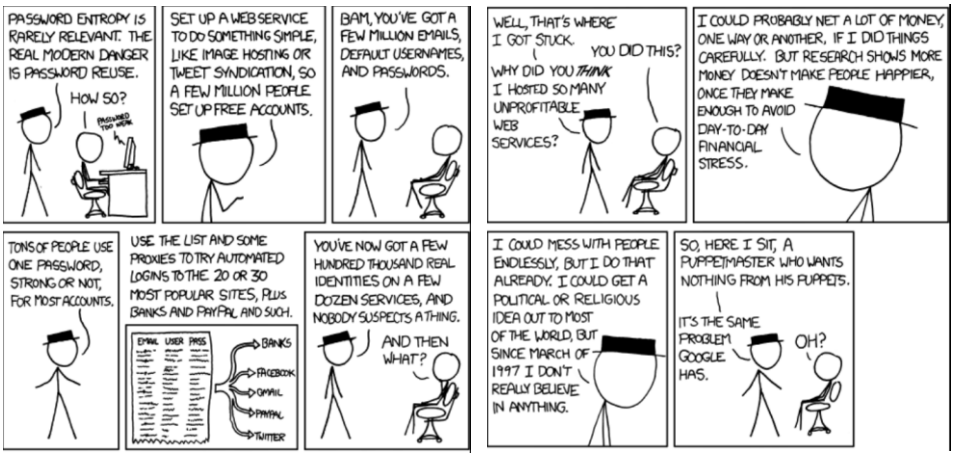
Cryptology (10)

Secure Browsing, HTTPS and TLS

John York, Blue Ridge Community College
Weyers Cave, VA
http://www.brcc.edu

Obligatory XKCD Cartoon



Off-topic, but very important…

Password reuse is a bad thing. When a major site it breached, its password hashes are often released on the web. People crack those hashes and then publish the usernames and passwords. Then people try the username/password combinations on any site that has monetary value (think Amazon, eBay, banking sites.)

So if you use the same username and password at both your chat site and your bank, if your chat site is compromised it is possible that your bank account will be compromised as well.

This happens so often that the reuse of stolen/cracked passwords has been mistaken for a breach. If site A is compromised, the attackers will try all the stolen credentials on site B. It will appear that site B has also been breached because so many people reuse their passwords.

Don't reuse passwords!

Use a password manager to keep unique, random passwords for all your accounts.

Password reuse really is a bad thing.

https://xkcd.com/792/

"It'll be hilarious the first few times this happens."

Transport Layer Security (TLS)

- Secure Sockets Layer (SSL) was original encryption for browsers
  - SSL now deprecated
  - The term SSL is often used to mean both TLS and SSL
- Current browsers use TLS 1.2, beginning to move to TLS 1.3
  - TLS 1.3 was approved March 2018
- TLS is a suite of several protocols, used for different reasons
  - Browser and server negotiate which suite they will use
- TLS provides encryption while the data is in transit
- TLS does not provide encryption while the data is at rest (ie., on your computer or the server)

The SSL protocol has been out of use for a few years, but the term is still used to refer to encrypted web traffic.

TLS 1.2 is the most common protocol, but it has been replaced by TLS 1.3. TLS 1.3 includes improvements for both security and speed, so it should soon be the dominant protocol.

It is important to note that TLS only provides encryption while the data is on the network. Once it reaches your computer, your computer decrypts the data and could well store the data unencrypted. You would have to use something else to store the data in an encrypted format.

Data in transit: data being sent over the network
Data at rest: data sitting on your computer

TLS is used by services other than web

- Some Virtual Private Network (VPN) clients are built on TLS
- FTPS is a secure version of FTP that runs over TLS/SSL
  - SFTP is FTP running over SSH
- Messages between email servers (SMTP) can use TLS (STARTTLS)
- Since TLS libraries are freely available, many services use them

Since TLS libraries are available commercially and open source, TLS has been incorporated into many protocols.

Here is a listing of TLS libraries.
https://en.wikipedia.org/wiki/Comparison_of_TLS_implementations

TLS Handshakes and negotiation

- Normally, the server presents a digital certificate to authenticate (in some cases, clients also authenticate with a certificate)
- TLS Handshake (simplified somewhat)
  - Client Hello. Client sends a list of cipher suites it supports
  - Server Hello. Server selects a suite from client list (or ends connection)
  - Server Certificate. Server presents its digital certificate to the browser
  - Server Key Exchange. Server's info to create shared key, depends on cipher suite, may be omitted if not Diffie-Hellman
  - Server Hello Done. (Certificate, Exchange, and Done may be in same packet)
  - Client Key Exchange. Client's info to create shared key
  - Change Cipher Spec. Sent by both sides, says "We're going encrypted now."

The TLS handshake must accomplish four things:

1) Client and server must agree on a suite of protocols that they will use for key exchange, hashing, message authentication codes, and the symmetric encryption they will use (see next slide.)
2) Server (and sometimes client) authenticate with a digital certificate
3) Client and server establish a session key that will be used for symmetric encryption
4) Client and server switch to symmetric encryption

The messages shown in the slide are versions used in TLS 1.2. TLS 1.3 accomplishes the same essential steps https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/ but compresses them into three messages so the handshake is much quicker.

It is very useful for network analysts to be able to view the server certificates during the TLS handshake. That way they can determine who the traffic is bound for, even if they cannot decrypt it. The certificate is now encrypted in TLS 1.3. Additionally, TLS 1.3 allows clients and servers to bypass the handshake if they have communicated recently and still have copies of the keys from the previous session. Both of these changes make life harder for network analysts, although they do improve security.

TLS Cipher Suites

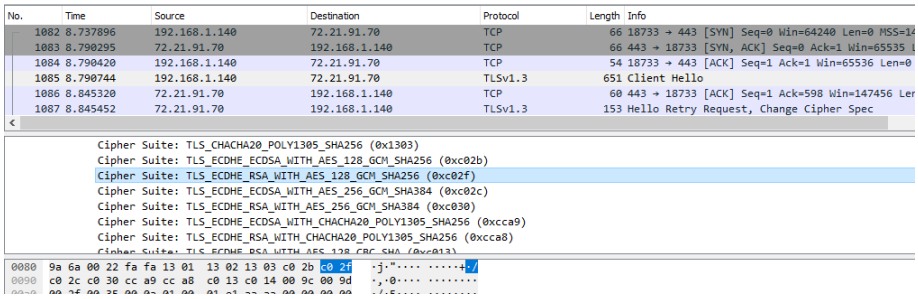
- Different functions need different cipher protocols
- Key Exchange
  - Usually Diffie-Hellman (DH), Elliptic Curve DH (ECDH), or RSA
- Digital certificates
  - Most often RSA, also DSA and ECDSA, must use SHA-256 or better as hash
- Symmetric Block Cipher (primary data exchange)
  - AES-GCM (AES in Galois Counter Mode) is common
- Hash Method
  - often SHA-256 or better
- Data Integrity (Message Authentication Code (MAC))
  - HMAC or AEAD

TLS uses many encryption protocols, and they are grouped together in suites. The protocols are used for different portions of the connection, and the purposes are shown in the slide. When your browser negotiates a cipher suite with a server, it just gives the server the number of the suite; it doesn't specify the individual components.

TLS 1.3 made major changes to the cipher suites to improve security. RSA is no longer allowed for key exchange, although it is still allowed for digital certificates/authentication. Also, many methods for computing MACs are no longer allowed. Some of the methods in TLS 1.2 were subject to padding, or oracle attacks.

Generally the browser, web server, or operating system should disallow older TLS versions and cipher suites as new ones become available and older ones become vulnerable. They are sometimes slow to update, so administrators may have to do this manually.
https://www.cloudinsidr.com/content/tls-1-3-and-tls-1-2-cipher-suites-demystified-how-to-pick-your-ciphers-wisely/

TLS example—Client Hello

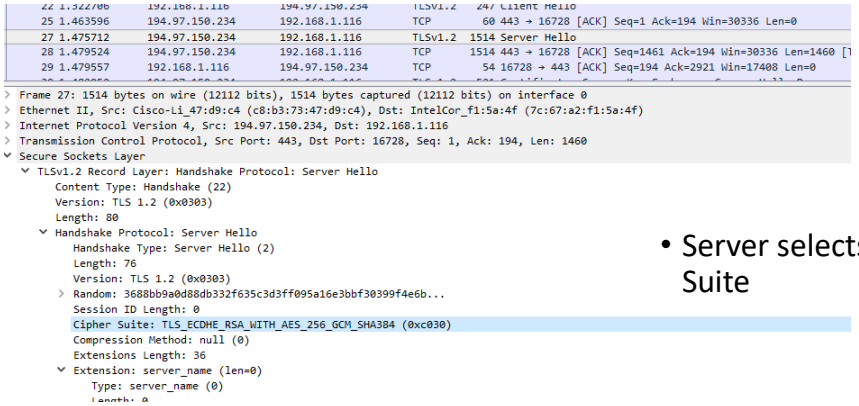


- Client lists Cipher Suites

In the client hello, the client lists the cipher suites it can use. Each suite has a number (0xc02f for the one highlighted in the packet capture.) That suite uses:

1. TLS as the overall protocol
2. Elliptic Curve Diffie-Hellman (ECDHE) for key exchange
3. RSA for certificates/authentication
4. AES 128 with Galois Counter Mode (AES128_GCM) for both symmetric encryption and message authentication codes (MAC). When GCM is added to AES, both the encryption and MAC are computed simultaneously
5. SHA256 for hashes.

TLS example—Server Hello



- Server selects Cipher Suite

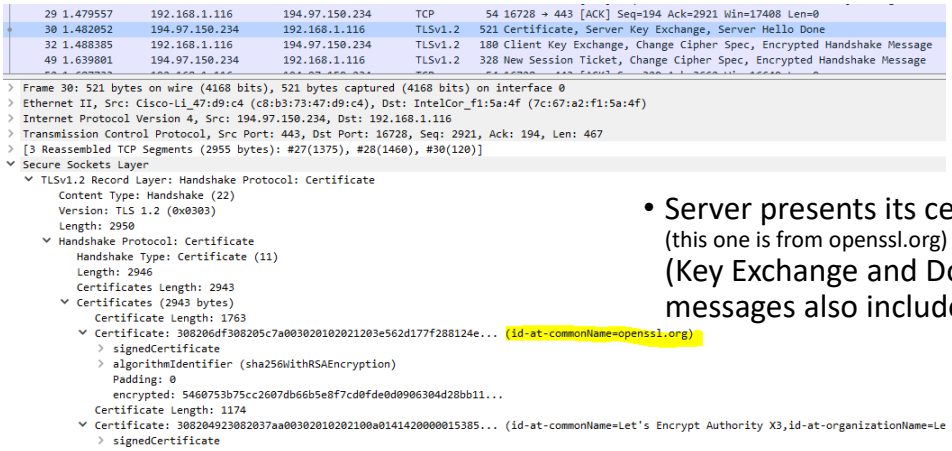
TLS Example—Cipher Suite

```
Version: TLS 1.2 (0x0303)
> Random: 3688bb9a0d88db332f635c3d3ff095a16e3bbf30399f4e6b...
Session ID Length: 0
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
```

- Overall—TLS 1.2
- Key exchange—ECDHE, Elliptic Curve Diffie-Hellman Exchange
- Certificate—RSA
- Symmetric encryption—AES256 with Galois Counter Mode
  - GCM includes authenticity checking
- Hash--SHA-384
- Note: the random number is used for Diffie-Hellman key generation

Here is another sample cipher suite that the client and server may choose.

TLS example—Server Certificate



- Server presents its cert
  (this one is from openssl.org)
  (Key Exchange and Done messages also included)

This portion of the handshake is unencrypted in TLS 1.2, and provides useful information for network analysts. Unfortunately it is encrypted in TLS 1.3 and we won't be seeing it much longer.

Server certificate is important

- Most traffic is now encrypted and cannot normally be examined by Intrusion Prevention System
  - Traffic with RSA encryption may be examined if server private key is known
  - Diffie Hellman Encryption monitoring requires proxy and Man in the Middle attack (MITM), or for the browser to save the keys as it goes
- Server certificate is a good indication of use
- Examining server certificates may be very useful in detecting malicious traffic

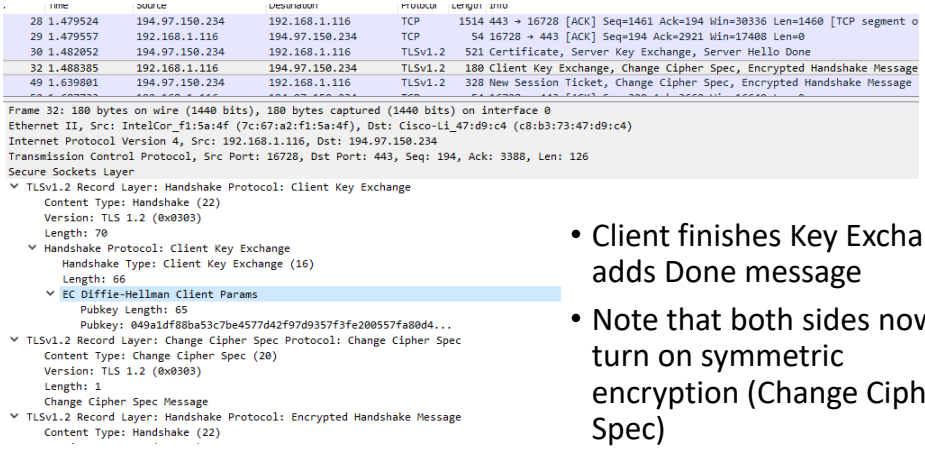
Unfortunately, it has become harder to monitor server certificates in TLS 1.3 since the certificate is now in the encrypted portion of the message.

While TLS 1.2 is still in use, monitoring the server certificates is still a good method for gaining general information about the traffic.

The most effective method for an organization to monitor its traffic is to require all web traffic to go through a proxy server, and to require all hosts in the organization to trust the certificate assigned to the proxy server. Then the proxy server can decrypt the outbound traffic, examine it for malware, and then establish a new encrypted session with the destination. This is essentially performing a Man in the Middle (MITM) attack against your users, but is often necessary since so much malware uses encrypted communications.

Using a proxy server or Next Generation Firewall (NGFW) can be expensive, however. It takes powerful hardware to do all this decryption and encryption for a high bandwidth link.

TLS example—Client Key Exchange, Done



- Client finishes Key Exchange, adds Done message
- Note that both sides now turn on symmetric encryption (Change Cipher Spec)

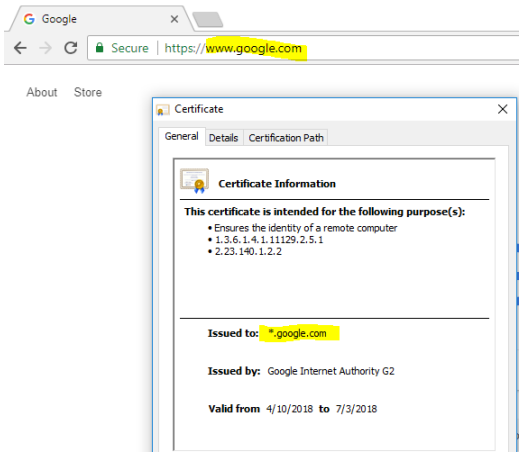
This is an example from a TLS 1.2 handshake. The TLS 1.3 handshake has been shortened, and I find the TLS 1.2 handshake easier to understand. The essential components are present in both versions:

1. Agree on a cipher suite (in TLS 1.3, the client tries to guess what suite the server will prefer to shorten the process.)
2. Authenticate--verify that the server certificate is correct. In some implementations the client also authenticates with its own certificate. Certificates are often RSA, but versions of DSA and Diffie-Hellman are also used.
3. Exchange a key for symmetric encryption. In TLS 1.3, this is usually done with a version of Diffie-Hellman. In TLS 1.2, RSA is often used.
4. Switch to symmetric encryption.

Note that every packet in symmetric encryption is authenticated and integrity checked with a MAC, either GCM or by a separate computation like HMAC. This detects corrupted packets and prevents adversaries from inserting packets in an attempt to break the encryption. MACs use symmetric keys, which means we can no longer prove which side sent the packet (non-repudiation), but it is no longer necessary. Non-repudiation was established at the beginning of the connection with digital certificates. MAC is much faster, so is use the authenticate individual packets once the connection is established.

Browser checks server certificate

- Common Name (CN) or Subject Alternate Name (SAN) on cert must match the DNS name in browser navigation bar. Note: "*" is a wild card
- If they do not match, the browser generates an error
- If HTTP Strict Transport Security (HSTS) is set on the server, user cannot click through the warning
  - Makes MITM attacks more difficult



The browser performs several checks that we discussed in the last lesson. Here is a partial list:

1. The DNS name in the browser's navigation bar must match the name in either the Subject (Common Name, or CN) or in one of the Subject Alternative Names (SANs)
2. The certificate must be issued by a Certificate Authority (CA) that the browser or OS trusts.
3. The certificate must not be expired (the current date is within the validity dates on the certificate)
4. The certificate must use current protocols and hashes (certificates with SHA-1 hashes are disallowed, for example.)

HTTP Strict Transport Security (HSTS) is designed to prevent a MITM attacker from intercepting your communications before your browser switches from HTTP to HTTPS (SSL-strip attack.) If your browser has seen the desired site previously and the site uses the HSTS header, the browser has cached that information. In later connections, the browser knows it is supposed to use HTTPS and the valid certificate for that site. It will not allow the user to click through warnings if the attacker tries to keep the connection or HTTP, or uses a forged certificate.

Forward Secrecy, RSA vs Diffie-Hellman

- RSA uses public and private keys to create secure channel
  - After channel is created, symmetric key is chosen
- If the server's private RSA key is known, the channel can be decrypted on any stored packets and any packets in the future (forward)
- In Diffie-Hellman, each side contributes a piece to the key exchange
  - The key $(\alpha^b)^a = \alpha^{ab} \bmod p$ is different every session
- With DH, the entire key is not in the traffic, so stored traffic cannot be decrypted
- Sometimes called "Perfect" Forward Secrecy (PFS)
- TLS v. 1.3 requires PFS, ie. no RSA key exchange, only DH and ECDH

In RSA, the server's private key is used to create the session key that will be used for the rest of the connection (asymmetric, AES for example.) If the network traffic is recorded, anyone who has the server's private key can recreate the session key and decrypt both sides of the traffic.

In Diffie-Hellman each side contributes to the key exchange with their private keys. Additionally, new private keys are selected for each new connection. The result is that you cannot recreate the session key by examining the traffic, even if you know one side's private key. Thus DH has Forward Secrecy because recorded sessions cannot be decrypted.

There are two ways to decrypt DH key exchange traffic. One is to configure your browser to record the session keys it uses (SSL session key logging.) Then you can decrypt traffic to and from your browser. It does not help you decrypt other people's traffic, however. The second method is to require your users to browse through a proxy server, and you require them to trust the proxy server's certificate. The is the MITM attack we discussed before.
https://redflagsecurity.net/2019/03/10/decrypting-tls-wireshark/