Symantec

TECHNICAL BRIEF

Responsibly Intercepting TLS and the Impact of TLS 1.3

Author: Roelof Du Toit

Introduction

TLS is an inherently complex protocol due to the specialized knowledge required to implement and deploy it correctly. TLS stack developers must be well versed in applied cryptography, secure programming, IETF and other standards, application protocols, and network security in general. Application developers using TLS should have the same qualities; especially if the application is TLS intercept. Throw government regulations, unique customer requirements, misbehaving legacy endpoints, and performance requirements into the mix, and it soon becomes clear that mature systems engineering skills and strong attention to detail are prerequisites to building a reliable and trustworthy TLS intercept solution. Paraphrasing Bruce Schneier: security requires special design considerations because functionality does not equal quality[1]. The goal of this paper is to contribute to the security community by listing areas of concern in TLS intercept solutions and by highlighting the impact of TLS 1.3 on TLS intercept.

Most people think of TLS *intercept* as a mechanism to decrypt TLS, but it is important to qualify the meaning of the term in the context of this paper. The term can refer to the intercept of TLS with the cooperation of one of the endpoints, usually through configuration, or to a malicious or clandestine intercept, typically by exploiting a vulnerability in the protocol or a specific TLS implementation or both. The term **TLS Intercept Application (TIA)** appears throughout the paper, and without exception, it will refer to an application with a security purpose. Whenever we use TLS intercept in a malicious context, it will appear as just **Man-In-The-Middle (MITM)**.

Secondly, TLS intercept could either be active, where the TIA controls the flow of the protocol, or passive, where the application only has access to a copy of the network traffic (either real-time or using data-at-rest). Passive TLS decrypt is usually deployed in environments where the TLS server endpoint shares private key material with the TIA. Theoretically, you can passively decrypt TLS sessions if one of the endpoints share the TLS session secrets with the TIA, although this is not practical in most cases. We will not consider the class of applications that use machine learning to detect anomalies and malware without decrypting TLS. However, you should review the information in this paper when compiling best practices for machine learning products in the TLS space.

The focus of this paper will be active TLS intercept with TLS client endpoint configuration – commonly found in antivirus products and middlebox (TLS relay, forward proxy, NGFW, and more) deployments. Although it is not the focus, many of the principles in this paper also apply to TLS *offload* deployments, where "offload" is referring to the stripping of the TLS layer before forwarding the traffic, e.g., HTTPS to HTTP (a.k.a. reverse proxy deployments). The paper assumes a basic knowledge of TLS, as well as the concept of TLS intercept using an emulated X.509 certificate.

Responsible TLS Intercept

Cryptography is harder than it looks, and TLS intercept is complex. Vendors of security products must act responsibly in general but should take extra care during the development of TLS intercept applications. A few basic principles would go a long way towards improving security, but vendors must be willing to invest the time and resources to follow the principles. The techniques and proposals described in this paper should by no means be considered a definitive list, but rather as a starting point. Good security benefits all – vendors, consumers, and all of the industry.

Given that TLS intercept applications have a **security** purpose it should go without saying that those applications **should not downgrade the security attributes** of the TLS session. The principles below will qualify that statement.

Principle #1: Do not downgrade the cryptographic strength

First, follow secure coding guidelines and cryptography best practices. Uninitialized data and buffer overflows would more than likely put your product on the CVE Hall of Fame (or rather Hall of Shame). TIAs should **always** use a cryptographically secure PRNG, seeded with a TRNG. The importance of a strong RNG cannot be overstated - weak random numbers open up holes in many parts of the TLS protocol.

Second, the TIA should attempt to retain as much of the client TLS attributes, advertised in CH[2], as possible. This retention is called the *limiting-modifications* principle. Take, as an example, the CH cipher-suite list. Modern browsers take special care when crafting the CH cipher-suite list; accounting for cryptographic strength as well as the potential performance impact (especially on mobile

devices). The order of each cipher-suite is critical, and the TIA should limit modifications to the list. Reordering the list is not good practice, but acceptable modifications include removing weak, deprecated, and unsupported cipher-suites. Unfortunately, many TIAs use cipher-suite lists that are independent of the original CH cipher-suite list, usually with only a short list of supported cipher-suites. It is imperative to add support for as many modern cipher-suites as possible to prevent situations like downgrading a client with GCM/CCM cipher mode support to CBC mode[3].

The same argument about limiting modifications to the CH can be made for other TLS attributes, e.g., TLS version, key exchange algorithm, ECDHE curve, signature algorithm, and TLS extensions. Certain known TLS vulnerabilities can only be mitigated by later TLS versions and specific TLS cipher modes and TLS extensions – a good example is the padded oracle attack called Lucky13[4], which can be mitigated with GCM/CCM cipher modes or support for the *encrypt-then-mac* TLS extension[5]. It is the responsibility of the TIA to support all the latest security-enhancing features in TLS. In general, modern non-malicious TLS client applications advertise TLS attributes that are beneficial to security, including the order in which TLS attributes are presented to the server. The TIA should trust the client to some degree, but in practice, this should balance with detection and prevention of malicious TLS sessions.

As further background, the techniques used by most TIAs could be reduced, on a high level, to the protocol flow in either Figure 1 or Figure 2. Gray text in braces, e.g., {FIN}, is an indication that those specific messages are encrypted. Note that variations in timing would not change the high-level classification.

Technique 1 (Figure 1) defers the upstream TLS session until the first application level payload is processed, but it prevents the server from influencing the TLS attribute negotiation on TLS session #1.

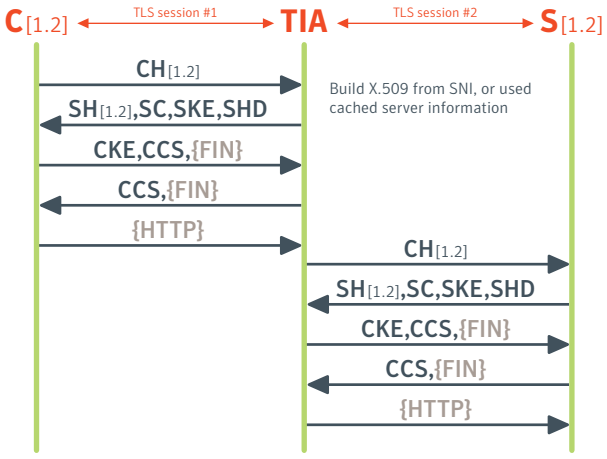

Figure 1 - TLS Intercept Technique 1

The approach taken with technique 2 (Figure 2) is to create the upstream TLS session #2 as soon as the TLS session #1 CH is received, which allows the TIA to modify the upstream CH according to the *limiting-modifications* principle. The SH on TLS session #1 also follows the SH on TLS session #2, which allows the server endpoint some level of control over the attributes of both TLS sessions.

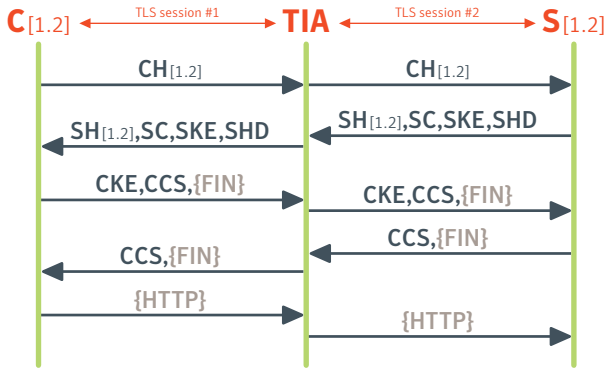

Figure 2 - TLS Intercept Technique 2

Third, the emulated X.509 certificate sent to the client should retain as much as possible of the original server X.509 certificate, including attributes like key type, key size, and subject. It is especially important to retain the validity status of the original X.509 certificate for client applications to enforce endpoint policies and properly present TLS session errors to users (see Principle #4 for more detail). The emulated X.509 certificate should retain the original *not-before* and *not-after* dates, and self-signed certificates should remain self-signed. Special care must be taken when adding X.509 extensions to the emulated X.509 certificate because some X.509 extensions are not appropriate for emulated certificates, specifically those extensions that are added by a public CA as part of extended validation (EV). You should retain the values of certain X.509 extensions without modification, e.g., *SubjectAltNames*, *BasicConstraints*, *KeyUsage*, and *ExtKeyUsage*.

Principle #2: Actively track, fix, and protect against known vulnerabilities

Malicious players rapidly exploit vulnerabilities in applications and protocols on an almost daily basis. TIA vendors and the developers, in particular, have a responsibility to actively track vulnerabilities in their systems and respond in a timely manner. Issues in TLS stack implementations, or the TLS protocol itself, usually have far-reaching implications because other applications depend on the confidentiality, authentication, and data integrity properties of TLS. The Heartbleed[6] vulnerability emphasizes the point about responding promptly, but active tracking was not necessary in that case due to the widespread news coverage. TIA developers should

go beyond just reading technical news articles, and should also: (a) follow and participate in discussions about current and future TLS related standards, (b) track changes in open source TLS stacks as well as major open source TLS endpoint applications (e.g., Chrome, Firefox), (c) educate themselves on application level protocols, especially in the context of how endpoint applications integrate with the TLS protocol.

Principle #3: Respect regulations and privacy

Privacy must be a fundamental issue for TIA designers and developers - not only from an ethical viewpoint but also due to the regulatory framework in which the TIA will operate.

Due to the location of the TIA, it has the responsibility to protect the user's PII (Personally identifiable information) as well as PHI (Protected Health Information). Protection must be two-pronged: (1) allow for masking of PII by all the security tools that touch the decrypted payload, and (2) limit intercept of certain categories of data (e.g., PHI), which includes evaluation of risk level and geolocation.

In deployments where the TIA is used to feed other security tools, it would be possible for the TIA to provide a data integrity guarantee by preventing any changes to the **decrypted** content from being propagated to the endpoints. Not only would it reduce the impact of misbehaving security tools, but it could also simplify compliance with certain regulations.

Principle #4: Validate, validate, validate

The following documents standardize X.509 certificate path validation:

RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

RFC 6818: Updates to the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

RFC 6960: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP

The proliferation of public sites switching to HTTPS is arguably good, but unfortunately, some of the mechanisms described in those standards, especially revocation checking, are not reliable or practical in such an environment. This situation has been a problem for some time, and browser vendors have implemented different mechanisms with varying levels of success. Sending the OCSP response in the TLS handshake mitigates part of the problem. This method is known as OCSP stapling, and was added to TLS in RFC 3546 and defined in its current form in RFC 6066 (updated by the pending TLS 1.3 standard). TIAs should support OCSP stapling, which means that the stapled OCSP response must determine the revocation status of the server X.509 certificate. As mentioned in Principle #1, the TIA should propagate the status of the original X.509 certificate in the emulated X.509 certificate – this includes the revocation status. The TIA should generate emulated OCSP responses at the same as generating the emulated X.509 certificate. Some client applications even have the policy to require either a stapled OCSP response or a valid X.509 *CRLDistributionPoint* extension. As an interim workaround, the TIA must, at a minimum, ensure that the client would view the emulated X.509 certificate as invalid if the original X.509 certificate has been revoked – it is up to the respective TIAs how to implement this – otherwise malicious sites might present as valid to the client application.

Client applications usually go further than just checking the X.509 path validity by also validating the server name (SNI extension sent in CH) against the names listed in the X.509 certificate. TIAs should follow the guidelines in the following document:

RFC 6125: Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)

The minimum requirements for the TIA: (a) validate the SNI against the X.509 certificate before matching a TIA policy rule against SNI, (b) only perform SNI matching against X.509 *SubjectAltNames* (SANs), with possible exceptions if the X.509 certificate does not have any SANs.

Some browsers implement SNI checks for known attacks, e.g., Punycode checks for Cyrillic-only international domain names (IDNs) that could be used to spoof legitimate sites for malicious reasons – a recent example[7] is: *https://xn--80ak6aa92e.com* would look almost identical to *https://apple.com* unless the browser implements special security checks. Responsible TIAs would also implement those extra security checks, or at least give the customer the option to configure special rule matching when the SNI is a Punycode IDN.

Principle #5: Be secure by default

The configuration of TIAs require special knowledge, and few customers have the know-how to do it securely and correctly. Educating customers would go a long way, but the TIA designers and developers could also help by anticipating which areas of configuration could potentially cause confusion, and then simplifying some of the decisions and presenting configuration choices alongside clear explanations of the impact of each option. That is easier said than done, but at a minimum, the TIA must ensure that it is secure by default.

The Impact of TLS 1.3

The primary purpose of the IETF Transport Layer Security (tls) working group, according to their charter[8], is (currently) to develop TLS 1.3[9] while considering the following design goals:

- Reduce observable data
- Reduce session setup latency, primarily for HTTP
- Address known payload protection issues (CBC, RC4)
- Reevaluate TLS handshake content
- Improve privacy, e.g., padding and less long-term-identifying values

After a few years of intense discussion and analysis (which is putting it mildly), the result is an elegant, efficient, and extensible protocol that should provide the required level of security for the foreseeable future. Extensibility is especially relevant in the context of adding post-quantum cryptography algorithms.

Where appropriate, this paper will refer to important differences between TLS 1.3 and TLS 1.2, but it will not attempt to list every detail – that information is readily available from many online sources, and it would distract from the goal of highlighting the impact of TLS 1.3 on TLS intercept.

TLS 1.3 poses unique challenges to TLS intercept applications. Vendors will have to commit to significant investment in R&D, even if the TIA is using an open source TLS stack implementation. TLS 1.2 is not going away anytime soon, so most vendors will probably take the position that it is acceptable to downgrade TLS 1.3 to TLS 1.2 – that would be a valid strategy in the short term, but it forces the TLS intercept vendor to be extra diligent when enforcing the policy so as not to downgrade security beyond responsible levels. Downgrading TLS 1.3 is perceived as not keeping up with standards, and knowledgeable consumers and customers would demand support for TLS 1.3 intercept. Also refer to the downgrade discussion later in this section for a description of a scenario where TLS 1.3 downgrade is risky.

The first TLS working group design goal for TLS 1.3 was to reduce observable data, and they have succeeded for the most part. The problem is that TIAs have come to rely on the availability of the server X.509 certificate to make proper policy decisions – especially policies that prevent intercept of certain categories of TLS sessions due to regulations. A TLS 1.3 server sends the X.509 certificate, with an optional stapled OCSP response, during the encrypted phase of the handshake – this prevents extraction and validation of policy information without advanced mechanisms. These mechanisms will invariably require extra TLS 1.3 sessions originating from the TIA to access and optionally cache the hidden X.509 information, adding to the load on TLS 1.3 servers. Independent of TLS 1.3, TLS servers might already be configured to return many different X.509 certificates depending on the negotiated session attributes, which further complicates the mechanisms to obtain the correct policy information. Complexity adds risk, so special care should be taken by TIA developers when designing these mechanisms.

The lack of plaintext X.509 information would impact other application classes as well:

- Security tools that use machine learning to enforce malware detection policy without TLS intercept.
- Inline tools that do not intercept TLS, but still require X.509 information in policy.
- Tools that monitor and classify TLS sessions based on category. The SNI information might be available, but cannot be validated (see Principle #4: Validate, validate, validate).

All those tools would require new mechanisms similar to traditional TIAs, and for some tools, it would not be possible to add such mechanisms (specifically tools that do not intercept TLS).

Whichever application or tool uses the new mechanisms, performance weighs against security during design – developers should favor the latter.

TLS 1.3 Protocol Flow

For reference, the protocol flow differences between a full (non-abbreviated) TLS 1.2 session and a TLS 1.3 session is shown in Figure 3. Interesting observations from the diagram are: (a) TLS 1.3 reduces the handshake setup from 2-RTT to 1-RTT, (b) TLS 1.3 does not need an explicit *ChangeCipherSpec* (CCS) signal to switch to the encrypted phase, (c) in TLS 1.3 the application payload is encrypted with different keys than the handshake messages.

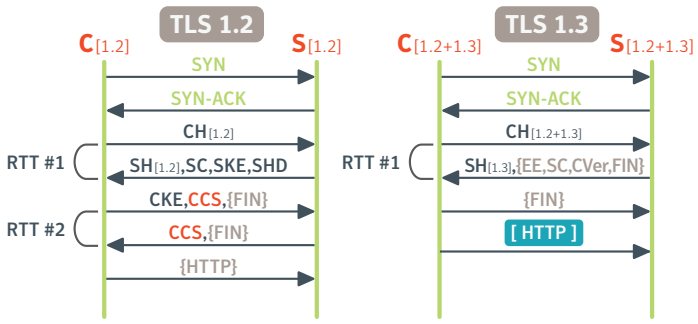

Figure 3 - TLS 1.2 2-RTT vs TLS 1.3 1-RTT

TLS 1.3 Intercept Protocol Flow

An intercepted TLS 1.3 session (technique 2 used in Figure 2) would follow the protocol flow depicted in Figure 4.

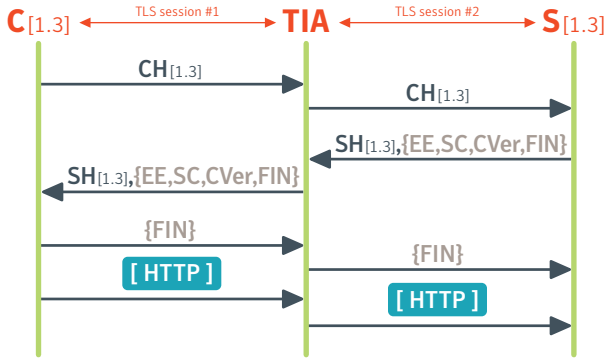

Figure 4 - TLS 1.3 Intercept

TLS 1.3 adds a complication: the CH must include the (EC)DHE public value in the *KeyShare* extension. If the server does not support or does not prefer the algorithm pre-selected by the client, then it will trigger a *HelloRetryRequest* (HRR). For the TIA to use the same (EC)DHE attributes on both TLS sessions (according to the *limiting-modifications* principle), it must also send an HRR to the client, as shown in Figure 5, even if the TIA does support the algorithm selected by the client.

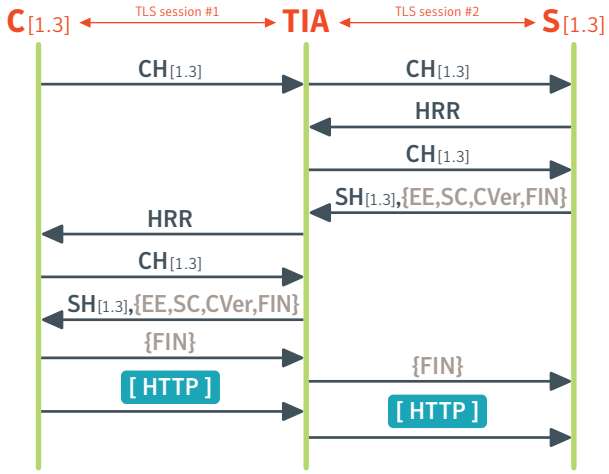

Figure 5 - TLS 1.3 Intercept Requiring HelloRetryRequest

TLS 1.3 Passive Decrypt

TLS 1.3 deprecates the use of RSA key exchange in favor of (EC) DHE, which implies that, for practical purposes, a TIA must be inline to participate in the TLS handshake. One alternative would be to share per-session secret information with one of the TLS endpoints. This sharing would only be practical and useful if the TIA has the cooperation of all the possible endpoints under the legal control of the customer and enterprise – not to mention the infrastructure and security measures needed to safely transfer the information in a mostly non-homogeneous environment. Another alternative that required installing a shared static DH private key was presented to the IETF TLS working group recently[10]. It sparked many discussions and prompted the creation of a document listing the perils of TLS intercept[11], with a bias towards preventing TLS intercept. The argument from enterprise operational teams is that they require a mechanism to aid in troubleshooting, compliance checking, and performance management. More proposals and open debate are required.

TLS 1.3 Downgrade Detection

An attacker would downgrade TLS to a protocol version lower than what is supported by both endpoints to exploit some known vulnerability in the lower version. A TLS 1.3 client would advertise support for TLS 1.3 in the CH and then wait for the SH from the server to indicate which protocol version should be used. The dilemma for the TLS 1.3 client is that it must avoid being tricked into using a lower protocol version. TLS 1.3 attempts to mitigate the protocol downgrade attack vector through a new TLS version downgrade detection mechanism (TLS 1.2 uses the TLS_FALLBACK_SCSV mechanism[12]). A TLS 1.3 server that negotiates TLS 1.2 encodes the word `DOWNGRD` followed by 01_{16} in the last 8 bytes of the SH random field – this is called the downgrade marker. A TLS 1.3 client that receives the marker should abort the handshake with an *illegal_parameter* alert.

A TIA should implement the TLS 1.3 downgrade detection mechanism as well as logic to prevent false aborts, as outlined in Figure 6 through Figure 10. In Figure 6 the TIA downgrades TLS 1.3 to TLS 1.2 for some unspecified reason, e.g., the TIA does **not** yet have support for TLS 1.3. The TLS 1.3 server adds the downgrade marker, but the TIA does not propagate the marker to the TLS 1.3 client. The TIA acts as a TLS 1.2 server on the client side and as a TLS 1.2 client on the server side, which explains why it does not alert (on the server side) or propagate the marker (on the client side).

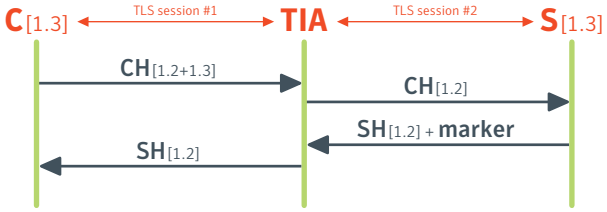

Figure 6 - TLS Downgrade Scenario 1

In Figure 7 the client does not support TLS 1.3. The TIA propagates the server's downgrade marker, but the client ignores the marker and continues with the TLS 1.2 handshake. The TIA acts as a TLS 1.3 server on the client side and as a TLS 1.2 client on the server side – in fact, the TIA should act as a TLS 1.3 server (in how it handles the downgrade marker) even if it does not support intercept of TLS 1.3!

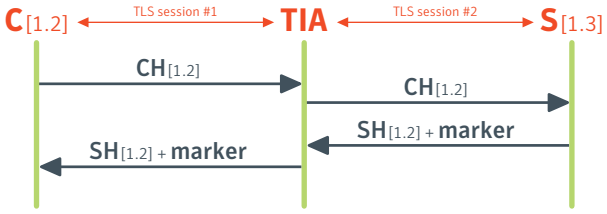

Figure 7 - TLS Downgrade Scenario 2

In Figure 8 the client supports TLS 1.3, but an attacker (MITM) between the TIA and the server attempts to downgrade the session to TLS 1.2 (or lower). The TLS 1.3 server sends the downgrade marker, which then traverses the MITM and reaches the TIA. Since the TIA is acting as a TLS 1.3 client on the server side, it will detect the marker and abort the TLS handshake on both sides.

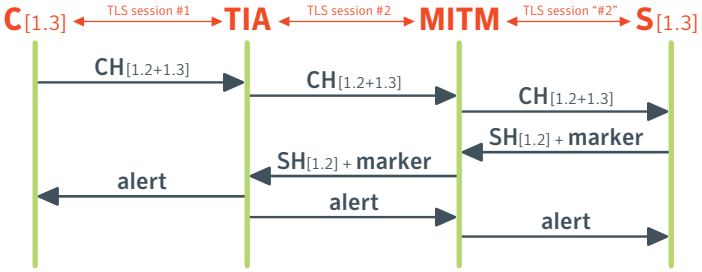

Figure 8 - TLS Downgrade Scenario 3

The attacker (MITM) can also be positioned between the TLS 1.3 client and the TIA, as shown in Figure 9. The client supports TLS 1.3, but the MITM attempts to downgrade the session to TLS 1.2 (or lower). From the viewpoint of the TIA, is looks identical to Figure 7, where the TIA propagates the downgrade marker on the client side. The marker traverses the MITM and reaches the TLS 1.3 client, at which point the client aborts the TLS handshake. The TIA also propagates the alert to the server.

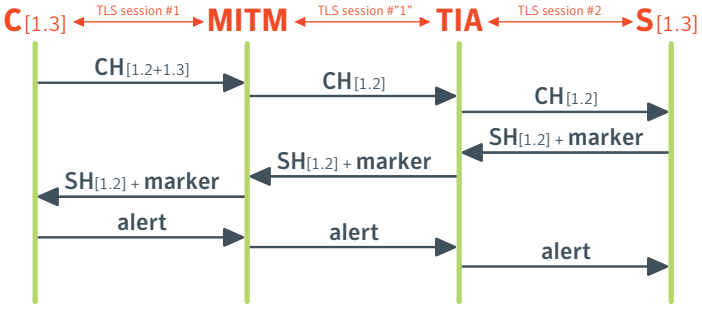

Figure 9 - TLS Downgrade Scenario 4

The scenario in Figure 10, from the point of view of the TIA, is identical to the scenario in Figure 6 where the TIA is responsible for the downgrade from TLS 1.3 to TLS 1.2. The attacker (MITM) is between the client and the TIA, and the MITM is aware of the presence of the TIA. Furthermore, the MITM knows that the TIA does not support TLS 1.3, and it utilizes this to its advantage – the MITM essentially abuses the fact that the TIA will downgrade TLS 1.3 to TLS 1.2. Even worse, the TIA does not propagate the marker (per the logic in the description of Figure 6), which then prevents the client from detecting that the server supports TLS 1.3. This is a very good example of the **security risk of not supporting TLS 1.3**. The same argument can be used to explain why TIAs should support the latest algorithms, cipher-suites, and TLS extensions.

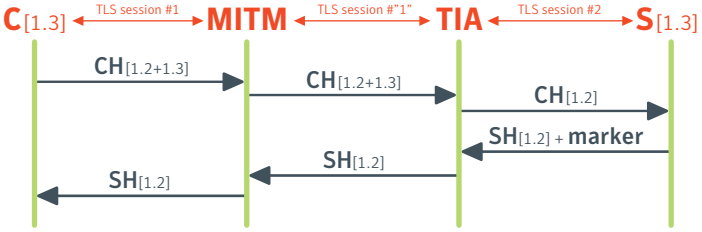

Figure 10 - TLS Downgrade Scenario 5

TLS 1.3 0-RTT Data

TLS 1.3 introduces the concept of 0-RTT data (a.k.a. early data), which is referring to encrypted application level payload that is sent as part of the first flight of data from the client to the server – implying that it does not require any negotiation with the server before it can be sent. The early data is encrypted with traffic keys derived from a pre-shared-key (PSK), and the PSK is typically extracted from a previous TLS 1.3 session, which gives the early data weaker security properties than the application level payload sent after the 1-RTT handshake has completed. There is no delivery guarantee associated with early data because the server might not have access to the PSK needed for decrypt. A TIA that encounters TLS 1.3 early data on TLS session #1 could opt to discard it, but to enhance endpoint application performance, the TIA should forward

the early data to the server on TLS session #2. The TIA should take special care not to erroneously upgrade the security properties of the early data.

Figure 11 depicts a scenario where the TIA can decrypt the early data ($HTTP_1$), but then sends the early data to the server as part of the post-handshake application data, essentially upgrading the security properties of the early data.

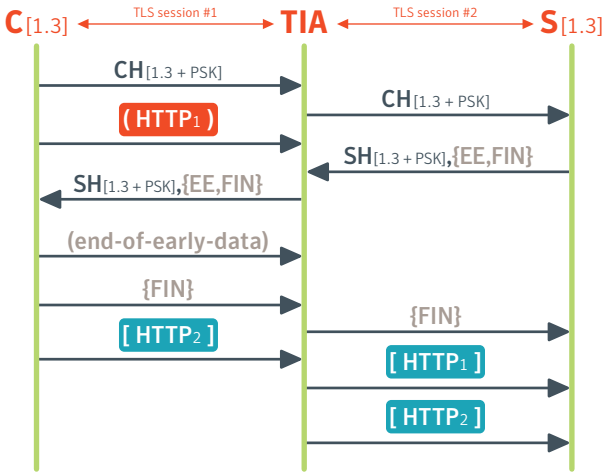

Figure 11 - TLS 1.3 0-RTT Intercept: Erroneous Properties Upgrade

Figure 12 shows what the TIA is supposed to do to retain the security properties of the early data ($HTTP_1$), which is to send the TLS session #1 early data as early data on TLS session #2.

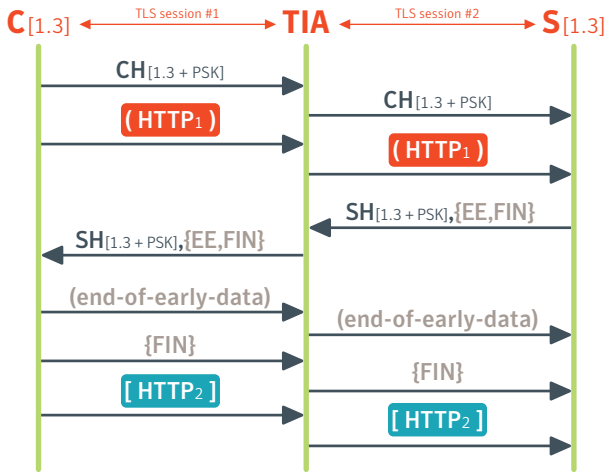

Figure 12 - TLS 1.3 0-RTT Intercept: Retain Properties

There is another complication with TLS 1.3 0-RTT data: if the TIA is used to feed decrypted payload to a third-party security stack, then special markers must be added to demarcate decrypted payload with weaker security properties (the 0-RTT data) from the subsequently decrypted payload. The problem also exists when the TIA performs internal security processing of the decrypted stream – the internal application layer (e.g., HTTP) should still be aware of the demarcation point and should treat the application layer payload with weaker properties differently.

Conclusion

Cryptography is hard, and security is hard. The combination is extra hard, which implies that TLS intercept application developers must have strong attention to detail. TLS 1.3 adds another layer of complexity, which makes intercepting TLS 1.3 even harder. Security vendors, and TLS intercept vendors, in particular must ensure to put enough focus on proper design.

TLS intercept vendors and TLS endpoint application vendors must both act responsibly. Part of that responsibility is to work at paving the way towards closer cooperation between the security industry and the endpoint application industry, while also involving the designers of the TLS (and related) standards.

References

[1] https://www.schneier.com/essays/archives/1997/01/why_cryptography_is.html
[2] CH=ClientHello, SH=ServerHello, SC=ServerCertificate, SHD=ServerHelloDone, etc.
[3] https://tools.ietf.org/html/rfc7457
[4] http://www.isg.rhul.ac.uk/tls/Lucky13.html
[5] https://tools.ietf.org/html/rfc7366
[6] http://heartbleed.com
[7] https://www.xudongz.com/blog/2017/idn-phishing
[8] https://datatracker.ietf.org/wg/tls/charter
[9] The TLS 1.3 specification is still in draft: https://tools.ietf.org/html/draft-ietf-tls-tls13-21
[10] https://tools.ietf.org/html/draft-green-tls-static-dh-in-tls13-01
[11] https://github.com/sftcd/tinfoil
[12] https://tools.ietf.org/html/rfc7507

About Symantec

Symantec Corporation (NASDAQ: SYMC), the world's leading cyber security company, helps organizations, governments and people secure their most important data wherever it lives. Organizations across the world look to Symantec for strategic, integrated solutions to defend against sophisticated attacks across endpoints, cloud and infrastructure. Likewise, a global community of more than 50 million people and families rely on Symantec's Norton and LifeLock product suites to protect their digital lives at home and across their devices. Symantec operates one of the world's largest civilian cyber intelligence networks, allowing it to see and protect against the most advanced threats. For additional information, please visit www.symantec.com or connect with us on Facebook, Twitter, and LinkedIn.

Symantec

350 Ellis St., Mountain View, CA 94043 USA | +1 (650) 527 8000 | 1 (800) 721 3934 | www.symantec.com