Inspection-Friendly TLS/HTTPS

by

Joseph Turcotte and Eda Zhou

A Major Qualifying Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Computer Science

by

Joseph Turcotte

Eda Zhou

March 2020

APPROVED:

Lorenzo De Carli

Abstract

As the Internet grows, Transport Layer Security (TLS) is becoming the standard to secure, end-to-end encryption. Compared to plaintext communication, TLS offers increased privacy to communicating parties because other parties cannot understand or manipulate the data being sent. However, end-to-end encryption can detract from user privacy; many Internet of Things (IoT) devices have been revealed to track excessive user data that is not required for their function. With their manufacturers' collecting data through an encrypted connection, users cannot see or control what information is being sent. Thus, a new approach is needed to address growing concerns over violations of user privacy by smart devices while inheriting the security properties of TLS.

In this project, we propose Inspection-Friendly TLS (IF-TLS), a TLS-based protocol that preserves the encryption offered by TLS while allowing middleboxes to observe traffic. We delegate control to the user to decide which devices should be inspected and what middleboxes have decryption capabilities through an access control list (ACL). The IF-TLS manager uses the ACL to obtain and share devices' session keys with trusted middleboxes. Additionally, we allow multiple middleboxes to be involved in the decryption process from anywhere in the network, including the cloud and the local area network.

We created a stable, comprehensive implementation of the IF-TLS protocol using Python. We included features such as session key establishment and sharing, lookups in an access control list that specifies decryption privileges for middleboxes, and tunneling traffic through middleboxes between a client and server. Additionally, we developed a working test bed to collect performance data for different IF-TLS

scenarios, such as comparing IF-TLS with middleboxes in the local network versus in the cloud. The data we collected shows that IF-TLS can perform reasonably for IoT devices in residential networks. Without a middlebox in the connection, we observed a 21 percent increase in the total IF-TLS initialization time compared to TLS 1.3, as well as similar data round-trip times compared to TLS 1.3. Finally, we conducted a qualitative security analysis of the IF-TLS protocol using the STRIDE and DREAD threat models; since the protocol's security depends on the devices running IF-TLS themselves, we claim that IF-TLS generally preserves the security properties of TLS 1.3.

Acknowledgements

We would like to thank our advisor, Professor Lorenzo De Carli, for his unwavering support and guidance. We would also like to thank Vishwajeet Bhosale of Colorado State University for providing us with traffic captures that we used for experiments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The total number of Internet-connected devices is expected to reach 20 billion by 2020 [1]. The growing number of available, connected devices means an even greater abundance of data that these devices generate and send to each other. Although the growing interconnectedness of the world's technology presents opportunities for rapid development and improvement of the quality of life, it also presents issues related to privacy. Personally identifiable information, such as usernames, passwords, locations, and addresses, is not only found on the devices themselves, but also in the streams of data these devices transmit. If there is no protection or security on the data transmitted, this information can be used to harm individuals, families, and organizations.

Fortunately, the network security community has developed feasible and secure methods for protecting data in transit. The Transport Layer Security (TLS) protocol provides security for "Internet of Things" (IoT) communications. With end-to-end data encryption and authentication, it becomes much more difficult to extract personally identifiable information from transmitted data, and thus owners of IoT devices face lower risks of having their information compromised while the devices

are in use. The assurance of data security is especially important for IoT streaming devices used in households, such as smart home cameras and smart baby monitors, that continuously transmit photos or videos of a user's home or family members with the expectation that other parties cannot access or manipulate the streams.

In some cases, however, the cryptographic mechanisms intended to protect user privacy can have the opposite effect. Although end-to-end encryption protects data while it is in transit, this data is not being inspected by any entities other than the two communicating endpoints. Therefore, it is not only possible, but likely that the IoT devices users keep in their homes are transmitting excessive data containing personal information; in many cases, this behavior would qualify as a privacy violation. It has been shown that streaming services utilize device tracking for advertising [2], and that home security systems contain hidden microphones for recording [3]. Unfortunately, end-to-end encryption cannot detect these violations because no other entity has access to the unencrypted data for inspection and detection purposes.

Currently, the commonly accepted approach is to sacrifice potential user privacy violations in favor of data security. Some approaches, such as BlindBox [4] and Multi-Context TLS [5], provide middlebox decryption capabilities in a secure manner but introduce infeasible overhead for IoT devices. Other approaches, such as man-in-the-middle proxies [6], use risky workarounds to give middleboxes decryption privileges; these approaches often introduce new security risks and challenges that outweigh the benefits of traffic inspection. Given the significant limitations and risks of current approaches, one idea is to rethink the TLS protocol in a way that specifies middleboxes as secure entities in IoT communication.

Inspection-Friendly TLS (IF-TLS) is a TLS-based protocol that preserves the secure data transit property from TLS, but also allows authenticated entities called middleboxes to inspect the data while it is in transit. Most importantly, the user

has control over which middleboxes have the ability to inspect traffic from each of their devices. Furthermore, these middleboxes can be located anywhere, including the local area network and the cloud.

Some of the challenges we faced were related to the protocol design; these challenges included determining efficient methods to share keys, generate and store access control rules, and transmit data. We often referred to our use case—IoT devices in a household—to make the most rational design choices. Other challenges involved implementation and testing details, including configuring routing rules and incorporating a cloud-based middlebox into the communication. We simplified some implementation and testing aspects and left those details for future work.

We built a proof-of-concept and ran performance experiments to see if the balance between security and user privacy could be feasibly obtained. Additionally, we conducted a security analysis of the protocol's design and identified potential attacks and countermeasures. We observed that, compared to TLS 1.3, IF-TLS performs reasonably for IoT devices in residential networks. Without a middlebox in the connection, we observed a 21 percent increase in the total IF-TLS initialization time compared to TLS 1.3, as well as similar data round-trip times compared to TLS 1.3. Additionally, we claim that IF-TLS generally preserves the security properties of TLS 1.3 while also benefiting from increased user privacy.

We outline the rest of the report in the following manner. Chapter 2 provides a summary of the background concepts and related work we reviewed to inform our protocol design. Chapter 3 provides a detailed discussion of the protocol design and implementation. Chapter 4 outlines our two-part evaluation plan that focuses on the performance and security of IF-TLS. In Chapter 5, we present and discuss our performance and security results. Chapter 6 describes opportunities for future work related to IF-TLS. Finally, we offer concluding remarks in Chapter 7.

Chapter 2

Background

2.1 TLS

Transport Layer Security (TLS) is the standard protocol for creating an end-to-end secure connection using encryption. It is built on top of the TCP/IP protocol suite that provides data transport services used on the Internet. Web applications and services use HTTPS, or HTTP web traffic encrypted with TLS, to protect plaintext data and avoid revealing sensitive user data to unauthorized parties. In 2018, 72.2% of all network traffic used HTTPS; this reflects an increase of nearly 20 percent from 2016 [7]. TLS/HTTPS provides secure communication in a variety of contexts, such as online banking, web browsing, and email, that rely on TLS to encrypt data intended to be private.

From 2008 to 2018, TLS 1.2 was the default standard for secure communications over the Internet [8]. According to RFC 5246, TLS 1.2 "allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery" [8]. Figure 2.1 shows how a TLS connection is initiated; a client and server first perform the TCP handshake, and then exchange a

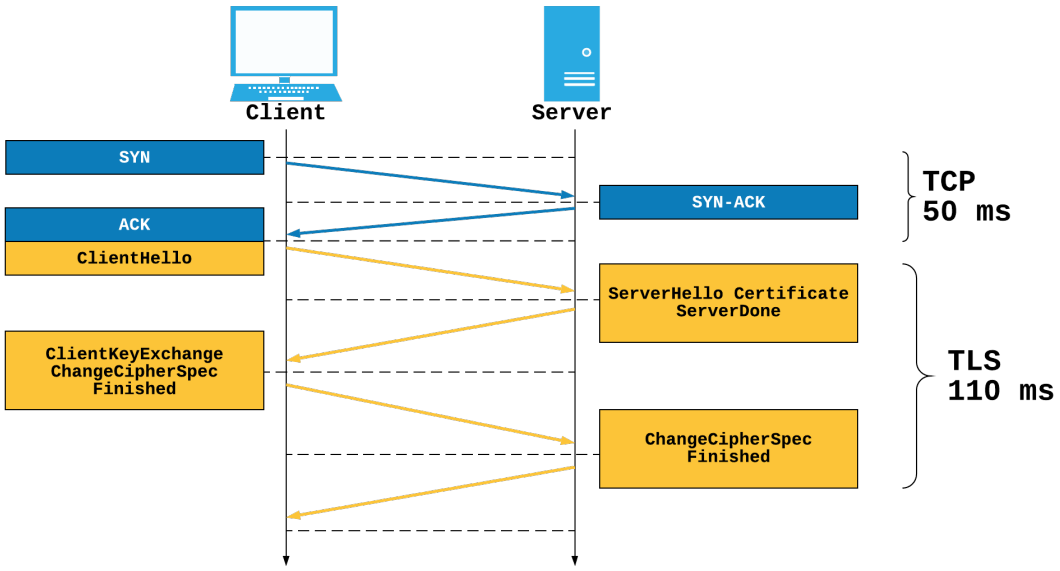

Figure 2.1: In the TLS Handshake Protocol, a client and server must agree on an encryption cipher suite and successfully establish a session to exchange encrypted data using TLS. Adapted from [9].

series of messages that are used to authenticate the entities and establish session keys to be used for the secure communication. The session keys are directly derived from a pre-master secret hash that is shared by the client and encrypted with the server's public key. Once the connection is initiated, the client and server exchange encrypted data for the duration of the session; RFC 5246 recommends an upper limit of 24 hours per session. TLS 1.2 provides authentication through the key exchange handshake, confidentiality with encryption of session data, and integrity by including a message authentication code (MAC).

In August 2018, RFC 8446 specified the latest version of TLS, known as TLS 1.3 [10]. TLS 1.3 introduced a number of changes from TLS 1.2; the most notable is the removal of legacy symmetric encryption algorithms that no longer provided a sufficient level of security. As a part of this, TLS 1.3 removed support for algorithms that do not provide forward secrecy, or the property that a compromised key cannot

be used to decrypt past or future traffic. This change removed static RSA and Diffie-Hellman cipher suites from the set of encryption algorithms that a client and server can agree to use when conducting the TLS Handshake Protocol.

The changes in TLS 1.3 were intended to create a more secure Internet by removing algorithms that undermined the power of encrypting connections at the network endpoints. However, it also introduced new challenges for network-based security solutions. For example, the static RSA and Diffie-Hellman suites allowed a server to pre-share its private key with network-based services that relied on inspecting un-encrypted data [11]; in TLS 1.3, this is no longer possible. Scenarios such as troubleshooting and malware protection become more difficult to handle when only the endpoints have access to plaintext. It is also possible that attacks could go undetected by allowing encrypted data to pass through the network without inspection.

Enterprise and university networks have developed workarounds to include network-based services in a TLS session, but they often undermine the security that TLS 1.3 was designed to provide. The most common workaround is a man-in-the-middle (MITM) proxy (Figure 2.2). A MITM proxy works by masquerading as the client and server, terminating the original TLS connection, and creating a new one with the receiving entity [6]. This process requires manually registering the MITM as a trusted authority with the client by installing the MITM's certificate. If the proxy were to misbehave or be compromised, the client would still be forced to trust the proxy instead of the real server; thus, adopting this workaround sacrifices authentication for decryption capabilities. In spite of this trade-off, an estimated 5-10% of all traffic is intercepted [12], and the network security community is often at odds with enterprise networks and Internet Service Providers with respect to this issue.

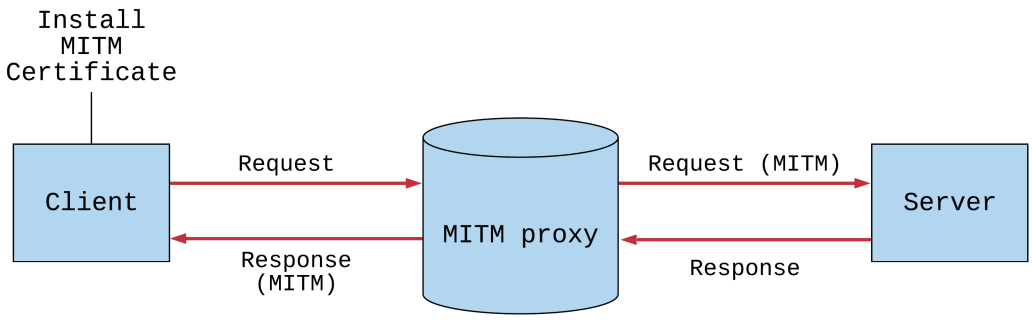

Figure 2.2: A MITM proxy pretends to be the client when interacting with the server so it can decrypt the traffic between the two entities. Adapted from [13].

2.2 IoT Devices

Internet of Things (IoT or "smart") devices are generally characterized by three qualities: 1) their ability to be controlled remotely; 2) their ability to communicate sensor data, such as the weather or heart rate, to users; and 3) their ability to connect to other devices over the Internet [14]. For example, smart speakers use voice recognition systems such as Amazon's Alexa or Apple's Siri to listen and respond to user requests using remote search engines. Additionally, smart cameras can extract information from images and send this information off to remote systems that can detect and raise warnings if anomalous behavior is detected. Recently, IoT devices have emerged in consumer markets as appliances that users can program to fit their personal needs. The diversity of IoT devices has appealed to homeowners who can utilize these devices to increase home security and make some aspects of domestic life more convenient. Even a smart egg tray, a device that tells a user when eggs in the fridge are going bad, can make day-to-day tasks simpler. However, networked IoT devices behave differently from web browsers in the context of TLS. IoT devices contain "always-on" sensors that transmit information to the Internet, regardless of whether the device is actively being used.

Researchers have raised multiple concerns about the potential for smart devices to violate user privacy because of their constant monitoring and transmitting of data. Apthorpe et al. [15] present a variety of case studies to explore how traffic send and receive rates from smart devices can be used to identify user behaviors. For example, analyzing peaks in traffic from the Sense sleep monitor allowed the researchers to correctly identify when the user got into and out of bed. Ren et al. [16] analyzed 81 IoT devices for exposure of personally identifiable information and found that some devices exposed their device IDs, locations at the city level, and user-specified or related device names, both in encrypted and un-encrypted traffic. Given that this information is transmitted frequently, an Internet Service Provider has the capability to track a device and see where and how often it is being used. Moghaddam et al. [2] developed a smart device crawler to intercept TLS traffic from TV streaming devices; they discovered that many streaming services, such as Roku and Amazon Fire, employ device tracking for advertising purposes, and that unique identifiers are often collected and transmitted over un-encrypted communications. Finally, Google revealed in February 2019 that its Nest home security system was equipped with a microphone that was responsive to voice commands; users did not know about this microphone and expressed concerns that they were being recorded without consent [3].

The case studies above represent significant IoT privacy vulnerabilities and potential violations. The pervasive nature of these vulnerabilities requires a new approach to protect user privacy while also allowing the users to enjoy the benefits of smart devices.

2.3 Middleboxes

The term *middlebox* refers to any device that intercepts traffic for purposes other than forwarding, such as inspecting or filtering traffic. When endpoints communicate, the traffic is often sent through user-owned, Internet Service Provider-owned, or company-owned devices. Common examples of middleboxes include firewalls, load balancers, network address translators, and intrusion detection systems. These middleboxes can be placed in the network, rather than at the endpoints, to augment security and performance by inspecting the data sent between the endpoints. In some cases, however, deploying middleboxes can introduce unneeded complexity in the network topology; additionally, middleboxes can exhibit high failure rates in larger networks, such as data center networks [17]. Several research efforts have explored methods to maximize the effectiveness of middleboxes while avoiding additional network complexity and failure management.

One effort involves cloud computing. Sherry et al. [18] explore services that outsource enterprise middlebox data processing to the cloud. The APLOMB system consists of a cloud provider that holds the enterprise's middleboxes and the APLOMB gateway that redirects traffic to the cloud for processing. The system is able to outsource over 90% of middlebox hardware to the cloud while only imposing 1.1ms of latency penalty on average. Additionally, Sherry et al. propose Embark [19], a system that also outsources middlebox processing with the goal of protecting the client's confidentiality. Embark augments the APLOMB architecture by encrypting the traffic as it passes through the cloud; the cloud provider processes the encrypted traffic using a set of keyword and prefix matching rules so that it never has access to the client's plaintext data. This helps protect the client in the event of unintentional or purposeful data leaks while still benefiting from cloud computing.

Another effort involves software-defined networking (SDN), a methodology that makes a network flexible and configurable through a series of programmable controllers [20]. Taylor et al. propose TLSDeputy [21], a system that incorporates middleboxes to augment security in residential networks by protecting clients from spoofed TLS servers. TLSDeputy monitors the TLS handshake process described in Section 2.1 by checking TLS certificates and verifying that these certificates have not been revoked. Compared to APLOMB and Embark, SDN offers a simpler approach to outsourcing middleboxes in residential networks because SDN only requires software modifications, rather than requiring a specialized network gateway in the case of APLOMB and Embark. Additionally, Taylor et al. address the feasibility of combining SDN controllers with cloud computing in residential networks [22]. The researchers considered the impacts of placing SDN controllers in the cloud with respect to page load times and latency for residential networks. In a study of 270 residential users across the U.S., the controllers provided 90% of the users with acceptable performance. By combining the decreased costs associated with cloud computing and the increased network flexibility associated with SDN, the system was able to achieve a higher level of security without sacrificing performance.

End-to-end encryption offered by TLS can hinder user privacy because it is impossible for any other party to ensure that unnecessary traffic or personally identifiable information is not being transmitted. With end-to-end encryption, it is even simpler for IoT device manufacturers to covertly collect user data. Thus, a new approach is needed to balance the encryption benefits offered by TLS and the privacy benefits offered by a trusted detection system. While TLS 1.3 is meant to protect the end-points, our goal is to protect the user. As we discuss in Chapter 3, our proposed design inherits the properties of TLS while also enabling trusted middleboxes to decrypt and filter data from the user's IoT devices.

2.4 Related Work

In Chapter 3 we discuss our proposed design for a new version of the TLS protocol that balances encryption and privacy benefits for IoT devices in residential networks. In this section we discuss efforts related to key points of our design, including access control and local key sharing.

Naylor et al. [5] propose Multi-Context TLS (mcTLS) to provide decryption capabilities to middleboxes without requiring the client to install a root certificate. The notion of an encryption "context", or a set of symmetric encryption and message authentication code (MAC) keys, functions as an access control mechanism that allows for flexible configuration of middlebox decryption capabilities. Endpoints can limit the kinds of data to which a middlebox has decryption access and restrict permissions to read-only for the purpose of preventing illegal data modifications. We omit some details from mcTLS in our approach because of our application. Since the middleboxes that are used with IF-TLS are intended to perform inspection (reads) on IoT device traffic and not modification (writes), we do not provide write permission and individual flow decryption to middleboxes. Instead, we specify the granularity of the IF-TLS protocol at the device level and route all flows from a device through the specified middleboxes.

Bierma et al. [13] discuss Locally Operated Cooperative Key Sharing (LOCKS), a mechanism that allows clients to share TLS session keys with a trusted agent, such as a security monitoring system, inside an enterprise network. The trusted agent stores these keys and forwards them to middleboxes as needed. Thus, LOCKS allows deep packet inspection (DPI) to be performed on traffic entering and exiting an enterprise network without introducing authentication risks associated with MITM proxies. However, this approach has only been demonstrated for enterprise environ-

ments and would not work with middleboxes in the cloud. We still use the idea of sharing session keys in our proposal.

Sherry et al. [4] propose BlindBox, a system that combines the benefits of middleboxes and encryption by allowing DPI on encrypted traffic. Rather than providing decryption capabilities to middleboxes, BlindBox utilizes searchable encryption to inspect encrypted traffic using keywords or according to pre-defined rules, much like an IDS such as Bro or Snort. This mechanism restricts the capabilities of middleboxes to inspect and analyze user data, which may make users more comfortable trusting the services that would ordinarily have access to plaintext emails or search history. However, the connection setup happens on the order of minutes for large IDS systems with thousands of search rules and thus is impractical for configuration on smart devices. In our approach, we give full, unconditional decryption access to middleboxes; this feature is the main difference between IF-TLS and TLS 1.3.

Wilson et al. [23] propose TLS-Rotate and Release (TLS-RaR), a system that allows device owners to decrypt and audit TLS traffic using a key rotation mechanism for middleboxes. The user can request that the TLS connection keys are rotated and released to an in-network middlebox that decrypts and analyzes past communications under that set of connection keys. Like other implementations that enable middlebox decryption capabilities, this prevents an IoT vendor from having complete control of TLS communications. We use the notion of short-lived session keys in our approach, and we allow middleboxes to decrypt traffic in real time with a short transition period to re-establish session keys.

Our approach draws on the related work by taking the beneficial aspects of each work and applying them to the IF-TLS protocol design. Like the other works, we also use delay as a metric to evaluate our protocol; we discuss this in Chapters 4 and 5.

Chapter 3

Implementation Plan

3.1 System Design

In this section, we explain the design of IF-TLS, the assumptions we make, and the novel contributions of our protocol. At a high level, IF-TLS works by sharing the IoT device and server session keys with authorized and trusted middleboxes defined in an access control list (ACL). These middleboxes are permitted to inspect all traffic from a device. The user first provides an ACL file that designates which middleboxes will be able to inspect traffic from their IoT devices. Each IoT device using IF-TLS will share its client-server session key with the IF-TLS manager. The IF-TLS manager then reads the ACL file and shares the device's session key with each authorized middlebox. Finally, all traffic from the IoT device will be routed through the middleboxes by the manager (Figure 3.1). These steps can be broken into three phases: (1) user initialization where trusted middleboxes are identified and assigned to IoT devices; (2) key sharing, the core of IF-TLS, where session keys are established and shared; (3) data sending where packets from the client to the server are redirected through the middlebox. The server's traffic to the client is also

sent using IF-TLS, but it is not rerouted. This is intentional because the purpose of IF-TLS is to enable users to view data from their IoT devices. The server's commands to the client are outside the scope of this project; however, we discuss this possible extension in Chapter 6.

Before we detail the protocol, we need to establish what systems need to run IF-TLS:

1. IoT devices: The smart devices we want to examine traffic from. We will refer to these as the clients who will share their session keys with the manager. These could be Google Nests, Echo Dots, smart egg trays, etc.

2. Manager: The device that runs the IF-TLS manager. The manager is responsible for configuring the middleboxes and the client. While we assume the application is on a local router, it can be put anywhere where all local area network traffic can be processed (such as on a specialized hardware device).

3. Middleboxes: The trusted middleboxes that the user wants to inspect their IoT device traffic. The middleboxes will receive the decryption (session) key from the manager. We expect the middleboxes to run intrusion detection or prevention systems; these systems can look for anomalies in network traffic. More advanced systems could also detect undesired privacy leaks. However, the specific function of the middlebox beyond being able to decrypt data is outside the scope of our project.

The following subsections expand on the three steps in IF-TLS: user initialization, key sharing, and data sending.

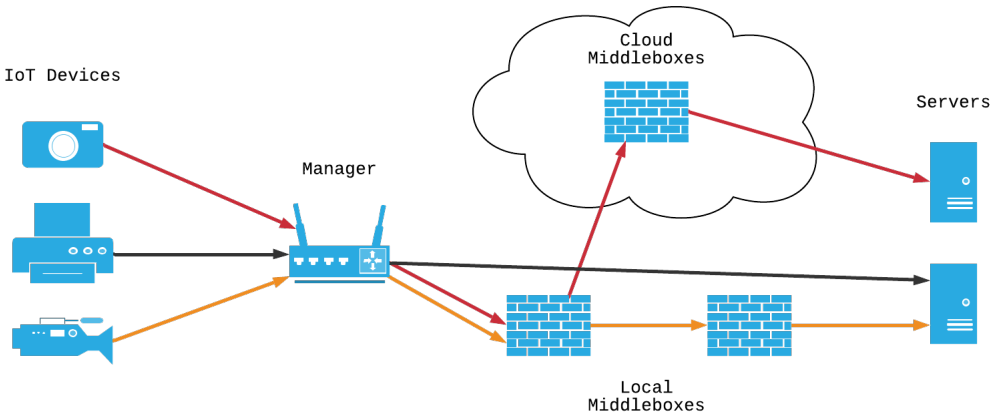

Figure 3.1: High-level system vision of IF-TLS protocol

3.1.1 User Initialization

The first step in IF-TLS is providing the access rules that determine which middleboxes will be able to decrypt which IoT devices' traffic. As mentioned in Section 2.4, the granularity for our protocol is at the IoT device level. This information is supplied as an Access Control List (ACL) by the user. The ACL is a dictionary where the keys are IoT device media access control (MAC) addresses and the values are a list of middlebox IPs as seen in Table 3.1. This way we can efficiently determine the order in which the middleboxes will inspect traffic. The device traffic will be sent to the middleboxes in the same order as they appear in the list. For example, the first device, 00:11:22:33:44:55, in Table 3.1 will have its traffic routed to middlebox 1.1.1.1, followed by 1.1.1.2, and finally to the server. The ACL also incorporates our two access levels: no access or full decryption access. The access level depends on whether the IP is in the device's list. We chose this design because it is the job of the middlebox to analyze packets. If we were to enable partial decryption, we would have to pre-process a device's traffic to determine what is suspicious enough to allow a middlebox to inspect. This technique is used by [4], but we determined

Key: Device MAC	Value: List of Middlebox IPs
00:11:22:33:44:55	[1.1.1.1, 1.1.1.2]
11:22:33:44:55:66	[1.1.1.2]
22:33:44:55:66:77	[1.1.1.2, 1.1.1.3]

Table 3.1: Example Access Control List

for our project that the inspection is the task of a middlebox. The middlebox may need access to entire flows in order to classify what is suspicious or unnecessary. The purpose of our protocol is to allow middleboxes to decrypt TLS traffic and not to detect misbehaving IoT devices; we leave that for future work.

We use IP addresses as unique identifiers for middleboxes in the ACL because we need the network layer IP for packet forwarding to middleboxes outside the local network. These IP addresses need to be updated when they are changed. However, we expect middleboxes to be constantly running on a cloud instance or within the user's private network; therefore, they are less prone to being assigned new IP addresses. If a middlebox IP is not updated in the ACL when it changes, the old IP will receive the initialization requests from the IF-TLS manager. If the old IP is not configured to use IF-TLS, then the manager will terminate the connection and return an error. However, if the old IP becomes owned by an entity using IF-TLS, the entity will be able to decrypt the device traffic, given that there is no middlebox authentication. We further discuss this attack vector and its countermeasure in Section 5.2.

Since IoT devices are on the local area network, we can use the more stable media access control (MAC) addresses in our ACL. MACs are locally unique to each device and we assume that they are immutable. Since the MAC address is assigned when a device is made, an access rule can be set for a device even before it is installed. This enables the user to have all traffic, including setup, from a device inspected. We discuss the possibility and implications of MAC spoofing in Section 5.2.

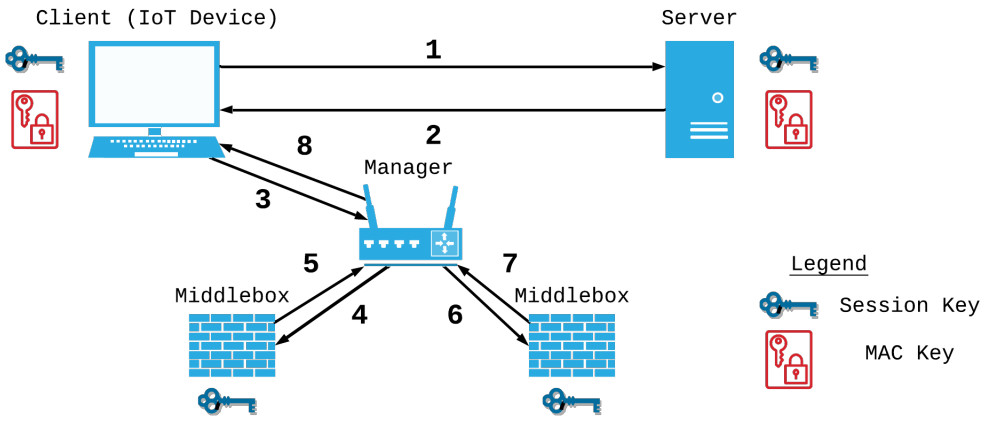

Figure 3.2: IF-TLS session and MAC keys are established and shared between the network entities.

The initialization process requires that the user defines an ACL for their current configuration. They can locate their devices' MAC addresses on the products themselves or through the devices' applications. The middlebox IPs will be provided once a service has been setup with their provider. The ACL is never final; once created, its rules can be altered as needed to include new devices or change middlebox permissions. Modifying access control rules requires modifying the ACL file and resetting the updated IF-TLS connections. Allowing modifications to the ACL enables the user to control their data and thus achieves one of the core goals of IF-TLS. If the user has a device they do not want to be inspected, they can simply exclude it. Any traffic that does not match a MAC address in the dictionary will be processed as normal and forwarded towards its destination. Once an ACL is provided, IF-TLS will function according to the ACL. We discuss changing access control rules dynamically in Section 6.1.

3.1.2 Key Sharing

Another technique IF-TLS introduces is key sharing. To prepare for data sending, each participating entity needs the IF-TLS session key. There are three initializations that need to occur: the client/server, client/manager, and manager/middlebox initializations. These are shown in Figure 3.2 as connections 1 & 2, 3 & 8, and 4-7, respectively. The client/server initialization establishes the cipher suites and keys for the IF-TLS session. Then, during the client/manager initialization, the client shares its session key pre-master secret with the manager. Finally, the manager/middlebox initialization is when the pre-master secret is distributed to the middleboxes. At the end of the initialization procedures, the client, server, and middlebox will have the session key. The manager does not store the session key since it does not need to inspect the client/server traffic.

Client/Server Initialization

The first connection to be established is the client/server connection. This is the base connection that determines the cipher suites and keys used. An IoT device, which we will refer to as the client, first establishes a TCP connection with the server. It then performs a series of exchanges similar to TLS to negotiate the cipher suites and share the pre-master secret. The following is an outline of the client/server IF-TLS handshake:

1. Client Hello: The client sends the cipher suites it wants to use (for encryption and the MAC) along with the length of the pre-master secret.

2. Server Hello: The server responds to the client if it agrees on the cipher suites; otherwise, the connection is closed. The server sends its certificate, its public key, and the asymmetric cipher suite used to the client.

3. Client Pre-master: The client responds to the server if it agrees on the cipher suite and verifies the server; otherwise, the connection is closed. The client then creates a pre-master secret, encrypts it with the server's public key, and sends it to the server.

4. Server Receive Pre-master, Server ACK: The server receives, decrypts, and computes the pre-master message from the client to create the session and MAC keys; this is explained in detail in Section 3.2. Finally, the server sends an acknowledgement (ACK) message encrypted using the session key and MAC key to the client. The server is now ready to send and receive data through IF-TLS.

5. Client Receive ACK: The client computes the session and MAC keys from the pre-master and receives the ACK message from the server. Then, the client decrypts the message and verifies the message and is now ready to send and receive data through IF-TLS.

The client/server initialization establishes the ciphers and keys the client and server use for IF-TLS. The user determines for how long the session keys last. Once that threshold is met, the manager sends a message to the client to reset its IF-TLS session; TLS 1.2 has a recommended upper limit of 24 hours per session [8]. Once the session and MAC keys have been computed, the client needs to share that session key with the manager to distribute to the trusted middleboxes.

The pre-master secret is a string of random bytes generated using os.urandom(), a cryptographically secure number generator. Half of the characters are used to generate the session key, whereas the other half is used to generate the MAC key. They are separately generated because we only want to share the session key with the middleboxes. If the keys were derived from the same hash, then the middleboxes

could generate the MAC key by guessing the cipher and gain write capabilities. Each half is then split again to be used for the key's generation and the salt. For example, a 64 byte (128 hexadecimal character) pre-master would be split as follows: the first 32 characters are used for the session key, the following 32 are for the session key salt, the next 32 are for the MAC key, and the final 32 are for the MAC key salt. The user can set the pre-master length to be compatible with the chosen ciphers. More details about our specific implementation are explained in Section 3.2.

Client/Manager Initialization

The IoT device shares its session key, an idea drawn from [13], with the manager during the client/manager initialization. This process is done over a TLS session to securely transfer the key and cipher information. Since a device needs to resend its session key to the manager with every new generation, this connection may remain open between IF-TLS sessions depending on the time-to-live set by the user on the TLS session.

In order to keep the packet length consistent, the client sends a portion of the pre-master secret to the manager, rather than the computed key itself. As explained in the client/server initialization section, the sent segment is used to generate the session key. The remaining portion of the pre-master secret is used to generate the client/server MAC key and is not shared because we only want the middlebox to have decryption capabilities, not writing capabilities. If a device fails to send a session key, the user can decide if the packets should still be forwarded to a middlebox, such as one that works on encrypted traffic or if all communications from the device should be dropped until the device complies. The manager does not compute the session key, but simply forwards the pre-master session key and cipher used to the middleboxes.

The manager/middlebox initialization is encapsulated within the client/manager initialization. Once the manager finishes sharing the pre-master session key with the specified middleboxes, the manager sends an ACK or NACK to the client. The acknowledgement message from the manager completes the client/manager initialization as well as the IF-TLS initialization process.

Manager/Middlebox Initialization

The final initialization is between the manager and the middleboxes. The manager consults the ACL to determine which middleboxes need a device's session key. Then, the manager initializes each middlebox listed in sequence by creating a TLS session with that middlebox and sharing the device's information (MAC address, cipher used) and pre-master session key. The middlebox computes the session key and respond with an ACK message if the calculation is successful. If any of the middleboxes responds with a NACK (i.e. because of an unsupported cipher), the manager stops its manager/middlebox initialization and sends a NACK to the client. The user can decide whether to remove the middlebox from the ACL or require the client to use a supported cipher suite. Additional security checks could also be added to ensure the authenticity of a middlebox, this is discussed in Chapter 6.

If all middleboxes respond with ACKs, the manager sends an ACK to the client and complete the IF-TLS initialization procedure. By sharing session keys, IF-TLS gives the middleboxes the ability to perform their inspections without giving them write abilities since they cannot create MACs. The delegated middleboxes now have the ability to inspect traffic.

3.1.3 Data Sending Procedure

The final step in IF-TLS is traffic forwarding. Once the ACL has been configured and the device's session key has been shared, packets can be redirected through the middleboxes (Figure 3.1). When a packet arrives at the manager running on the local router, the packet's source MAC address can be found in the data link layer. The MAC address is used to identify the originating IoT device and look it up in the ACL. If there is no match, then the packets are forwarded as usual. If there is a match, the corresponding dictionary value is retrieved. The packet is then forwarded to the middlebox(es) in the order they appear in the retrieved list. If there are multiple middleboxes, then they receive the traffic in sequential order according to their value (see Table 3.1). The destination of the packets is not checked, so all traffic from a matched device is re-routed.

A device's flows are inspected in sequence by middleboxes to reduce overhead. We considered having the data processed in parallel; however, this would require another application to analyze each middlebox's response and determine if a device's flows should still be sent to the server. This could be implemented at the router, but it would add another round-trip time delay for each middlebox. The user would then need to write rules on how to manage conflicting middlebox decisions and middleboxes that take too long to respond. Instead, middleboxes forward the data that they have inspected to the next middlebox; this process establishes a priority order for the middleboxes.

The packets that are routed through the middleboxes all contain a message authentication code to ensure integrity. While the middleboxes are able to decrypt the payloads, they are unable to create a new matching hash. If any middlebox manipulates the traffic, the server would detect the modifications by comparing the message hash to the message authentication code. Only the client and server are able

to generate and authenticate hashes. This enforces the policy that middleboxes can inspect data and drop flows, but cannot write to the packets. One of our intended use cases for IF-TLS is for middleboxes to drop traffic containing excessive user information. Therefore, we do not manipulate the contents of packets, but we do allow flows to be dropped in transit. We do not consider dropping flows as a form of manipulating packets since the contents of the packets themselves are not changed.

Finally, the client can send a closing message to the server. This procedure is similar to the concept of FIN packets for TCP. The client first sends a closing packet to the server. The server then responds with a closing ACK and closes the socket on its side. When the client receives the server's ACK, it closes its side of the socket as well. To reset a connection, the client re-establishes a TCP connection with the server and starts the IF-TLS initialization procedure from the beginning.

3.2 Software Implementation

This section explains the details of our implementation. The IF-TLS protocol itself is a Python module containing roughly 800 lines of code. Our initial codebase presented IF-TLS as a library of publicly callable functions. We later changed IF-TLS to a class for the purpose of preserving private fields and features, such as the session and MAC keys used for message encryption, decryption, and verification. We also wrote short modules for each of the entities participating in the communication (client, manager, middlebox, and server); each of these modules invoke an instance of the IF-TLS class and call the relevant procedures. Figure 3.3 shows the IF-TLS calls made between a client and server for the client/server initialization, as well as the subroutines that are called for each entity. The blue arrows pointing to the right represent the client sending messages to the server; the yellow arrows in the

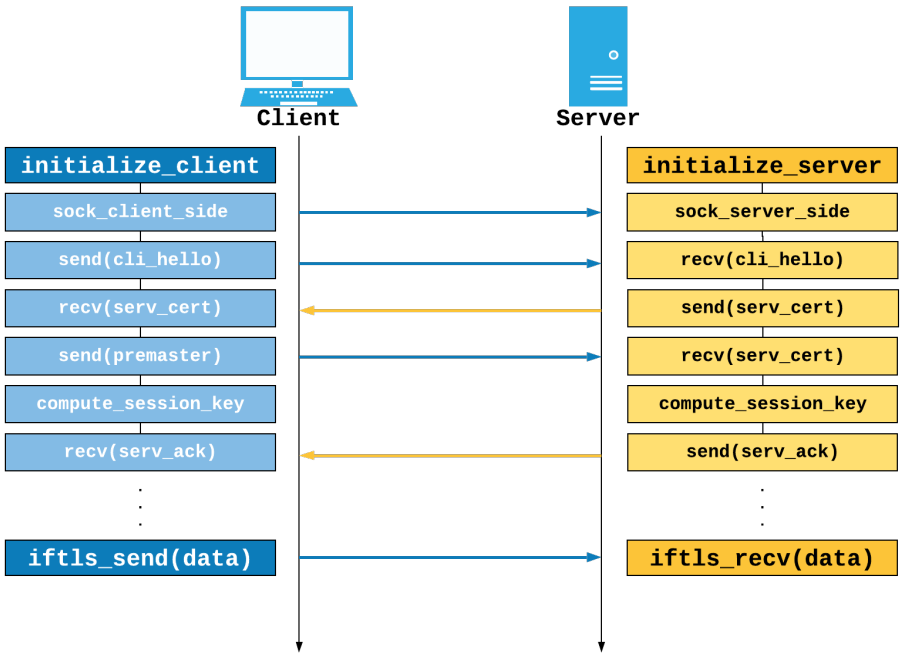

Figure 3.3: A sequence diagram of client-server calls using the IF-TLS API.

opposite direction represent messages sent from the server to the client. We assume in this diagram that the messages sent between the client and server are verified through the process we described in Section 3.1.2; otherwise, the connection is shut down before reaching the data sending step.

For our ciphers, we use PKCS1_OAEP (RSA) for the asymmetric cipher, AES for the symmetric cipher, and Poly1305 for the MAC hash. These ciphers are all actively supported in the pycryptodome library. The pre-master secret is comprised of 128 hexadecimal characters (64 bytes). Its characters are used as follows: the first 32 characters are for the session key, the following 32 are for the session key salt, the next 32 are for the MAC key, and the final 32 are for the MAC key salt. The key and salt are used to create 32 byte hashes (the session key and MAC key) used by AES and Poly1305. The 32 byte hash is the longest length key we can use for AES and allows us to use AES-256. When the client sends the pre-master secret

to the manager, it only sends the first 64 characters that are used to compute the session key.

Our set-up for IF-TLS included virtual machines for a client, server, manager, and middlebox. The client and manager ran on one network, while the server ran on a separate network. The middlebox ran either in the same local network as the client and manager or in the cloud. We only used one middlebox for our proof-of-concept. In order to route traffic to a designated local middlebox, we added default gateway rules to the client and manager's routing tables. On the client VM, we added a rule to forward all traffic on eth1 to the manager. Then, on the manager, we added a rule to forward all traffic to the local middlebox VM. This worked because the middlebox was within the local network and discoverable by the manager. The middlebox then used Scapy to passively inspect the data passing through. The middlebox did not hold any packets; rather, it simply decrypted and printed the data it saw using the IF-TLS session key.

We had to use a different technique in order to forward traffic to a cloud middlebox. For experimental purposes, we created direct TCP sockets between the client and middlebox and between the middlebox and server to use a cloud middlebox. In this set-up, the client sends data through the socket to the middlebox and the middlebox forwards the data to the server. Like before, the middlebox inspects the packets it receives, but does not hold any. Routing rules would not work with this arrangement because the middlebox is outside the local network. We also could not use the *via* flag option, which allows a gateway to the destination to be specified, because the network router had security settings that would not allow us to use it as a hop.

3.3 Tools and Devices

In our initial phases of testing, we experimented with a Sonos One Speaker to obtain packet captures using Wireshark. These experiments helped us gain an understanding of IoT device traffic patterns and behavior, including how long the device maintains TLS and TCP sessions and how many packets are transmitted in a TCP session. Later, we analyzed packet captures in Wireshark from a variety of IoT devices; in Chapter 4, we describe the packet captures we used for evaluating IF-TLS from a performance standpoint.

We decided to use virtual machines to host our network-based entities because standardizing the operating system and environment made our performance evaluation much simpler. We chose TinyCore virtual machines to host the client, manager, and middleboxes based in the local network. The TinyCore virtual machines are based on a lightweight Linux distribution that allowed us to run our manager and middlebox modules after installing the necessary Python packages. It also allowed us to easily configure our routing rules using basic Linux commands. A more detailed set of instructions can be found in Appendix A.

We used the Amazon Elastic Compute Cloud, or EC2, web service to create and run a server in the cloud that hosted our middlebox. We chose the Amazon Linux AMI as a server template to launch our instance, and t2.micro as the instance type. We connected to our instance in the cloud using a public/private SSH key pair and ran the middlebox module on the server. We also had to modify the security group to prevent TCP traffic from being blocked. Like TinyCore, Amazon EC2 was a suitable choice for our cloud-based middlebox because it was simple to configure and the middlebox itself did not require much processing power or resources to function properly.

Chapter 4

Evaluation Plan

4.1 Metrics for Evaluation

We evaluated IF-TLS's effectiveness as an alternative protocol to TLS 1.3 in two different ways. First, we conducted a quantitative analysis of the protocol's performance; this involved timing how long the protocol components took to complete, as well as testing packet throughput. Then, we conducted a qualitative analysis of the protocol's security. We discuss each part in detail below.

4.1.1 IF-TLS Performance

The primary metric we used to assess IF-TLS's performance was delay. By introducing additional configuration and processing in both parts of IF-TLS (the initialization and data sending phases), the system will experience computational overhead and delay. Our goal was to measure the wall clock delay associated with each component of the protocol. To do this, we isolated portions of the protocol that involved a sending procedure and acknowledgement procedure; breaking up the components in this way allowed us to measure wall clock time on a single device. By

measuring time one a single entity, we avoid potential inconsistencies of the Network Time Protocol (NTP) on different devices. Table 4.1 shows the different protocol components we measured, as well as the start and end events that demarcate each part. These protocol components are the same components we described in Section 3.1.2.

The Client-Server initialization measure is the time between the client-server connection establishment and when an ACK is received from the server finalizing the creation of session keys. This is the time it takes the client to establish a TCP connection and IF-TLS connection, with a session key and mac key, with the server. Our next measurement is the Client-Manager initialization. This measures the amount of time it takes the client to send the session key to the manager and then for the manager to successfully share the key with all middleboxes. The Client initialization time is then the time it takes to fully initialize IF-TLS on the client, which includes the components covered in Client-Server initialization and Client-Manager initialization. The Manager-Middlebox initialization component measures the individual transmission times between just the manager to a middlebox. Finally, the round-trip time measures the time it takes for IF-TLS to send and receive data.

4.1.2 IF-TLS Security

We performed a qualitative analysis of the IF-TLS protocol by identifying strengths and weaknesses related to the security goals that IF-TLS aims to provide: confidentiality, integrity, and authentication. First, we explored a variety of threat models that could be applied to IF-TLS. Shevchenko [24] outlines thirteen different threat modeling methods that we considered; we include the discussion of DREAD in addition to the twelve threat models explicitly analyzed. Many of the models involve business/organizational objectives (PASTA, VAST, OCTAVE), detailed

Component	Start Event	End Event
Client-Server initialization	Client starts connection with server	Client receives ACK from server
Client-Manager initialization	Client starts connection with manager	Client receives ACK from manager
Client Initialization	Client starts connection with server	Client sends first message to server
Manager-Middlebox initialization	Manager sends session key to middlebox	Manager receives ACK from middlebox
Round-trip time (RTT)	Client sends a message to server	Client receives message ACK from server

Table 4.1: The components we used to measure delay.

scoring calculations (CVSS, Trike, Quantitative TMM), or creating attack personas (PnG, Security Cards, hTMM) that would be difficult to apply to our user-oriented, proof-of-concept project. We narrowed our search to the following frameworks: STRIDE, DREAD, LINDDUN, and Attack Trees. LINDDUN maps data flows and then applies a threat analysis. However, much of IF-TLS's data flow is done over TLS and assumed to be secure, so we determined that this was the wrong area to focus on. Attack Trees depict how attacks can be achieved from a goal (the root of the tree). Similar to LINDDUN, since IF-TLS relies on TLS and the security of devices themselves, we wanted to focus on the attacks and their threat to IF-TLS components; centering on the novel IF-TLS components would create thin trees and diminish the purpose of Attack Trees.

Finally, we identified STRIDE and DREAD as the most comprehensive and relevant threat models for our security analysis. STRIDE analyzes threats based on six mnemonic categories: (S)poofing, (T)ampering, (R)epudiation, (I)nformation Disclosure, (D)enial of Service, and (E)levation of Privilege. This allows us to analyze threats to IF-TLS based on the type of attack. Then, for each threat, we apply DREAD to assess their risk based on five criteria: (D)amage, (R)eproducibility,

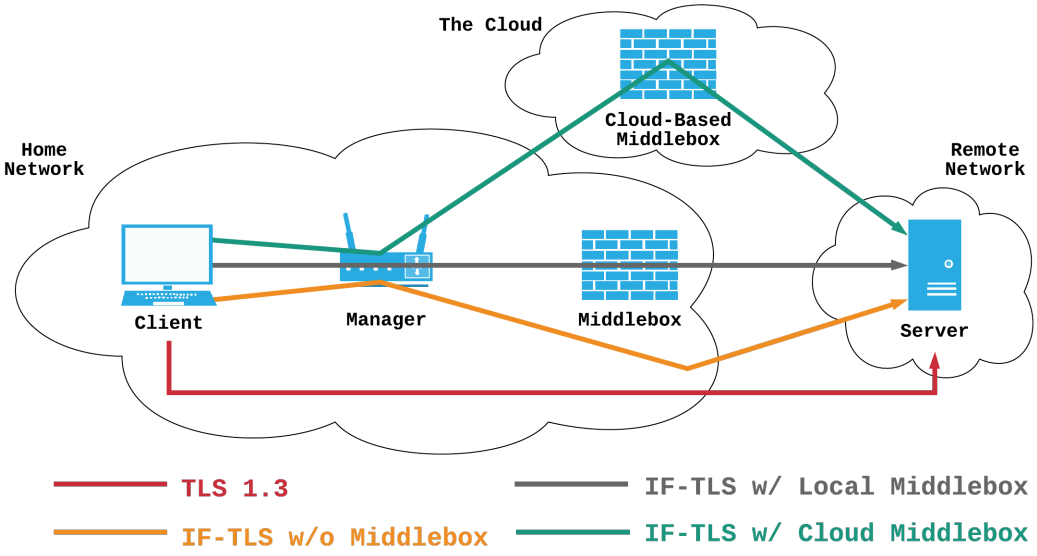

Figure 4.1: The four evaluation scenarios we used for performance testing.

(E)xploitability, (A)ffected users, and (D)iscoverability. DREAD provides the framework we need to determine the capabilities of attackers and the damage they can cause. We also identified potential countermeasures that could mitigate these vulnerabilities, as well as areas for future work.

4.2 Experimental Setup

We constructed four different evaluation scenarios (Figure 4.1). In the first scenario, we obtained performance data related to delay under a TLS 1.3 connection; these measurements provided a baseline against which we could compare measurements with the IF-TLS protocol. The second scenario involves an IF-TLS connection with no middlebox; this scenario allowed us to directly measure how much overhead the IF-TLS initialization procedure adds compared to TLS 1.3. In the third scenario, we added a middlebox residing in the local network and routed IF-TLS traffic through that middlebox. Finally, we moved the middlebox into the cloud and routed traffic

Capture Name (File Extension: pcap)	Capture Size (# packets)	Session Length (seconds)
Cap A	24	0.5
Cap B	1070	39.5
Cap C	2052	12.1

Table 4.2: The packet capture samples we used for measuring delay.

in a similar manner.

To obtain quantitative performance data for TLS 1.3, we added timing components to the client to measure how long the client-server and client-manager initialization took to complete. We also measured the round-trip time, or the time it took for the client to send a message to the server and receive it echoed back. The contents of the message corresponded to the number of the generated message ("1" for the first message, "2" for the second message, etc.). We collected 30 measurements of the initialization time.

To more accurately mimic how a real IoT device would transmit data, the client transmitted messages to the server at intervals from real IoT device pcap files. Vishwajeet Bhosale from Colorado State University allowed us to use a collection of IoT device packet traces; we chose to use 3 of these traces. We filtered each trace for bursts of TCP traffic because the data transmissions occurred close enough to each other in time for rapid testing, and because IF-TLS would run on top of this TCP traffic in practice. The samples we used for testing are outlined in Table 4.2; these samples encompass a variety of TLS sessions, from sessions that last less than a second and transmit a small burst of packets to sessions that last closer to a minute and transmit a much larger burst.

For each scenario, we first collected 30 measurements of the initialization components in Table 4.1. Then, we performed the data sending procedure three times with each of the packet captures and recorded the round-trip times.

Chapter 5

Results and Discussion

In this chapter, we present an overview of our findings and discuss the implications of these results. The first part of our findings relates to the IF-TLS performance results we obtained. In Chapter 3, we discussed how our original IF-TLS implementation differed from the current implementation. We obtained performance results for both implementations and found similar results, so we will only present results for the current implementation. We found that IF-TLS introduces minor additional delay to the initialization procedure compared to TLS 1.3. Additionally, we found that IF-TLS experiences similar round-trip time delays to TLS 1.3. The second part of our findings relates to our qualitative security analysis; we analyzed IF-TLS using the STRIDE and DREAD threat models and developed potential attack vectors for a variety of threat categories. We observed that compromising the access control list (ACL) would cause the most damage to a system utilizing the IF-TLS protocol, since the ACL governs data transit paths and specifies the middleboxes that have decryption capabilities.

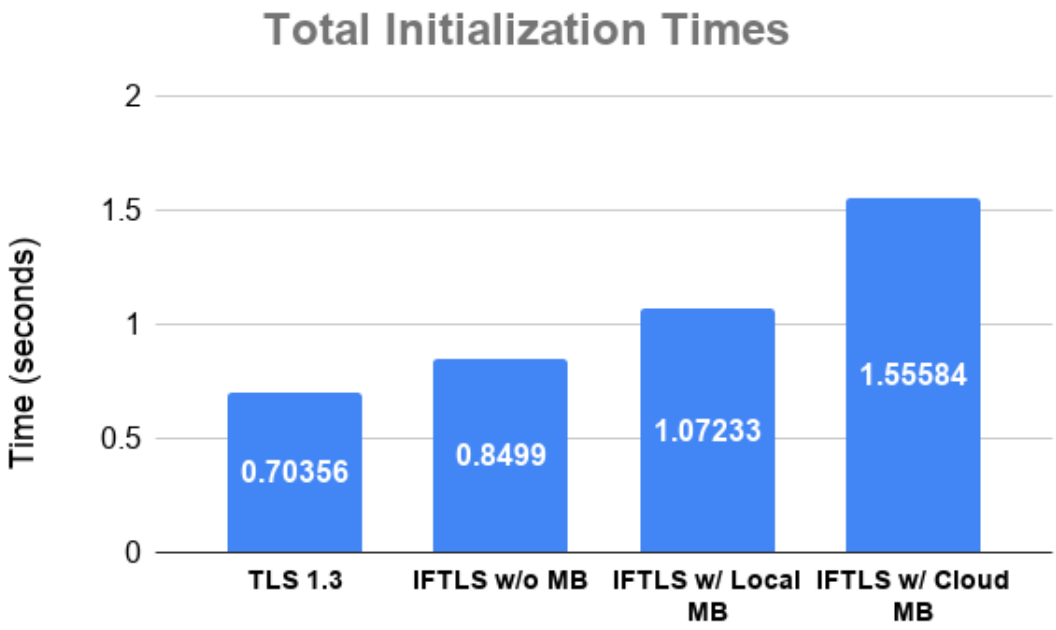

Figure 5.1: Average initialization times for the four scenarios we tested.

5.1 Performance Results

5.1.1 Total Initialization Time

Our first performance result highlights the total initialization overhead for IF-TLS; this includes the client-server, client-manager, and manager-middlebox initialization times. Figure 5.1 shows the average amount of time it took for the initialization procedure to complete under the four scenarios we tested: TLS 1.3, IF-TLS without a middlebox, IF-TLS with a local middlebox, and IF-TLS with a cloud middlebox. On average, IF-TLS without a middlebox adds 21 percent additional delay to the initialization procedure compared to TLS 1.3. The percentage increases for IF-TLS with a local middlebox and cloud-based middlebox are 52 and 121 percent, respectively.

Each scenario adds a layer of complexity to the initialization procedure, and thus

Middlebox Location	Cli-Manager Init. Time (s)	Manager-MB Init. Time (s)	% of Cli-Manager Init. Time
Local Network	0.25980	0.18227	70.2%
The Cloud	0.37665	0.29664	78.5%

Table 5.1: The manager-middlebox initialization as a sub-component of client-manager initialization.

the average initialization times increase compared to TLS 1.3. It is important to note that the initialization time for the cloud we present here does not factor in the time to establish TCP sockets for the purpose of forwarding traffic through the middlebox (Section 3.2). Therefore, the time here would represent the measurements obtained through the traditional use of forwarding rules. It is also important to note that the latter two measurements are representative of having one middlebox in the path to the server; adding more middleboxes to the chain of decryption would increase the initialization time, which we will discuss in Section 5.1.2.

The effect of the initialization procedure on the protocol's feasibility partially depends on how often IF-TLS sessions need to be re-established between the IoT device and the server. This frequency is synonymous with how long the session keys are valid before new keys are required or how often the ACL rules are modified. In Section 3.1.2, we mentioned that the user has control over how long their generated session keys are valid for, with an upper limit of 24 hours mutuated from the TLS 1.2 standard. As the time period for the keys' validity increases, the impact of initialization overhead decreases because this procedure is performed less often. In Section 5.2, we will discuss why a user may want to fine-tune this parameter to achieve a desirable balance between performance and security.

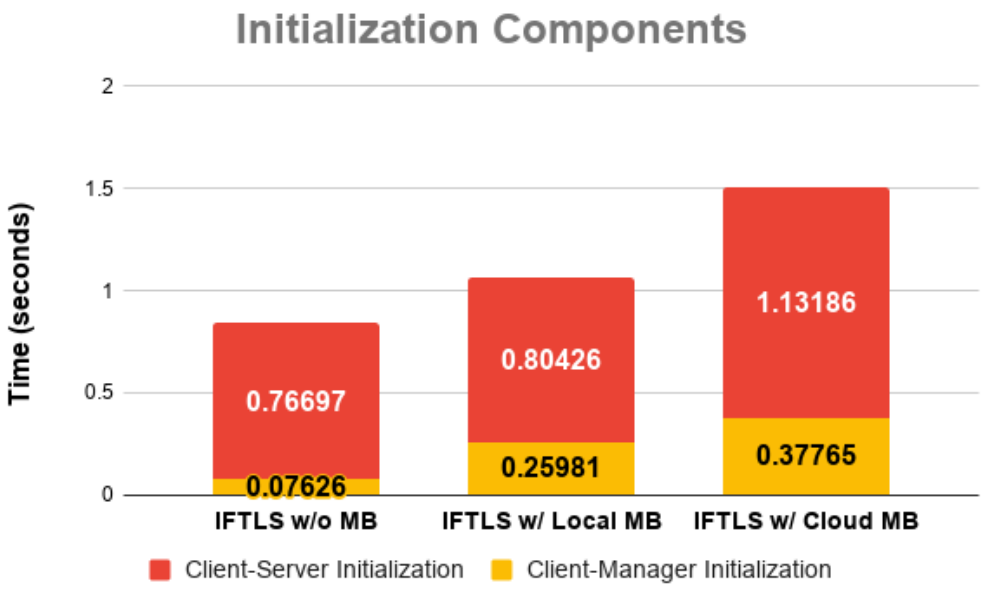

Figure 5.2: A comparison of the initialization components for the scenarios using IF-TLS.

5.1.2 IF-TLS Initialization Components

In addition to analyzing the total initialization time before data sending, we also measured the individual components of the initialization across the three IF-TLS scenarios. Figure 5.2 shows each component's proportion of the total initialization time for these scenarios; from left to right, the client-server initialization makes up 91, 76, and 75 percent of the initialization procedure. The absence of a middlebox in the first IF-TLS scenario implies that the client-manager initialization consists of a simple lookup in the access control list, followed by an acknowledgment to the client that all middleboxes (in this case, 0) have computed the session key.

As in Section 5.1.1, the latter two measurements are representative of having one middlebox in the path; thus, the client-manager proportion will increase by a roughly constant factor as more middleboxes are added to the path. Table 5.1 shows the quantitative factor we obtained by measuring the manager-middlebox

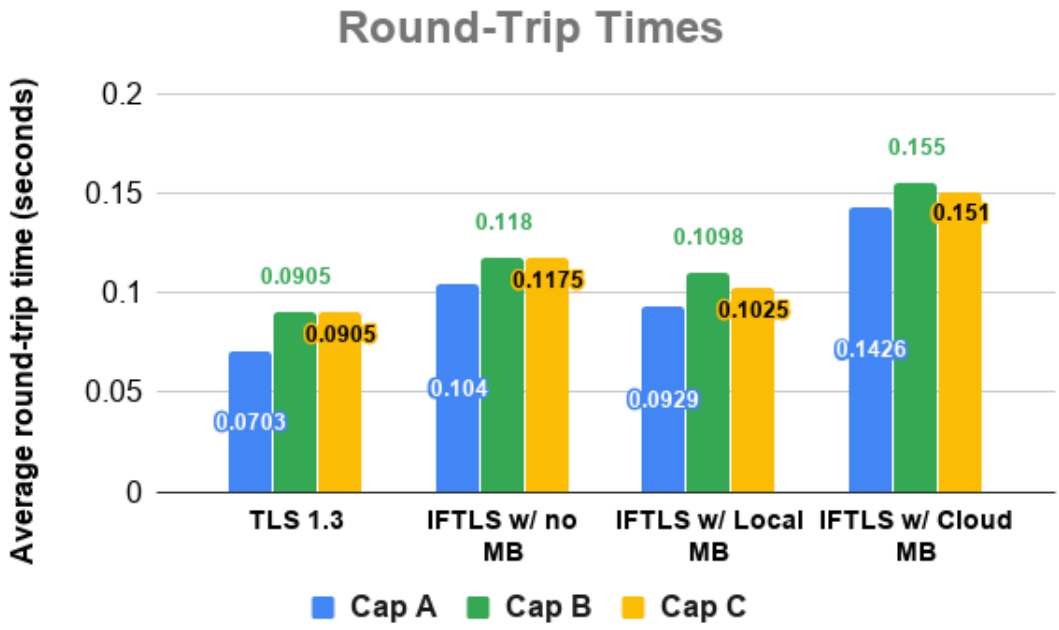

Figure 5.3: A comparison of the data sending round-trip times for the four scenarios we constructed using three capture files.

initialization time from Table 4.2. Because the manager-middlebox initialization is a sub-component of the client-manager initialization, we also show the former as a percentage of the latter. The times added for a middlebox in the local network and the cloud are roughly 0.2 and 0.3 seconds, respectively. The cloud factor may vary slightly depending on the physical location of the cloud server.

5.1.3 Round-Trip Time

Finally, we obtained round-trip time (RTT) data for each of the three packet captures from Section 4.2. Figure 5.3 shows a comparison of the average RTTs for the four scenarios. We also averaged the RTT increase for the three IF-TLS scenarios compared to TLS 1.3. From left to right, the RTT increased by 36, 22, and 80 percent. Interestingly, IF-TLS with a local middlebox produced slightly faster RTTs

than IF-TLS without a middlebox in the connection; this may be due to the fact that routing traffic through the local middlebox may be faster than routing directly to the server. It is also important to note that the RTT for the cloud scenario is an overestimation because we routed traffic through TCP sockets rather than through traditional forwarding rules.

Similar to Sections 5.1.1 and 5.1.2, the RTT will increase with more middleboxes in the connection. We assume that the middleboxes in the path do not hold packets before forwarding them to the next middlebox, so the only factor that contributes to the RTT increase would be the time to forward packets from one hop to the next, which includes middlebox analysis/processing. The increase will be negligible for a group of middleboxes that are all located in the local network because the next-hop time is negligible. For a group of cloud-based middleboxes, this increase will depend on where the middleboxes are located; it would take more time to forward packets between two middleboxes that are geographically distant. Thus, it is more difficult to estimate the RTT increase when the cloud is involved.

5.1.4 IF-TLS with Cloud-Based Middleboxes

One of our goals in our performance testing was to determine whether the additional delay introduced by IF-TLS would still allow for an acceptable user experience with an IoT device, even when placing a series of middleboxes in the cloud. One accepted metric for how long users are willing to wait for loading a web page is 2 seconds [25]; this is analogous to the IF-TLS initialization time, which does not exceed 2 seconds in the case of a single cloud-based middlebox. Furthermore, the majority of data transmitted using IoT devices is sensor or streaming data, and does not require frequent interaction, such as clicking between web pages, with the user. Other studies [26] [27] have shown that the frequency of interaction with IoT devices is

not high enough for a user to notice a few seconds of initialization overhead, which thus does not negatively impact the quality of the user experience. Based on the initialization and round-trip time data we obtained, we conclude that IF-TLS can still perform reasonably with a series of cloud-based middleboxes.

Our second goal was to determine the performance impacts of placing a middlebox in the local network versus placing it in the cloud. In the previous three sections, we observed that placing a middlebox in the cloud resulted in higher initialization and data sending times; thus, utilizing the cloud presents a performance trade-off between cloud-based benefits, such as scalability and reduced IT costs, and a faster connection. Depending on the use case for IF-TLS, the middlebox could remain in a local network or be placed in the cloud; a homeowner with a few IoT devices may forego the benefits of cloud computing for faster data sending and receiving, whereas a company making use of IoT devices can tolerate a slight decrease in IF-TLS performance to reap the benefits cloud computing offers.

5.2 Security Analysis

In this section, we present a security analysis of IF-TLS. This analysis includes potential threat models that span adversaries with varying capabilities and resources, as well as potential attacks on the system and their associated defenses. After exploring different threat modeling methods [24], as discussed in Section 4.1.2, we chose to use Microsoft's STRIDE threat modeling [28] accompanied by their DREAD method [29]; these models allow us to evaluate different threats and their associated risks. STRIDE analyzes threats based on six mnemonic categories: (S)poofing, (T)ampering, (R)epudiation, (I)nformation Disclosure, (D)enial of Service, and (E)levation of Privilege. Then, for each threat, we use DREAD to assess

attacks on five criteria: (D)amage, (R)eproducibility, (E)xploitability, (A)ffected users, and (D)iscoverability.

One of our core goals for IF-TLS is to preserve the security properties offered in TLS 1.3. This includes authentication, integrity, and confidentiality; we will discuss how these could be violated in IF-TLS. The security of IF-TLS depends partially on the security of its components: the IoT device, the manager/router, the middleboxes, and the server. These are briefly discussed as they relate to each threat, but we will not discuss them in detail as they are outside the control of our protocol. Finally, some of the defenses we mention will be further evaluated in Chapter 6.

5.2.1 (S)poofing

The first threat category we evaluate is identity spoofing. Spoofing is when an adversary attempts to conceal their identity or purposefully takes the identity of something else. In networking, this commonly means modifying the source IP of a packet so it cannot be easily traced back or appears to be from a trusted source. This manipulation violates the property of authentication since it is more difficult to differentiate between the actual source and a spoofed source.

In IF-TLS, it is possible for an attacker to use media access control (MAC) address spoofing to imitate an IoT device. The IF-TLS manager uses MAC addresses to identify the IoT device and look it up in the ACL. If an adversary impersonates an IoT device, their traffic would also be routed through the assigned middleboxes. However, since the IF-TLS manager never sends any confidential information to the device, the attacker has little incentive to spoof an IoT device if that is their only capability. The adversary would not be able to trick any device besides the IF-TLS manager without the IoT device's session and message authentication code

keys. Given that both media access control and message authentication code are abbreviated as MAC, we will only refer to the former as MAC address for the remainder of this section. The MAC address is assigned to identify devices, whereas a message authentication code is a hash that is created to check the integrity of a packet's payload.

Since the session and message authentication code keys are separately derived, they can be independently compromised; we will consider these different scenarios. If the attacker has the capability to attain a device's session key, then they will be able to decrypt any messages between the device and server. However, they will not be able to spoof messages since they do not have the message authentication code to create hashes; the server will reject messages with unverifiable hashes. The attacker, on the other hand, would only gain the ability to verify messages if they just have the message authentication code key. In order to fully imitate a device, they need both keys to get decryption/encryption and message authentication capabilities. However, it is difficult to obtain the keys. Either the client, server, client/server IF-TLS session, or a manager/middlebox's TLS session would need to be compromised to obtain the session key. We assume the TLS protocol is secure and free of vulnerabilities. The client and server's security depends on the entities themselves. An attacker would need to access the device's memory to extract the stored keys. If an attacker has compromised a device to this point, then our security goals have already been violated. The security of the client/server IF-TLS session depends on the ciphers used and the randomness of the pre-master secret. As detailed in Section 3.2, we use actively supported cipher suites, salts when generating keys, and a cryptographically secure random number generator. To obtain the message authentication code key, the client, server, or client/server IF-TLS session would need to be undermined. The security of these components is equivalent to

the explanation above. The attacker would also need to re-obtain the session and message authentication code keys every session in order to continue spoofing the device. While the damage is severe, the adversary needs advanced capabilities in order to carry out a full device spoofing attack.

Another MAC spoofing attack could be from the IoT device itself. If the IoT device changes the MAC of its own packets, it can bypass whatever rules were assigned for it in the ACL. By applying the DREAD model, we can assess this risk. The damage is significant since this enables a misbehaving client to circumvent IF-TLS. This evades the purpose of IF-TLS—to allow users to control their devices' data. MAC spoofing is also a very simple attack since the MAC address is unencrypted in the data link layer. However, this attack can be mitigated. The IF-TLS manager can implement finer device authentication so it does not rely solely on the MAC address. We will discuss this possibility in Chapter 6.

Finally, since the ACL relies on the IP of the middlebox, if a middblebox changes their IP, it is possible for another device to take the old and trusted IP. In this scenario, the other device would receive session keys and traffic because the IF-TLS manager believes it is still the assigned middlebox. This would continue until the user updates the IP in the ACL. If the new device is owned by an adversary, then they will be able to read all of the user's data from the designated IoT device. While this spoofing attack would be easier than compromising a device to obtain the session key, it can be mitigated by implementing middlebox authentication. This is also discussed in Chapter 6.

5.2.2 (T)ampering

Tampering encompasses a variety of attacks that are related to intentional modification of a product or system, especially modifications that would cause harm to

an end user. Tampering violates the integrity property of a system by making it difficult or impossible to discern between legitimate and illegitimate data or system processes.

In the context of IF-TLS, the most dangerous form of tampering is unauthorized modification of the access control list (ACL) on the manager. If an adversary were able to obtain write permission on the ACL, a simple attack may consist of wiping the ACL's contents, thus removing all middleboxes as points of IF-TLS communication. A more subtle attack would be to strategically modify the order or addresses of middlebox processing, such as re-routing a user's IoT traffic to the adversary's personal network. An adversary could also tamper with the IoT devices themselves and change the cipher suites used for an IF-TLS session, forcing servers that use IF-TLS to reject sessions initiated by the client.

Following the DREAD model, tampering attacks can pose grave effects on an IF-TLS session, because the ACL acts as the central point for access control in an IF-TLS session. Once the attacker seizes control of the ACL, they can add, delete, or modify routes easily since write permissions are enabled on the ACL. Tampering attacks are typically difficult to discover as well; it would take a diligent end user who checks the ACL frequently to verify that the routes have not been modified by an unauthorized entity. However, the attack is reproducible and exploitable only to the extent to which the manager itself is vulnerable. Since the manager is designed to run on the user's home router, the attacker would have to compromise the user's router to launch this type of attack. A router's overall security depends on a variety of factors, such as the router's age and the presence or absence of security holes; thus, it is difficult to indicate how much of a threat tampering poses to IF-TLS.

5.2.3 (R)epudiation

In information security, non-repudiation is the guarantee that someone cannot deny the validity of an action or a piece of data. Thus, repudiation would allow an adversary to commit a malicious action and successfully deny that they performed the action. Repudiation is closely related to tampering because an adversary can more easily tamper with a system, such as modifying the access control list, if the system does not have the non-repudiation property. In other words, the system cannot authenticate which user performed the action, so all changes made to the system would have to be either allowed or prohibited. Obtaining access and tampering with the ACL is also considered a repudiation attack because the adversary can now make changes to the ACL without needing to establish their identity.

Repudiation follows the DREAD analysis for tampering. Attacks related to repudiation are damaging because it cannot be proven whether or not a specific entity launched an attack. The security of the ACL on the manager, and thus the router, determines how reproducible and exploitable a repudiation attack is. Finally, because of the inability to associate an entity with an action, repudiation attacks cannot be discovered. As long as each access to the manager is authenticated and logged, non-repudiation holds.

5.2.4 (I)nformation Disclosure

Information disclosure encompasses intentional attacks on a system, such as a data breach, or unintentional system vulnerabilities, such as data leaks. These kinds of disclosures violate the security property of confidentiality because the data is no longer contained in a private session. Furthermore, information disclosures are a violation of the end user's privacy, a property that IF-TLS was designed to protect.

By virtue of enabling middleboxes to access traffic, IF-TLS introduces inherent risks related to information disclosure that do not exist when using TLS 1.3. First, the risk of information disclosure increases as the number of devices with access to system-specific data increases. Although we assumed in our protocol design that the middleboxes in the path of communication are trusted and authenticated, adding even one middlebox to the path increases the attack space that adversaries can exploit. As one example, an exceedingly liberal ACL configuration may grant a middlebox access to more IoT data streams than to which it needs access. The access control policies should be optimized to only include middleboxes in the communication that need access to unencrypted payloads; this would help minimize the extent to which device- and user-specific data becomes accessible over a network. Furthermore, if we assume middleboxes can misbehave or be compromised, sharing application-specific data now poses a harm to the user, since the middlebox can decrypt and analyze payloads for malicious purposes, such as tracking a user based on the data their IoT devices transmit.

The damage that information disclosure can cause depends on the amount and types of data that are exposed. Accessing a stream of encrypted data is much less useful than obtaining the pre-master that is used to generate the IF-TLS session key. A compromised middlebox, on the other hand, has access to the IF-TLS session key and decrypted payloads, and is free to share this information with malicious parties. Similar to repudiation, a disclosure attack is only reproducible and exploitable to the extent which the system component is vulnerable. For example, obtaining the session and MAC keys would require breaching the secure session established by the client and server during the IF-TLS initialization procedure. Disclosure attacks are typically difficult to discover, although explicitly breaching the database is easier to detect than a data leak that the end user is unaware of.

5.2.5 (D)enial of Service

Denial of service (DoS) attacks are when a service or entity is rendered unavailable. An adversary often accomplishes this by flooding a device with packets that occupy all of its resources. Then, there is none available for any real communication. In IF-TLS, we will focus on a DoS against the IF-TLS manager. While it is also possible to launch a DoS on a middlebox or server, those entities exist independently of IF-TLS and thus are outside our scope.

We will evaluate the risk of a DoS using DREAD. The level of damage is high since the IF-TLS manager routes all the traffic on a local network. In our implementation, we assume it is running on the local router itself. If an entity successfully disables the router, then all devices connected to it would be unusable. Nonetheless, using IF-TLS will not make a router more vulnerable to conventional DoS attacks than it already is. Since the IF-TLS manager only performs lightweight dictionary lookups to the ACL, it would take the same amount of traffic that would crash the router to DoS the IF-TLS manager.

5.2.6 (E)levation of Privilege

As the name implies, elevation of privilege refers to granting a party the authority to do something that that party should not have the ability to do. Elevating privileges in this manner violates the authorization security property, which would allow privileged users to perform a variety of actions that could infringe on the main user's security or privacy. Although protecting against elevation privilege is orthogonal to the goals of IF-TLS, we still consider the effects of such attacks on IF-TLS's functionality.

The attacks that are possible through elevation of privilege depend on the ac-

tions or information that other parties have been granted access to, as well as the intentions of the parties that have these privileges. Similar to tampering, a party could have the ability to read from and write to the access control list, and thus have control over the IoT device flows. The party could also instruct middleboxes to forward IoT traffic to unknown locations, such as to an attacker that is interested in learning about an end user based on their IoT device traffic patterns.

Similarly, the amount of damage a privilege elevation can do to IF-TLS depends on the privilege that was granted, and to whom the privilege was granted. In most situations, the end user has control over the parties that receive the increased privilege; thus, attacks in this category largely depend on the end user's knowledge of who should have the ability to perform specific actions and who should not.

Chapter 6

Future Work

In this chapter we present opportunities for future work associated with IF-TLS. The protocol is still in its infancy, and there are many areas to explore related to the IF-TLS performance and security measures we described in Chapter 5. Some of the ideas we describe here focus on the implementation details of IF-TLS that we were unable to implement, whereas other ideas focus on further improvements to IF-TLS.

6.1 Implementation Challenges

As we created our proof-of-concept implementation of IF-TLS, we encountered challenges that led to us being unable to fully implement certain features. The forwarding functionality of IF-TLS, described in Section 3.2, still needs to be expanded. With a middlebox in the cloud, we were only able to use a crude solution that directly connected the client to the middlebox and the middlebox to the server. Ideally, this traffic could be forwarded by the router and sustain multiple middleboxes on the path.

Also, as outlined in Section 3.1.3, an IF-TLS connection needs to be manually

reset by the user in order to apply changes to the ACL. Instead, whenever a modification is made to the ACL file, any devices that have new values should have their IF-TLS connections reset automatically by the IF-TLS manager to apply the changes immediately.

Additionally, authentication should be included for all entities in IF-TLS. The IF-TLS manager should incorporate IoT device and middlebox authentication. IoT device authentication would prevent misbehaving IoT devices from circumventing IF-TLS, as explained in Section 5.2.1. Middlebox authentication would prevent unauthorized entities from purposefully using a man-in-the-middle attack to receive IF-TLS traffic. It would also prevent data from being sent accidentally to formerly used middlebox IP addresses, as mentioned in Section 5.2.6, if a middlebox were to have its IP address changed. Finally, the client should authenticate the server it is communicating with. While we are sending the server's certificate as part of the IF-TLS handshake, we are not actively verifying it in our implementation.

6.2 Extensions to IF-TLS

Other implementation details are more accurately described as limitations of the IF-TLS protocol. Currently, our implementation of IF-TLS only routes traffic from the client to the server through the middlebox(es). We created this stipulation because we deemed that only the data from the IoT devices was relevant to IF-TLS's purpose of inspecting collected user data. However, routing rules could be added to redirect and inspect traffic from the server to the client as well. This may require additional configurations on the server to support routing through the middlebox(es). Also, allowing bi-directional traffic inspection may impact the performance results and security analysis we presented in Chapter 5.

As we discussed in Section 5.2, the access control list (ACL) is also the target of most threats and creates a single point of security failure for the system. If the ACL is compromised, the attacker can control IF-TLS as if they were the user. Decentralization is not as critical for a small home network because there are fewer connected devices in the system. However, in a larger setting such as a smart city or company office, measures should be incorporated to mitigate the effects of a single point of failure. One preventive measure is to de-centralize the ACL over an entire system. A distributed system could reduce the risk of tampering and elevation of privilege threats to IF-TLS, since the access control rules exist and can be cross-verified on multiple systems.

6.3 Additional Performance Measures

The primary performance metric we collected and analyzed for IF-TLS was delay. Measuring initialization and round-trip times allowed us to draw comparisons between IF-TLS and TLS 1.3 for a variety of scenarios, including the placement of a middlebox in the cloud. However, additional network performance measurements may also be useful to more accurately observe the overall quality of service IF-TLS provides. One potentially useful measurement is jitter, or the variation in time delay for data sent over a network [30]. IoT devices that transmit continuous streams of data, such as smart video cameras, may experience network disruptions in the presence of high jitter values, and thus decrease the quality of service to the user. Since jitter depends on the average spread of packet delays, an analysis of the round-trip delays we collected in our performance measurements can provide insight on how much jitter IF-TLS introduces on average.

6.4 Formal Methods for Security Verification

The last area of future work we consider is applying formal verification methods to IF-TLS. Formal methods aim to prove or disprove the correctness of cryptographic algorithms related to system properties or specifications. Previous efforts have used various tools to verify components of the TLS protocol. [31] provided libraries to formally verify C-written TLS packet processing applications using Coq, a theorem prover that uses the Gallina specification language. [32] utilized the Casper/FDR2 toolbox and model verification techniques to conduct a formal verification of the TLS Handshake protocol. [33] utilized the Tamarin prover, a security protocol verification tool, to perform automated analysis of revision 10 of the TLS 1.3 specification prior to the specification's deployment. Together, these methods provide a more detailed analysis of a security protocol; in some cases, this analysis can uncover vulnerabilities or implementation errors in a protocol before its deployment, thus decreasing the attack space that adversaries can exploit.

The same methods used to verify components of TLS can be applied to IF-TLS. The components of interest for verification in IF-TLS are the IF-TLS handshake, the sharing of session and MAC keys, and the transmission of data from the IoT device to the server. Each component involves one or more cryptography schemes, such as AES for generating the MAC key and RSA asymmetric encryption for the IF-TLS handshake. Thus, the implementation details for each of these components could be verified using any of the methods mentioned in the previous efforts above. Although formal methods are primarily used for proving the correctness of cryptographic algorithms, they can also provide insight into weak points for information leaks or breaches, since an incorrectly implemented algorithm may expose sensitive information. As we discussed in Section 5.2.4, it is critical that the session keys,

MAC keys, and access control list remain confidential in an IF-TLS session; proving that this property holds would minimize the risk of violating confidentiality.

Chapter 7

Conclusion

In this paper, we presented IF-TLS, a protocol that balances the security benefits that TLS offers and the privacy considerations that analysis by network-based middleboxes help to enforce. Delegating control to the end user regarding traffic processing allows the protocol to be parametrized in a way that meets each end user's needs. IF-TLS is designed particularly for users who have concerns over how much identifying information is sent from their smart devices; the protocol empowers users to make more informed decisions about which devices they elect to use. The concepts we introduce in our protocol, such as key sharing according to a set of access control policies, do not introduce unreasonable complexity or undermine the security properties of TLS 1.3.

One of our hopes with the IF-TLS protocol design is to find a medium between the goals of service providers and network security researchers. With most of the modern Internet using TLS 1.2, there are still opportunities to revise properties of TLS 1.3 before it becomes mainstream. End users deserve to know how and when their data is being collected, in the same way that they deserve reliable and secure service for the devices they use in their homes. Workarounds to the restrictions

TLS 1.3 imposes are not secure, and do not fix the fundamental issues related to end-to-end encryption. Our design aims to shed light on how these issues can be handled feasibly.

Bibliography

[1] R. van der Meulen. (2017) Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-20-31-percent-from-2016

[2] H. M. Moghaddam, G. Acar, B. Burgess, A. Mathur, D. Y. Huang, N. Feamster, E. W. Felten, P. Mittal, and A. Narayanan, "Watching you watch: The tracking ecosystem of over-the-top tv streaming devices," 2019.

[3] S. Fussell. (2019, February) The microphones that may be hidden in your home. [Online]. Available: https://www.theatlantic.com/technology/archive/2019/02/googles-home-security-devices-had-hidden-microphones/583387/

[4] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 213–226, Aug. 2015. [Online]. Available: http://doi.acm.org/10.1145/2829988.2787502

[5] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste, "Multi-context tls (mctls): Enabling secure in-network functionality in tls," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 199–212, 2015.

[6] V. Zakharevich and M. Rakhmanov, "Intercepting ssl and https traffic with mitmproxy and sslsplit," Apr 2016. [Online]. Available: https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/intercepting-ssl-and-https-traffic-with-mitmproxy-and-sslsplit/

[7] Fortinet. (2018) Threat landscape report q3 2018. [Online]. Available: https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/threat-report-q3-2018.pdf

[8] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," Internet Requests for Comments, RFC Editor, RFC 5246, August 2008. [Online]. Available: https://tools.ietf.org/rfc/rfc5246.txt

[9] "What happens in a tls handshake?" https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/, accessed: 2019-10-07.

[10] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, August 2018. [Online]. Available: https://tools.ietf.org/rfc/rfc8446.txt

[11] F. Andreasen, N. Cam-Winget, and E. Wang, "Tls 1.3 impact on network-based security," Working Draft, IETF Secretariat, Internet-Draft draft-camwinget-tls-use-cases-00, October 2017, http://www.ietf.org/internet-drafts/draft-camwinget-tls-use-cases-00.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-camwinget-tls-use-cases-00.txt

[12] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of https interception." in *NDSS*, 2017.

[13] M. Bierma, A. Brown, T. DeLano, T. M. Kroeger, and H. Poston, "Locally operated cooperative key sharing (locks)," in *2017 International Conference on Computing, Networking and Communications (ICNC)*, Jan 2017, pp. 356–362.

[14] M. Rouse, S. Shea, and M. Haughn. (2018) Iot devices (internet of things devices). [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/IoT-device

[15] N. Apthorpe, D. Reisman, and N. Feamster, "A smart home is no castle: Privacy vulnerabilities of encrypted iot traffic," *CoRR*, vol. abs/1705.06805, 2017. [Online]. Available: http://arxiv.org/abs/1705.06805

[16] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, "Information exposure from consumer iot devices," 2019.

[17] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 350–361, 2011.

[18] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 13–24, 2012.

[19] C. Lan, J. Sherry, R. A. Popa, S. Ratnasamy, and Z. Liu, "Embark: Securely outsourcing middleboxes to the cloud," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 255–273. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/lan

[20] Cisco. (2019) Software-defined networking. [Online]. Available: https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html

[21] C. R. Taylor and C. A. Shue, "Validating security protocols with cloud-based middleboxes," in *2016 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2016, pp. 261–269.

[22] C. R. Taylor, T. Guo, C. A. Shue, and M. E. Najd, "On the feasibility of cloud-based sdn controllers for residential networks," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2017, pp. 1–6.

[23] J. Wilson, R. S. Wahby, H. Corrigan-Gibbs, D. Boneh, P. Levis, and K. Winstein, "Trust but verify: Auditing the secure internet of things," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2017, pp. 464–474.

[24] N. Shevchenko. (2018, Dec) Threat modeling: 12 available methods. [Online]. Available: https://insights.sei.cmu.edu/sei_blog/2018/12/threat-modeling-12-available-methods.html

[25] F. F.-H. Nah, "A study on tolerable waiting time: how long are web users willing to wait?" *Behaviour & Information Technology*, vol. 23, no. 3, pp. 153–163, 2004. [Online]. Available: https://doi.org/10.1080/01449290410001669914

[26] C. Axel, G. Ravindra, and O. W. Tsang, "Towards characterizing users interaction with zoomable video," in *Proceedings of the 2010 ACM Workshop on Social, Adaptive and Personalized Multimedia Interaction and Access*, ser. SAPMIA 10. New York, NY, USA: Association for Computing Machinery, 2010, p. 2124. [Online]. Available: https://doi.org/10.1145/1878061.1878069

[27] N. Q. M. Khiem, G. Ravindra, and W. T. Ooi, "Towards understanding user tolerance to network latency in zoomable video streaming," in *Proceedings of the 19th ACM International Conference on Multimedia*, ser. MM 11. New York, NY, USA: Association for Computing Machinery, 2011, p. 977980. [Online]. Available: https://doi.org/10.1145/2072298.2071917

[28] A. Shostack. (2007, Sep) Stride chart. [Online]. Available: https://www.microsoft.com/security/blog/2007/09/11/stride-chart/

[29] Microsoft. (2018, June) Threat modeling for drivers. [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/drivers/driversecurity/threat-modeling-for-drivers#the-dread-approach-to-threat-assessment

[30] Viavi, "How to measure network performance," https://www.viavisolutions.com/en-us/how-measure-network-performance, accessed: 2019-10-18.

[31] R. Affeldt and N. Marti, "Towards formal verification of tls network packet processing written in c," in *Proceedings of the 7th Workshop on Programming Languages Meets Program Verification*, ser. PLPV 13. New York, NY, USA: Association for Computing Machinery, 2013, p. 3546. [Online]. Available: https://doi.org/10.1145/2428116.2428124

[32] M. Tobarra, D. Cazorla, F. Cuartero, and G. Díaz, "Formal verification of tls handshake and extensions for wireless networks," in *Proc. of IADIS International Conference on Applied Computing (AC06), San Sebastian, Spain, IADIS Press*, 2006, pp. 57–64.

[33] C. Cremers, M. Horvat, S. Scott, and T. v. d. Merwe, "Automated analysis and verification of tls 1.3: 0-rtt, resumption and delayed authentication," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 470–485.

[34] P. Singh, "Configure tiny core linux as dhcp server using udhcpd," https://iotbytes.wordpress.com/configure-dhcp-server-on-microcore-tiny-linux/, accessed: 2019-10-18.

[35] ——, "Configure tiny core linux as nat (p-nat) router using iptables," https://iotbytes.wordpress.com/configure-microcore-tiny-linux-as-nat-p-nat-router-using-iptables/, accessed: 2019-10-18.

[36] J. Turcotte and E. Zhou, "If-tls," https://github.com/edamamez/IF-TLS, 2020.

Appendix A: IF-TLS Setup Instructions

Configuring the TinyCore Virtual Machines

To mimic the behavior of a router in a network, we configured the management TinyCore VM to act as a NAT router so that the management system can observe traffic from the client and redirect it to a middlebox (or series of middleboxes). First, we followed the instructions from [34] to configure the VM as a DHCP server using udhcpd; unlike the tutorial, we used eth1 instead of eth0. Then, we configured the VM as a P-NAT router using [35].

Additionally, the network settings need to be modified in VirtualBox. Figure 7.1 below shows the network adapter settings that should be changed. The gateway VMs first adapter (eth0 interface) should be configured with NAT to enable communication with the host machine. The second adapter (eth1 interface) should be configured with the internal network to allow other local machine traffic to be routed through the gateway.

The other TinyCore virtual machines representing the client and local middlebox do not need to be configured as NAT routers, but should be assigned their own static IPs. The network settings in VirtualBox should also be modified as above to attach

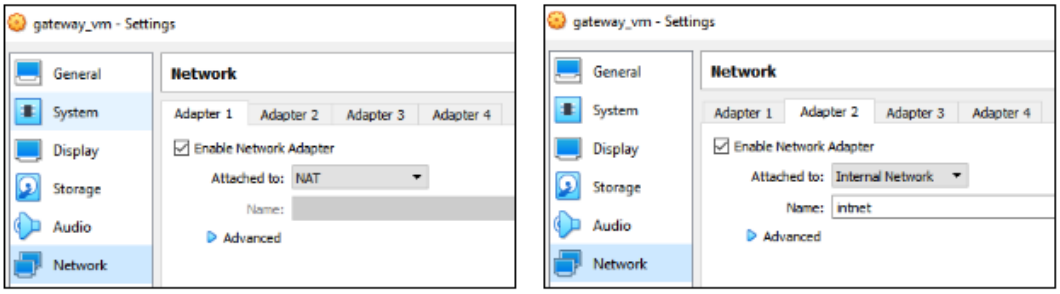

Figure 7.1: The VirtualBox settings we configured for our TinyCore VMs.

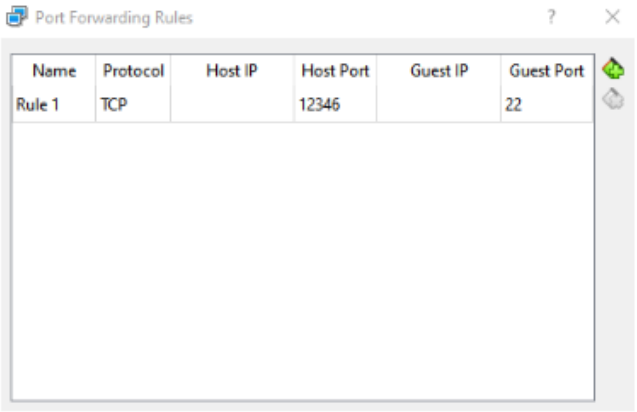

Figure 7.2: Adding a port forwarding rule on VirtualBox to allow for SSH.

one adapter to the internal network. Additionally, to enable SSH for the TinyCore machines, port forwarding needs to be enabled, which allows SSH to be used through an application such as PuTTY. The TinyCore VM has some trouble with scrolling and full-screening, so using SSH on the host machine is more convenient. In Figure 7.1 above, clicking the drop-down for Advanced under the NAT tab shows a Port Forwarding button. This will bring up a screen similar to Figure 7.2; adding a new rule simply requires pressing the green plus button in the top right part of the screen and filling in the rule. The guest port must be 22, but the host port can be any number (a number larger than 5000 is generally safe). Each VM use a different port.

Configuring the IF-TLS Repository

The IF-TLS repository [36] can be cloned or downloaded from GitHub and transferred to each of the virtual machines using the scp command, or through a file transfer utility such as WinSCP or FileZilla. The repository comes with a setup script called setup.sh that installs all the necessary Python libraries that the IF-TLS API relies on. This script must be run every time the TinyCore VM is restarted, since these libraries are not stored in the virtual machine at shutdown. Any files that should be saved, including the IF-TLS repository and the setup script, need to be backed up in /opt/.filetool.lst before shutting the machine down. The command *$ filetool.sh -b* will save the changes to filetool and the network configurations (note that this command may take a while to finish). Once the setup script finishes, utilizing the protocol simply requires changing into the source directory and running the script that corresponds to the machine (e.g. client.py for the client VM). Note: python3.6 and python3.6-dev can be installed using *$ tce-load -wi [package name].*