PADERBORN UNIVERSITY

The University for the Information Society

Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
Research Group System Security

Bachelor's Thesis

Submitted to the System Security Research Group
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science

Large Scale Scanning of TLS Session Ticket Confusion

Tim Storm

Supervisors: Prof. Dr.-Ing. Juraj Somorovsky
Prof. Dr. Eric Bodden
Sven Hebrok M.Sc.

Date: May 14, 2023

Abstract

Session tickets are a resumption mechanism, which can speed up repeated TLS connections. To do so, information is stored client-side, encrypted with an additional symmetric key, which is separate from existing private keys. A server only has to store this key, making session tickets stateless for the server. If the key is shared between servers, a client can be misled into resuming a session with a different, less secure server. In this thesis, we design and implement a scan for detecting prerequisites to such an attack, by requesting and redeeming tickets for pair-wise servers. We find that 17,901 out of 22,127 scanned (virtual) hosts are potentially vulnerable to this attack because they share their keys and accept tickets issued for other domains. We discuss the difficulties of detecting such an attack and show that unfortunately, our approach does not scale to larger sample sizes.

Official Declaration

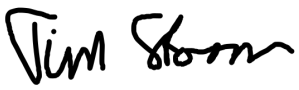
I hereby declare that I prepared this thesis entirely on my own and have not used outside sources without declaration in the text. Any concepts or quotations applicable to these sources are clearly attributed to them. This thesis has not been submitted in the same or a substantially similar version, not even in part, to any other authority for grading and has not been published elsewhere.

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

05/14/23

Date



Tim Storm

Contents

1 Introduction

Transport Layer Security (TLS) [1, 2] is a widely used protocol that provides confidentiality, integrity, and authenticity to applications on the internet. Over time there have been different versions of the TLS protocol, TLS 1.2 [1] and 1.3 [2] are most used today. The most common usage of TLS is secure web browsing via HTTP over TLS (HTTPS). Before traffic can be encrypted, however, a handshake must be performed in which the necessary parameters for a TLS session are negotiated. This includes a key exchange, which is authenticated using public key certificates. The handshake introduces an overhead of up to two roundtrip times. With high latency, this can be a noticeable delay in addition to the time-intensive public key cryptography.

Session resumption can be used to abbreviate the handshake between two parties if they have already performed a handshake recently. The key exchange and the certificate validation in particular are cut. Instead, both partners determine the master secret for the new session from the previous master secret. Up until TLS 1.2, the master secret was simply reused, since TLS 1.3 a new secret is derived from the previous secret.

In TLS 1.2, this eliminates a full roundtrip for subsequent handshakes, effectively cutting the time until the first application data can be received in half. TLS 1.3 has refined the handshake protocol and requires a single round trip in both the regular and resumed handshake, making it significantly faster than TLS 1.2. 1-RTT mode also benefits from session resumption, since public key cryptography can be avoided. There is even 0-RTT mode based on session resumption, which, when resuming a session, allows a client to send so-called “early data” together with the handshake messages themselves. This is inherently vulnerable to replay attacks but achieves even lower latency.

Session tickets are a session resumption mechanism, which is stateless for a server: After completing an initial handshake, the server encrypts any state relevant to the session with a dedicated session ticket encryption key (STEK). It then sends this ticket to the client. In a later handshake, the client presents their session ticket, which the server attempts to decrypt with its STEK. If the decryption is successful, they can fully recover the session from the ticket and resume the session without storing any information server-side. As long as the STEK is only ever available to the original server, the client can be sure that they are still talking to the same server even after resumption [3, 2].

However, if two servers share their STEK, a session negotiated with one server could be decrypted and then resumed by another. This presents a possible attack in which an attacker reroutes traffic meant for one server to another server, which shares the STEK, but is less trustworthy (for example by hosting user-submitted content) that then resumes the session. This would look like a regular session resumption to the affected client, potentially misleading them into trusting data supplied by that other server. The server itself would not detect that they were not the intended recipient. This kind of attack can be referred to as *session ticket confusion*. Delignat-Lavaud and Bhargavan first identified the vulnerability in 2015 and proved that it was feasible in practice [4].

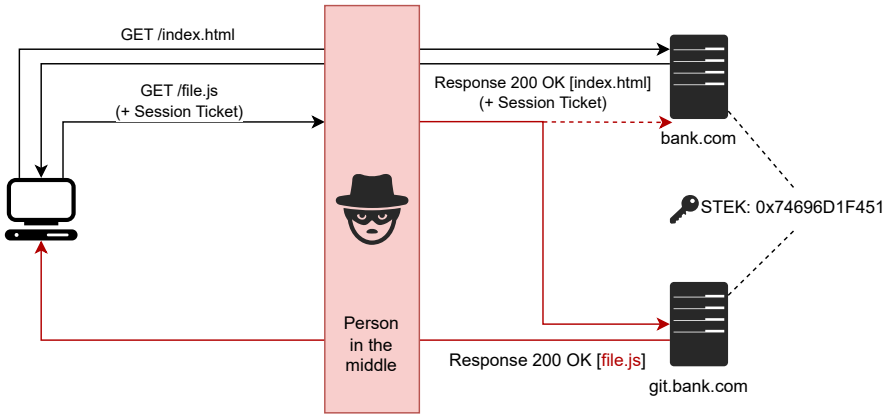


Figure 1.1: The basic idea behind session ticket confusion. Some time after an initial request, a client sends another request to the same server using session resumption but is redirected to a second server with the same STEK. The second server accepts and faithfully answers the request with a malicious file planted by the attacker. The client receives the file and processes it as if it was served by the original server.

It is unlikely that servers would share a STEK purely by chance. Therefore it is interesting whether related servers, for example, servers maintained by the same administrator or on similar domains, share their STEK. The aforementioned different "servers" do not have to be physically distinct: Through *virtual hosting* multiple websites may be hosted on the same machine, so it might be possible to accidentally share STEKs through misconfiguration. To distinguish between virtual servers, TLS offers the SNI extension, which explicitly includes the target domain in the client's first message to the server. This, depending on the implementation, may prevent the success of a host confusion attack, since a server could recognize the mismatch between the domain of the original website and its own.

1.1 Current State of Research

The problems presented by session resumption mechanisms are well explored: They compromise Perfect Forward Secrecy [5] and can be used to track users across multiple sessions [6]. The introduction of TLS 1.3 has overhauled session tickets and addressed some of the problems, but also included even riskier schemes such as the 0-RTT mode mentioned earlier.

Despite its known issues, industry [7] keeps on using session resumption purely for its performance benefits. Current research even proposes extending session ticket reuse across different applications connected to the same hostname [8] or even across hostnames [9].

Rather than being vulnerable on a protocol level, there appears to be a widespread misuse on an implementation and deployment level. The session ticket confusion attack above, as described by Delignat-Lavaud and Bhargavan [4], could be prevented by correctly implementing SNI. In practice, implementations seem to be inconsistent and lenient compared to the specification [10, 2].

A 2016 study conducted by Springall, Durumeric, and Halderman [11] revealed high exposure to attacks based on the reuse of cryptographic state, including improper rotation and sharing of STEKs. For this, they used the *key_name* field of a session ticket, queried over a six-hour window. They then grouped sites, which shared their identifier at any point and found that a large number of websites appeared to share their STEKs.

1.2 Contributions

While the *key_name* field used by Springall, Durumeric, and Halderman in [11] is a strong indicator for STEK sharing, it is not definitive: Our work improves upon this by practically verifying whether STEKs are being reused. Furthermore, we focus on analyzing the practical exploitability of STEK reuse via session ticket confusion. We implement a scan, which tries to authenticate with a session ticket to determine whether two servers share their STEK. Our results support the claims made by Springall, Durumeric, and Halderman; We were able to find a surprising number of servers, which shared their STEKs. Most of the potentially vulnerable servers are situated in Content Delivery Network (CDN) and appear to be vulnerable to practical session ticket confusion. Unfortunately, we found that our approach does not scale well to large sample sizes, which restricted our analysis to a modest 22,000 servers. We still included a less thorough scan of another 55,000 servers, which supports our results.

1.3 Organization of the Thesis

This thesis is divided into 6 chapters. In Chapter 2, we provide the necessary background for session tickets, STEK reuse and the resulting session ticket confusion attack. Based on this knowledge we then derive certain design goals for our scan, which we discuss in Chapter 3. In Chapter 4, we outline how our implementation addresses these goals. Using our implementation, we collected data for several thousand hosts. We show and discuss these results in Chapter 5. Finally, we summarize our results and outline possible future work in Chapter 6.

2 Background

In this chapter, we introduce all necessary background for the session ticket confusion attack. This includes what session tickets are, how they are used in a TLS handshake and the security considerations that apply. We will use this later to motivate design decisions for our scanning application. We also introduce TLS-Attacker, the library that we used to implement our application.

2.1 Transport Layer Security

TLS [2, 1] is a widely used cryptographic protocol used to secure communication over the internet by providing encryption, authentication and integrity to an application. It is independent of the communication on the application layer and can be used for a variety of protocols, but one of its most common usages is HTTPS. HTTPS allows accessing and interacting with websites on the internet securely.

2.1.1 TLS Handshake

To secure any interaction using TLS, the communicating parties perform a handshaking phase in which the cryptographic parameters, such as the cipher to be used, and keys needed for encryption are negotiated. Depending on the interaction either of the parties can authenticate themselves using a public key certificate during this phase.

Once the handshake has been completed, both parties can then exchange encrypted traffic based on the previously negotiated parameters.

In this work, we consider the setting of web browsing, where an anonymous client connects to an authenticated web server. Here the handshake aims to accomplish two important goals:

1. After the handshake finishes both parties share a set of symmetric keys, which they can use to securely communicate with each other.
2. After the handshake finishes, the client is sure that they are communicating with the right server.

A typical TLS 1.2 handshake, as shown in Figure 2.1, begins with the client sending a *ClientHello* message to the server in which they list, amongst others, the different key exchange methods and ciphers they support. The server responds with a *ServerHello* message in which it chooses from the options suggested by the client. Together with the *ServerHello* the server sends a public key certificate to be used as part of the key exchange.

There are different key exchange schemes based on RSA or Diffie-Hellman. For our purposes, the differences between the two are not important[1]. Depending on the key exchange, the certificate can be used to encrypt a key (RSA), as part of the key (Diffie-Hellman) or to sign part of a key (Diffie-Hellman Ephemeral). With every method, the client and server negotiate a shared secret called the Pre-Master secret (PMS) from which they ultimately derive the symmetric keys required for further communication. By verifying all signatures and the certificate chain of the server's certificate itself, a client confirms the authenticity of the received messages. After exchanging keys, the *Server-* and *ClientFinished* messages confirm that no messages were added to the handshake or otherwise modified by a third party.

This handshake always requires two roundtrips from the client to the server and back. It is only then that both parties can use the negotiated keys to safely communicate.

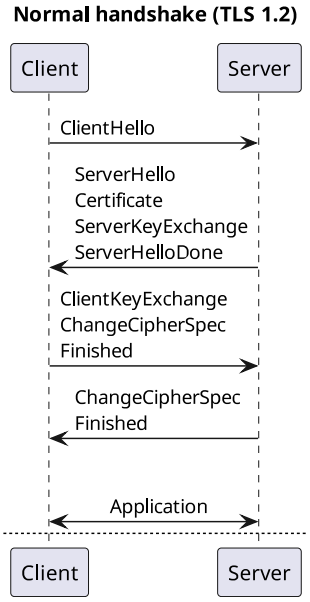


Figure 2.1: Full handshake using TLS 1.2

[1]Our illustrations will depict an ephemeral Diffie-Hellman key exchange for consistency.

Differences in TLS 1.3 TLS 1.3 [2] has overhauled the TLS protocol, including the handshake. The RSA key exchange has been deprecated and Diffie-Hellman key exchange is now used for every connection. This means that the key exchange can be performed immediately in the *ClientHello* and *ServerHello* messages. But since the certificate is only sent after the *ServerHello* message, the server has to authenticate retrospectively with the newly introduced *CertificateVerify* message. In Figure 2.2 we can see the TLS 1.3 equivalent of the handshake shown in Figure 2.1. In conclusion, the improved handshake protocol in TLS 1.3 is quite similar to its predecessor and achieves the same aforementioned objectives, but manages to do so in a single roundtrip.

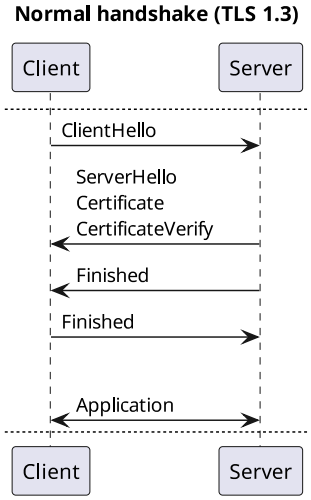


Figure 2.2: Full handshake using TLS 1.3

2.1.2 SNI

SNI is an extension to the TLS handshake, which allows a client to include the name of the server it is trying to reach, in the handshake itself [10]. This is useful in the context of "virtual hosting", where multiple distinct web services may be hosted on the same physical machine. A server can then use the domain name from the extension to handle the request appropriately, for example by serving a certificate responsible for the requested domain.

When faced with an unknown server name, a server is recommended to either ignore the mismatch (and let the client resolve it on the application layer) or abort the handshake completely.

A majority of TLS connections use SNI [12] and as of TLS 1.3, the extension is mandatory to implement [2].

In this work, we denote the domain name communicated via SNI to reach a host X as “the SNI for X”.

2.1.3 Session Resumption using Session Tickets

Session tickets are a form of session resumption mechanism, introduced to TLS 1.2 in RFC5077 [3]. When a client handshakes with a server, they can request a session ticket from the server. On the next handshake, the client can then include said ticket in the *ClientHello* message to abbreviate and speed up the handshake. This can come at the cost of security: Depending on the protocol version and implementation, attacks can range from decrypting singular sessions to fully impersonating servers during session resumption [5, 13].

To use session resumption, a client includes the empty *SessionTicket* extension in their *ClientHello* message during the initial handshake. If the server wishes to issue a ticket, it encrypts all cryptographic state relevant to the session using the STEK and passes it to the client in the *NewSessionTicket* message. This interaction is shown in Figure 2.3. The encrypted ticket contains everything the server needs to resume the current session later, most importantly the negotiated master secret. The ticket is stored *client-side* together with the respective parameters required by the client to resume the session later. Notably, the server issuing the ticket does not store any data after ending the initial session, making this *stateless* for the server.

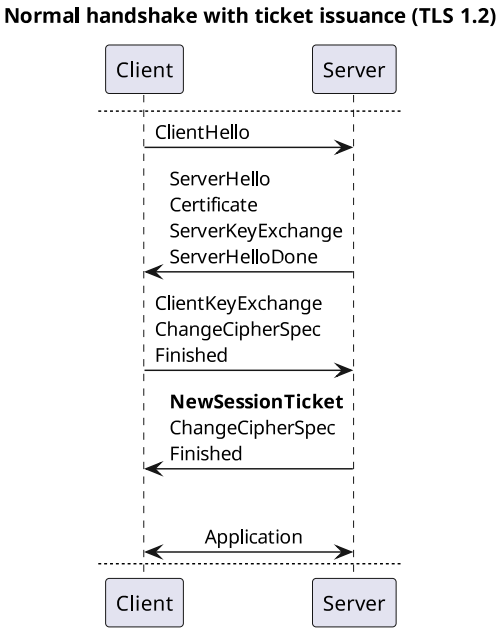


Figure 2.3: Ticket issuance in TLS 1.2

The next time the client wishes to begin a connection with the same server, the client includes the ticket as an extension to the *ClientHello*. When the server receives the *ClientHello* with the session ticket, it will attempt to decrypt the ticket using the STEK. Depending on the circumstances (more on that in Subsection 2.1.3.3) the server can either choose to reject the ticket (and fall back to the full handshake as seen in Figure 2.1) or honor the ticket, in which case they perform an abbreviated version of the handshake (see Figure 2.4).

The abbreviated handshake is significantly faster: Since the client and server already share a secret (the master secret from the previous connection, encrypted in the ticket), they do not need to perform a key exchange again. The server has already authenticated itself to the client in the prior connection, so it does not have to send its certificate again. This means that both parties can proceed to the *ChangeCipherSpec* message right after their respective *Hellos* and then start exchanging application data immediately. This not only saves a full roundtrip of latency but also saves CPU time, because it gets rid of expensive calculations for the key exchange and the verification of the certificate chain. In practice, session tickets have been shown to reduce the total time for a handshake by up to 50% and CPU time by up to 95% [7].

The additional speed has risks associated with it, however: A server's STEK allows an attacker to decrypt any session ticket and therefore the master secret for a session. This makes the STEK a long-term secret, which (when compromised) may be used to decrypt sessions recorded in the past. This violates "Perfect Forward Secrecy", even with an otherwise perfect forward secure key exchange[2]. It is crucial for both parties that the STEK and the ticket contents themselves remain a secret: From the server's point of view, any client with an encrypted ticket and the master secret contained within can impersonate the identity of the original client. From the client's point of view, any server capable of decrypting an encrypted ticket can impersonate the original server. To prevent an attacker from decrypting existing tickets or maliciously creating their own, session tickets are protected using strong encryption and integrity mechanisms [3].

Essentially, session tickets are a speed-security tradeoff: A server can speed up sessions with previous clients by reusing previous secrets, without storing any additional session information. Because session tickets tie a server's identity and the security of all sessions conducted with it to the STEK itself, the STEK becomes an additional secret to keep safe. Regularly changing ("rotating") the STEK and invalidating prior tickets can limit the impact of an exposed STEK.

[2]Perfect Forward Secrecy is the property that "compromise of long-term keys does not compromise past session keys"[14]. This is highly desirable and it is recommended to use perfect forward secure cipher suites in TLS by the German government [15].

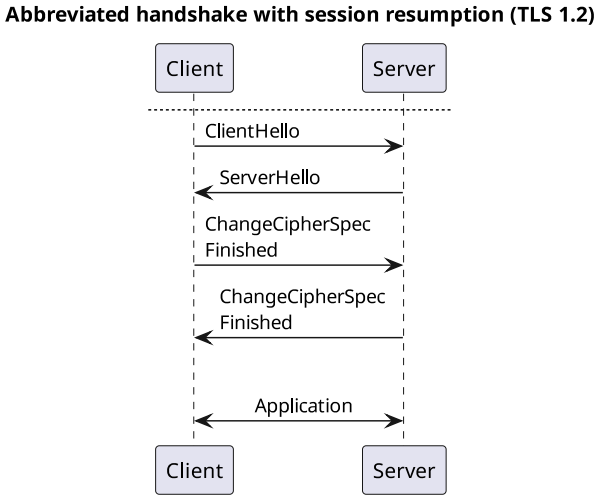


Figure 2.4: Abbreviated handshake in TLS 1.2

2.1.3.1 Session resumption using PSKs

TLS 1.3 has deprecated session tickets in favor of the new Pre-Shared Keys (PSKs) mechanism: Instead of negotiating a secret for the session, a secret can be shared out-of-band or derived from a previous session, which is then used for communication. There even exists a combination of (EC)DHE and PSKs, which preserves "Perfect Forward Secrecy".

PSKs address many of the aforementioned problems with session tickets, but we omit any further details here: For our purposes, PSKs *are* session tickets. Whenever we refer to "session tickets", we actually mean "TLS 1.2 session tickets or TLS 1.3 PSKs" (unless explicitly stated otherwise).

Note that PSKs are formalized as a key "exchange" method. While the resulting handshakes may look nearly identical, the underlying implementation may differ significantly between session tickets and PSKs. Therefore, *implementation-specific* behavior does not necessarily apply to both [2]. We outline what this means for us in Subsection 2.2.1.

2.1.3.2 Recommended Ticket Construction

Servers are only ever expected to receive and understand their own session tickets, therefore their format could be custom for every server implementation. Nonetheless, RFC 5077 recommends a structure for TLS 1.2 session tickets, shown in Figure 2.5. While we omit the cryptographic details for simplicity, we would like to point out the *key_name* field: The RFC states that it "serves to identify a particular set of

key_name [16]	IV [16]	encrypted_state [0..$2^{16}-1$]	MAC [32]

Figure 2.5: Structure of ticket as recommended by RFC 5077, byte size in brackets

keys used to protect the ticket" and that it "should be randomly generated to avoid collisions between servers" [3]. Its purpose is to quickly identify tickets that will not be accepted, for example, foreign or expired tickets.

Previous work [16] has shown that eight out of nine TLS libraries implemented a *key_name* of varying length at the beginning of their tickets. In practice, this meant that 98.7% of the hosts supporting session tickets implemented a *key_name* of 16 bytes. We will consider the *key_name* field to be an important indicator that a ticket was issued using a certain STEK.

TLS 1.3 The TLS 1.3 specification [2] does not include any recommendation regarding session ticket structure, but the aforementioned study concluded that most implementations follow the structure recommended for TLS 1.2 in RFC 5077 [16].

2.1.3.3 SNI in Session Resumptions

SNI has a special role in session resumptions, as it is the only TLS extension, which directly affects whether sessions may be resumed [10]. In particular, "A server that implements this extension MUST NOT accept the request to resume the session if the server_name extension contains a different name". Here "MUST NOT" means "an absolute prohibition of the specification" [17].

However, the specification for TLS 1.3 notes that "in reality the implementations were not consistent on which [...] supplied SNI values they would use", i.e. whether they would use the SNI value implicitly associated with the session (see Subsection 2.1.2) or the value explicitly provided within the *ClientHello*. Therefore, starting with TLS 1.3 "the SNI value is always explicitly specified in the resumption handshake" [2]. This makes it especially interesting to analyze how SNI affects the resumption handshake and whether this behavior is consistent between protocol versions.

2.2 Session Ticket Confusion Attack

Session ticket confusion is a type of *virtual host confusion attack*, first introduced by Delignat-Lavaud and Bhargavan in 2015. A virtual host confusion is "when any request is routed to a virtual host that wasn't intended to serve the domain of the request, without causing any authentication failure on an honest client" [4] (cf. Subsection 2.1.2). Session ticket confusion is possible when a modern web server (acting as a HTTP multiplexer) incorrectly handles an incoming request. Because session resumptions do not (re-)validate certificates, a client can be tricked into a session with a different host this way.

In the worst case, this could mean that a resumption request sent to website $\mathcal{A} = (\texttt{IP}_A, \texttt{domain}_A)$ can be rerouted to another website $\mathcal{B} = (\texttt{IP}_B, \texttt{domain}_B)$ by a man-in-the-middle, where $\mathcal{B}$ shares its STEK with $\mathcal{A}$. Because $\mathcal{B}$ can decrypt the session ticket, the client would then establish a session with $\mathcal{B}$. The client would believe the session to be authenticated under $\mathcal{A}$'s certificate from the original handshake. That certificate does not have to cover $\mathcal{B}$: Other than sharing a STEK, $\mathcal{A}$ and $\mathcal{B}$ do not have to be related in any way.

This only works if $\mathcal{B}$ routes the request to an unintended virtual host: Theoretically, $\mathcal{B}$ could notice that the SNI contained in the *ClientHello* (i.e. $\texttt{domain}_A$), does not exist on the target device. Similarly, the ticket itself could indicate the domain of the original request. In practice, Delignat-Lavaud and Bhargavan did find a pair of servers where a TLS session with one "high-trust" server could be continued with a different "low-trust" server because they shared a session cache[3]. This means that a client can be tricked so that it believes to be communicating with an intended "high-trust" server when in reality its requests are answered by a "low-trust" server [4].

This can but does not have to, be an exploitable vulnerability (more on that in Subsection 2.2.2)

2.2.1 Differences in TLS 1.3

There are no fundamental differences in TLS 1.3's design that prevent the same attack from happening with PSKs. However, as mentioned in Subsection 2.1.3.1, PSKs are a separate mechanism from TLS 1.2 session tickets. If two servers are vulnerable to a TLS 1.2 session ticket confusion, this does not imply that the same applies to PSKs and vice-versa. Theoretically, tickets usable on both versions are possible, in practice no servers allow switching between the two for resumption [16].

[3] Session caches are used by a different session resumption mechanism, called "session IDs"[1]

2.2.2 Attacker Model

Our attacker model is based on the assumptions made by Delignat-Lavaud and Bhargavan [4].

Attack Scenario A client has connected to server $\mathcal{A}$ in the past and has been issued a session ticket. There exists a separate server $\mathcal{B}$, which hosts different content but shares its STEK with $\mathcal{A}$. Both $\mathcal{A}$ and $\mathcal{B}$ are not compromised by the attacker. The client now attempts to resume its session with server $\mathcal{A}$. The attacker observes this attempt and intercepts it.

Attacker Capabilities The attacker can redirect the corresponding TCP traffic to $\mathcal{B}$, but they cannot change the redirected traffic itself, because it is protected by TLS. The TLS session is considered secure, in particular, there are no vulnerabilities from leaked certificate secret keys.

Attack Goals The objective is to have the client connect to $\mathcal{B}$ and perform a session resumption to establish a TLS session with $\mathcal{B}$. Because session resumption skips the certificate verification, the client does not realize that the server has changed and sends requests meant for $\mathcal{A}$ to $\mathcal{B}$. $\mathcal{B}$ has to accept and process these requests differently from $\mathcal{A}$.

Since the attacker does not control $\mathcal{B}$ in any meaningful way, the impact of the attack heavily depends on $\mathcal{B}$'s behavior: $\mathcal{B}$ could for example visibly log requests and thereby leak potentially confidential requests meant for $\mathcal{A}$. Alternatively, $\mathcal{B}$ could serve user-submitted data, causing the client to request and process potentially malicious files. Depending on the messages sent by the client this could either happen organically or could be only made possible through a separate weakness like a CSRF attack.

2.3 TLS-Attacker

TLS-Attacker [18] is a Java framework designed for analyzing TLS libraries. It allows users to define arbitrary sequences of TLS messages and gives fine control over the respective content. We employ TLS-Attacker to dynamically handshake with different servers and observe their reaction to certain messages. We use not naturally occurring combinations of SNI and session tickets, which are not necessarily supported by other TLS client implementations. The details are outlined in Section 3.3.

3 Design

Our objective is to implement a scanning application, which identifies servers that are potentially vulnerable to a session ticket confusion attack because they share STEKs (as described in Section 2.2). In this chapter, we will examine previous work and, together with the background knowledge from the section above, determine design goals for our application.

We present a high-level description of our proposed application, which we describe in more technical detail in the following Chapter 4.

3.1 Previous Work

In 2016, Springall, Durumeric, and Halderman conducted a study where they investigated the use of "security shortcuts", which served "to reduce the costs of cryptographic computations and network round trips" [11]. This included identifying "service groups", larger groups of domains, operated by the same organization, that shared their STEKs for extended amounts of time. They queried each domain on the Alexa Top Million list ten times and grouped domains that issued tickets with a matching *key_name* field during any one of the ten attempts.

Amongst the one million potential domains, they found 354,697 sites that supported session tickets. 184,603 of those sites were grouped, meaning they shared their *key_name* with at least one other site. Based on their study, the authors concluded that "the magnitude of sharing [STEKs] across domains was surprising" [11].

3.2 Design Goals

Springall, Durumeric, and Halderman relied on the *key_name* field in order to identify shared STEKs. While the *recommended* ticket construction (Subsection 2.1.3.2) suggests that the *key_name* should "identify a particular set of keys", this does not mean that matching *key_names* guarantee a matching STEK.

We propose to simulate a ticket confusion attack (Section 2.2) to confirm whether servers share their STEK: If a server can extract the master secret from another

server's securely encrypted ticket, then both servers had to have used the same symmetric key.

Unfortunately, a matching STEK is no guarantee for accepting a ticket: Outside factors, such as the information encrypted in the ticket itself, or the client's handshake (cf. Subsection 2.1.3.3) may cause a server to reject tickets that they were able to decrypt. This means that our approach can only prove that servers share a STEK but never rule it out.

However, even if two servers share a STEK and accept each other's tickets, this only constitutes an attack if the destination server processes the requests accordingly (see Subsection 2.2.2). Therefore, we want to refine our scan by also looking at different scenarios regarding SNI: Unlike an attacker, we can freely manipulate the SNI, because we are acting as both the attacker and the victim. As we discussed in Subsection 2.1.3.3, SNI is the most relevant factor in getting a ticket accepted apart from the STEK. Because SNI has a record of being poorly implemented in session resumption, we expect that some servers will likely handle SNI values inconsistently [2]. By experimenting with different SNIs, we will try to identify different levels of vulnerability.

This motivates our first major design goal: Instead of relying only on the *key_name*, we want to check whether two domains share a STEK by requesting a session ticket from server $\mathcal{A}$ and then attempting to resume a session using that ticket on server $\mathcal{B}$ with $\mathcal{B}$'s SNI. Only if this succeeds, the servers can be considered *potentially* vulnerable to a session ticket confusion attack. If the servers share a STEK, we want to try different SNI values to assess the possibility of a session ticket confusion attack.

This approach has one caveat: Since we have to check pairs of two domains, we scale quadratically in the number of domains. For common datasets such as the Tranco Top 1 Million list [19], this number becomes prohibitively large.

We cannot avoid the scaling factor, but we do not have to consider every single combination: The odds of two servers sharing a randomly generated 128-bit AES key (a reasonable choice for encrypting/securing tickets) by chance are negligible. Rather we are interested in domains where the STEKs are shared intentionally or accidentally through misconfiguration. Our overall goal here is to consider systemic misuse, rather than finding singular instances. In this case, there has to be some logical proximity between the hosts, e.g. servers hosted in the same data center, multiple domains hosted on the same physical machine, domains maintained by the same system administrator, etc.

Thus, our second design goal: Rather than checking every possible combination, we only want to analyze promising candidate pairs that are related. Here we want to implement pre-sorting hosts into potential "service groups" such as the ones found by [11]. We present criteria for this in Section 3.4.

3.3 Scanning for STEK Sharing

We consider pairs of *virtual* hosts, meaning pairs of IP addresses and hostnames for our scans. To check whether hosts $\mathcal{A}$ and $\mathcal{B}$ share a STEK, we first initiate an ordinary connection to host $\mathcal{A}$ in which we request a session ticket from the server.

We then use the ticket issued by $\mathcal{A}$ with $\mathcal{B}$, by treating our request to $\mathcal{B}$ as an ordinary resumption request, except we include $\mathcal{A}$'s ticket in the extension. If $\mathcal{B}$ performs the abbreviated handshake (cf. Figure 2.4), that means that $\mathcal{B}$ was able to decrypt the session ticket, proving that they use the same STEK as $\mathcal{A}$.

Since our overarching goal is to evaluate the likelihood of session ticket confusion attacks, this may not be enough: Especially SNI may interfere (see Subsection 2.1.3.3). Therefore, we have to consider how adjusting different elements of the handshake influences whether the foreign ticket is accepted. Changing other parameters, such as the protocol version or the used cipher suites before resuming, has been shown to lead to rejection with almost all servers [16], so SNI is the only value that we chose to manipulate between the initial handshake and resumption.

With these considerations in mind, our tool should perform the following steps:

1. Initiate handshake with $\mathcal{A}$
2. Prepare resumption handshake to $\mathcal{A}$
3. Adjust SNI accordingly
4. Send resumption handshake to $\texttt{IP}_B$
5. Check if abbreviated handshake succeeds

There are four different SNI scenarios worth examining (see Subsection 2.1.3.3 and Subsection 2.2.2), which correspond to different attack paths:

"Adjusted confusion" Set the SNI for the resumption to $\texttt{domain}_B$. This check is the most likely to succeed (as it eliminates all rejection factors within our control) and therefore is the most reliable indicator of whether the two servers share their STEK. To exploit this as an attack, an attacker needs to outright change the SNI, which requires strong control over requests sent by the client. Under such circumstances, session ticket confusion would be an unlikely attack choice.

"Original confusion" Set the SNI to $\texttt{domain}_A$. This check is the closest to our attacker model because it simulates an attacker being able to reroute a client's messages without any changes in the messages themselves. On its own, it does not tell us a lot about the possibility of an attack, because of load balancing and virtual hosting: If $\texttt{IP}_A = \texttt{IP}_B$ or if $\texttt{IP}_A$ is a mirror of $\texttt{IP}_B$, this is a legitimate resumption attempt without any reason to fail.

"Invalid confusion" Change the SNI to a domain unknown to the server (in our case: `falscherservername.de`). This check helps us differentiate whether an "original confusion" succeeded, only because the server recognized the name (in which case this fails) or due to an improper fallback (in which case this check also succeeds). In case an attacker gains limited influence on the SNI, invalid confusion may become an attack vector. Across different servers, this is equivalent to "original confusion", but it can also work on virtual hosts of the same server when there is an insecure fallback in place.

"Missing confusion" Do not include the SNI extension at all. This attack path requires an insecure fallback, when there is no SNI supplied. In the past, downgrade attacks were used to effectively remove SNI from a handshake [4]. If an exploit to remove SNI is discovered in the future, this may become a viable attack scenario.

Our base attacker model (Subsection 2.2.2) only considers "original confusion", where the attacker does not have any TLS level capabilities. If an attacker can manipulate the SNI accordingly, by abusing another weakness like CSRF, each of these scenarios can become a full-fledged attack. Depending on the changes made to the SNI and the behavior of the destination server, different targets are possible. We summarize the circumstances needed in Table 3.1. In any case, an attacker needs to be able to redirect traffic between two servers that share a STEK to perform a ticket confusion attack.

Type	**Changes to SNI**	**Destination accepts**	**Required routing**	**Possible targets**
Adjusted	Arbitrary	–	–	Other servers + VHosts
Missing	Remove SNI extension	without SNI	Fallback	Other servers + VHosts
Invalid	Set to unknown	invalid SNI	Fallback	Other servers + VHosts
Original	–	invalid SNI	Fallback	Other servers

Table 3.1: A summary of the different attack types. In any case, the source and destination host need to share a STEK, but with additional control over the SNI, more possible attacks open up. Notably, virtual hosts can only become a target when there is some kind of control over the SNI because otherwise, the request becomes a proper resumption with the original host.

Based on the importance of the different variations and to improve speed, we have chosen to execute "adjusted confusion" as a base test: We only ever perform the other three if adjusted confusion succeeded.

Note that the test is directional: While STEK sharing necessarily applies to both parties, that is not the case for actually accepting any tickets. To meaningfully evaluate the danger of session ticket confusion attacks we, therefore, perform the described scan from $\mathcal{A}$ to $\mathcal{B}$ and vice-versa. We also repeat the scan for both TLS versions 1.2 and 1.3, due to the differences outlined in Subsection 2.2.1.

3.4 Restricting the Search Space to Promising Candidates

As explained above, our scan has to be performed in both directions, making an exhaustive search of all $n \cdot (n-1)$ pairs of hosts infeasible for any common sample size. Instead, we are interested in pairs of hosts that have some degree of logical proximity. Their STEKs might be shared due to external factors besides pure chance. Possible reasons include human error, such as misconfiguration of multiple virtual hosts, intentional configuration, such as mirrored servers for a load-balancing setup, or automated deployment with unchanged default values. We believe that CDNs are especially likely to share their STEK because they directly profit off of the increased speed with little associated risk, as long as all entry points to the network behave the same.

We need some indicators to decide whether we consider a pair to be worth scanning, so we introduced a pre-processing stage: There we collected different data points and grouped different hosts based on these. We then perform pairwise, bi-directional scans for all hosts within a group.

We will now explain the different factors we originally considered and why they are a potential indicator for STEK sharing:

Keyname As explained in Subsection 2.1.3.2, the *key_name* field is used to identify a "set of keys [3]", meaning a STEK. Therefore equal *key_names* are the strongest indicator that two servers share their STEKs.

IP blocks Hosts with IP addresses that are near (or even equal) to each other indicate that the hosts may be on the same physical machine or in the same network. Here, different hosts may be serviced by the same admin or deployed within the same infrastructure, which increases the chance of accidentally sharing the STEK. At the same time STEKs may be shared intentionally, e.g. with different, load-balancing servers. We especially considered IPs from the same /24 block, i.e. IPs that only differ in the last 8 bits (for a total of 255 distinct addresses) to limit the number of pairs.

Domain similarity Hosts with similar domains are likely to belong to the same organization. "Similar" can refer either to subdomains, e.g. `panda.upb.de` and `paul.upb.de`, different top-level domains (`google.de` and `google.com`), or lexicographically similar second-level domains like `upb.de` and `upb50.de`.

Ticket length It is very unlikely that implementations would issue tickets with different content lengths between contents (cf. Figure 2.5), so we considered the ticket length as a possible criterion.

Certificate names Certificates can contain different domain names for which they are valid (through wildcards or the "Subject Alternative Name" field), so a web server can serve different domains with a single certificate and potentially use the same STEK.

4 Implementation

Based on the design presented in Chapter 3, we have implemented a pipeline consisting of Preprocessing, Grouping and Scanning. In the Preprocessing step, we collect data points, as described in Section 3.4, about all domains from the Tranco Top 1 Million list. We use these data points to create groups of promising hosts. Finally, we perform a scan, as described in Section 3.3, for each host pair within each group.

In this chapter, we dive into implementation details for each of the steps. The resulting implementation will eventually be the base for our evaluation presented in Chapter 5.

4.1 Preprocessing Phase

We first filtered out invalid entries in our domain list by using ZDNS [20]. This gave us a list of all resolving domains and each of their responsible IP addresses[1]. We then used ZMap [21] to discard all gathered IP addresses, which expose TCP port 443. Finally, we recombined the remaining IP addresses and their respective domains.

This left us with all responding combinations of IPs and domains from our initial list: Using ZGrab[2], we performed 3 handshakes each with the `--session-ticket` option enabled, which requests a session ticket during the handshake. For each host, we determined the *key_name* by taking the prefix shared amongst the most tickets [13]. This way, we reduced the handshake data down to tuples of (`ip, domain, key_name, ticket_length`).

ZGrab only supports TLS up to version 1.2 at the moment, so we only consider hosts that support TLS 1.2 tickets, regardless of PSK support. This means that the *key_name* for any host is only based on TLS 1.2 tickets, hosts that exclusively support PSKs are filtered out. It also means that we need to check for PSK support for any given host before performing the scan.

[1] We restricted ourselves to IPv4 addresses because including the IPv6 space increases the search space significantly.

[2] `https://github.com/zmap/zgrab2`

4.2 Grouping Phase

We constructed groups of hosts using the aforementioned tuples in a simple Python script via `itertools`' `groupby`[3] function. Our final implementation used the *key_name*, the *ticket_length*, and the first 24 bits of the IP address as a grouping key.
The output was a list of all pairs of different hosts from within each generated group, each tagged with an arbitrary index for their respective group. We discarded groups of size one because they do not contain a *pair* of candidate hosts.

4.3 Scanning Phase

4.3.1 Overall Architecture

Each ticket confusion scan is independent of the others and we can trivially parallelize the scanning workflow described in Section 3.3. Therefore, we modeled each scan job as a separate `SessionTicketConfusionScanner` object. Each consists of a unique combination of source and destination hosts.

The `SessionTicketConfusionScanner` class implements the `Callable` interface. Using Java's inbuilt `ExecutorService` class, we can simply submit all of our scan jobs and leave multithreading to the operating system. We go into detail about the implementation of a single job in Subsection 4.3.2. To prevent data loss from unforeseen interruptions, we immediately collect the results of each job and write the results to an external database. In our case, we used the Java driver for MongoDB, a document-oriented NoSQL database. The resulting architecture can be seen in Figure 4.1.

4.3.2 Scanning Job

In our implementation of an individual scan job, we have taken the SNI variations proposed in Section 3.3, but added additional steps to improve robustness and speed. We first check whether we have the technical prerequisites to perform a ticket confusion, that is whether we get a ticket from $\mathcal{A}$ and whether we can redeem tickets with $\mathcal{B}$. While this should have been ensured by our pre-processing, we encountered some servers that behaved inconsistently between connections. This also serves as a check for PSK support in the TLS 1.3 case (cf. Section 4.1). The resulting scan job is shown in Figure 4.2. It is the same for both TLS 1.2 and 1.3.

We implemented each of the steps as its own `Probe` class, derived from two abstract superclasses (see Figure 4.3): `ResumptionProbe` and `SessionTicketConfusionProbe`.

[3] `https://docs.python.org/3/library/itertools.html#itertools.groupby`

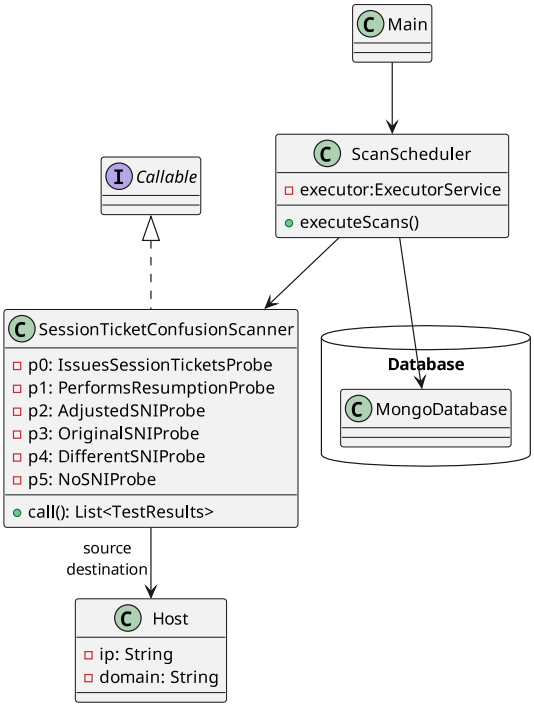


Figure 4.1: Simplified class diagram depicting the architecture of our scanning application. For brevity, we have omitted and simplified classes surrounding `SessionTicketConfusionScanner`.

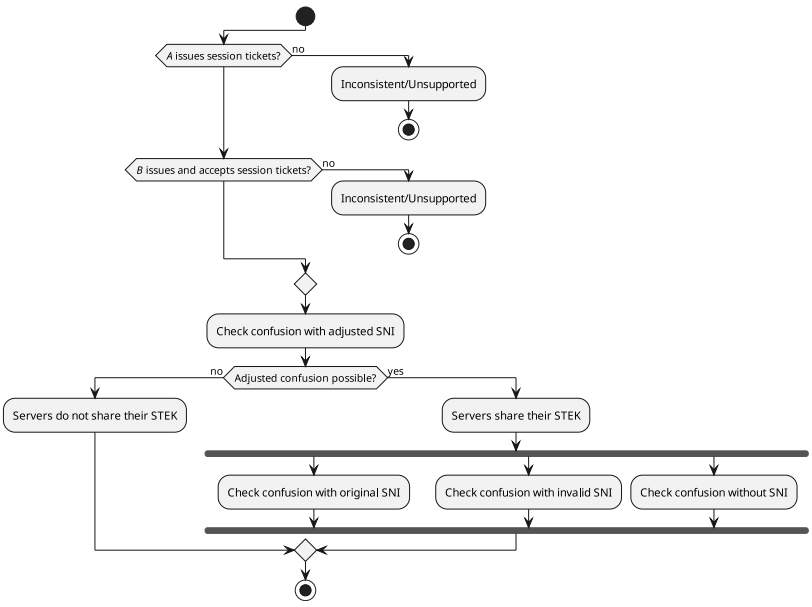


Figure 4.2: Activity diagram depicting the flow for a single scan job.

`ResumptionProbe` provides utility functions to perform a session resumption, which we adapted from Hebrok et al. [13]. Based on the IP and domain of the target, as well as the protocol version, it can generate a `State`, which represents a sequence of messages the client will send and expect to receive ("workflow"), within the TLS-Attacker framework. We generate different `States` for the different handshakes of the session resumption. Once executed, we can then verify if our workflow succeeded, meaning the server responded in the way we expected/wanted it to. Our initial handshake workflow succeeds if the server accepts the handshake itself and issues a session ticket (see Figure 2.3). The resumption handshake succeeds if the server accepts the ticket without falling back to a full handshake (see Figure 2.4).

To capture the notion of ticket confusion, where the host for the initial handshake does not match the host for the resumption, we introduced the subclass `SessionTicketConfusionProbe`: It implements Section 3.3, by exposing the abstract function `configureResumptionHandshake()`, to change the `Config` used for the initial handshake before attempting to resume. This allows us to freely manipulate the SNI and the destination host between handshakes by subclassing `SessionTicketConfusionProbe`.

A possible resulting entry of our scan, as saved in the database, is shown in Figure 4.4.

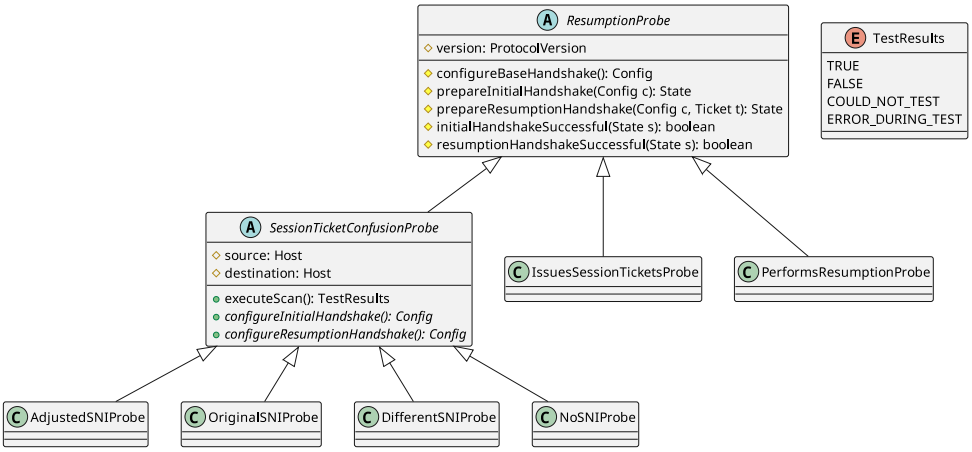


Figure 4.3: Class diagram depicting each probe. This diagram omits technical details related to multithreading and serialization.

```
{
    _id: ObjectId('640258c642b1c23a59ab15a0'),
    source: { domain: 'zerotothree.org', ip: '141.193.213.20', index: 21447 },
    destination: { domain: '6sense.com', ip: '141.193.213.21', index: 21447 },

    scanResult: {
        confusionResultMap: {
            TLS12: {
                sourceIssuesSessionTickets: 'TRUE',
                destinationPerformsResumption: 'TRUE',
                allowsTicketConfusionWithOriginalSNI: 'COULD_NOT_TEST',
                allowsTicketConfusionWithInvalidSNI: 'COULD_NOT_TEST',
                allowsTicketConfusionWithoutSNI: 'COULD_NOT_TEST',
                allowsTicketConfusionWithAdjustedSNI: 'FALSE'
            },
            TLS13: {
                sourceIssuesSessionTickets: 'FALSE',
                destinationPerformsResumption: 'FALSE',
                allowsTicketConfusionWithOriginalSNI: 'COULD_NOT_TEST',
                allowsTicketConfusionWithInvalidSNI: 'COULD_NOT_TEST',
                allowsTicketConfusionWithoutSNI: 'COULD_NOT_TEST',
                allowsTicketConfusionWithAdjustedSNI: 'COULD_NOT_TEST'
            }
        }
    }
}
```

Figure 4.4: A possible resulting database entry for a job. Note that this might not appear consistent to Figure 4.1, because the class diagram was simplified.

5 Evaluation

We used the implementation described in Chapter 4 to perform a large-scale evaluation of hosts in the wild. In this chapter, we present the results of our evaluation and analyze which hosts are potentially vulnerable and why. We do so by taking a closer look at the host pairs we found and which probes they were susceptible to on which TLS version.

5.1 Scans

Host Groups Selection We used the standard Tranco Top 1 Million list obtained on February 22, 2023 [19][1] for our scan. After our initial pre-processing step, we were left with 1,239,420 unique combinations of IP and domain ("hosts") to consider. To test our implementation and to provide an initial benchmark, we first applied our scan to around 6,000 entries. Based on the runtime of this preliminary scan we settled on the choice of grouping criteria outlined in Section 4.2, to reduce the runtime to a feasible level. This included limiting ourselves to the first 100,000 entries of the Tranco Top 1 Million list. This gave us 122,701 unique combinations of IP and domain, resulting in 87,677 unique hosts, spread across 7143 groups.

Excluding CDNs While much more manageable, this still would have required 43 million individual scan jobs (approximately 90 days). A closer examination revealed that a few strongly related clusters accounted for a majority of those: Three clusters of sizes 3810, 3304, and 2949 made up close to 10% of all scans. We were able to track these hosts to a few major CDN providers: CloudFlare, Fastly, and Amazon CloudFront (AWS). We assumed that CDN endpoints would follow a streamlined implementation and would behave consistently across clusters. We decided to focus our efforts on hosts on the outside and filtered out hosts belonging to either of the three providers (using their publicly available IP ranges).

Scan Setup This left us with 22,132 hosts in 4,470 groups for a total of 1.2 million individual jobs. They were completed using an Intel Xeon CPU E5-2695 v3 @ 2.30GHz in around 9 days. The tool used 200 Java threads on 22 virtual CPU cores. We later performed equally many scans, randomly sampled from the CDN hosts we

[1] available at `https://tranco-list.eu/list/LYVG4/1000000`

filtered out, to validate our assumptions. These scans were performed in three days with the same setup.

5.2 Results

5.2.1 STEK Sharing

We say that a host is vulnerable, if it accepted a probes resumption handshake in at least one instance. As shown in Figure 5.1, we found that a majority of surveyed hosts were vulnerable to adjusted confusion, which indicates a high amount of STEK sharing. In particular, 80.9% of the hosts shared their STEK at least once on either TLS version 1.2 or 1.3. 44.7% of all vulnerable hosts did so only on TLS 1.2, while another 30.4% did so on both versions of the protocol. Only 5.8% of scanned hosts were found to share their STEKs on TLS 1.3 exclusively.

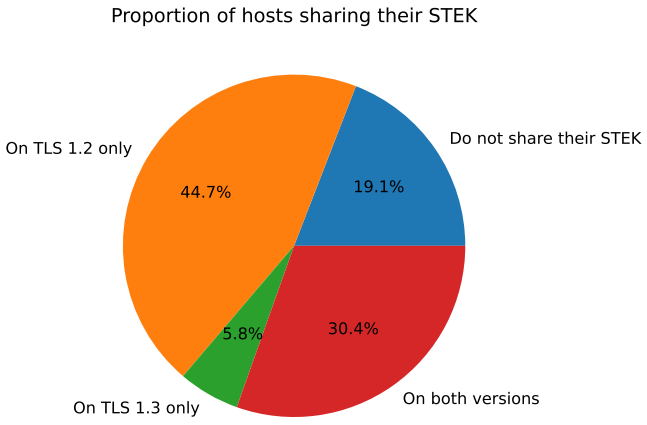


Figure 5.1: Proportion of hosts, sharing their STEK. Two hosts share their STEK when adjusted confusion is successful.

We see that the host vulnerable to the adjusted and original confusion probes are closely related, which means most hosts with shared STEKs would also be vulnerable to a real attack. However, we also found a noticeable distance to the missing and invalid confusion probes, which indicates that our number of “true” original confusions is inflated: There must have been servers, that accepted the original SNI confusion because they recognized it (i.e. virtual hosts or mirrors), rather than falling back because of an unknown SNI.

Because of our preprocessing, all scanned hosts supported TLS 1.2 session tickets, but only 43.4% of the hosts also supported PSKs[2]: By looking at the *relative* number of vulnerable hosts, we see that the share of vulnerable servers is similar between versions. In fact, TLS 1.3 has a higher proportion of hosts with STEK sharing. Compared to 75.1% of TLS 1.2 STEK sharing, 83.4% of hosts with PSK support shared their STEK (see Figure 5.2b). Only fallback confusion (i.e. missing or invalid confusion) is noticeably less prevalent in TLS 1.3 both in the share of vulnerable hosts and the difference between "proper" and fallback confusion. We believe that this is due to the requirement of explicit SNI, outlined in Subsection 2.1.3.3.

5.2.2 Behavior Within Groups

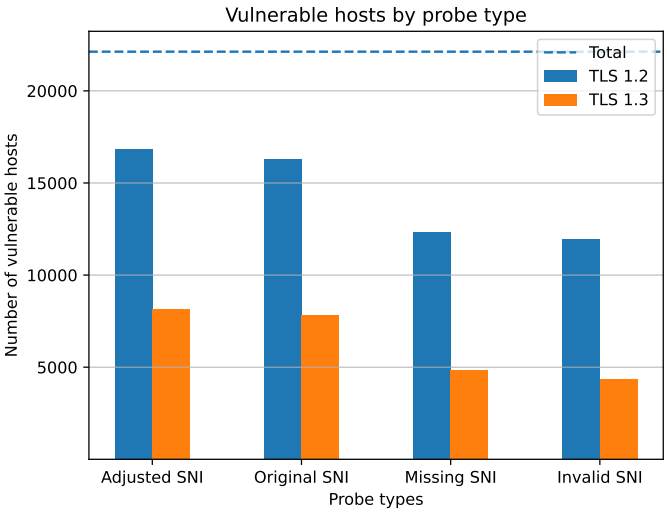
The notion that one occurrence of confusion makes a host vulnerable ignores the dynamics within the scanned groups themselves: We found that the hosts we identified as vulnerable by each of our probes (Figure 5.2a), behaved consistently within their groups: For adjusted confusion and original confusion, the share of hosts that accepted the tickets of all other hosts within their group was 69.4% and 70.2% respectively, with invalid and missing confusion sitting even higher at 81.0% and 80.4% (see Figure 5.3a). This means that when two hosts share a STEK, most often the whole group does.

Additionally, we tried visualizing groups to find irregular structures in terms of the possible directions but were unable to do so. Most groups resulted in a highly connected graph, although it has to be noted that not all groups were fully connected. We also found service groups that would be split into subgroups of vulnerable combinations – both can be seen in Figure 5.3b. This is likely a consequence of our scanning methodology, as server configurations could have changed during the execution (for example due to STEK rotation).

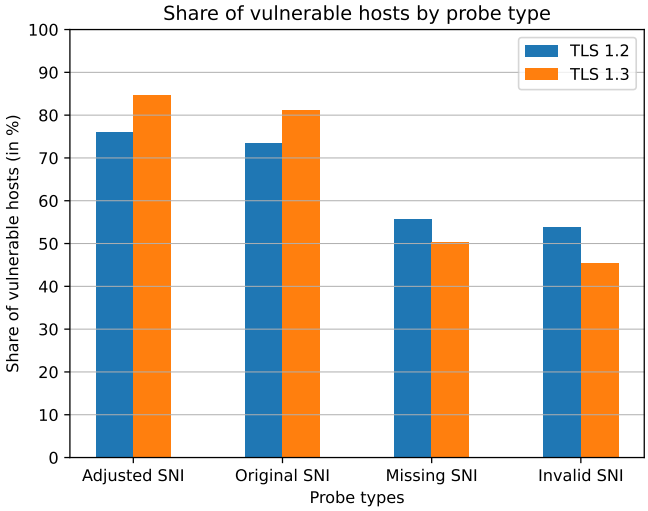
5.2.3 Identifying Common Properties

To understand which hosts were most likely to share their STEK, we looked at the IP addresses of successful adjusted confusion attempts (Figure 5.4). We found that a majority of hits happened with nearly consecutive IPs – 63.5% of detections occurred between hosts with a difference of less than four. 35.7% of these can be accounted to hosts with an IP difference of zero, i.e. virtual hosts on the same machine.
While these results support our initial assumption that close IP addresses are more likely to be vulnerable to confusion, it must be considered that they are somewhat

[2]A similar proportion can be seen amongst Tranco Top 1 Million as a whole: 65.9% of support TLS 1.2 session tickets, while only 33.8% support TLS 1.3 session tickets, according to `https://tls-scanner.cs.uni-paderborn.de/properties/sessionResumption` on April 5, 2023



(a) The absolute number of vulnerable servers per version for each of our probes.

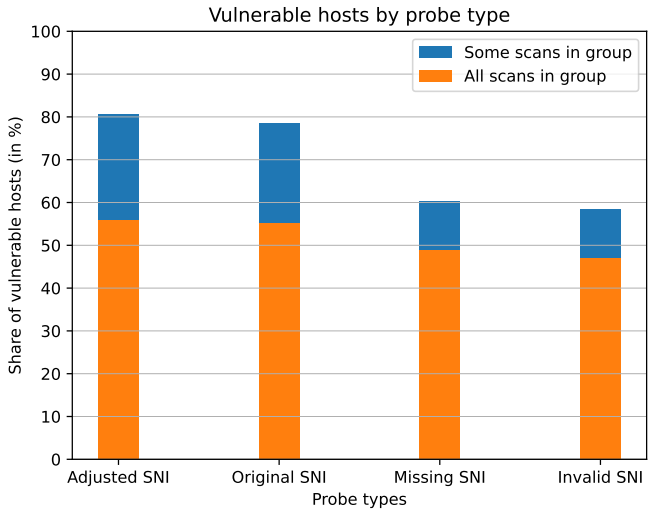


(b) The share of vulnerable servers compared out of the servers that support TLS 1.2/1.3 tickets respectively.

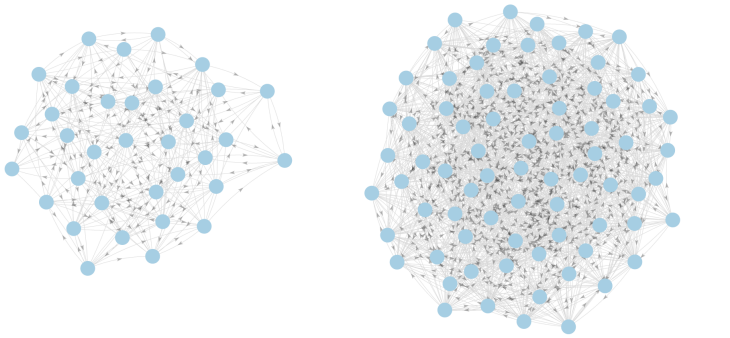
Figure 5.2: Analysis of vulnerable servers per version per probe type.

skewed: Since the number of individual scans scales quadratically with the group size, larger groups tend to be overrepresented.

We also manually inspected the largest groups of vulnerable hosts to understand what types of services/organizations shared STEKs on a larger scale: By looking at the IPs and the certificates used by the respective hosts, we were able to



(a) Comparison of the number of hosts that are vulnerable in at least one instance to the number of hosts consistently vulnerable for all pairs within their group. Our results indicate that groups as a whole tend to be vulnerable rather than singular pairs.



(b) Visualization[a] of accepted tickets for *one* service group. Directed edges indicate successful adjusted confusion. Here we can see a strongly (but not necessarily perfectly) connected structure within a group. Some groups split into disjunct subgroups like this, presumably because of STEK rotation over time.

[a] using `https://github.com/vasturiano/force-graph`

Figure 5.3: Analysis of the vulnerability of whole service groups.

identify the entities responsible for each of the major groups. They are shown in Table 5.5.

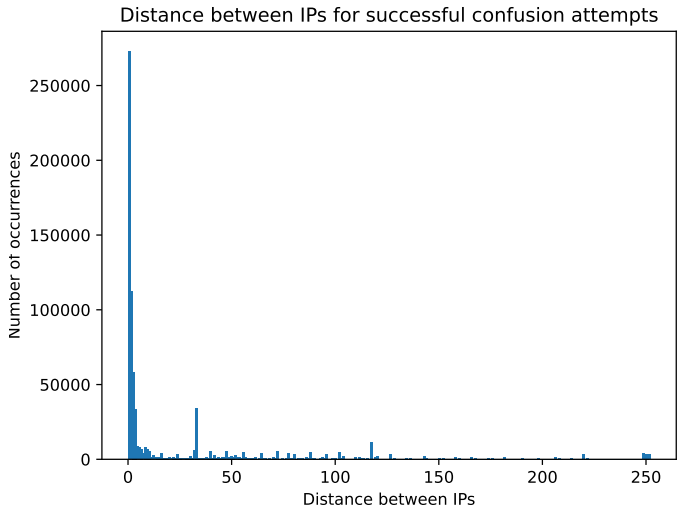


Figure 5.4: IP distances between successful confusion attempts

Most identified parties appear to be either CDN providers (Pantheon, Vercel, AliCDN and StackPath/Highwinds) or hosters for user content (such as WPEngine, WordPress, Shopify and GitHub). Surprisingly enough, the majority of the actual underlying infrastructure turned out to be provided by a few major providers - including the ones we previously filtered out. Both WPEngine and Shopify are served on CloudFlare infrastructure, although with IP ranges outside of the public CloudFlare ranges. GitHub uses Fastly for their repositories as well as GitHub Pages. Highwinds Network Group was acquired by StackPath and their "platform [was] integrated with existing StackPath offerings", so we consider them as one combined service provider [22]. Tallying these up gives us the numbers portrayed in Table 5.6.

5.2.4 Vulnerablity to Real-World Ticket Confusion

As mentioned earlier, almost all hosts vulnerable to adjusted confusion are also vulnerable to original confusion: To understand whether these were "true" confusions, we analyzed instances of original confusion from $\mathcal{A}$ to $\mathcal{B}$ by requesting certificates from $\mathcal{B}$ with $\texttt{domain}_A$ and comparing them to the certificates supplied by $\mathcal{A}$ in the first place. If they are the same, $\mathcal{B}$ would have been able to perform the exact same handshake as $\mathcal{A}$, making them a legitimate stand-in.

Our results show that for a majority of these confusions, the certificates did not match, which means that the client was confused into trusting $\mathcal{B}$, as can be seen in Figure 5.7. Note that there are two major limitations to this observation: We did not query the different certificates on the same day, which may falsify the results,

Table 5.5: The twenty largest vulnerable groups we found. All of them are either part of a CDN or host user pages.

Service group	#hosts	Service group	#hosts
WPEngine #1	460	WPEngine #4	108
Pantheon	428	Highwinds	101
WPEngine #2	405	AliCDN #2	101
Shopify	280	StackPath	101
WordPress	231	WordPress VIP	86
xvideos	154	CloudFlare	80
Vercel #1	152	GitHub #1	74
AliCDN #1	147	GitHub #2	74
Vercel #2	137	GitHub #3	72
WPEngine #3	120	GitHub #4	72

Table 5.6: Twenty largest vulnerable groups, summed by the underlying infrastructure. Most groups identified in Table 5.5 are realized using the same underlying infrastructure.

Infrastructure	# hosts
CloudFlare	1453
Pantheon	428
WordPress	317
Fastly	292
Vercel	289
AliCDN	249
StackPath	202
xvideos	154

because certificates may have changed in the meantime. Secondly, we only checked for exact certificate matches: Technically multiple distinct certificates could be issued for the same domain, so $\mathcal{B}$ might have had a different certificate for $\texttt{domain}_A$, possibly through a wildcard certificate or multiple Subject Alternative Names. This would possibly make it legitimate for $\mathcal{B}$ to handle the resumption request, even when the actual certificate may be different.

It would be possible to inspect the names of the certificate, but even then, $\mathcal{B}$ has to serve the same content as $\mathcal{A}$ (cf. Subsection 2.2.2). Unfortunately, this is best verified by also taking the HTTP content served into account, which was out of the scope of this thesis.

Due to these limitations, we cannot say for certain whether a host is vulnerable. Nonetheless, we conclude that it is dangerous for hosts to accept tickets under a

different certificate because the client only verifiably trusts the original certificate.

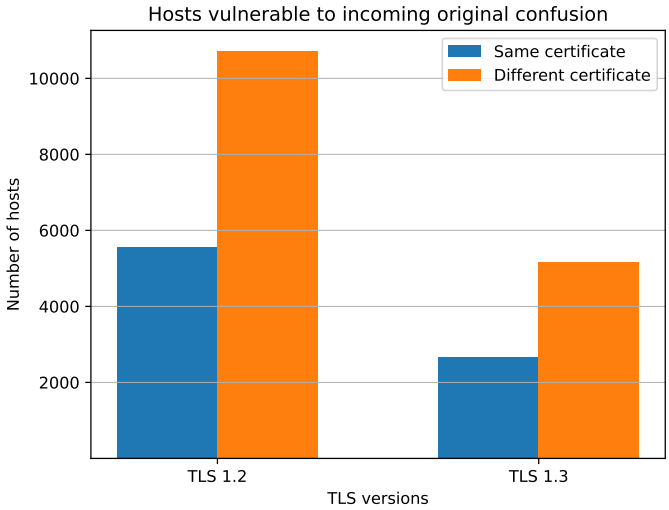


Figure 5.7: Comparison of successful original confusions

The same cannot be said for the fallback variants (invalid and missing confusion): On TLS level (compare Figure 5.2a), we encountered a clear increase in rejected connections. Furthermore, we found that all major infrastructure providers identified in Table 5.6, employed countermeasures on the Application layer at least: These ranged from specialized error codes (CloudFlare) to HTTP 404 (WordPress, Pantheon, Fastly to an extent) to central redirects (Vercel). In either case, we were unable to identify evidence for unusual and potentially vulnerable behavior.

5.2.5 Supplementary CDN Scan

As mentioned in Subsection 2.1.3, we performed a second scan of only CDN hosts that we had previously filtered out. These scans were picked randomly and accounted for 2.9% of possible scans in this set. They are not meant to be exhaustive, but to validate our assumption, that CDN nodes tend to behave homogenously regarding ticket confusion.

Our random sample contained 52,302 new unique hosts: 29.9% belonged to AWS, 60.2% to CloudFlare and 9.9% to Fastly. We found a lower percentage of hosts sharing STEKs (see Figure 5.8). We attribute this to the sparse coverage of potential scans. This still amounted to 24,370 new hosts sharing their STEK. Only 5.0% of vulnerable hosts belong to AWS, 75.0% to CloudFlare and 20.0% to

Fastly. This shows, that STEK sharing is a problem among all evaluated CDN providers.

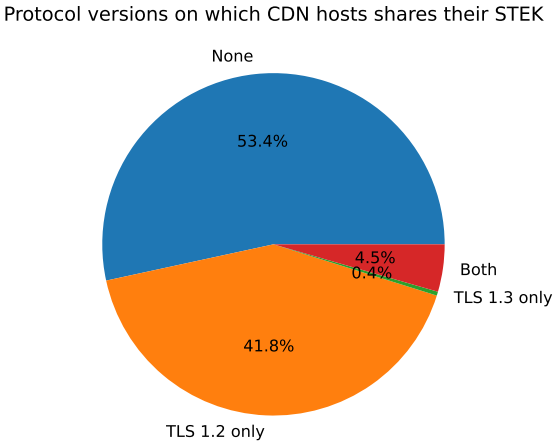


Figure 5.8: The portion of CDN hosts, sharing their STEK on a specific version.

Categorizing the hosts by vulnerability to the different probes (see Figure 5.9) is far less uniform than before (Figure 5.2a). For each CDN, adjusted confusion is closely related to original confusion. Missing and invalid confusion differ between CDNs: AWS CloudFront and an overwhelming part of CloudFlare require a correct SNI to accept the resumption. Fastly servers on the other hand servers seem to be indifferent to the supplied SNI and always react the same. Overall, the success rate varies strongly between the different CDNs, especially CloudFront is an outlier with a rate of only 7.8%. This may be due to the low sample size.

However, these additional scans still confirm, that CDNs are equally vulnerable to STEK sharing and ticket confusion (although at varying degree). This aligns with our previous findings, indicating that our results generalize to the Tranco Top 100k list (at least).

5.3 Discussion

In our evaluation, we confirmed 17,901 instances of STEK sharing for 22,127 of the scanned hosts, which were spread evenly between protocol versions[3]. The sheer scale of the vulnerability indicates a general disregard for STEKs as truly unique secrets.

[3]Relative to the overall support of session tickets.

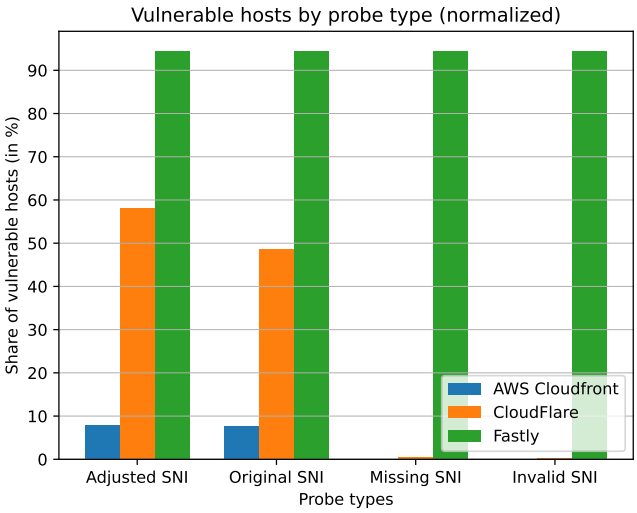


Figure 5.9: The portion of CDN hosts, that are vulnerable to one of the probes, grouped by CDN providers.

We believe, however, that this is intentional: We saw, that the largest identified groups were all associated with CDNs, which were positioned on closely related IPs and behaved consistently across their respective groups. A less exhaustive scan of servers directly hosted by AWS, CloudFlare and Fastly supports this hypothesis: Although the exact numbers differed, we found another 24,370 potentially vulnerable hosts, distributed across all three. It makes sense for these kinds of services to do so - as long as each entry node is capable of handling every domain on the network, this can improve speed across wildly different websites without actually compromising security.

One important prerequisite to a session ticket confusion attack is improper routing on the destination: If rerouting a client's connection does not change the received content, there is simply no effect. In practice, we found inconclusive evidence, which suggests that rerouting circumvented authentication under the original certificate, which violates expectations to an extent. To properly evaluate this risk, proper fingerprinting on the HTTP level is recommended.

Ultimately we found STEK reuse on 56.8% of all surveyed hosts - although we did not find a way for direct exploitation, this is an avoidable risk to security in the future. The results confirm that the *key_name* is indeed a strong indicator for STEK sharing as proposed in [11].

5.3.1 Limitations

During the implementation of our scanning approach, we ran into two main limitations outlined below, which should be considered when interpreting the results.

Scaling and speed The resulting application is quite slow and scales quadratically by design: We overestimated the effect our pre-processing would have in cutting down the total number of scans and we were not able to perform a scan without heavily restricting the search space. Furthermore, we could only perform our scan once, which makes us highly vulnerable to any changes to the system during the runtime, as well as external factors such as connection loss. Even though our findings are strong within our selected sample, our findings do not necessarily generalize to a larger population of hosts.

TLS-only scanning Our TLS-level approach to the problem proved to be insufficient for conclusively recognizing any real attack vectors and made it hard to extract meaningful, general statistics which properly represented the widely different groups we inspected.

It is possible to address the latter, by extending the scan to include HTTP-level fingerprinting, but it seems unlikely to solve the former problem. Due to SNI, it is impossible to eliminate the pairwise scanning and therefore the poor scaling, which unfortunately makes our approach unsuitable for further experiments of untargeted, large-scale scanning.

6 Conclusion

In this thesis, we explored an approach for scanning pairs of (virtual) hosts for the reuse of STEKs in TLS session resumption and the resulting danger of a session ticket confusion attack. We found that 80.9% of hosts and 46.6% of large CDN hosts we evaluated, shared their STEK with at least one other host. In both sets, the largest vulnerable groups were placed within CDN/load-balancing setups, where the STEKs seem to have been shared deliberately. In most of these cases, practical session ticket confusion appears to be possible on TLS level.

Overall, our results show that STEK sharing is quite common and might be the stepping stone for attacks in the future. However, our scanning approach is severely limited by the sheer number of possible host pairs and does not scale to a desirable number of hosts (upwards of 100 thousand), which makes future internet-wide scanning unrealistic.

Future Work Instead, we propose to concentrate the efforts on single organizations: We still believe to have uncovered a widespread issue amongst CDN providers, which do not consider STEKs to be a proper long-term secret. A closer examination of services such as CloudFlare, with a focus on STEK sharing and the different certificates used for hosts within the network, may result in a practically exploitable session ticket confusion attack. In particular, the evaluation would need to be extended to the Application layer, to analyze the routing behavior of servers affected by ticket confusion. This would require comparing the served hypertext depending on the SNI and the ticket.

Bibliography

[1] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). RFC. Obsoleted by RFC 8446, updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919, 8447. Fremont, CA, USA: RFC Editor, Aug. 2008. DOI: 10.17487/RFC5246. URL: https://www.rfc-editor.org/rfc/rfc5246.txt.

[2] E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446 (Proposed Standard). RFC. Fremont, CA, USA: RFC Editor, Aug. 2018. DOI: 10.17487/RFC8446. URL: https://www.rfc-editor.org/rfc/rfc8446.txt.

[3] J. Salowey et al. *Transport Layer Security (TLS) Session Resumption without Server-Side State*. RFC 5077 (Proposed Standard). RFC. Obsoleted by RFC 8446, updated by RFC 8447. Fremont, CA, USA: RFC Editor, Jan. 2008. DOI: 10.17487/RFC5077. URL: https://www.rfc-editor.org/rfc/rfc5077.txt.

[4] Antoine Delignat-Lavaud and Karthikeyan Bhargavan. "Network-Based Origin Confusion Attacks against HTTPS Virtual Hosting". In: *Proceedings of the 24th International Conference on World Wide Web*. WWW '15. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, May 2015, pp. 227–237. ISBN: 978-1-4503-3469-3. DOI: 10.1145/2736277.2741089. (Visited on 10/30/2022).

[5] *We Need to Talk about Session Tickets*. https://words.filippo.io/we-need-to-talk-about-session-tickets/. Sept. 2017. (Visited on 10/30/2022).

[6] Erik Sy et al. "Tracking Users across the Web via TLS Session Resumption". In: *Proceedings of the 34th Annual Computer Security Applications Conference*. Dec. 2018, pp. 289–299. DOI: 10.1145/3274694.3274708. arXiv: 1810.07304 [cs]. (Visited on 10/30/2022).

[7] *TLS Session Resumption: Full-speed and Secure*. http://blog.cloudflare.com/tls-session-resumption-full-speed-and-secure/. Feb. 2015. (Visited on 10/30/2022).

[8] Jens Hiller et al. "The Case for Session Sharing: Relieving Clients from TLS Handshake Overheads". In: *2019 IEEE 44th LCN Symposium on Emerging Topics in Networking (LCN Symposium)*. Osnabrueck, Germany: IEEE, Oct. 2019, pp. 83–91. ISBN: 978-1-72812-561-9. DOI: 10.1109/LCNSymposium47956.2019.9000667. (Visited on 10/30/2022).

[9] Erik Sy et al. *Enhanced Performance for the Encrypted Web through TLS Resumption across Hostnames*. Feb. 2019. DOI: 10.48550/arXiv.1902.02531. arXiv: 1902.02531 [cs]. (Visited on 10/30/2022).

[10] D. Eastlake 3rd. *Transport Layer Security (TLS) Extensions: Extension Definitions*. RFC 6066 (Proposed Standard). RFC. Updated by RFCs 8446, 8449. Fremont, CA, USA: RFC Editor, Jan. 2011. DOI: 10.17487/RFC6066. URL: https://www.rfc-editor.org/rfc/rfc6066.txt.

[11] Drew Springall, Zakir Durumeric, and J. Alex Halderman. "Measuring the Security Harm of TLS Crypto Shortcuts". In: *Proceedings of the 2016 Internet Measurement Conference*. IMC '16. New York, NY, USA: Association for Computing Machinery, Nov. 2016, pp. 33–47. ISBN: 978-1-4503-4526-2. DOI: 10.1145/2987443.2987480. (Visited on 10/30/2022).

[12] Sergey Frolov and Eric Wustrow. "The Use of TLS in Censorship Circumvention". In: Jan. 2019. DOI: 10.14722/ndss.2019.23511.

[13] Sven Hebrok et al. "We Really Need to Talk About Session Tickets: A Large-Scale Analysis of Cryptographic Dangers with TLS Session Tickets". In: Apr. 2023.

[14] Alfred J. Menezes and Scott A. Vanstone. *Handbook of Applied Cryptography*. 1st edition. Boca Raton: CRC Press, Oct. 1996. ISBN: 978-0-8493-8523-0.

[15] *BSI TR-02102-2 "Kryptographische Verfahren: Verwendung von Transport Layer Security (TLS)" Version: 2023-1*. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102-2.html?nn=132646 (visited on 05/07/2023).

[16] Simon Nachtigall. "Evaluation of TLS session tickets". MA thesis. Paderborn University, Aug. 2021. (Visited on 03/20/2023).

[17] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. RFC 2119 (Best Current Practice). RFC. Updated by RFC 8174. Fremont, CA, USA: RFC Editor, Mar. 1997. DOI: 10.17487/RFC2119. URL: https://www.rfc-editor.org/rfc/rfc2119.txt.

[18] Juraj Somorovsky. "Systematic Fuzzing and Testing of TLS Libraries". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1492–1504. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978411. (Visited on 04/30/2023).

[19] Victor Le Pochat et al. "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation". In: *Proceedings 2019 Network and Distributed System Security Symposium*. San Diego, CA: Internet Society, 2019. ISBN: 978-1-891562-55-6. DOI: 10.14722/ndss.2019.23386. (Visited on 10/30/2022).

[20] *ZDNS | Proceedings of the 22nd ACM Internet Measurement Conference*. https://dl.acm.org/doi/10.1145/3517745.3561434. (Visited on 04/06/2023).

[21] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. “ZMap: Fast Internet-Wide Scanning and Its Security Applications”. In: *Proceedings of the 22nd USENIX Conference on Security*. SEC'13. USA: USENIX Association, Aug. 2013, pp. 605–620. ISBN: 978-1-931971-03-4. (Visited on 04/06/2023).

[22] *Highwinds Joins StackPath | StackPath*. https://www.stackpath.com/blog/highwinds-joins-stackpath. (Visited on 05/08/2023).

List of Figures

List of Tables