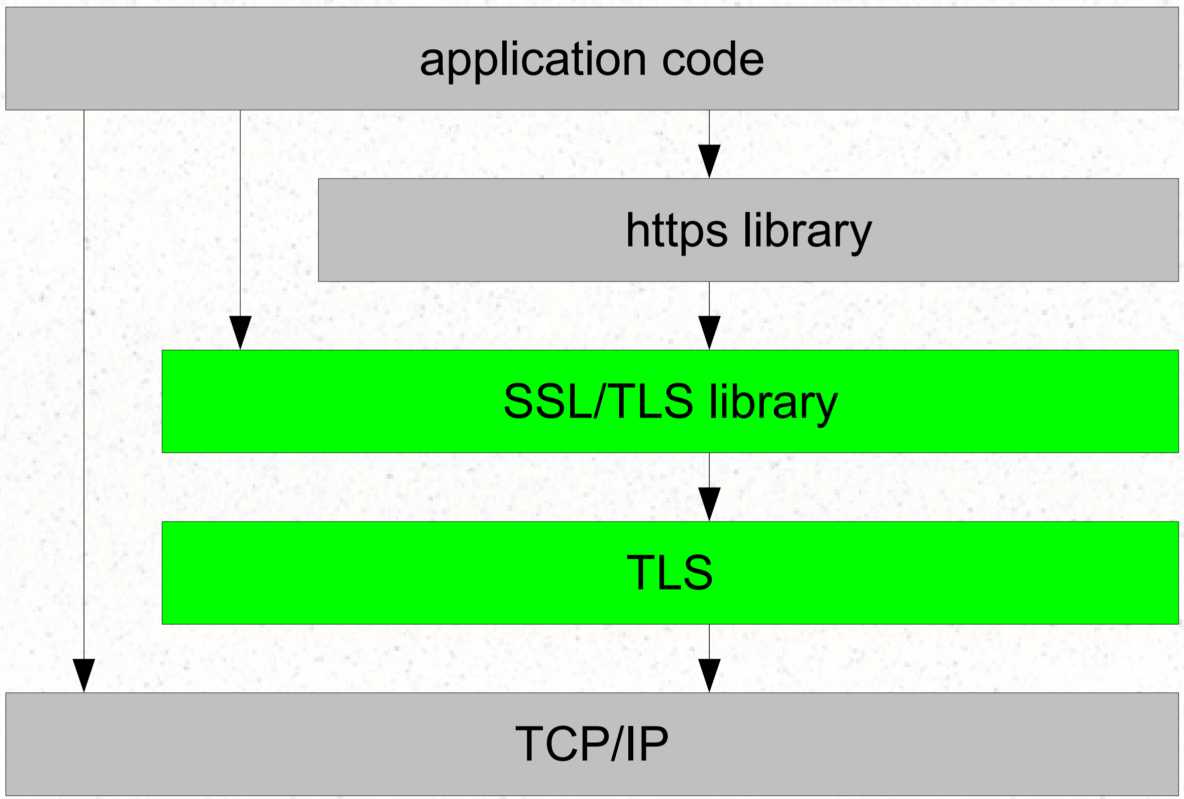
History of the TLS Authentication Gap Bug

Marsh Ray
Steve Dispensa
PhoneFactor

Customary Protocol Stack Diagram



TLS Details

- Exploitable MitM attack results from authentication gap in renegotiation
- TLS overview
- Discovery, demo, details
- Vulnerable code
- Fixes

SSL

- The "Secure Sockets Layer"!
- Originated with Netscape in 1994
- Version 1 not released publicly

SSLv2 Spec "0.2"

- First shipped version
- Spec revised a few times!
  - November 29, 1994
  - December 22, 1994
  - January 17, 1995
  - January 24, 1995
  - February 9, 1995

SSLv2 Spec "0.2"

- Basic handshake From SSL 0.2 Protocol Spec
  - C -> S: client-hello challenge, cipher_specs
  - S -> C: server-hello conn-id,server_certificate,cipher_specs
  - C -> S: client-master-key {master_key}server_public_key
  - C -> S: client-finish   {connection-id}client_write_key
  - S -> C: server-verify   {challenge}server_write_key
  - S -> C: server-finish   {new_session_id}s_write_key

SSLv2 Spec "0.2"

- Uses only MD5 for PRF and record data MACs
- Client Finished message is just an echo of the connection-id from the Server Hello
  - which was just sent in plaintext
- Server Finished message is the session-id-data
  - client has no way to validate it
- MitM can freely manipulate many fields in the handshake

SSLv2 Spec "0.2"

- Mandatory strong server authentication
- Provides for optional strong client authentication
- Satisfies export regulations by sending a portion of the key in the clear

SSLv3

- November 1996
- Multiple versions of the spec were circulated
  - Disagreements persist to this day!
- wp.netscape.com/eng/ssl3/3-SPEC.HTM
  - Some implementers worked from this version
  - It did not allow any extension of Client Hellos

SSLv3

- This was the last spec version driven by Netscape
- One recent sample indicated that it may still represent 22% of SSL/TLS handshakes!
  - Even though the vast majority of clients and servers actually support newer versions

SSLv3

- wp.netscape.com/eng/ssl3/draft302.txt
  - Published in IETF's Internet Draft system as draft-freier-ssl-version3-02.txt
  - Still seen at mozilla.com
  - Allows for "extra data" at the end of the Client Hello
  - But there was nothing that used it
  - Nothing to implement and test with
  - Specifies that this "extra data" is included in the hash calculations

SSLv3

- Completely breaking change
  - Client Hello message reformatted!
    - Caused some servers to hang the connection in a slow fail condition
    - Some implementations still send SSLv2-compatible hello
  - Cipher suite is shortened from 3 to 2 bytes
  - Record layer now defines distinct record types for handshake, app data, alerts, etc

SSLv3

- Introduces support for Diffie-Hellman and Fortezza (aka "clipper chip") key exchange
- Uses MD5 and SHA together in most places
- Multiple choices for record layer MAC
- Satisfies export regulations by using only 40 bits for key generation

SSLv3

- New message: Change Cipher Spec
  - In SSLv2, handshake messages modified the crypto parameters incrementally.
  - CCS enables Handshake messages are used to build a new "pending" connection state and switch to it all at once.

SSLv3

- New message: Finished
  - Exchanged (C->S then S->C) to complete the handshake
  - Sent in the new connection state right after CCS
  - Content is MD5||SHA-1 (36 bytes) over all previous handshake messages
  - Resists MitM by detecting early manipulations

SSLv3

- Introduced the new concept of "renegotiation"!
  - Who knew?!
- Not heavily advertised, the substring "renego" only appears twice in the spec!
- Very elegant, reuses the exact the same handshake protocol!
- Allows application data to be intermingled with renego handshake messages

SSLv3

- New message: Hello Request
  - Server-initiated renegotiation
- Only one mention in the 63 pages spec about client-initiated renegotiation:
  - "The client can also send a client hello in response to a hello request or on its own initiative in order to renegotiate the security parameters in an existing connection."

TLS 1.0

- January 1999 - RFC 2246 - TLS 1.0
- IETF renamed SSL to TLS
  - But everyone still calls it SSL
- Removes support for Fortezza

TLS 1.0

- Defines an abstract Pseudorandom Function (PRF)
  - Replaces assortment of MD5 and SHA combinations
  - Uses MD5 and SHA together
    - Survives breaks of either one
  - For key expansion and Finished
- Shortens Finished message data from 36 to 12 bytes

TLS 1.0 Extensions

- June 2003 - RFC 3546 TLS Extensions
- Updates TLS 1.0 about 4.5 years after-the-fact to define the general extension format and a few initial extensions
- Oops - some existing servers abort or hang the connection
- Doesn't apply retroactively to SSLv2 or 3 (in practice)

TLS 1.1+

- April 2006 - RFC 4346 TLS 1.1
- April 2006 - RFC 4366 TLS Extensions
- August 2008 - RFC 5246 TLS 1.2
- Not widely supported at this time

Authentication Gap

- *Man-in-the-Middle in Tunnelled Authentication Protocols*
  - N. Asokan, Valtteri Niemi, and Kaisa Nyberg
- Uses the example of PEAP to show that signing the protocol in one direction and simply tunneling the authentication of the other independently does not provide the strongest mutual authentication.

Example: HTTPS Login Form

1. Client strongly authenticates the server with TLS and PKI

2. Server authenticates the client with username/password.

HTTPS Login Form

- Server presents a certificate which client verifies through his trusted root CAs. Client uses that public key to securely negotiate the session key for the session.
  - This simultaneously authenticates the server to the client and defends the session against MitM. The session key is strongly bound to the certificate that the client decided to trust.

HTTPS Login Form

- Password-based credentials are passed over https to the server to authenticate the client. The session key is not strongly bound to this transaction.
- The client can ensure the non-existence of a MitM using PKI.
- But the server has no way to prove the non-existence of a MitM. He can only rely on the client to do a good job of this.

HTTPS Login Form

- But what if the client is a bozo? What if the client trusts an evil root CA?
- In this model, the server transitively trusts every root CA that is trusted by the client!

HTTPS Login Form

Trusting a key is not the same as trusting the key's owner. Trust is not necessarily transferable; I have a friend who I trust not to lie. He's a gullible person who trusts the President not to lie. That doesn't mean I trust the President not to lie. This is just common sense. If I trust Alice's signature on a key, and Alice trusts Charlie's signature on a key, that does not imply that I have to trust Charlie's signature on a key.

– Philip Zimmerman

Authentication Gap

- When form-based and HTTP authentication is simply carried through a TLS connection as an application protocol, it does not provide strong mutual authentication.
- Strong mutual authentication requires that each endpoint be able to independently prove the absence of a MtiM. This can only be done by ensuring that the authentication process in both directions contributes to the generation of the session key.

Authentication Gap

- Packet captures

Vulnerable Client

- Is it really a problem with the TLS spec?
- Maybe HTTPS is just using it wrong?

Vulnerable Client

- What's wrong with this client code?

```
String dnsName = "secure.example.com";
IpAddress ip = dnsResolve(dnsName);
TcpSocket s = connectTo(ip);
SSL ssl = connectSSL(s, REQUIRE_SERVER_CERT);
Cert serverCert = ssl->getPeerCert();
if (serverCert->getSubjectName() != dnsName)
    dieWithError("cert mismatch");
exchangeCriticalData(ssl);
```

Vulnerable Client

- Nothing!
- This code was secure with SSLv2
- Silently becomes vulnerable when used with SSLv3+ and an SSL/TLS stack that handles renegotiation transparently for the app (most of them).
- Secure with disabled or patched renegotiation

Vulnerable Client

- There is not one tutorial on the web of "Here's how to use this SSL/TLS library safely" which provides example code that does everything correctly.
- Strongly suggests that there are plenty of vulnerable client apps out there.
- Probably all Perl apps that use SSL/TLS directly

Vulnerable Client

- Patching both the client and server to support RI makes this application code secure again. It also re-enables interesting new possibilities.
- not vulnerable -> vulnerable -> not vulnerable

  SSLv2 SSLv3+ SSLv3+RI
- Something changed with SSLv3 to break the apps
  - Renegotiation was added

Authentication Gap

- TLS Terminology: Session
  - Uniquely identified by session id given to client in Server Hello message
  - Client can request to resume any session at any time
- Session resumption is orthogonal to renegotiation!
- No session identifier is carried across renegotiation

Authentication Gap

- TLS Terminology: Connection
  - Netscape defined what developers wanted:

    a "Secure Sockets Layer"
  - Sockets are well understood as a "connection-oriented" protocol
  - OO APIs tend to derive the SSL and TCP objects from a common IO interface

Authentication Gap

- TLS Terminology: Connection State
  - Sessions and connections are many-to-many
  - An instance of a session on a connection
  - The specs do not give an explicit name to this thing of great importance!
  - Connection State is the best we can come up with but isn't perfect
    - It excludes the initial, null CS

Authentication Gap

- Issues of identity
  - An authenticated server has an identity.
  - An authenticated client has an identity.
  - Does an anon endpoint have an identity?
  - Does an anon-anon connection have identities?
  - In what ways can identity change?

TROOPERS 10
IT-Security Conference & Workshops

Authentication Gap

- Yes, and yes.
- Even an anon endpoint can have some identity
  - "The same guy as was on the endpoint number of secure records ago"
- Identities can change across renegotiation
  - But are they additive?

Authentication Gap

- Designers who developed renegotiation expected identity would be additive across renegotation.
- Credentials could be "stacked"
  - An anonymous endpoint could be upgraded to authenticated through renegotiation
    - e.g. HTTPS
  - An endpoint could renegotiate to provide multiple certificates for their identity

Authentication Gap

- Renegotiation was developed for three purposes:
- 1. To refresh crypto keys
  - Probably the most commonly given justification
  - Not the most commonly used in practice
  - Necessary with the way TLS is used?

Authentication Gap

- Renegotiation was developed for three purposes:

- 2. To change cipher spec
  - Upgrading crypto strength
  - SGC
  - Not used much
    - Danger sign

Authentication Gap

- Renegotiation was developed for three purposes:

- 3. To allow dynamically specifying client cert requirements
  - Probably most important to Netscape's business case
  - Most commonly used case today

Authentication Gap

- Another commonly-cited justification for renegotiation:

  "To protect the client certificate"

- Questionable
  - Supposed to be a "public key" right?
  - Only protects against passive eavesdropping
  - Most apps will provide their client certificate cert if asked nicely

TROOPERS 10 IT-Security Conference & Workshops

Attack

- Blind plaintext injection
- Client cert stealing/redirection

Attack

- Primary attack allows "blind plaintext injection"

  The ability to insert attacker-chosen plaintext at a specific point in the protocol stream

  - Relatively limited and unusual capability
  - Some protocols are affected worse than others

Blind Plaintext Injection Attack

- HTTPS is particularly badly affected
  - *A relatively important case*
  - Allows session cookie stealing
  - Much like CSRF
    - Some mitigations may help
    - Some may not (GET -> POST)

Blind Plaintext Injection Attack

- Server sees the renegotiation
  - In fact, he may have requested it
- Client generally sees no renegotiation
- Either
  - Server accepts client-initiated
  - Server requests renegotiation
    - various techniques

Client Cert Redirection

- Client's client cert credentials can be redirected
  - From any client that will provide a cert
  - To any TLS server that will accept it
- Retroactively authenticates client's request
- Potentially a huge compromise
- Possibly without user interaction
- Client does not see result

Authentication Gap

- Questions!

Mitigations

- Forbid renegotiation entirely

Mitigations

- Forbid renegotiation entirely
  - Easy to implement

Mitigations

- Forbid renegotiation entirely
  - Easy to implement
  - What most devs expected anyway

Mitigations

- Forbid renegotiation entirely
  - Easy to implement
  - What most devs expected anyway
  - Works great for probably 95%+ of sites

Mitigations

- Forbid renegotiation entirely
  - Easy to implement
  - What most devs expected anyway
  - Works great for probably 95%+ of sites
  - Really, really bad!

    First round of patches broke stuff

Mitigations

- Things depending on renegotiation:
  - Tor
    - Wasn't vulnerable code

Mitigations

- Things depending on renegotiation:
  - Web Services
    - Widely deployed in B2B
    - Microsoft has a big investment
    - MS shops can use integrated auth
    - It's cross-platform interop that needs client cert auth the most!

Mitigations

- Things depending on renegotiation:
  - Smart cards
    - Some deployments have millions of cards and thousands of servers!
    - Work by storing client certs on the chip
    - Usually accessed by a PIN
    - Used for high-security websites

Mitigations

- The only correct mitigation is to fix renegotiation!

  Restore the continuity-of-identity guarantee.

Mitigations

- One method:

  At the record layer, have the renegotiated keys depend on both the new and the old key material.

  - Possibly as simple as replacing = with ^=

Mitigations

- One method:

  At the record layer, have the renegotiated keys depend on both the new and the old key material.

  - Possibly as simple as replacing = with ^=

- Too good to be true.
  - PKCS#11 API doesn't support the change
  - Burned into Si

Mitigations

- Another method:

  Inject the previous Finished message into the beginning of the handshake messages for constructing the Finished verify_data

  - Technically clean, simple to describe
  - Doesn't require new protocol structures
  - Only requires one endpoint to patch

Mitigations

- Another method:

  Inject the previous Finished message into the beginning of the handshake messages for constructing the Finished verify_data

  - Technically clean, simple to describe
  - Doesn't require new protocol structures
  - XXXXX
  - Not deployable by some sites for years!
    - Changes crypto calculations

Mitigations

- Method suggested by Project Mogul
- Defines a TLS extension which sends the verify_data from the previous Finished message on the Client and Server Hellos
  - Client and server cooperate to exclude MitM
  - Requires both client and server to patch
  - Requires support for TLS extensions
  - A few hundred lines of code

Mitigations

- RFC 5746!
  - Accepted by IETF/IESG
  - Several vendors have shipped code!
    - Opera 10.50
    - www.mikestoolbox.org
    - Firefox (alpha/beta??)
  - Many others have it internally

TLS Authentication Gap

Additional resources

- IETF [TLS] mailing list

  https://www.ietf.org/mailman/listinfo/tls
- [mogul-open]

  http://lists.links.org/mailman/listinfo/mogul-open
- PhoneFactor (status of patches)

  http://www.phonefactor.com/sslgap
- Marsh's blog

  http://extendedsubset.com/

Authentication Gap

- More questions!