]
ver	45
ipconfig	327
tasklist	13,344

to FakeTLS identification. On the other hand, raw-feature methods generally require larger data sizes for feature acquisition. Our proposed method can detect FakeTLS from smaller data pieces than that of raw-feature methods.

6.5 The Design of the Experiment

6.5.1 Datasets

There are several publicly available data sets that contain traffic encrypted by malware [104, 85, 9]. However, none of them cover malware families that use the FakeTLS technique.

To evaluate the effectiveness of our proposed method, we decided to create our original dataset. We generated regular TLS communication data as benign (normal) data and encrypted data mimicking actual FakeTLS as malicious (novelty) data. The dataset generation process has four steps: origin data creation, benign encrypted data generation, malicious encrypted data generation, and feature extraction. Figure 6.5 shows the overview of our datasets.

For reproducibility and further research development, we have made the dataset used in our experiment publicly available [105]. In addition to the dataset itself, it includes codes to build the experimental environment for recreating the dataset.

6.5.1.1 Origin Data Creation

As a source of our dataset, we used Windows commands' output on a real Windows OS environment (See Table 6.8). These are the commands that attackers often execute to gather information in the early stages of intrusion [106].

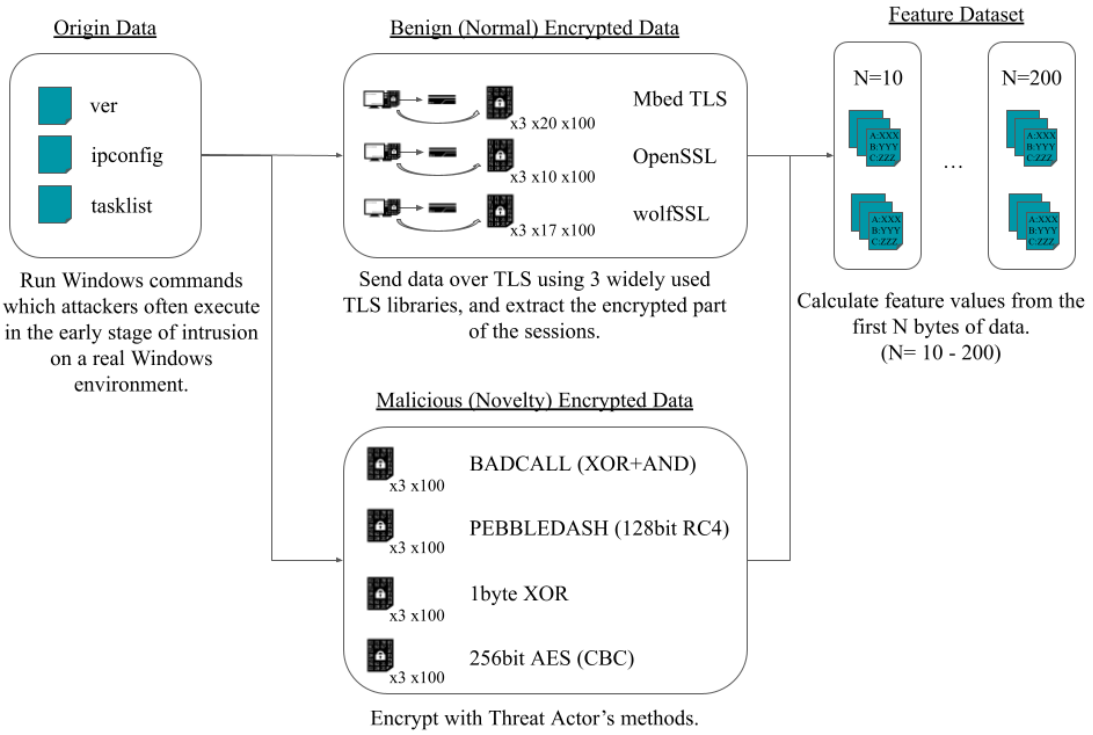


Figure 6.5. Experimental dataset generation for TLS Lie Detector

6.5.1.2 Benign Encrypted Data Generation

For benign data, we constructed a TLS server/client environment based on a container (Alpine Linux 3.18), generated TLS communication, and collected packets using tcpdump. The specifications of the experimental environment are shown in Table 6.9.

We used three TLS libraries: OpenSSL, Mbed TLS, and wolfSSL. These are all widely used TLS libraries and have been reported to be used by Lazarus. All were built with

Table 6.9. Benign data collection environment for TLS Lie Detector

TLS Library	Server version	Client version
OpenSSL	3.1.2	3.1.2
Mbed TLS	development (As of 8/4/2023)	3.4.1
wolfSSL	5.6.0	5.6.0

settings that enabled TLS 1.2 and TLS 1.3, and we used the cipher suites available with standard settings. Only the Mbed TLS server was built using the development branch on the GitHub repository because TLS 1.3 was not available in the released version at the time of our experiment.

Next, we generated TLS communications in the server-client environment where the server and the client use the same TLS library. By including only one candidate in ClientHello's Cipher Suites list each time, we controlled the cipher suites used in each session. While changing cipher suites, we generated TLS communications with all available patterns with standard configuration, which are 10-20 cipher suites for each TLS library. For each experimental setting, we sent and received three types of origin data 100 times each and collected the PCAP data. In total, we collected 14,100 sessions. Lastly, we extracted only the encrypted TLS Application Data from each session.

6.5.1.3 Malicious Encrypted Data Generation

For malicious data, we created Python programs that encrypt data with four algorithms often used by threat actors.

Two of the algorithms are exactly the same algorithms as Lazarus's methods. One is replicating malware BADCALL [11] which uses XOR/AND operations, and the other is replicating malware PEBBLEDASH [13] which uses 128-bit key RC4 for encryption.

The other two algorithms are selected from the encryption methods frequently listed in MITRE ATT&CK [107]. We chose XOR (1-byte XOR) and AES (AES-256-CBC).

We encrypted the three types of origin data with four encryption methods 100 times each. For each encryption, we randomly generated an encryption key. Since we only need the encrypted data part for this experiment, we directly encrypted the origin data using our encryption program without actually generating the TLS communication. In total, we collected encrypted data equivalent to 1200 sessions.

We acknowledge that the volume of malicious data is extremely small compared to that of benign data. However, in a real enterprise environment, malicious traffic is usually much less common than benign traffic. Therefore, it is feasible to test under such an imbalanced situation.

6.5.1.4 Feature Extraction

From the benign and malicious encrypted data, we calculated the randomness features we discussed in Section 6.4.1.

The cipher suites included in the benign dataset and the statistics of extracted features are summarized in Tables 6.10 to 6.12. Also, statistics of the malicious dataset are summarized in Table 6.13.

Table 6.10. The benign (normal) dataset in TLS Lie Detector experiment (MBed TLS)

Cipher suite (Removed leading "TLS_" from IANA description name)	Entropy			$\bar{S}_n$			$V_n(\bar{obs})$		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
AES_128_CCM_8_SHA256	6.44	7.37	7.99	-0.56	0.01	0.63	3.64	3.99	4.33
AES_128_CCM_SHA256	6.63	7.42	7.99	-0.57	-0.00	0.60	3.76	4.01	4.40
AES_128_GCM_SHA256	6.60	7.43	7.99	-0.64	-0.01	0.71	3.73	4.00	4.21
AES_256_GCM_SHA384	6.69	7.46	7.99	-0.72	-0.00	0.67	3.68	4.00	4.23
CHACHA20_POLY1305_SHA256	6.59	7.43	7.99	-0.46	-0.01	0.66	3.67	4.00	4.24
ECDHE_ECDSA_WITH_AES_128_CBC_SHA	5.90	7.16	7.99	-0.76	-0.02	0.79	3.63	4.00	4.37
ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	6.06	7.22	7.99	-0.62	0.00	0.67	3.67	4.00	4.40
ECDHE_ECDSA_WITH_AES_128_CCM	5.45	6.98	7.99	-1.45	-0.33	0.13	3.23	3.82	4.09
ECDHE_ECDSA_WITH_AES_128_CCM_8	5.13	6.92	7.99	-1.77	-0.39	0.17	3.18	3.81	4.11
ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	5.40	6.99	7.99	-1.83	-0.35	0.24	3.26	3.83	4.10
ECDHE_ECDSA_WITH_AES_256_CBC_SHA	5.90	7.17	7.99	-0.81	0.00	1.12	3.60	4.00	4.37
ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	6.22	7.29	7.99	-0.57	-0.01	0.70	3.64	4.00	4.29
ECDHE_ECDSA_WITH_AES_256_CCM	5.41	6.99	7.99	-1.77	-0.36	0.23	3.10	3.82	4.13
ECDHE_ECDSA_WITH_AES_256_CCM_8	5.24	6.92	7.99	-1.84	-0.40	0.17	3.11	3.80	4.12
ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	5.33	6.99	7.99	-1.45	-0.34	0.19	3.12	3.83	4.12
ECDHE_ECDSA_WITH_CAMELLIA_128_CBC_SHA256	6.02	7.22	7.99	-0.71	-0.00	0.54	3.67	4.00	4.36
ECDHE_ECDSA_WITH_CAMELLIA_128_GCM_SHA256	5.44	6.99	7.99	-1.68	-0.35	0.22	3.26	3.83	4.06
ECDHE_ECDSA_WITH_CAMELLIA_256_CBC_SHA384	6.17	7.29	7.99	-0.54	-0.00	0.48	3.44	4.00	4.43
ECDHE_ECDSA_WITH_CAMELLIA_256_GCM_SHA384	5.41	6.98	7.99	-1.62	-0.36	0.42	3.19	3.83	4.08
ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	5.41	7.02	7.99	-0.85	-0.01	0.85	3.69	4.01	4.43

Table 6.11. The benign (normal) dataset in TLS Lie Detector experiment (OpenSSL)

Cipher suite	Entropy			$\bar{S_n}$			$V_n(\bar{obs})$		
(Removed leading "TLS_" from IANA description name)	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
AES_128_GCM_SHA256	6.57	7.40	7.99	-0.55	0.01	0.71	3.73	3.99	4.31
AES_256_GCM_SHA384	6.54	7.40	7.99	-0.55	-0.01	0.43	3.63	4.01	4.36
CHACHA20_POLY1305_SHA256	6.53	7.40	7.99	-0.68	0.01	0.56	3.71	4.01	4.25
ECDHE_ECDSA_WITH_AES_128_CBC_SHA	6.54	7.40	7.99	-0.63	-0.01	0.53	3.76	4.01	4.39
ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	6.61	7.40	7.99	-0.56	-0.00	0.69	3.67	4.00	4.33
ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	6.57	7.40	7.99	-0.57	0.00	0.60	3.78	4.00	4.38
ECDHE_ECDSA_WITH_AES_256_CBC_SHA	6.52	7.40	7.99	-0.56	-0.00	0.57	3.76	4.00	4.35
ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	6.57	7.40	7.99	-0.51	0.01	0.61	3.72	4.00	4.28
ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	6.56	7.40	7.99	-0.60	0.01	0.63	3.77	4.01	4.31
ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	6.52	7.40	7.99	-0.59	-0.01	0.87	3.72	4.00	4.37

Table 6.12. The benign (normal) dataset in TLS Lie Detector experiment (wolfSSL)

Cipher suite (Removed leading "TLS_" from IANA description name)	Entropy			$\bar{S}_n$			$V_n(\bar{o}bs)$		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
AES_128_CCM_8_SHA256	6.07	7.23	7.99	-1.01	-0.00	0.75	3.61	4.01	4.44
AES_128_CCM_SHA256	6.28	7.30	7.99	-0.66	-0.00	0.92	3.59	4.00	4.29
AES_128_GCM_SHA256	6.26	7.30	7.99	-0.64	0.01	0.92	3.63	4.00	4.26
AES_256_GCM_SHA384	6.44	7.35	7.99	-0.72	-0.00	0.90	3.64	4.00	4.31
CHACHA20_POLY1305_SHA256	6.20	7.30	7.99	-0.68	0.00	0.68	3.63	4.00	4.30
ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	5.76	7.12	7.99	-0.87	-0.01	0.74	3.53	4.01	4.50
ECDHE_ECDSA_WITH_AES_128_CBC_SHA	5.95	7.17	7.99	-0.76	0.02	1.02	3.71	4.01	4.42
ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	6.04	7.23	7.99	-0.79	-0.01	0.58	3.56	4.00	4.41
ECDHE_ECDSA_WITH_AES_128_CCM	5.42	6.98	7.99	-1.74	-0.36	0.29	3.14	3.83	4.06
ECDHE_ECDSA_WITH_AES_128_CCM_8	5.18	6.92	7.99	-2.07	-0.41	0.24	2.98	3.80	4.09
ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	5.71	7.08	7.99	-0.87	-0.03	0.81	3.58	4.01	4.43
ECDHE_ECDSA_WITH_AES_256_CBC_SHA	5.91	7.17	7.99	-0.74	-0.01	0.62	3.61	4.00	4.35
ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	6.24	7.29	7.99	-0.59	-0.00	0.73	3.75	4.01	4.38
ECDHE_ECDSA_WITH_AES_256_CCM_8	5.17	6.92	7.99	-1.84	-0.40	0.24	2.89	3.81	4.10
ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	5.59	7.07	7.99	-0.93	0.01	0.81	3.57	4.00	4.45
ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	5.49	7.01	7.99	-0.82	0.02	0.95	3.49	4.00	4.39
ECDHE_ECDSA_WITH_RC4_128_SHA	5.56	7.05	7.99	-0.92	-0.00	0.68	3.57	4.00	4.49

Table 6.13. The malicious (novelty) dataset in TLS Lie Detector experiment

	Entropy			$\bar{S_n}$			$V_n(\bar{obs})$		
Encryption scheme	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
XOR/AND	3.12	4.06	4.56	-3.91	-0.01	4.41	1.68	3.95	6.47
1-byte XOR	3.12	4.06	4.56	-4.36	0.09	4.28	2.84	3.97	6.03
RC4(128bit)	5.12	6.88	7.99	-1.07	0.00	1.20	3.49	4.00	4.56
AES(256bit, CBC)	5.15	6.92	7.99	-1.25	-0.00	1.08	3.54	4.02	4.56

To consider the impact of data length on performance, we set the maximum data length for feature extraction. One of the aspects we considered in our experiments was a trade-off between processing time and data length. Generally, the longer the data, the more features can be expected to be obtained, but it takes more time to extract the features. Also, in TCP/IP, data can be divided into multiple packets, so if the data length is long, there will also be time to wait for subsequent packets. Although we assumed the use case in the threat-hunting process where real-time detection performance is not that critical, the detection process needs to be finished in a reasonable time.

In related studies of feature extraction from raw packet data [66, 67, 68], source data is extracted up to several hundred to several thousand bytes. Lopez-Martin *et al.* verified the trade-off between the data amount used for traffic identification and its performance and concluded that 5 to 15 packets are sufficient [108].

As our research focuses on the randomness of encrypted payloads, we believe there also is a trade-off between the features' distribution and data length. That is, the longer the data, the closer the data gets to true randomness, reducing its characteristic variability. In the NIST SP800-22 test that we select in our proposed method, 100 bytes is specified as the recommended minimum data length, and we believe that the neighborhood of this value is expected to provide the most feature variation.

From these two perspectives, we adjusted the maximum data length in increments of 10 bytes, ranging from 10 to 200 bytes, and extracted the features.

6.5.2 Evaluation Methodology

We evaluated the TLS Lie Detector using a variant of five-fold cross-validation. First, we divided the benign dataset into five groups. We performed stratified partitioning so

that each divided group contains equal proportions of TLS libraries, cipher suites, and command results. Then we selected four of the groups and trained the TLS Lie Detector in the training phase. In the testing phase, we combined the last group of benign data with malicious data and tested the performance. The image of the experimental dataset when the benign data is divided into five groups, A, B, C, D and E, is as shown in Table 6.14.

We adopted the general metrics of binary classification shown in Table 6.15 as the evaluation metrics.

Where TP is the number of correctly determined FakeTLS (detected), FP is the number of falsely determined legitimate TLS as FakeTLS (over-detected), TN is the number of correctly determined legitimate TLS, and FN is the number of falsely determined FakeTLS as legitimate TLS (missed).

In the experiment, we conducted a total of five verifications by switching the data used for testing among the five divided data. Then, for each value, we calculated the average value of the five verifications. Note that this validation data has a large bias in the ratio of benign to novelty data. Therefore, we only use the accuracy rate as a reference and measure the performance of the methods by the F scores (F0.5, F1, F2) and the Matthews Correlation Coefficient (MCC), which can be used even for imbalanced datasets. The F scores are indices that measure the balance between "minimization of false positives (precision)" and "minimization of false negatives (recall)". It takes a value between 0 and 1, with higher values indicating better performance. In threat hunting, the tolerance for over-detections and misses varies depending on the analysis resources and the organization's risk appetite, and it is not always appropriate to view recall and precision with the same weight. So we decided to calculate 3 types of F scores, F0.5 score (high weight on precision), F1 score (even weight), F2 score (high weight on recall). The MCC is an index that measures the correlation between predicted values and correct values in binary classification problems. It takes values from -1 to 1, with higher values indicating better performance.

6.6 Experimental results

The results of the experiment are shown in Table 6.16, Figure 6.6 and Figure 6.7.

Table 6.14. The combination of the datasets in TLS Lie Detector experiment

		Experiment				
Dataset		1	2	3	4	5
Benign	A	•	•	•	•	○
	B	•	•	•	○	•
	C	•	•	○	•	•
	D	•	○	•	•	•
	E	○	•	•	•	•
Novelty		○	○	○	○	○

•:Training, ○:Testing

Table 6.15. The evaluation metrics in TLS Lie Detector experiment

Metrics	Formula
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F0.5 Score	$1.25 \cdot \frac{Precision*Recall}{0.25Precision+Recall}$
F1 Score	$2 \cdot \frac{Precision*Recall}{Precision+Recall}$
F2 Score	$5 \cdot \frac{Precision*Recall}{4Precision+Recall}$
MCC	$\frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

Table 6.16. The results of the TLS Lie Detector experiments (Max Byte Length from 60 to 100)

Detector	Max Byte Length	TP	FN	FP	TN	Accuracy	Precision	Recall	F0.5	F1	F2	MCC
One-Class SVM	60	956.40	243.60	1410.40	1409.60	0.59	0.40	0.80	0.45	0.54	0.67	0.28
	70	958.40	241.60	1413.40	1406.60	0.59	0.40	**0.80**	0.45	0.54	0.67	0.28
	80	950.00	250.00	1412.40	1407.60	0.59	0.40	0.79	0.45	0.53	0.66	0.27
	90	945.20	254.80	1412.80	1407.20	0.59	0.40	0.79	0.44	0.53	0.66	0.27
	100	939.40	260.60	1409.60	1410.40	0.58	0.40	0.78	0.44	0.53	0.66	0.26
Isolation Forest	60	804.40	395.60	271.60	2548.40	0.83	0.75	0.67	0.73	0.71	0.68	0.59
	70	825.20	374.80	237.40	2582.60	0.85	0.78	0.69	0.76	0.73	**0.70**	0.63
	80	816.20	383.80	264.00	2556.00	0.84	0.76	0.68	0.74	0.72	0.69	0.61
	90	819.60	380.40	273.00	2547.00	0.84	0.75	0.68	0.74	0.72	0.70	0.60
	100	811.80	388.20	281.20	2538.80	0.83	0.74	0.68	0.73	0.71	0.69	0.59
LOF	60	731.20	468.80	22.60	2797.40	0.88	0.97	0.61	0.87	0.75	0.66	0.70
	70	790.00	410.00	27.00	2793.00	**0.89**	0.97	0.66	**0.88**	**0.78**	0.70	**0.74**
	80	764.80	435.20	21.40	2798.60	0.89	0.97	0.64	0.88	0.77	0.68	0.73
	90	751.40	448.60	18.80	2801.20	0.88	**0.98**	0.63	0.88	0.76	0.67	0.72
	100	752.20	447.80	25.20	2794.80	0.88	0.97	0.63	0.87	0.76	0.67	0.72

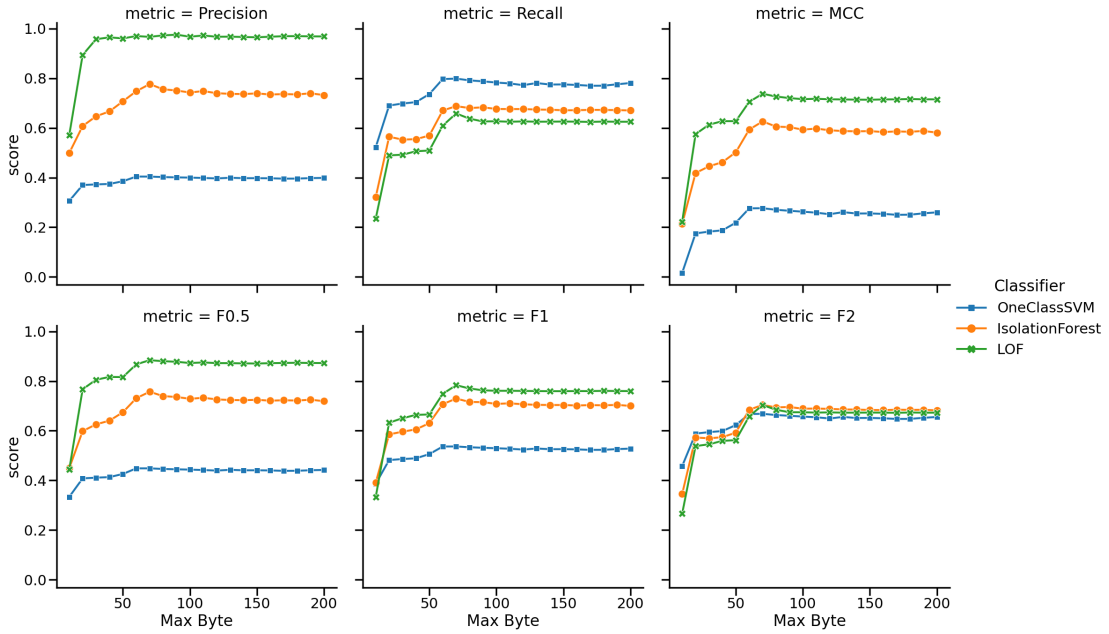


Figure 6.6. Evaluation metrics transition by maximum byte length in TLS Lie Detector experiment

Comparing the performance of the detectors, the F0.5 score, the F1 score, and the MCC were higher in the order of LOF, Isolation Forest, and One-Class SVM. Regarding the F2 score, the best-performing detector differs depending on the maximum byte length. However, at the best score, LOF and Isolation Forest had almost equivalent performance. Upon closer inspection, One-Class SVM judged more data as FakeTLS than the other two methods, regardless of whether they were correct or incorrect. One-Class SVM had the most TPs among the three methods, but also the most FPs. Although the recall was the highest among the three methods due to the large number of TPs, the precision was the lowest due to the extremely large number of FPs. As a result, the F-scores were the lowest among the three methods. LOF had the highest number of missed detections (FN) among the three methods, but the lowest number of false detections (FP). Hence, the F-scores and MCC showed high scores due to the high precision. Isolation Forest scores were intermediate between One-Class SVM and LOF on most metrics.

Comparing the performance according to the maximum byte length, all the 3 F-scores and the MCC for all detectors became the highest when the length was 70 bytes. When the maximum length was less than 70 bytes, there was a tendency to have fewer TPs and

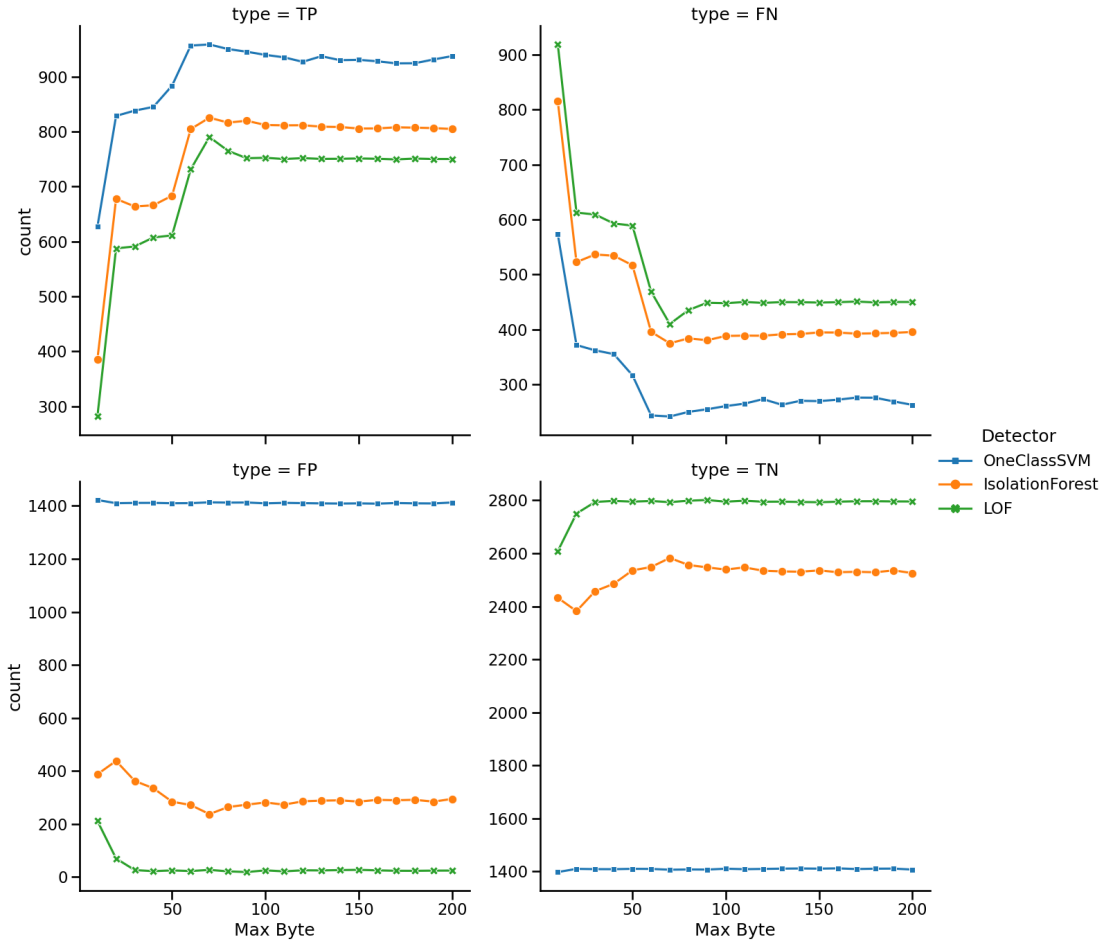


Figure 6.7. Confusion Matrix value transition by maximum byte length in TLS Lie Detector experiment

more FNs. We can see that the detectors misjudged legitimate TLS as FakeTLS when the data length was not so long enough. In the area of over 100 bytes, the metrics are almost flat or slightly degrading.

Figure 6.8 and Figure 6.9 show the confusion matrices of LOF and Isolation Forest that showed the highest performance (with a maximum byte lengths of 70 bytes).

6.7 Consideration

In terms of the maximum byte length, the proposed method is most effective around 70 bytes, where the randomness of the encrypted data is seen to be more likely to differ.

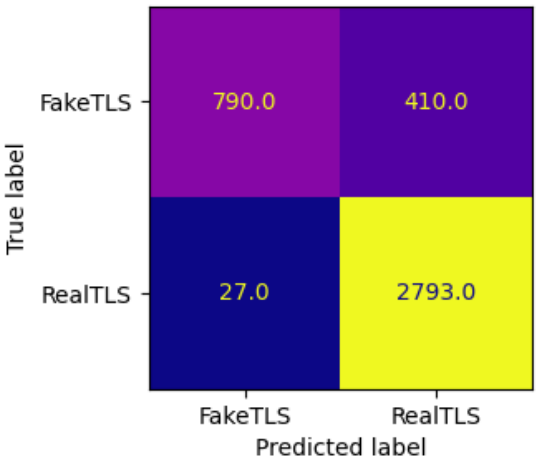


Figure 6.8. Confusion Matrix of TLS Lie Detector experimental result (LOF, Max 70 byte)

For data lengths shorter than 70 bytes, false positives become particularly noticeable. On the other hand, in areas of 100 bytes or more, the metrics are almost flat or slightly degrading, suggesting that obtaining more data may not contribute much to detection performance. Although further analysis is needed, it suggests the existence of a difference in the way randomness increases between normal TLS and FakeTLS. Furthermore, it is conceivable that by dividing into short blocks of data and making multiple judgments, it may be possible to further emphasize the difference.

When we focus on the differences among encryption algorithms, the results differ greatly depending on the algorithm strength. The precision and recall scores for each algorithm used in FakeTLS are described in Figure 6.10. For the simple algorithms using XOR or XOR/AND operation, all three detectors (with a maximum length of 50 bytes or more) achieve precision and recall with a score of 1.0, which means our proposed method can completely detect FakeTLS. In contrast, detecting FakeTLS using strong algorithms such as RC4 or AES remains a challenge that needs to be solved. While Isolation Forest and LOF detectors can accurately detect normal TLS, they often miss FakeTLS with strong algorithms. However, as exemplified by the malware Toneshell [109], XOR-based FakeTLS remains a real-world attack tactic even in 2025. Therefore, the current performance of the TLS lie detector is considered sufficient for practical use for the time being.

When looking at the differences according to the classification methods, One-Class SVM

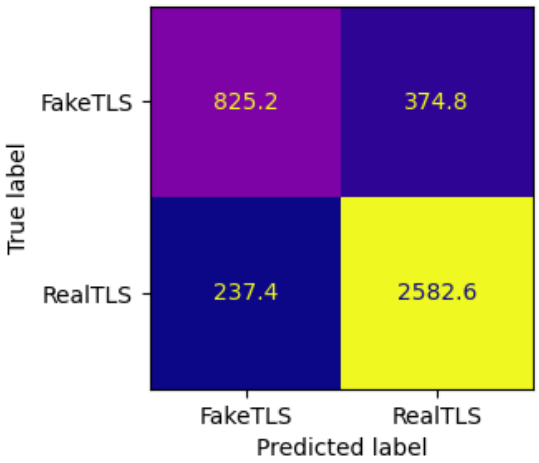


Figure 6.9. Confusion Matrix of TLS Lie Detector experimental result (Isolation Forest, Max 70 byte)

shows characteristics that differ from the other two methods. While the LOF detector has the highest detection performance for the overall experiment, it is inferior to the One-Class SVM and the Isolation Forest detectors when limited to FakeTLS using strong encryption algorithms. For FakeTLS using strong encryption algorithms, the Isolation Forest detector achieves the highest F0.5 score, and the One-Class SVM detector achieves the highest F1 and F2 scores. This suggests that the areas of expertise vary by detector, indicating the potential for performance improvement by combining the judgments of multiple classification methods.

The F-scores of the best-performing detector (LOF with a maximum length of 70 bytes) are higher in the order of F0.5, F1, and F2 scores, which suggests that our proposed method is more suitable for situations where precision is required more than recall. In other words, our proposed method is more effective in situations where an organization has multi-perspective analysis capabilities, meaning there is a relative tolerance for false negatives from individual perspectives, but limited human resources lead to a low tolerance for wasted analysis efforts caused by false positives.

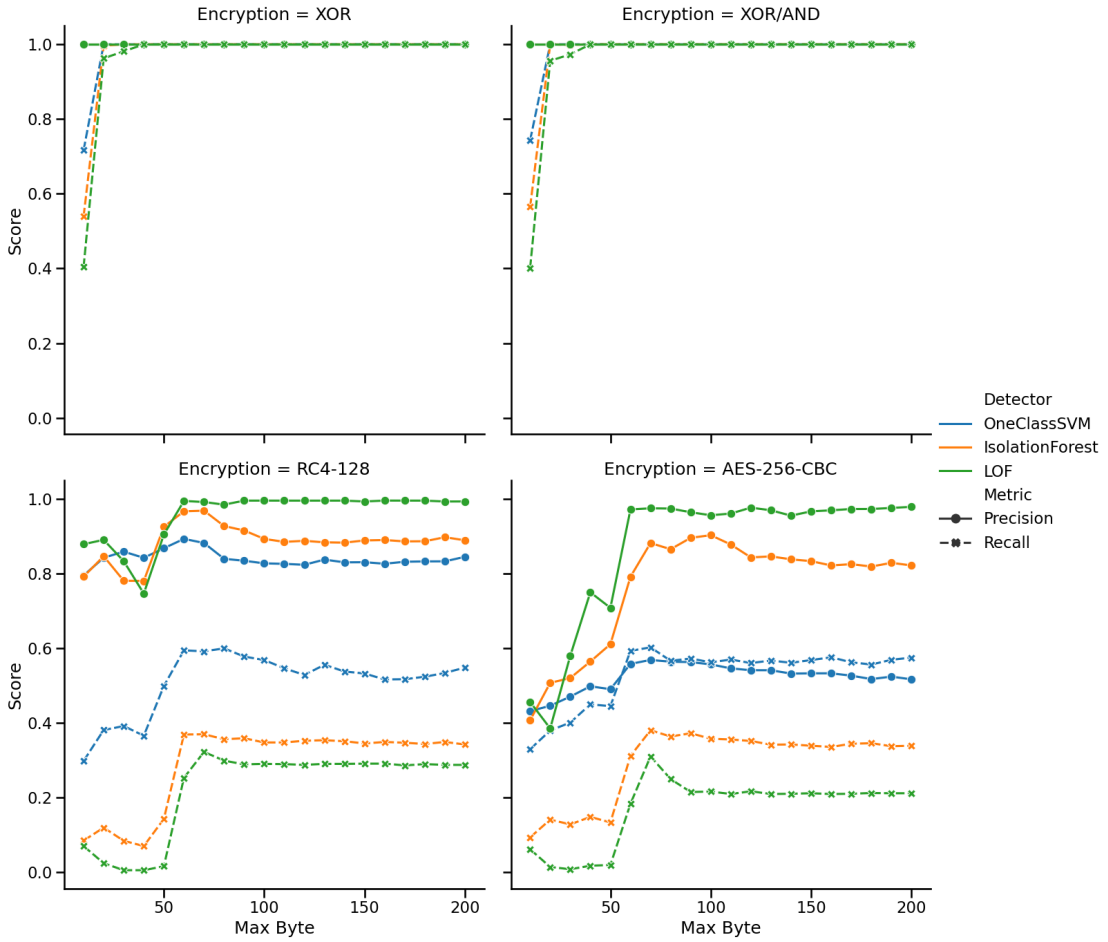


Figure 6.10. Precision and Recall by encryption algorithm (XOR, XOR/AND, RC4-128, AES-256-CBC)

6.8 Conclusion

In this chapter, we propose a TLS Lie Detector that identifies FakeTLS communications using only encrypted data. Our research question is: **"Is it possible to detect spoofed TLS communications from encrypted exchanges over the communication path, under conditions where a TLS handshake does not exist, or even if it does exist, its contents cannot be trusted?"**

At the best score in the overall experiment, the detector performs at an F0.5 score of 0.88, an F1 score of 0.78, an F2 score of 0.70, and a Matthews correlation coefficient

of 0.74. In particular, our proposed method can completely detect FakeTLS using weak encryption algorithms with low randomness.

End-to-end communication encryption is expected to become more widespread in the future as security threats grow and privacy awareness increases. Not surprisingly, threat actors will adapt to this trend and change their TTPs (Tactics, Techniques, Procedures) to achieve their goals by blending their activities into benign encrypted communications. In such cases, this proposed method can serve as a new logic for detecting threats along the communication path. Although our proposed method is considered sufficiently effective against current threats, detecting FakeTLS using strong encryption algorithms remains a challenge to prepare for future threat evolution. We believe there is significant room for improvement, such as adding more features and combining multiple classification methods to create an ensemble classifier.

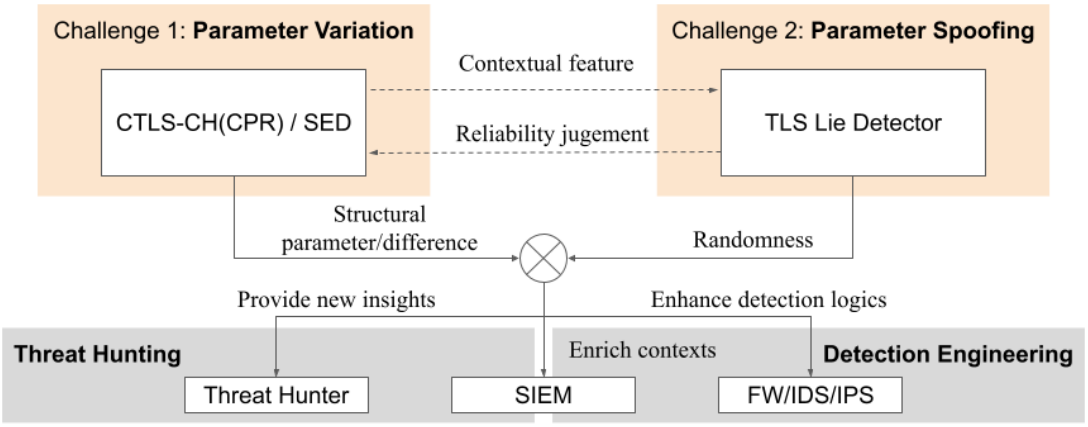


Figure 7.1. Overview of the research in this thesis

Chapter 7
Overall Discussion

7.1 Relationship between Parameter Variation and Parameter Spoofing

This thesis addresses two complementary yet distinct challenges in identifying encrypted C2 communications used by malware.

Chapter 5 focuses on the problem of parameter variation in TLS handshake metadata. It demonstrates that both benign and malicious software exhibit frequent, sometimes unpredictable changes in handshake parameters—such as cipher suites, extensions, and protocol versions—due to factors like session context, software updates, and the operational environment. This natural variability undermines the reliability of static, hash-based fingerprinting methods. To overcome this, Chapter 5 introduces structure-aware techniques (CPR and SED) and adaptive matching strategies (e.g., EM-D), enabling robust identification even when parameters fluctuate or drift over time.

Chapter 6, on the other hand, addresses parameter spoofing, in which malware deliberately manipulates or fabricates TLS handshake parameters to evade detection. In extreme cases, such as FakeTLS, the handshake may mimic legitimate TLS while the actual pay-

load does not conform to protocol standards. This renders handshake-based identification ineffective. Chapter 6 proposes a fundamentally different approach: analyzing the statistical randomness of encrypted payloads using entropy and randomness tests (Monobit, Runs, etc.), allowing detection of spoofed communications even when handshake metadata is unreliable or absent.

The relationship between the two approaches is synergistic (as described in Figure 7.1). CPR and SED provide resilience against natural variability in handshake parameters, ensuring that legitimate changes do not lead to missed detections. TLS Lie Detector extends detection capabilities to adversarial scenarios, where attackers intentionally manipulate or bypass handshake features.

Together, this thesis forms a comprehensive framework for encrypted C2 identification. Structural fingerprinting in Chapter 5 is effective when handshake metadata is available and not maliciously altered. Payload randomness analysis in Chapter 6 is essential when handshake metadata cannot be trusted or is missing.

By integrating both approaches, this thesis addresses the full spectrum of challenges posed by encrypted malware communications in modern networks, enabling robust identification against parameter variation without relying on decryption.

7.2 Applicable Use Cases

The proposed methods can be integrated into modern Security Operations Center (SOC) workflows as additional detection signals. Structural fingerprints (CPR/SED) and payload-based novelty scores can be used to enrich SIEM/IDS alerts, support threat hunting activities, and enhance detection engineering.

In the field of detection engineering, eliminating the fluctuating parameters identified in this research can suppress variations caused by TLS specifications, extend the lifespan of IoCs, and generate more persistent intelligence. Furthermore, while the feasibility from a processing performance perspective requires future verification, employing a Nearest Neighbor (NN) strategy for detection is expected to reliably filter only communications closely resembling known threats while suppressing false positive risks, thereby improving blocking performance. Furthermore, CPR's reversible transformation feature can be leveraged to identify and improve flaws in detection logic at the detailed parameter level. This

feature is highly compatible with Detection Engineering and greatly supports a continuous cycle of detection performance improvement.

In the field of Threat Hunting, leveraging the distance concept from SED enables the extraction and prioritized investigation of encrypted communications that closely resemble known threats, thereby effectively capturing unknown threats. Furthermore, the FakeTLS probability information provided by the TLS Lie Detector can detect anomalies in communications that appear to be normal handshakes as potential signs of sophisticated concealment efforts, aiding analysts in determining investigation priorities.

Even during the log collection phase, which forms the foundation for both detection engineering and threat hunting, estimated labels for unknown threats based on SED proximity and FakeTLS probability scores can be used to add new context to logs. By correlating these signals with other contextual information (such as destination reputation, behavioral analytics, and host telemetry), organizations can improve detection accuracy and reduce alert fatigue.

The methods proposed in this research enable threat detection solely based on superficial parameter structures and statistical characteristics, without decrypting communications, making them effective for achieving compliance with security monitoring requirements. Due to privacy regulations in various countries, including the EU's General Data Protection Regulation (GDPR), and strict financial-sector policies, communication content should not be inspected without proper justification.

Furthermore, due to service specifications and other factors, use cases where TLS Inspection, which breaks end-to-end encryption and terminates communications at an intermediate point, cannot be applied are expected to persist. Our multi-perspective approach is especially valuable in environments where encrypted traffic dominates, and traditional content inspection is no longer feasible.

7.3 Limits and Open Issues

While the proposed methodologies, structural fingerprinting (CPR/SED, EM-D) and payload randomness-based detection (TLS Lie Detector), demonstrate significant promise in controlled experiments, their practical deployment and generalizability are subject to several important limitations and operational considerations.

This section critically examines the scope and boundaries of the evaluation, the challenges of real-world implementation, and the engineering trade-offs that must be addressed for effective adoption.

7.3.1 Dataset and External Validity

A key strength of this research lies in the use of a longitudinal, real-world dataset (the MTA dataset) spanning over a decade of malware traffic.

However, several limitations remain:

Browser-Centric Fluctuation Dataset

The controlled dataset for parameter fluctuation analysis was primarily constructed by using major web browsers (Chrome, Edge, Firefox) to access popular domains. While this approach enabled systematic induction and measurement of parameter variation, it may not fully capture the diversity of TLS implementations and behaviors found in enterprise, IoT, or malware environments. The long tail of client software and device types remains underrepresented.

Malware Family Coverage

The analysis focused on a subset of malware families (e.g., Trickbot, IcedID, Cobalt Strike) with sufficient longitudinal data. Many emerging or less prevalent malware strains, as well as rapidly evolving C2 infrastructures, may exhibit different patterns of parameter variation or spoofing not captured in the current study.

Labeling and Ground Truth

Session-level labeling in mixed PCAP files (especially in sandbox environments) is inherently challenging. The presence of both benign and malicious flows in the same capture can complicate the assignment of ground truth, potentially introducing noise into evaluation metrics. The lack of publicly available, well-labeled FakeTLS datasets further constrains the scope of validation for spoofing detection.

7.3.2 Sustaining Performance Over Time

The dynamic nature of TLS ecosystems, driven by protocol evolution, software updates, and adversarial adaptation, necessitates continuous maintenance of detection logic:

Fingerprint Drift and Model Aging

As observed in the parameter drift analysis, fingerprint databases and trained models can become obsolete as new TLS versions, extensions, or client behaviors emerge. Scheduled retraining, periodic database refreshes, and automated pipelines for updating de-fluctuation parameter lists are essential to maintain detection efficacy.

Emergence of New Variants

The introduction of features such as Encrypted Client Hello (ECH), ALPS, or delegated credentials in TLS 1.3 and beyond may introduce new sources of parameter variation or obfuscation. Ongoing monitoring and adaptation of structural representations and feature extraction logic are required to keep pace with protocol developments.

7.3.3 Engineering and Operational Considerations

For practical deployment in high-throughput or real-time environments, several engineering challenges must be addressed:

Computational Overhead

Structural fingerprinting (CPR/SED) and payload randomness tests (entropy, Monobit, Runs) introduce additional processing requirements compared to traditional hash-based fingerprinting. Efficient implementation (such as incremental parsing, caching, and parallelization) can mitigate performance impacts, but resource constraints (CPU, memory) must be carefully managed, especially in large-scale or latency-sensitive deployments.

Threshold Tuning and False Positive Management

The selection of SED thresholds, de-fluctuation parameter sets, and novelty detection sensitivity directly affects the balance between false positives and false negatives. Data-driven, environment-specific calibration is necessary, and thresholds should be periodically

re-evaluated as the ecosystem evolves.

Integration with Existing Security Workflows

The outputs of the proposed methods are most effective when combined with other contextual signals (e.g., destination reputation, behavioral analytics, host telemetry) within SIEM or IDS frameworks. Multi-context correlation can help mitigate alert fatigue and improve the actionable value of detections.

Chapter 8
Conclusion

In this thesis, we address the critical challenge of identifying encrypted C2 communications used by malware in the era of default-encrypted Internet traffic. The widespread adoption of encryption, while enhancing privacy and security for legitimate users, has also enabled threat actors to conceal malicious activities. This duality has rendered traditional security solutions, especially those that rely on payload inspection, less effective, necessitating the development of novel identification technologies that do not rely on decryption.

This thesis makes the following key contributions:

Clarification of Long-term Parameter Variation in Real-world Malware C2

By analyzing a large-scale, longitudinal dataset spanning over 11 years (the MTA dataset), we find that both benign and malicious software exhibit significant variability in TLS handshake parameters. We introduce the concepts of "Parameter Fluctuation" (short-term, session-level changes) and "Parameter Drift" (long-term evolution), demonstrating that these phenomena undermine the reliability of static, hash-based fingerprinting methods. Our findings highlight the need for more robust, structure-aware identification techniques.

Development of Robust Structural Fingerprinting Techniques

To address the limitations of existing fingerprinting approaches, we propose the Compacted Protocol Representation (CPR) and Structural Edit Distance (SED) as new frameworks for representing and quantifying differences in TLS handshake parameters. We further introduce the "Exact Match after De-fluctuation (EM-D)" strategy, which removes known fluctuating parameters prior to matching, significantly improving identification performance in environments with many unknown fingerprints. Our reproducible dataset-generation methodology enables systematic evaluation of robustness against parameter variation.

Establishment of Payload-Randomness-based Detection for Parameter Spoofing

Recognizing the emergence of advanced evasion techniques, such as FakeTLS, which forge or omit handshake metadata, we develop the TLS Lie Detector. This novelty detection model leverages statistical randomness tests (entropy, Monobit, Runs) on encrypted payloads to identify spoofed communications, even in the absence of a trustworthy handshake. Our experiments demonstrate that this approach is highly effective against weak encryption algorithms and provides a new detection logic for threat hunting in encrypted environments.

While our methodologies demonstrate significant promise in controlled experiments, several challenges remain for practical deployment. The diversity of TLS implementations in real-world environments, the emergence of new protocol features (e.g., Encrypted Client Hello), and the need for continuous model and fingerprint updates require ongoing research and engineering effort. Additionally, detecting FakeTLS with strong encryption algorithms remains an open problem, underscoring the need for further feature engineering and ensemble methods.

This thesis advances the state of the art in the identification of encrypted C2 communication by providing both theoretical insights and practical tools for robust detection. As encrypted traffic continues to dominate network communications, the approaches developed herein offer a foundation for future research and operational defense against evolving malware threats. Continued collaboration between academia and industry, along with the development of richer, well-labeled datasets, will be essential to sustain and extend these advances.

Bibliography

[1]Google. “HTTPS encryption on the web”. URL: https://transparencyreport.google.com/https/.

[2]Let’s Encrypt. “Let’s Encrypt Stats”. URL: https://letsencrypt.org/stats/.

[3]Sean Gallagher. “Nearly half of malware now use TLS to conceal communications”. Sophos Ltd. URL: https://www.sophos.com/en-us/blog/nearly-half-of-malware-now-use-tls-to-conceal-communications.

[4]Unit 42. “2023 Unit 42 Network Threat Trends Research Report”. Tech. rep. Palo Alto Networks, 2023.

[5]ThreatLabz. “ThreatLabz 2024 Encrypted Attacks Report”. Tech. rep. Zscaler, Inc., 2024.

[6]John Althouse, Jeff Atkinson, and Josh Atkins. “JA3 - A method for profiling SSL/TLS Clients”. Salesforce, Inc. URL: https://github.com/salesforce/ja3.

[7]John Althouse. “JA4+ network fingerprinting”. FoxIO. URL: https://github.com/FoxIO-LLC/ja4.

[8]The MITRE Corporation. “Data Obfuscation: Protocol or Service Impersonation”. The MITRE Corporation. URL: https://attack.mitre.org/versions/v18/techniques/T1001/003/.

[9]Brad Duncan. “Malware Traffic Analysis”. URL: https://www.malware-traffic-analysis.net/.

[10]Andrew Rukhin et al. “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”. Tech. rep. SP 800-22 Revision 1a. National Institute of Standards and Technology, 2010. URL: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final.

[11]Department of Homeland Security. “North Korean Trojan: BADCALL”. Malware Analysis Report MAR-10135536-10. Department of Homeland Security, 2019.

[12]Department of Homeland Security. “North Korean Trojan: TAINTEDSCRIBE”. Malware Analysis Report MAR-10288834-2.v1. Department of Homeland Security, 2020.

[13]Department of Homeland Security. “North Korean Trojan: PEBBLEDASH”. Malware Analysis Report MAR-10288834-3.v1. Department of Homeland Security, 2020.

[14]Cisco. “Cisco 2018 Annual Cybersecurity Report”. 2018.

[15]WatchGuard Threat Lab. “Internet Security Report - Q2 2025”. 2025.

[16]National Security Agency. “Managing Risk from Transport Layer Security Inspection”. 2019.

[17]Robert M. Lee. “The Sliding Scale of Cyber Security”. SANS Institute, 2015.

[18]Sqrrl. “A Framework for Cyber Threat Hunting”. 2016.

[19]Alan O. Freier, Philip Karlton, and Paul C. Kocher. “The Secure Sockets Layer (SSL) Protocol Version 3.0”. RFC 6101. 2011. URL: https://www.rfc-editor.org/info/rfc6101.

[20]Christopher Allen and Tim Dierks. “The TLS Protocol Version 1.0”. RFC 2246. 1999. URL: https://www.rfc-editor.org/info/rfc2246.

[21]Tim Dierks and Eric Rescorla. “The Transport Layer Security (TLS) Protocol Version 1.1”. RFC 4346. 2006. URL: https://www.rfc-editor.org/info/rfc4346.

[22]Eric Rescorla and Tim Dierks. “The Transport Layer Security (TLS) Protocol Version 1.2”. RFC 5246. 2008. URL: https://www.rfc-editor.org/info/rfc5246.

[23]Eric Rescorla. “The Transport Layer Security (TLS) Protocol Version 1.3”. RFC 8446. 2018. URL: https://www.rfc-editor.org/info/rfc8446.

[24]Richard Barnes et al. “Deprecating Secure Sockets Layer Version 3.0”. RFC 7568. 2015. URL: https://www.rfc-editor.org/info/rfc7568.

[25]Kathleen Moriarty and Stephen Farrell. “Deprecating TLS 1.0 and TLS 1.1”. RFC 8996. 2021. URL: https://www.rfc-editor.org/info/rfc8996.

[26]“Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations”. SP800-52 Revision 2. 2019.

[27]“Guideline for TLS server configuration settings”. Ver. 3.1.1. 2025.

[28]Qualys SSL Labs. “SSL Pulse”. URL: https://www.ssllabs.com/ssl-pulse/ (visited on 2025-12-20).

[29]Donald E. Eastlake 3rd. “Transport Layer Security (TLS) Extensions: Extension Definitions”. RFC 6066. 2011. URL: https://www.rfc-editor.org/info/rfc6066.

[30]Daniel Kahn Gillmor. “Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)”. RFC 7919. 2016. URL: https://www.rfc-editor.org/info/rfc7919.

[31]Stephan Friedl et al. “Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension”. RFC 7301. 2014. URL: https://www.rfc-editor.org/info/rfc7301.

[32]Adam Langley. “A Transport Layer Security (TLS) ClientHello Padding Extension”. RFC 7685. 2015. URL: https://www.rfc-editor.org/info/rfc7685.

[33]Pasi Eronen et al. “Transport Layer Security (TLS) Session Resumption without Server-Side State”. RFC 5077. 2008. URL: https://www.rfc-editor.org/info/rfc5077.

[34]David Benjamin. “Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility”. RFC 8701. 2020. URL: https://www.rfc-editor.org/info/rfc8701.

[35]Yoav Nir, Simon Josefsson, and Manuel Pégourié-Gonnard. “Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier”. RFC 8422. 2018. URL: https://www.rfc-editor.org/info/rfc8422.

[36]Andrei Popov, Magnus Nyström, and Dirk Balfanz. “Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation”. RFC 8472. 2018. URL: https://www.rfc-editor.org/info/rfc8472.

[37]Richard Barnes et al. “Delegated Credentials for TLS and DTLS”. RFC 9345. 2023. URL: https://www.rfc-editor.org/info/rfc9345.

[38]David Benjamin and Victor Vasiliev. “TLS Application-Layer Protocol Settings Extension”. Internet-Draft draft-vvv-tls-alps-01. Work in Progress. Internet Engineering Task Force, 2020. 9 pp. URL: https://datatracker.ietf.org/doc/draft-vvv-tls-alps/01/.

[39]Eric Rescorla et al. “TLS Encrypted Client Hello”. Internet-Draft draft-ietf-tls-esni-25. Work in Progress. Internet Engineering Task Force, 2025. 53 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-esni/25/.

[40]L.Bassham, A.Rukhin, and J.Soto. “A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications”. Special Publication (SP) 800-22 Rev 1a. 2010.

[41]Robert G. Brown, Dirk Eddelbuettel, and David Bauer. “Dieharder: A Random Number Test Suite”. Duke University. URL: https://webhome.phy.duke.edu/~rgb/General/dieharder.php.

[42]Pierre L’Ecuyer and Richard Simard. “TestU01: A C library for empirical testing of random number generators”. In: *ACM Trans. Math. Softw.* 33.4 (2007). URL: https://doi.org/10.1145/1268776.1268777.

[43]Qualys SSL Labs. “HTTP Client Fingerprinting Using SSL Handshake Analysis”. 2009. URL: https://www.ssllabs.com/projects/client-fingerprinting/.

[44]Marek. “SSL fingerprinting for p0f”. 2012. URL: https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/.

[45]Lee Brotherston. “TLS fingerprinting”. URL: https://github.com/LeeBrotherston/tls-fingerprinting.

[46]Google. “VirusTotal”. URL: https://www.virustotal.com/.

[47]Shodan. “Shodan”. URL: https://www.shodan.io/.

[48]Wireshark Foundation. “Wireshark”. URL: https://www.wireshark.org/.

[49]Open Information Security Foundation. “Suricata”. URL: https://suricata.io/.

[50]SSL Blacklist project. “JA3 Fingerprints”. URL: https://sslbl.abuse.ch/ja3-fingerprints/.

[51]Sergey Frolov and Eric Wustrow. “The use of TLS in Censorship Circumvention”. In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS 2019. 2019.

[52]Blake Anderson. “The Generation and Use of TLS Fingerprints”. In: *FloCon*. 2019.

[53]Cisco. “Joy”. URL: https://github.com/cisco/joy.

[54]Cisco. “Mercury”. URL: https://github.com/cisco/mercury.

[55]David McGrew. “Bayes at 10+ Gbps: Identifying Malicious and Vulnerable Processes from Passive Traffic Fingerprinting”. In: *FloCon*. 2020.

[56]Richard Moore. “TLS Prober”. URL: https://github.com/WestpointLtd/tls_prober.

[57]John Althouse et al. “JARM”. Salesforce, Inc. URL: https://github.com/salesforce/jarm.

[58]Blake Anderson and David McGrew. “Accurate TLS Fingerprinting using Destination Context and Knowledge Bases”. 2020. URL: https://arxiv.org/abs/2009.01939 (visited on 2025-12-20).

[59]Susu Cui et al. “MVDet: Encrypted malware traffic detection via multi-view analysis”. In: *Journal of Computer Security* 32.6 (2024), pp. 533–555.

[60]Zihao Wang, Kar Wai Fok, and Vrizlynn L.L. Thing. “Machine learning for encrypted malicious traffic detection: Approaches, datasets and comparative study”. In: *Comput. Secur.* 113.C (2022). URL: https://doi.org/10.1016/j.cose.2021.102542.

[61]Jiayong Liu et al. “A Distance-Based Method for Building an Encrypted Malware Traffic Identification Framework”. In: *IEEE Access* 7 (2019), pp. 100014–100028.

[62]Susu Cui et al. “CBSeq: A Channel-Level Behavior Sequence for Encrypted Malware Traffic Detection”. In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 5011–5025.

[63]Jiyuan Liu et al. “MalDetect: A structure of encrypted malware traffic detection”. In: *Comput. Mater. Contin.* 60.2 (2019), pp. 721–739.

[64]Andrey Ferriyan et al. “Encrypted Malicious Traffic Detection Based on Word2Vec”. In: *Electronics* 11.5 (2022), p. 679.

[65]Gibran Gomez et al. “Unsupervised Detection and Clustering of Malicious TLS Flows”. In: *Security and Communication Networks* 2023.1 (2023), p. 3676692. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2023/3676692.

[66]Zihao Wang and Vrizlynn L.L. Thing. “Feature mining for encrypted malicious traffic detection with deep learning and other machine learning algorithms”. In: *Computers & Security* 128 (2023), p. 103143.

[67]Jianyi Liu et al. “Spatial-Temporal Feature with Dual-Attention Mechanism for Encrypted Malicious Traffic Detection”. In: *Security and Communication Networks* 2023.1 (2023), p. 7117863. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2023/7117863. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/7117863.

[68]Gang Long and Zhaoxin Zhang. “Deep Encrypted Traffic Detection: An Anomaly Detection Framework for Encryption Traffic Based on Parallel Automatic Feature Extraction”. In: *Computational Intelligence and Neuroscience* 2023.1 (2023), p. 3316642. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2023/3316642.

[69]Akamai Security Intelligence Response Team. “Bots Tampering With TLS to Avoid Detection”. Akamai Technologies. URL: https://www.akamai.com/blog/security/bots-tampering-with-tls-to-avoid-detection.

[70]Blake Anderson, Subharthi Paul, and David McGrew. “Deciphering malware’ s use of TLS (without decryption)”. In: *Journal of Computer Virology and Hacking Techniques* 14 (2018), pp. 195–211.

[71]Diogo Barradas et al. “Extending C2 Traffic Detection Methodologies: From TLS 1.2 to TLS 1.3-enabled Malware”. In: *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*. RAID ’24. Association for Computing Machinery, 2024, pp. 181–196.

[72]Routa Moussaileb et al. “Ransomware network traffic analysis for pre-encryption alert”. In: *Foundations and Practice of Security*. FPS 2019. Springer. 2020, pp. 20–38.

[73]May Almousa, Janet Osawere, and Mohd Anwar. “Identification of ransomware families by analyzing network traffic using machine learning techniques”. In: *2021 Third international conference on transdisciplinary AI (TransAI)*. IEEE. 2021, pp. 19–24.

[74]Juncheng Guo et al. “MGEL: a robust malware encrypted traffic detection method based on ensemble learning with multi-grained features”. In: *International Conference on Computational Science*. Springer. 2021, pp. 195–208.

[75]Qualys SSL Labs. “SSL Pulse”. URL: https://www.ssllabs.com/ssl-pulse/.

[76]David Benjamin and David Adrian. “Feature: TLS ClientHello extension permutation”. URL: https://chromestatus.com/feature/5124606246518784.

[77]Leander Schwarz and Dennis Jackson. “Add Option for Randomizing TLS Client Hello Extension Order”. URL: https://bugzilla.mozilla.org/show_bug.cgi?id=1789436.

[78]In-Su Jung et al. “Enhanced Encrypted Traffic Analysis Leveraging Graph Neural Networks and Optimized Feature Dimensionality Reduction”. In: *Symmetry* 16 (2024), p. 733.

[79]Carsten Bormann and Paul E. Hoffman. “Concise Binary Object Representation (CBOR)”. RFC 8949. 2020. URL: https://www.rfc-editor.org/info/rfc8949.

[80]Michael B. Jones et al. “CBOR Web Token (CWT)”. RFC 8392. 2018. URL: https://www.rfc-editor.org/info/rfc8392.

[81]John Dickinson et al. “Compacted-DNS (C-DNS): A Format for DNS Packet Capture”. RFC 8618. 2019. URL: https://www.rfc-editor.org/info/rfc8618.

[82]Henk Birkholz, Christoph Vigano, and Carsten Bormann. “Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures”. RFC 8610. 2019. URL: https://www.rfc-editor.org/info/rfc8610.

[83]Paul C. Bryan and Mark Nottingham. “JavaScript Object Notation (JSON) Patch”. RFC 6902. 2013. URL: https://www.rfc-editor.org/info/rfc6902.

[84]Graham Cormode and S. Muthukrishnan. “The string edit distance matching problem with moves”. In: *ACM Trans. Algorithms* 3.1 (2007).

[85]Stratosphere. “Stratosphere Laboratory Datasets”. Stratosphere Research Laboratory. 2015. URL: https://www.stratosphereips.org/datasets-overview.

[86]Nimesha Wickramasinghe et al. “ SoK: Decoding the Enigma of Encrypted Network Traffic Classifiers ”. In: *2025 IEEE Symposium on Security and Privacy*. 2025, pp. 1825–1843.

[87]Selenium Project. “Docker images for the Selenium Grid Server”. URL: https://github.com/SeleniumHQ/docker-selenium.

[88]The Tcpdump Group. “TCPDUMP & LIBPCAP”. URL: https://www.tcpdump.org/.

[89]Victor Le Pochat et al. “Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation”. In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. NDSS 2019. 2019.

[90]Atsushi Kanda, Masaki Hashimoto, and Takao Okubo. “TLS Parameter Fluctuation Traffic Generator”. IISEC. URL: https://github.com/atsushikandan/TLSParameterFluctuationTrafficGenerator.

[91]Robert Falcone and Jen Miller-Osborn. “Scarlet Mimic: Years-Long Espionage Campaign Targets Minority Activists”. Palo Alto Networks, Inc. 2016. URL: https://unit42.paloaltonetworks.com/scarlet-mimic-years-long-espionage-targets-minority-activists/.

[92]Sudeep Singh and Atinderpal Singh. “The Return of the Higaisa APT”. Zscaler, Inc. 2020. URL: https://www.zscaler.com/blogs/security-research/return-higaisa-apt.

[93]Telegram Messenger Inc. “MTProxy: Simple MT-Proto proxy”. Telegram Messenger Inc. URL: https://github.com/TelegramMessenger/MTProxy.

[94]Sqrrl Data, Inc. “A Framework for Cyber Threat Hunting”. White Paper. 2018.

[95]Julien Olivain and Jean Goubault-Larrecq. “Detecting Subverted Cryptographic Protocols by Entropy Checking”. Research Report LSV-06-13. ENS Cachan, 2006.

[96]Andrew M White et al. “Clear and Present Data: Opaque Traffic and its Security Implications for the Future”. In: *NDSS 2013*. 2013.

[97]Han Zhang, Christos Papadopoulos, and Dan Massey. “Detecting encrypted botnet traffic”. In: *2013 Proceedings IEEE INFOCOM*. 2013.

[98]Shoufu Luo, Jeremy D. Seideman, and Sven Dietrich. “Fingerprinting Cryptographic Protocols with Key Exchange Using an Entropy Measure”. In: *2018 IEEE Security and Privacy Workshops (SPW)*. 2018.

[99]Liu Qinrang Zhao Bo and Liu Xiaomin. “Evaluation of Encrypted Data Identification Methods Based on Randomness Test”. In: *2011 IEEE/ACM International Conference on Green Computing and Communications*. IEEE, 2011.

[100]Satadal Sengupta et al. “Exploiting Diversity in Android TLS Implementations for Mobile App Traffic Classification”. In: *The World Wide Web Conference*. WWW ’19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 1657–1668.

[101]Bernhard Schölkopf et al. “Estimating the Support of a High-Dimensional Distribution”. In: *Neural Comput.* 13.7 (2001), pp. 1443–1471. URL: https://doi.org/10.1162/089976601750264965.

[102]Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. “Isolation Forest”. In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422.

[103]Markus M. Breunig et al. “LOF: Identifying Density-Based Local Outliers”. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’00. Dallas, Texas, USA: Association for Computing Machinery, 2000, pp. 93–104. URL: https://doi.org/10.1145/342009.335388.

[104]Arash Habibi Lashkari et al. “Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification”. In: *2018 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2018, pp. 1–7.

[105]Atsushi Kanda, Masaki Hashimoto, and Takao Okubo. “TLS Lie Detector Dataset”. IISEC. URL: https://github.com/atsushikandan/TLSLieDetector_Dataset.

[106]Shusei Tomonaga. “Windows commands that attackers abuse(2015-12-02)”. JPCERT/CC. 2015. URL: https://blogs.jpcert.or.jp/ja/2015/12/wincommand.html.

[107]The MITRE Corporation. “MITRE ATT&CK v18.1”. The MITRE Corporation. URL: https://attack.mitre.org/versions/v18.

[108]Manuel Lopez-Martin et al. “Network Traffic Classifier with Convolutional and Recurrent Neural Networks for Internet of Things”. In: *IEEE Access* (2017).

[109]Sudeep Singh. “Latest Mustang Panda Arsenal: ToneShell and StarProxy | P1”. Zscaler ThreatLabz. URL: https://www.zscaler.com/blogs/security-research/latest-mustang-panda-arsenal-toneshell-and-starproxy-p1.