SSL Renegotiation Attacks

Related PCAP File(s):

- `SSL_renegotiation_edited.pcapng`

Although HTTP traffic is unencrypted, we sometimes will run into encrypted HTTPs traffic. As such, knowing the indicators and signs of malicious HTTPs traffic is crucial to our traffic analysis efforts.

HTTPs Breakdown

Unlike HTTP, which is a stateless protocol, HTTPs incorporates encryption to provide security for web servers and clients. It does so with the following.

1. `Transport Layer Security (Transport Layer Security)`
2. `Secure Sockets Layer (SSL)`

Generally speaking, when a client establishes a HTTPs connection with a server, it conducts the following.

1. `Handshake:` The server and client undergo a handshake when establishing an HTTPs connection. During this handshake, the client and server agree upon which encryption algorithms to use, and exchange their certificates.
2. `Encryption`: Upon completion of the handshake, the client and the server use the prior agreed upon encryption algorithm to encrypt further data communicated between them.
3. `Further Data Exchange:` Once the encrypted connection is established, the client and the server will continue to exchange data between each other. This data could be web pages, images, or other web resources.
4. `Decryption:` When the client transmits to the server, or the server transmits to the client, they must decrypt this data with the private and public keys.

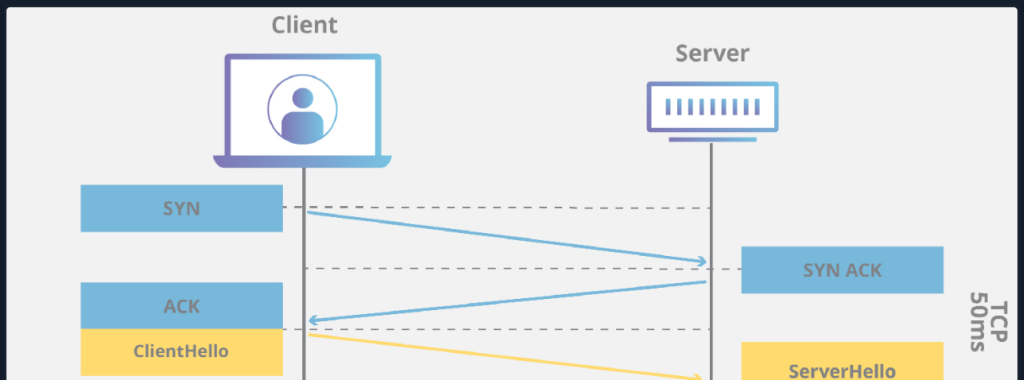
As such, one of the more common HTTPs based attacks are SSL renegotiation, in which an attacker will negotiate the session to the lowest possible encryption standard.

However there are other encryption attacks we should be aware of like the `heartbleed vulnerability`

The Heartbleed Vulnerability CVE-2014-0160

TLS and SSL Handshakes

In order to establish an encrypted connection, the client and server must undergo the handshake process. Fortunately for us, TLS and SSL handshakes are mostly similar in their steps.



To break it down further, we might observe the following occur during our traffic analysis efforts.

1. `Client Hello` - The initial step is for the client to send its hello message to the server. This message contains information like what TLS/SSL versions are supported by the client, a list of cipher suites (aka encryption algorithms), and random data (nonces) to be used in the following steps.
2. `Server Hello` - Responding to the client Hello, the server will send a Server Hello message. This message includes the server's chosen TLS/SSL version, its selected cipher suite from the client's choices, and an additional nonce.
3. `Certificate Exchange` - The server then sends its digital certificate to the client, proving its identity. This certificate includes the server's public key, which the client will use to conduct the key exchange process.
4. `Key Exchange` - The client then generates what is referred to as the premaster secret. It then encrypts this secret using the server's public key from the certificate and sends it on to the server.
5. `Session Key Derivation` - Then both the client and the server use the nonces exchanged in the first two steps, along with the premaster secret to compute the session keys. These session keys are used for symmetric encryption and decryption of data during the secure connection.
6. `Finished Messages` - In order to verify the handshake is completed and successful, and also that both parties have derived the same session keys, the client and server exchange finished messages. This message contains the hash of all previous handshake messages and is encrypted using the session keys.
7. `Secure Data Exchange` - Now that the handshake is complete, the client and the server can now exchange data over the encrypted channel.

We can also look at this from a general algorithmic perspective.

Handshake Step	Relevant Calculations
`Client Hello`	`ClientHello = { ClientVersion, ClientRandom, Ciphersuites, CompressionMethods }`
`Server Hello`	`ServerHello = { ServerVersion, ServerRandom, Ciphersuite, CompressionMethod}`
`Certificate Exchange`	`ServerCertificate = { ServerPublicCertificate }`
`Key Exchange`	• `ClientDHPrivateKey` • `ClientDHPublicKey = DH_KeyGeneration(ClientDHPrivateKey)` • `ClientKeyExchange = { ClientDHPublicKey }` • `ServerDHPrivateKey` • `ServerDHPublicKey = DH_KeyGeneration(ServerDHPrivateKey)` • `ServerKeyExchange = { ServerDHPublicKey }`
`Premaster Secret`	• `PremasterSecret = DH_KeyAgreement(ServerDHPublicKey, ClientDHPrivateKey)` • `PremasterSecret = DH_KeyAgreement(ClientDHPublicKey, ServerDHPrivateKey)`
`Session Key Derivation`	`MasterSecret = PRF(PremasterSecret, "master secret", ClientNonce + ServerNonce)`

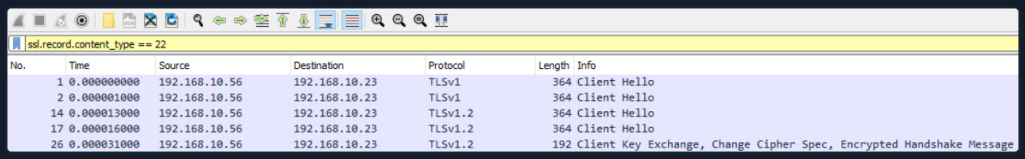
	KeyBlock = PRF(MasterSecret, "key expansion", ServerNonce + ClientNonce)
Extraction of Session Keys	• ClientWriteMACKey = First N bytes of KeyBlock • ServerWriteMACKey = Next N bytes of KeyBlock • ClientWriteKey = Next N bytes of KeyBlock • ServerWriteKey = Next N bytes of KeyBlock • ClientWriteIV = Next N bytes of KeyBlock • ServerWriteIV = Next N bytes of KeyBlock
Finished Messages	FinishedMessage = PRF(MasterSecret, "finished", Hash(ClientHello + ServerHello))

Diving into SSL Renegotiation Attacks

In order to find irregularities in handshakes, we can utilize TCP dump and Wireshark as we have done before. In order to filter to only handshake messages we can use this filter in Wireshark.

- `ssl.record.content_type == 22`

The content type 22 specifies handshake messages only. Specifying this filter we should get a view like the following.



When we are looking for SSL renegotiation attacks, we can look for the following.

1. `Multiple Client Hellos` - This is the most obvious sign of an SSL renegotiation attack. We will notice multiple client hellos from one client within a short period like above. The attacker repeats this message to trigger renegotiation and hopefully get a lower cipher suite.
2. `Out of Order Handshake Messages` - Simply put, sometimes we will see some out of order traffic due to packet loss and others, but in the case of SSL renegotiation some obvious signs would be the server receiving a client hello after completion of the handshake.

An attacker might conduct this attack against us for the following reasons

1. `Denial of Service` - SSL renegotiation attacks consume a ton of resources on the server side, and as such it might overwhelm the server and cause it to be unresponsive.
2. `SSL/TLS Weakness Exploitation` - The attacker might attempt renegotiation to potentially exploit vulnerabilities with our current implementation of cipher suites.
3. `Cryptanalysis` - The attacker might use renegotiation as a part of an overall strategy to analyze our SSL/TLS patterns for other systems.

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

+ 1 Inspect the SSL_renegotiation_edited.pcapng file, part of this module's resources, and enter the total count of "Client Hello" requests as your answer.

16

Submit

Previous Next

Mark Complete & Next