Matt Christensen (mrc621)

October 22, 2021

CS 465 (001) – Clift, Frederic M

Project #6 – TLS Report and Analysis

Introduction

Transport Layer Security (or TLS) is the current successor of the Secure Sockets Layer protocol (SSL). The vast majority of the internet employs TLS to establish secure connections, everything from web browsing services (https) to email and instant messaging (pop, imap, smtp, etc.).

TLS can be used by the client to setup a secure connection to the server. By way of a simplified explanation of the TLS three-way handshake – the client sends the server a hello message with random data and a list of cipher suites we want to use. The server responds with its hello message, which consists of random data, the cipher suite it chose, the server's certificates, and a session id. The client finishes with a client done message – consisting of the encrypted pre-master secret key (encrypted with the server's public key) before switching to communicating with encrypt. The server also sends a finished message (changing to speak in the cipher also).

It is also important to realize that between the server hello and client done that the client is responsible for a number of tasks (such as validating the certificate chain, checking for revocation, generating the pre-master secret key, etc.). Additionally, the server can also request certificates from the client (in which case the client would send certificates and the server would validate them).

The purpose of this report is to analyze the specific TLS qualities of various websites – focusing on the difference and similarities between implementations. The way TLS is structured allows for various different implementations – such as switching out cipher suites when a vulnerability is discovered. Additionally, we will analyze three websites trying to make claims to the security properties guaranteed by its implementation.

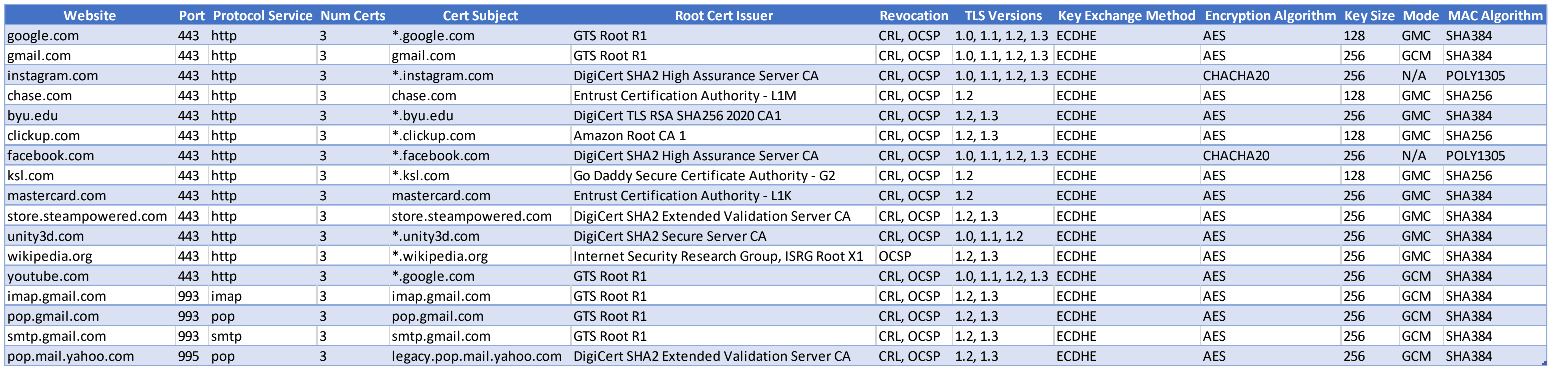
Report

First, it is important to describe the methodology for collecting the data for this study (for finding properties of TLS describe in the table below). Two tool were used to collect the data. First, the “*openssl s_client*” was used (in a Linux environment) as a means of generating debug TLS connection information. The data generation was generated using the following bash script:

```
#! /bin/bash

mkdir -p ./tls_results

echo "Establishing TLS connection with..."

for site in "google.com" "gmail.com" "instagram.com"; do
echo ${site}
timeout 30 openssl s_client -connect ${site}:443 &> ./tls_results/tls_details_for_${site:0:-4}.txt &
done

echo ""

secs=30
while [ $secs -gt 0 ]; do
   echo -ne "$secs secs before timeout \033[0K\r"
   sleep 1
   : $((secs--))
done

echo "Results saved in ./tls_results"
```

Additional information was gathered from the SSL Server Test tool from the SSL Labs website (https://www.ssllabs.com/ssltest/) – such as revocation data and determining all of the supported TLS versions (instead of just the one that was used).

The results of the two aforementioned tests are summarized in the following table for the following seventeen websites:



Website	Port	Protocol Service	Num Certs	Cert Subject	Root Cert Issuer	Revocation	TLS Versions	Key Exchange Method	Encryption Algorithm	Key Size	Mode	MAC Algorithm
google.com	443	http	3	*.google.com	GTS Root R1	CRL, OCSP	1.0, 1.1, 1.2, 1.3	ECDHE	AES	128	GMC	SHA384
gmail.com	443	http	3	gmail.com	GTS Root R1	CRL, OCSP	1.0, 1.1, 1.2, 1.3	ECDHE	AES	256	GCM	SHA384
instagram.com	443	http	3	*.instagram.com	DigiCert SHA2 High Assurance Server CA	CRL, OCSP	1.0, 1.1, 1.2, 1.3	ECDHE	CHACHA20	256	N/A	POLY1305
chase.com	443	http	3	chase.com	Entrust Certification Authority - L1M	CRL, OCSP	1.2	ECDHE	AES	128	GMC	SHA256
byu.edu	443	http	3	*.byu.edu	DigiCert TLS RSA SHA256 2020 CA1	CRL, OCSP	1.2, 1.3	ECDHE	AES	256	GMC	SHA384
clickup.com	443	http	3	*.clickup.com	Amazon Root CA 1	CRL, OCSP	1.2, 1.3	ECDHE	AES	128	GMC	SHA256
facebook.com	443	http	3	*.facebook.com	DigiCert SHA2 High Assurance Server CA	CRL, OCSP	1.0, 1.1, 1.2, 1.3	ECDHE	CHACHA20	256	N/A	POLY1305
ksl.com	443	http	3	*.ksl.com	Go Daddy Secure Certificate Authority - G2	CRL, OCSP	1.2	ECDHE	AES	128	GMC	SHA256
mastercard.com	443	http	3	mastercard.com	Entrust Certification Authority - L1K	CRL, OCSP	1.2	ECDHE	AES	256	GMC	SHA384
store.steampowered.com	443	http	3	store.steampowered.com	DigiCert SHA2 Extended Validation Server CA	CRL, OCSP	1.2, 1.3	ECDHE	AES	256	GMC	SHA384
unity3d.com	443	http	3	*.unity3d.com	DigiCert SHA2 Secure Server CA	CRL, OCSP	1.0, 1.1, 1.2	ECDHE	AES	256	GMC	SHA384
wikipedia.org	443	http	3	*.wikipedia.org	Internet Security Research Group, ISRG Root X1	OCSP	1.2, 1.3	ECDHE	AES	256	GMC	SHA384
youtube.com	443	http	3	*.google.com	GTS Root R1	CRL, OCSP	1.0, 1.1, 1.2, 1.3	ECDHE	AES	256	GCM	SHA384
imap.gmail.com	993	imap	3	imap.gmail.com	GTS Root R1	CRL, OCSP	1.2, 1.3	ECDHE	AES	256	GCM	SHA384
pop.gmail.com	993	pop	3	pop.gmail.com	GTS Root R1	CRL, OCSP	1.2, 1.3	ECDHE	AES	256	GCM	SHA384
smtp.gmail.com	993	smtp	3	smtp.gmail.com	GTS Root R1	CRL, OCSP	1.2, 1.3	ECDHE	AES	256	GCM	SHA384
pop.mail.yahoo.com	995	pop	3	legacy.pop.mail.yahoo.com	DigiCert SHA2 Extended Validation Server CA	CRL, OCSP	1.2, 1.3	ECDHE	AES	256	GCM	SHA384

Similarities

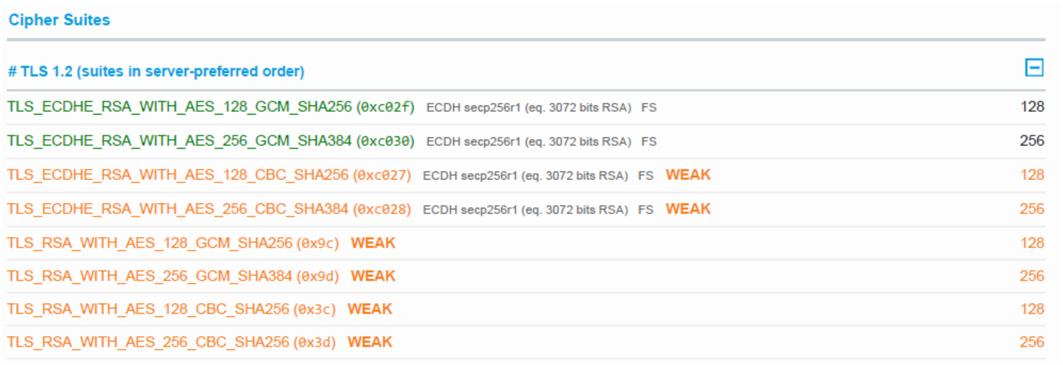
One immediate similarity across the data is the prevalence of block ciphers across companies and websites. All but two websites us a block cipher implementation for encryption – with the exception being ChaCha20 with Poly1305. Perhaps one reason that we see such a prevalence of block ciphers is because of the uniform length/size of data, where block ciphers can be more efficient. The ChaCha20 implementation is used for both Facebook and Instagram, which are owned by the same company. ChaCha20 provides unique protection from timing attacks because of the speed of the cipher (which consists of simple additions, fixed rotations, and XOR operations).

Another obvious similarity is the consistent use of a certificate revocation list (CRL) and Online Certificate Status Protocol (OCSP). OCSP offers a real-time protocol to check the revocation status of a given certificate while CRL provides a simple list of revoked certificates that typically expires every twenty-four hours (or less). The CRL is conventionally provided by the root certificate authority (CA), however it can also be provided by a different trusted source. The only outlier is Wikipedia.org, which only reports CRL technology, however this may be reporting error.

Additionally, it is interesting to see that websites across a given company share implementation details. For example, all Google owned websites (Google, YouTube, Gmail) all support TLS versions 1.0, 1.1, 1.2, and 1.3 as well as all sharing the same certificate, and cipher (TLS_ECDHE_AES_256_GCM_SHA384). I found it interesting to see that Google would use the same certificate for YouTube and Gmail for example. In my mind, if the certificate gets compromised then all websites' identities would also be compromised. It is interesting to see how security implementation is standardized across the different platform development teams – implementing a security standard.

Another similarity between all but one website is the lack of RC4 use. The Unity3D website however does include two weak cipher options in the list of acceptable ciphers that include RC4 as the encryption algorithm (TLS_RSA_WITH_RC4_128_SHA and TLS_RSA_WITH_RC4_128_MD5). RC4 has known variabilities, and as such its inclusion makes the list of cipher suites vulnerable.

Finally, it is interesting that every single site chose to use ECDHE as the key exchange method for the cipher suite. This make sense, because the browser and server always seek to pick the highest listed cipher suite possible and ECDHE is currently the industry standard. It should be noted however, that many times the server does have cipher suite options what don't include ECDHE (as noted in the figure below, which shows the potential cipher suites of Chase.com).



Cipher Suites

TLS 1.2 (suites in server-preferred order)

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)	ECDH secp256r1 (eq. 3072 bits RSA)	FS		128
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)	ECDH secp256r1 (eq. 3072 bits RSA)	FS		256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	WEAK	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028)	ECDH secp256r1 (eq. 3072 bits RSA)	FS	WEAK	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c)			WEAK	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d)			WEAK	256
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c)			WEAK	128
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d)			WEAK	256

Differences

The immediately evident difference across the sampled data is the variance in the supported TLS versions.

Version	Count
1.2, 1.3	8
1.0, 1.1, 1.2, 1.3	5
1.2	3
1.0, 1.1, 1.2	1

As seen in the figure above, the majority of implementations that were surveyed support 1.2 and 1.3. All non-http services use just the 1.2 and 1.3 versions, so excluding these results from our data we would have four sites that use said versions. Although supporting 1.2 and 1.3 and v1.0 - v1.3 are the most common we still see a fair amount of site only use 1.2 and one site use v1.0 - v1.2 (Unity3d). Perhaps we see a large distribution because of a desire to provide the most access possible. It should be noted however that forward secrecy is only included in TLS versions 1.2 and 1.3.

Additionally, another notable difference is that the subjects of the certificates are not always wildcards. Many of the certificates have wildcards in the alternate names if it doesn't appear as the subject. However, there does not appear to be any industry standard in where the wildcard appears.

Additionally, the non-http services seem distinct from http websites. All sampled non-http services (pop, imap, and smpt) all use TLS versions 1.2 and 1.3 in addition to sharing the same selected cipher suite (TLS_ECDHE_AES_256_GCM_SHA384).

Finally, we see a wide variety in certificate root authorities (such as DigiCert, Amazon, Google, GoDaddy, etc.). This is reasonable, as each company chooses to get their certificates from different companies for various reasons. Additionally, spreading the responsibility of issuing certificates across multiple companies is intuitively a good idea – as if one company is breached and compromised not all certificates are unsafe.

Case Study

We will now compare three different connection implementations and attempt to reason about the cryptographic guarantees of the given sample. The summarized details can be found in the table included below:

	Key Exchange Method	Encryption Algorithm	MAC Algorithm	TSL Version	X509 Certificate
Google.com	ECDHE	AES	SHA384	1.0, 1.1, 1.2, 1.3	RSA Digital Signature
Store.SteamPowered.com	ECDHE	AES	SHA384	1.2, 1.3*	RSA Digital Signature
Instagram.com	ECDHE	CHACHA20	POLY1305	1.0, 1.1, 1.2, 1.3	RSA Digital Signature
Security Property	Privacy/Confidentiality	Privacy/Confidentiality	Integrity and Authentication	*Forward Secrecy	Non-Repudiation

Google.com

First, we will analyze Google's TLS implementation. Google uses ECDHE for the Key Exchange Method, which guaranties the cryptographic properties of Privacy/Confidentiality by exchanging secrets successfully (guarding against passive attacks). AES is used as the Symmetric Encryption Cipher to the extent that Privacy/Confidentiality is guaranteed because of the ability to encrypt our data safely. The MAC Algorithm used is SHA384, which guarantees Integrity and Authentication by being able to detect when the message is modified after being sent and authenticating the validity of the sender. However, because the TLS 1.0 and 1.1 versions are including in the list of possible ciphers suites Forward Secrecy is not guaranteed.

Store.SteamPowered.com

Next, we take a look at the Steam implementation of TLS. Steam also uses ECDHE for the Key Exchange Method, which guaranties the cryptographic properties of Privacy/Confidentiality by exchanging secrets successfully (guarding against passive attacks). Additionally, AES is used as the Symmetric Encryption Cipher such that Privacy/Confidentiality is guaranteed because of the ability to encrypt our data safely. The MAC Algorithm used is SHA384, which guarantees Integrity and Authentication by being able to detect when the message is modified after being sent and authenticating the validity of the sender. Also, because the only TLS versions specified by the server are 1.2 and 1.3 Forward Secrecy *is* guaranteed. Finally, non-repudiation from the server is guaranteed because of the RSA Digital Signature from the X509 certificate.

Instagram.com

Finally, we take a look at the Instagram TLS implementation. Instagram also uses ECDHE for the Key Exchange Method, which guaranties the cryptographic properties of Privacy/Confidentiality by exchanging secrets successfully (guarding against passive attacks). However, ChaCha20 is used as the Symmetric Encryption Cipher such that Privacy/Confidentiality is guaranteed because of the ability to encrypt our data safely. Additionally, with ChaCha20 we are protected from timed attacks (as discussed and explained previous in this paper). The MAC Algorithm used is Poly1305, which guarantees Integrity and Authentication by being able to detect when the message is modified after being sent and authenticating the validity of the sender. However, because the TLS 1.0 and 1.1 versions are including in the list of possible ciphers suites Forward Secrecy is not guaranteed. Also, non-repudiation (just like Google and Steam) from the server is guaranteed because of the RSA Digital Signature from the X509 certificate.

Conclusion

In conclusion, we have seen how varied and complex TLS implementations can be. Because of the wide variety of implementation options possible with TLS it can be easy to include a possible configuration that is vulnerable. Additionally, the plethora of options can cause information overload and decision fatigue, causing developers to compromise their TLS implementation. It is important to understand the basic principles of TLS inn order to avoid such mistakes. Also, non-repudiation from the server is guaranteed because of the RSA Digital Signature from the X509 certificate.

Appendix

Questions

I do not understand the part about the server name extension, both how that saves IPv4 addresses and why that is needed.

I am also fuzzy on the understanding of how the master key was generated (with the two halves being separately hashed and the PRF function).

Additionally, does the server send the master key back to the client? Or does the client generate it from the pre-master key?