Early Draft: Breaking Client and Server Authentication through TLS Resumption across Hostnames

Erik Sy
University of Hamburg

ABSTRACT

KEYWORDS

ACM proceedings, LaTeX, text tagging

1 INTRODUCTION

2 BACKGROUND

In this section, we describe the TLS 1.3 0-RTT connection establishment and its known security limitations. Subsequently, we review the mechanism of TLS Client Certificate Authentication.

2.1 TLS 1.3 0-RTT Handshake

We begin this section by describing the protocol flow of the TLS 1.3 0-RTT connection establishment. Subsequently, we review known security limitations of this resumption handshake.

Protocol Flow. This TLS handshake allows a client to send encrypted application data without waiting for the server's response. To successfully establish TLS 1.3 0-RTT connections, the client and server exchanged prior to this handshake a symmetric pre-shared key (PSK). Figure 1 shows a schematic of this handshake where encrypted data are highlighted in grey. To begin with, the client sends its ClientHello message. Furthermore, the client signals its intention to directly send encrypted application data using the EarlyData extension. The application data are encrypted with a previously retrieved pre-shared key and a reference to this used key is provided in the PSK extension. Upon receiving the client's messages, the server retrieves the referenced pre-shared key and decrypts the presented application data. The Server responds with a ServerHello message containing the PSK and EarlyData extension to signal successful decryption of the received application data. In resumption handshakes, the peers authenticate each other based on their capability to decrypt/ encrypt data with the secret PSK. Thus, compared to an initial handshake the peers do not use certificates in the authentication process. To validate the integrity of the exchanged messages, the server provides hashes of the exchanged messages within its encrypted Finished message. Moreover, the server can start sending encrypted application data. Subsequently, the client indicates with the EndOfEarlyData message, that it stops using the PSK to send encrypted data. Then, it validates that the exchanged messages were not tampered with using the server's Finished message. Finally, it provides its own Finished messages to the server and connection establishment is completed.

Know Security Limitations. The TLS 1.3 0-RTT handshake does by design neither provide forward secrecy nor protect against replay attacks for application data encrypted under the PSK [1]. Forward secrecy describes the property of secure communication protocols, that compromised long-term key material does not lead to a compromise of the confidentiality of past sessions. In detail,

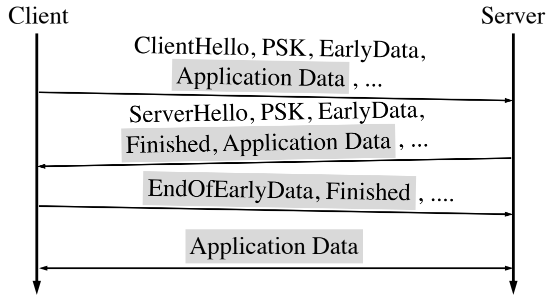
Client
Server
ClientHello, PSK, EarlyData,
Application Data , ...
ServerHello, PSK, EarlyData,
Finished, Application Data , ...
EndOfEarlyData, Finished , ...
Application Data

Figure 1: Schematic of the TLS 1.3 0-RTT handshake.

early application data sent by the client are only protected by the PSK without forward secrecy. However, the optional inclusion of a Diffie-Hellman key exchange within the 0-RTT handshake provides forward secrecy for application data encrypted after this key exchange completed. RFC 8446 [1] describes several replay attacks using 0-RTT data. The simplest attack assumes a network attacker that duplicates a flight of 0-RTT data and sends them to the same physical server. This attack can be prevented by ensuring that the server accepts 0-RTT data at most once. Other replay attacks take advantage of client retry behavior with the aim to provide multiple copies of the same application message to the server. Note, that this class of attacks cannot be prevented by TLS. Due to the limited security guarantees of 0-RTT data, this handshake mode must not be used by applications vulnerable to replay attacks [1]. As a result of this, TLS implementations deactivate the 0-RTT connection establishment by default.

2.2 TLS Client Certificate Authentication

This type of TLS authentication is used to establish a secure and trusted communication channel in both directions between a client and server. Thus, this handshake requires both peers to present proof for their identity during the mutual authentication. Use cases include IoT devices where the server authenticates the client before processing the provided requests/ data. Another use case presents mobile banking apps which provide each installed app a unique certificate for the purpose of client authentication.

The concept of Client Certificate Authentication (CCA) is supported by all TLS versions. Figure 2 provides a schematic of this authentication mechanism for the TLS 1.3 connection establishment. In this schematic encrypted messages of the handshake protocol are highlighted with grey color. The connection establishment is initiated by the ClientHello message. The server responds with its ServerHello message and its server certificate. To prove its ownership of the private key belonging to the presented certificate, the server uses this key to compute a fresh CertificateVerifiy message.

Subsequently, the server is requesting the client to authenticate itself indicated by the CertificateRequest message. Then, it computes the Finished message and optionally starts sending encrypted application data. Upon receiving these messages, the client validates the certificate presented by the server and the server's ownership of the corresponding private key using the obtained CertificateVerify message. Next, the client provides its own authentication certificate and uses the corresponding private key to generate a fresh CertificateVerifiy message. To detect modifications to the exchanged messages, the client validates the received Finished message and provides the server its own version of this message type. The server can now authenticate the client using the presented client certificate and CertificateVerifiy message. If this final validation is successful, the peers have established a secure connection with mutual authentication.

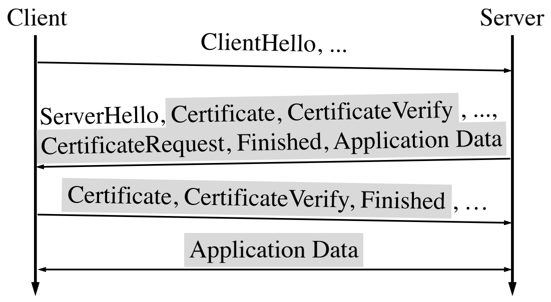
Client
Server
ClientHello, ...
ServerHello, Certificate, CertificateVerify , ...,
CertificateRequest, Finished, Application Data
Certificate, CertificateVerify, Finished , ...
Application Data

Figure 2: Client Certificate Authentication in TLS 1.3.

In the case of resumption handshakes, the TLS 1.3 specification explicitly does not permit the server to send a CertificateRequest message. Thus, the client has to authenticate its identity via a certificate only during a full TLS handshake. This practice assumes that a client that resumes a connection to the TLS server enforcing Client Certificate Authentication has been authenticated in this way during a prior connection. This assumption will be exploited in the following of this paper to circumvent TLS Client Certificate Authentication.

3 ATTACK ON TLS CLIENT CERTIFICATE AUTHENTICATION

In this section, we present a design flaw in TLS allowing an adversary without valid credentials to establish a TLS session with a server configured to conduct client certificate authentication. To begin with, we introduce our attacker model. Then, we describe the scenario leading to a successful circumvention of TLS Client Certificate Authentication.

3.1 Attacker Model

We consider the following adversary. Our adversary has no control over the attacked TLS server and cannot break the deployed cryptographic primitives. However, our attacker can learn whether different hostnames served via TLS share a cryptographic secret such as a session cache or Session Ticket Encryption Key (STEK) that allows to mutually resume TLS sessions across theses identified hostnames. To learn this information, the attacker may attempt to resume a TLS session previously established towards one hostname with a different hostname. Thus, our active attacker is capable to modify the TLS messages exchanged during the handshake. In our attack, we abstract the application layer on top of TLS. However, we assume that our attacker is able to read and write requests/responses using the deployed application protocol.

3.2 Attack Scenario

To be vulnerable against our attack, the victim TLS server must support TLS session resumption mechanism. Note, that about 96% of the TLS server hosting the Alexa Top Million Sites fulfills this requirement [3]. As another precondition, the attacked host must share its session cache or STEK across different hostnames. This practice of TLS secret state sharing is also very common on the web [4]. For example, CloudFlare uses a single STEK allowing the resumption of TLS sessions between more than 60 000 hostnames within the Alexa Top Million Sites [2].

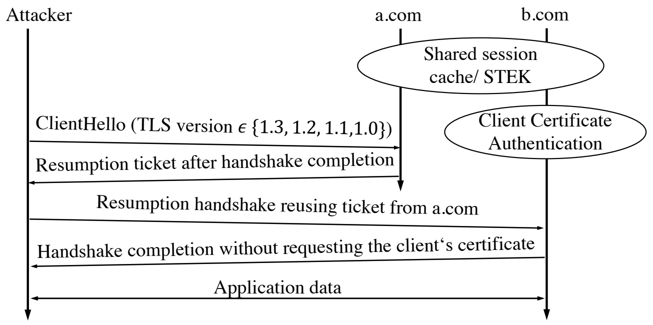
Attacker
a.com
b.com
Shared session cache/ STEK
Client Certificate Authentication
ClientHello (TLS version ϵ {1.3, 1.2, 1.1,1.0})
Resumption ticket after handshake completion
Resumption handshake reusing ticket from a.com
Handshake completion without requesting the client's certificate
Application data

Figure 3: Schematic of the attack on TLS CCA.

Figure 3 presents a schematic of the introduced attack on TLS Client Certificate Authentication. Here, the TLS secret state is shared between hostname *a.com* and *b.com*. Furthermore, *b.com* is configured to validate the identity of its clients via Client Certificate Authentication. The attacker starts by connecting to *a.com*, where it is legitimate for the attacker to establish a TLS session. During this session, the attacker receives a session resumption ticket. This ticket can be used to resume the session with *b.com* because of the secret state sharing between these hostnames. During this resumption handshake with *b.com*, the server does not repeat the process of client certificate authentication [1]. As a result, the attacker connects to *b.com* without presenting client credentials. In a real-world attack, the client can now use the established TLS session to retrieve sensitive material from the server or apply illegitimate modifications to the server's state. In total, this attack breaks the TLS design goal of client authentication.

4 CONFUSION ATTACK USING EARLY DATA

The TLS 1.3 0-RTT handshake provides the option to send early data before the connection establishment is completed. Thus, server- and client-side applications may already process exchanged data before it is validated that the exchanged messages are not tampered with. In this section, we present a confusion attack against this handshake design of TLS 1.3. We start by describing our attacker model and subsequently introduce our attack scenario.

4.1 Attacker Model

Our attacker has neither control over the client nor the TLS server. However, the attacker is able to modify the exchanged packets between client and server during transit. Moreover, we assume that the attacker cannot break the deployed cryptographic primitives. Furthermore, we suppose that the adversary is aware of a shared secret TLS state such as a session cache between different hostnames.

4.2 Attack Scenario

We consider the following scenario. Hostname *a.com* and *b.com* support TLS 1.3 0-RTT handshakes and share their secret TLS state with each other. Furthermore, we assume that these hostnames return different results for the same client request. For example, if the client provides two integer values to *a.com*, it will respond with the sum of these values. However, providing two integer values to *b.com* will return the difference between these values.

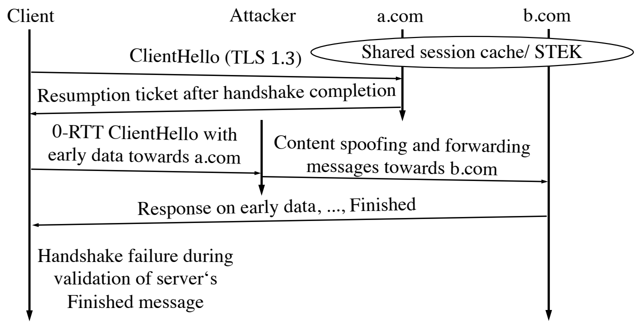
Client
Attacker
a.com
b.com
Shared session cache/ STEK
ClientHello (TLS 1.3)
Resumption ticket after handshake completion
0-RTT ClientHello with early data towards a.com
Content spoofing and forwarding messages towards b.com
Response on early data, ..., Finished
Handshake failure during validation of server‘s Finished message

Figure 4: Schematic of the confusion attack using early data.

Figure 4 provides a schematic of the proposed confusion attack using early data. Prior to the attack, the client retrieves resumption ticket from one of these hostnames. Then, we assume that the client wants hostname *a.com* to compute the sum of two presented integer values. The client attaches these values as early data within its TLS 1.3 0-RTT resumption request and sends them towards *a.com*. For some reason, the adversary wants the client to receive the difference of the provided values. Thus, it rewrites the *server name indication* from *a.com* to *b.com* and forwards these modified packets to *b.com*. Upon receiving these packets, *b.com* finds that it is able to resume the TLS session and can decrypt the attached early data. Subsequently, *b.com* computes the difference between the received values and sends them along with further handshake messages towards the client. The client forwards this first flight of data towards its application layer before the handshake is completed. Thus, the application layer receives the difference instead of the sum of the provided integers. Upon validating the server's Finished message, the client learns that the messages have been tampered with during transit because both peers observed different handshake messages. This results in a handshake failure. However, the client processed already data from the unauthenticated hostname *b.com* presenting a violation of the TLS design goals.

A variation of this attack could use the client's early data to modify the state of the TLS server. In this scenario, the proposed confusion attack would lead to a modification of the server state of the wrong hostname.

5 IMPACT AND LIMITATIONS

Impact: Replay protections across hostnames, if TLS state is shared?

Limitations: Application layer authentication such as HTTP Host Header, Cookies?

6 COUNTERMEASURES

Session resumption only to the same hostname of original session (secure default) by including hostname of prior session in session cache/ ticket-> Eventually breaks applications using resumption across hostnames as performance optimization

Deactivate TLS state sharing or session resumption if early data or Client Certificate Authentication is used -> Eventually performance impact

7 RELATED WORK

Selfie attack

HTTP Host Confusion Attack

8 CONCLUSION

REFERENCES

[1] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. https://doi.org/10.17487/RFC8446

[2] Drew Springall, Zakir Durumeric, and J. Alex Halderman. 2016. Measuring the Security Harm of TLS Crypto Shortcuts. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*. ACM, New York, NY, USA, 33–47. https://doi.org/10.1145/2987443.2987480

[3] Erik Sy, Christian Burkert, Hannes Federrath, and Mathias Fischer. 2018. Tracking Users Across the Web via TLS Session Resumption. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC '18)*. ACM, New York, NY, USA, 289–299. https://doi.org/10.1145/3274694.3274708

[4] Erik Sy, Moritz Moennich, Tobias Mueller, Hannes Federrath, and Mathias Fischer. 2019. Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames. *CoRR* abs/1902.02531 (2019). arXiv:1902.02531 http://arxiv.org/abs/1902.02531