Transport Layer Security (TLS)

NALINI ELKINS

INDUSTRY NETWORK TECHNOLOGY COUNCIL

PRESIDENT@INDUSTRYNETCOUNCIL.ORG

A few words about me

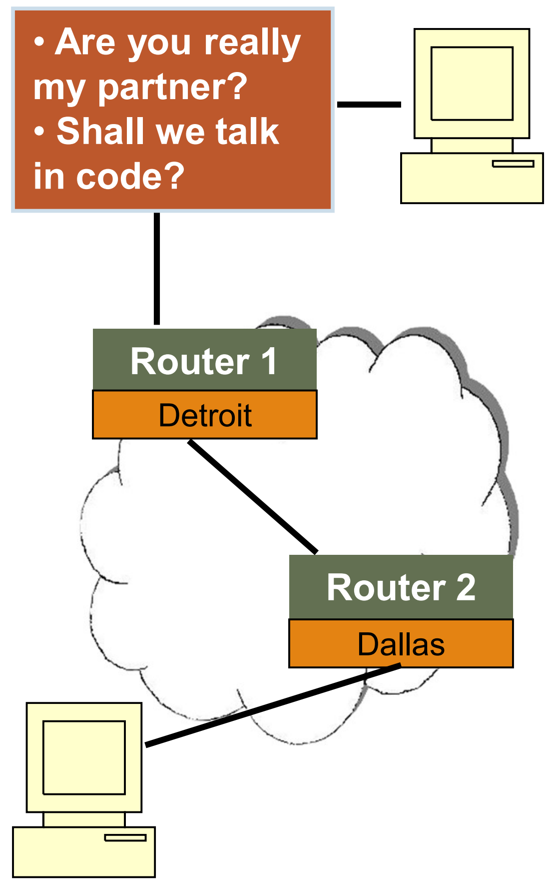
- President: Industry Network Technology Council
- Founder & CEO: Inside Products, Inc.
- Advisory Board: India Internet Engineering Society
- RFCs: RFC8250 (Embedded performance and diagnostics for IPv6) and others
- Product developer (OEMed by IBM and others)
- Working with IPv6 for 15 years
- Working with network management, diagnostic, performance issues at large brick-and-mortar enterprises for over 30 years

Agenda

- Introduction
- TLS handshake,
- Performance implications,
- Certificates,
- Server and client authentication,
- Case study

What is Secure Sockets Layer?

- Secure Sockets Layer (SSL) is a protocol developed by Netscape for transmitting private documents via the Internet.
- The main functions of SSL are:
  - Server authentication
  - Data privacy and integrity
  - Optional client authentication via digital certificate
- Multiple versions of SSL exist: SSL v2.0 and SSL v3.0.
- The SSL protocol became the Internet standard Transport Layer Security (TLS) described in RFC 2246 and updated in RFC 3546 and RFC5246.
- TLS v1.3 is the latest version of the secure sockets layer protocol (RFC8446).
- There are slight differences between SSL 3.0 and TLS 1.0-1.2, but the protocol remains substantially the same.
- TLS1.3 is quite different.

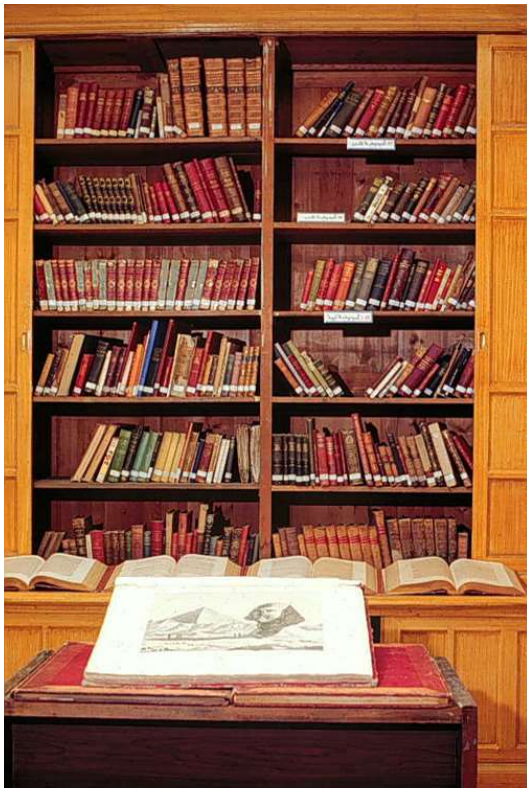


TLS Applications

- To implement TLS, the application program must use special TLS socket calls.
- As far as the TCP stack is concerned, TLS is just a TCP application. It is transparent to the stack.
- Languages such as C/C++ or Java provide application programming interfaces that interface with the sockets APIs for the platform (z/OS, Windows, Linux) to allow applications to establish secure sockets communications.
- TLS is available for TCP applications only. UDP, ICMP or other higher level protocols are not supported.

TLS Socket Library:

TCP Only



TLS Packet Encryption

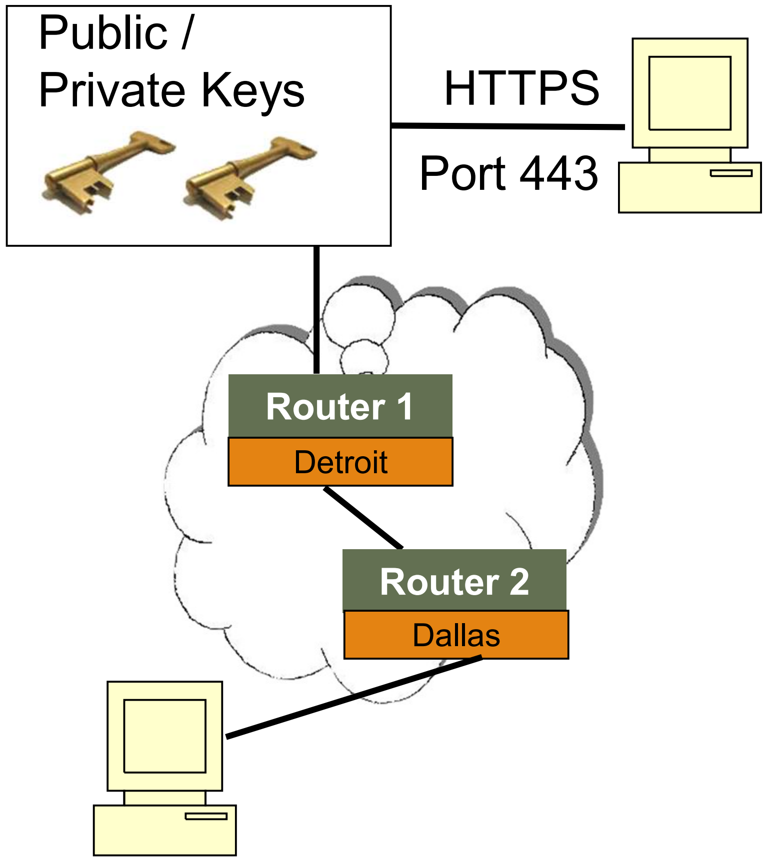
- TLS protects the TCP packet data. The IP header and TCP header are sent unencrypted.
- One of the benefits of using TLS is that if you do a packet trace, you will be able to see the TCP header, which means that you can see the ports which are being used.
- You may contrast this with IPSec which can encrypt the entire packet.

No.	Time	Source	Destination	Protocol	Info
259	167.343036	66.218.70.70	192.168.1.101	SSLv3	Application Data

```
⊞ Frame 258 (54 bytes on wire, 54 bytes captured)
⊞ Ethernet II, Src: AsustekC_39:29:2b (00:11:d8:39:29:2b), Dst: LinksysG_e4:ae:3
⊞ Internet Protocol, Src: 192.168.1.101 (192.168.1.101), Dst: 66.218.70.70 (66.2
⊟ Transmission Control Protocol, Src Port: 2259 (2259), Dst Port: https (443), S
    Source port: 2259 (2259)
    Destination port: https (443)
    Sequence number: 588    (relative sequence number)
    Acknowledgement number: 3825    (relative ack number)
    Header length: 20 bytes
```

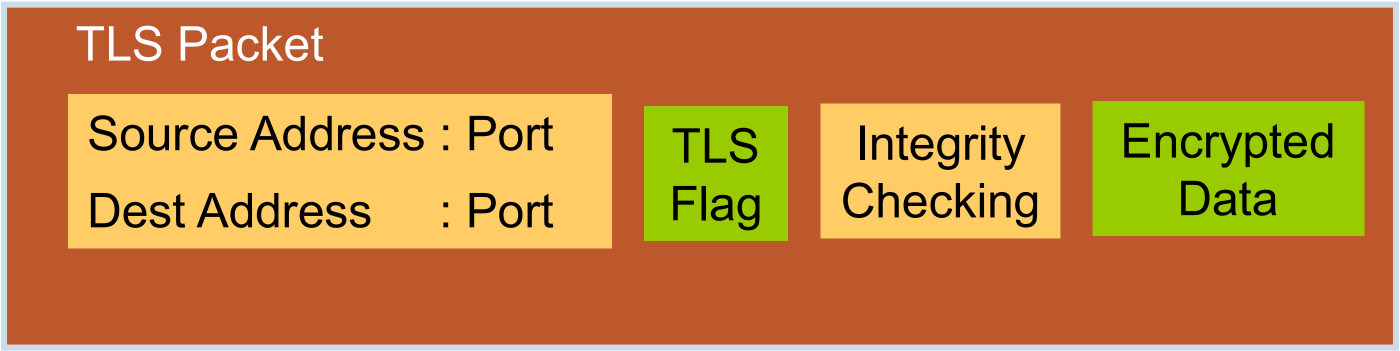
How Does TLS Work?

- Usually, TLS uses two keys to encrypt data – a public key known to everyone and a private or secret key known only to the recipient of the message.
- Both Netscape Navigator and Internet Explorer support TLS, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers.
- By convention, URLs that require an TLS connection start with https: instead of http:.
- An TLS-protected HTTP transfer uses port 443 instead of HTTP's normal port 80. Ports for TLS for TN3270 or FTP are assigned by the user.



TLS Packet Flow

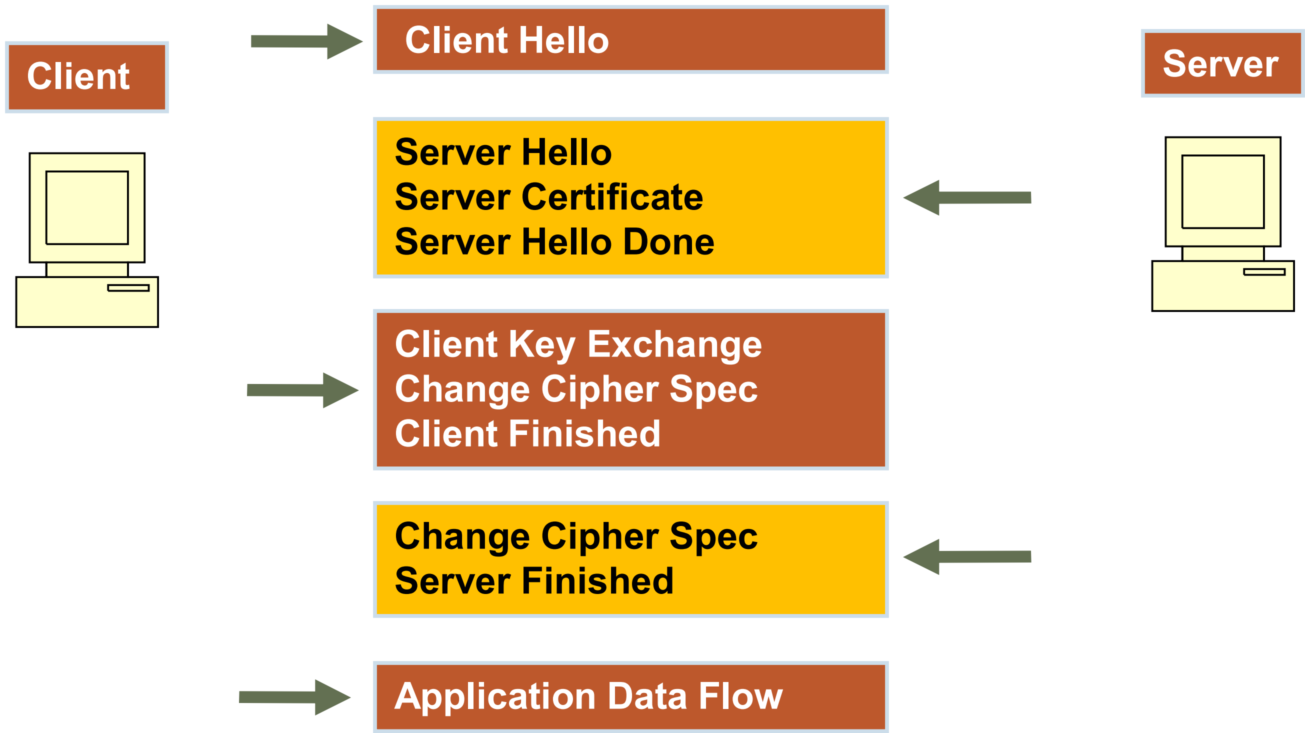
- A typical TLS packet may contain
  - **TLS flags indicating the type of information transported - the type of *TLS message*.**
  - **Cryptographic integrity checking (typically using the MD5 or SHA-1 algorithm).**
  - **Encrypted data (typically using the 3DES, AES or RC4 algorithms).**
- An TLS connection begins with a handshake which uses asymmetric (public key) cryptography.
- The handshake is followed by a data transfer phase, also called the TLS *record protocol*, which uses symmetric cryptography.



TLS Handshake

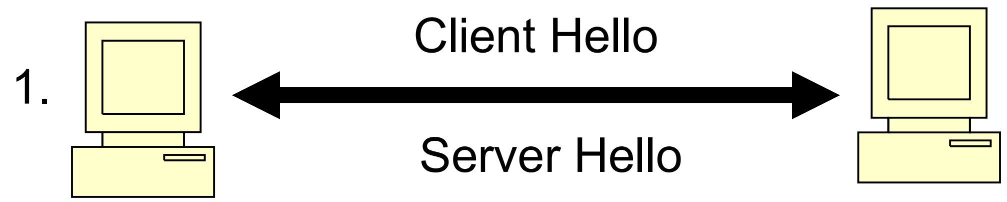
- An TLS connection begins with a handshake. As the name suggests, the handshake entails the initial setup. During the handshake, an exchange of information occurs that includes the following:
    - Authentication of the server.
    - Decision on how the data is to be encrypted.
    - Optionally, the authentication of the client.
- When the session begins, the client must know the public key of the server.
- No encryption is in use initially, so both parties (and any eavesdropper) can read this key, but the client can now transmit information to the server in a way that no one else could decode.

Packet in TLS Handshake Server Certificate

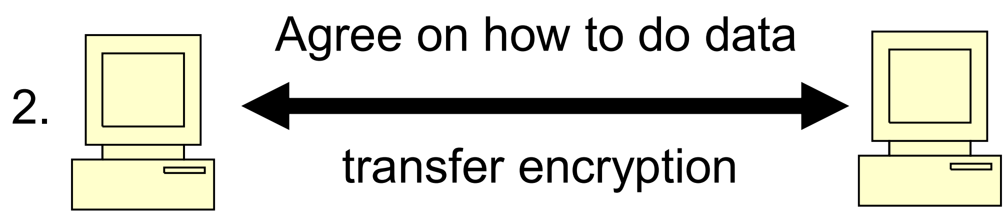


TLS Protocol Exchanges

1. The client connects to the server indicating that it wants to perform TLS. Contains sessionID. Server agrees ("server hello"). Handshake begins.



2. The client and server agree on a common symmetric algorithm to be used for the data transfer that follows the handshake. Both have a list of possible algorithms the order of preference



TLS Protocol Exchanges

3. The server provides its public key in its certificate to the client-this is also called *server authentication*, and is a required step of the TLS handshake.

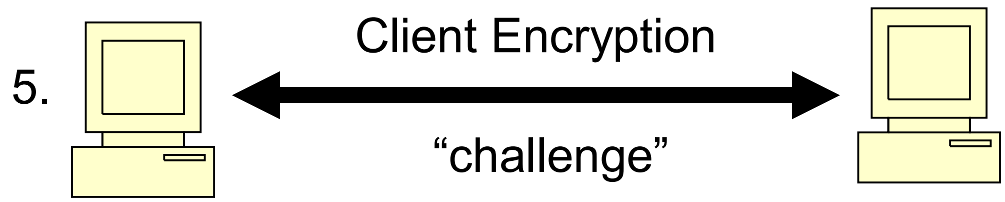
3.

4. The client verifies the integrity of the server's certificate Depending on the client design, if this certificate is not available, the client may ask the end user to agree to either pursue the communication or to abort it.

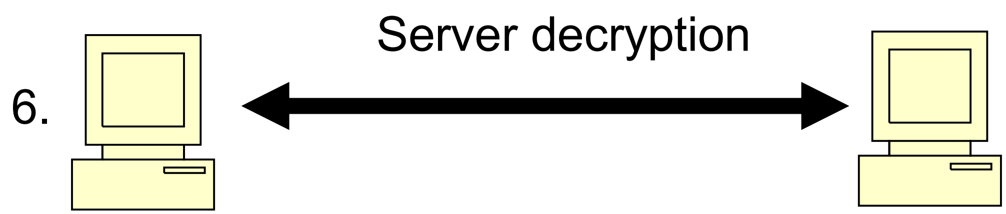
4.

TLS Protocol Exchanges

5. The client now has the server's public key and uses it to encrypt a random number, which is then sent to the server.

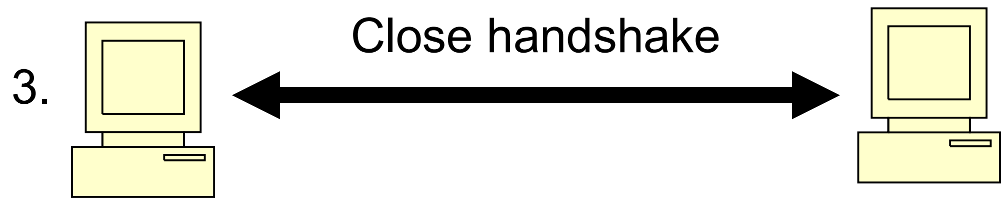


6. The server retrieves the value of this random number using its private key. The decryption of the secret random number using the server's private key can cost a great deal of computing resource during the TLS handshake.

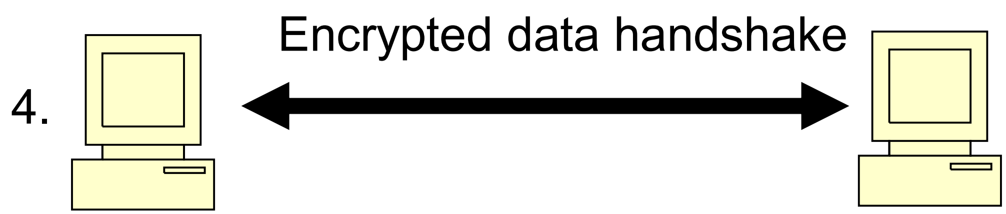


TLS Protocol Exchanges

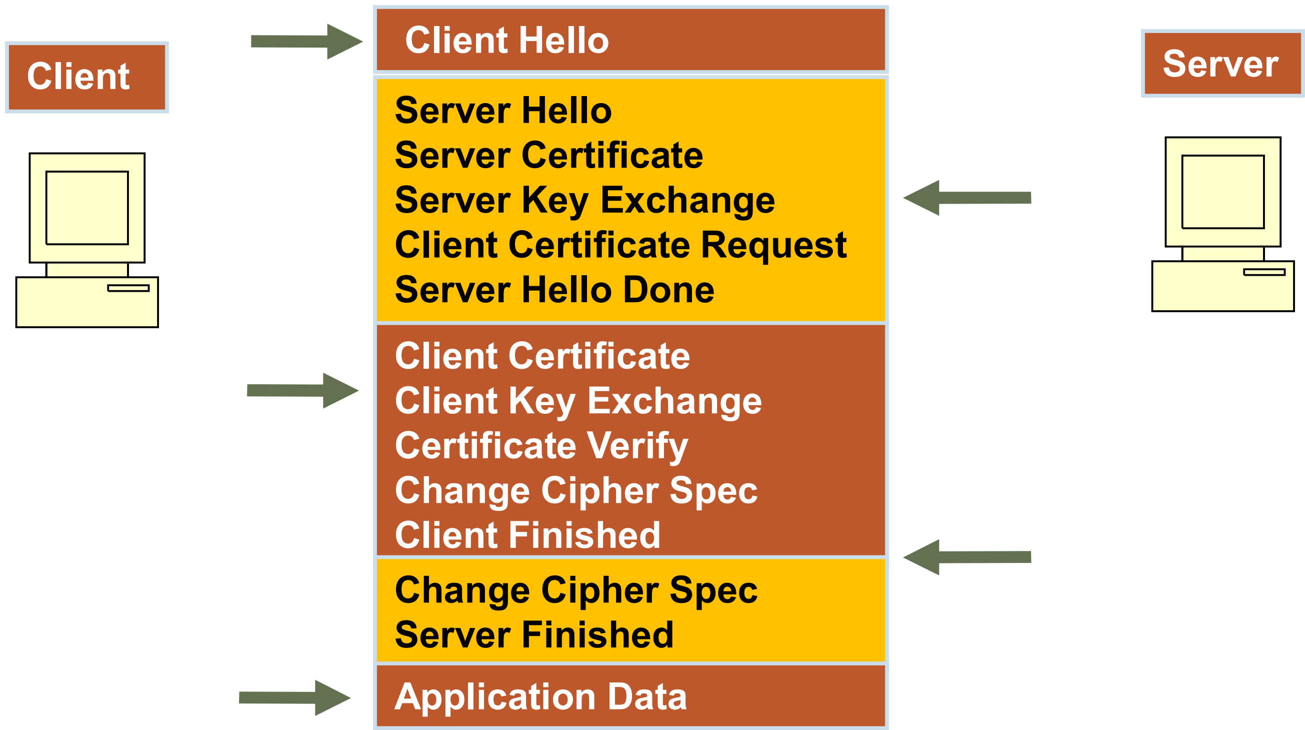
7. At this point only the client and server know this secret random number (the pre master secret), which can then be used to generate the keys to encrypt and decrypt the data, using the symmetric algorithm previously selected. The client and the server then close the handshake phase.



8. The data transfer phase (also called TLS record protocol) begins.



Packets in TLS Handshake
Server and Client Certificates



Mutual Authentication

- Client sends ClientHello message proposing TLS options.
- Server responds with ServerHello message selecting the TLS options.
- Server sends Certificate message, which contains the server's certificate.
- Server requests client's certificate in CertificateRequest message, so that the connection can be mutually authenticated.
- Server concludes its part of the negotiation with ServerHelloDone message.
- Client responds with Certificate message, which contains the client's certificate.
- Client sends session key information (encrypted with server's public key) in ClientKeyExchange message.
- Client sends a CertificateVerify message to let the server know it owns the sent certificate.
- Client sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
- Client sends Finished message to let the server check the newly activated options.
- Server sends ChangeCipherSpec message to activate the negotiated options for all future messages it will send.
- Server sends Finished message to let the client check the newly activated options

From: An Introduction to Mutual TLS Authentication: Elvin Cheng, 8 Feb 2012
http://www.codeproject.com/Articles/326574/An-Introduction-to-Mutual-TLS-Authentication

TLS / SSL Handshake

No.	Source	Destination	Protocol	Info
248	192.168.1.101	66.218.70.70	SSLv2	Client Hello
249	66.218.70.70	192.168.1.101	SSLv3	Server Hello, Certificate, Server Hello Done
252	192.168.1.101	66.218.70.70	SSLv3	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
253	66.218.70.70	192.168.1.101	TCP	https > 2259 [ACK] Seq=854 Ack=283 Win=8190 Len=0
254	66.218.70.70	192.168.1.101	SSLv3	Change Cipher Spec, Encrypted Handshake Message
255	192.168.1.101	66.218.70.70	SSLv3	Application Data
256	66.218.70.70	192.168.1.101	SSLv3	Application Data
257	66.218.70.70	192.168.1.101	SSLv3	Application Data
258	192.168.1.101	66.218.70.70	TCP	2259 > https [ACK] Seq=588 Ack=3825 Win=65535 Len=0
259	66.218.70.70	192.168.1.101	SSLv3	Application Data
260	192.168.1.101	66.218.70.70	TCP	2259 > https [ACK] Seq=588 Ack=5277 Win=65535 Len=0
261	66.218.70.70	192.168.1.101	SSLv3	Application Data
262	192.168.1.101	66.218.70.70	TCP	2259 > https [ACK] Seq=588 Ack=6584 Win=64228 Len=0
469	192.168.1.101	66.218.70.70	TCP	2259 > https [RST, ACK] Seq=588 Ack=6584 Win=0 Len=0

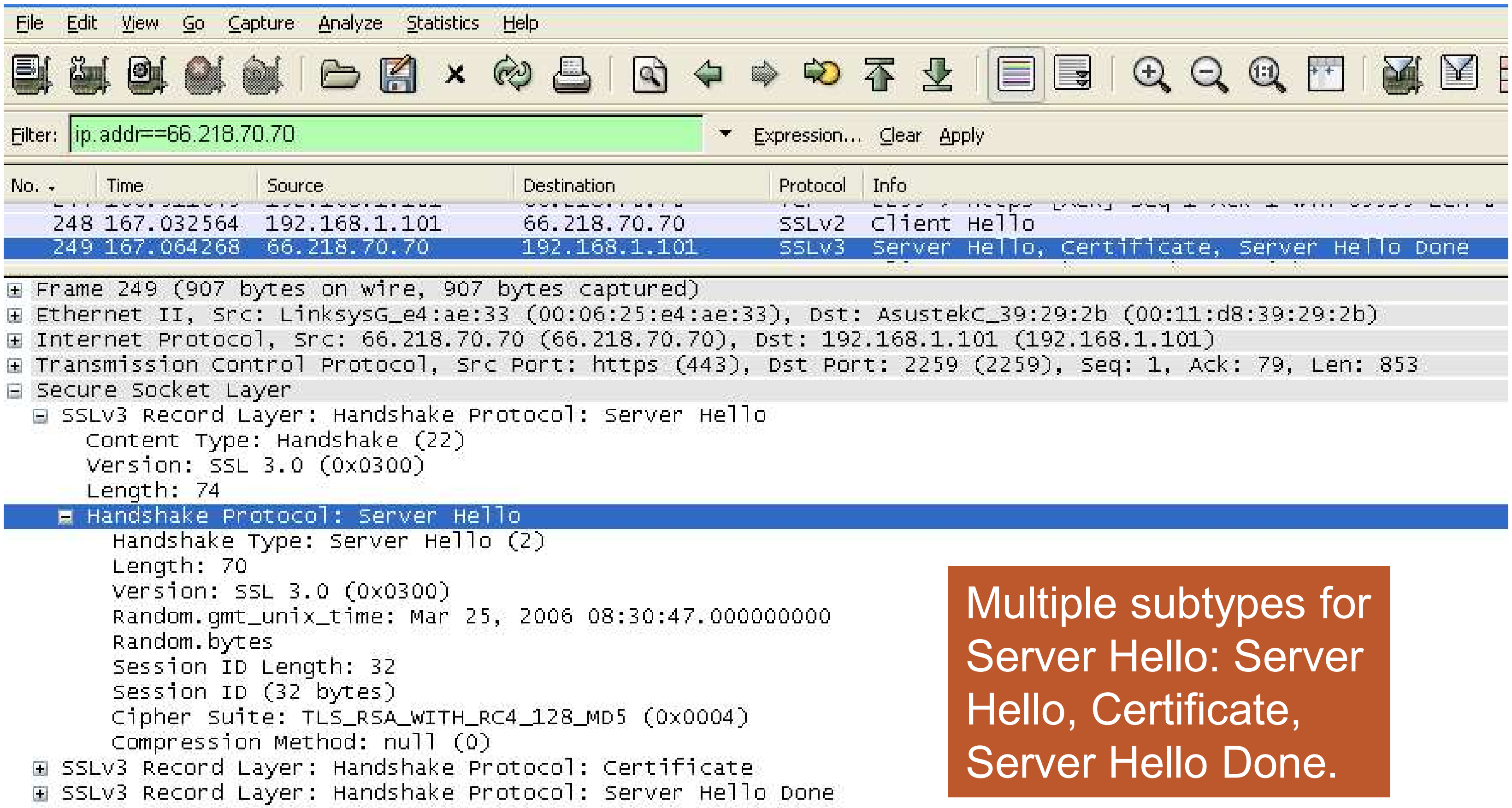
Notice the negotiation and then start of application flow

TLS / SSL Handshake

No.	Time	Source	Destination	Protocol	Info
248	167.032564	192.168.1.101	66.218.70.70	SSLv2	Client Hello

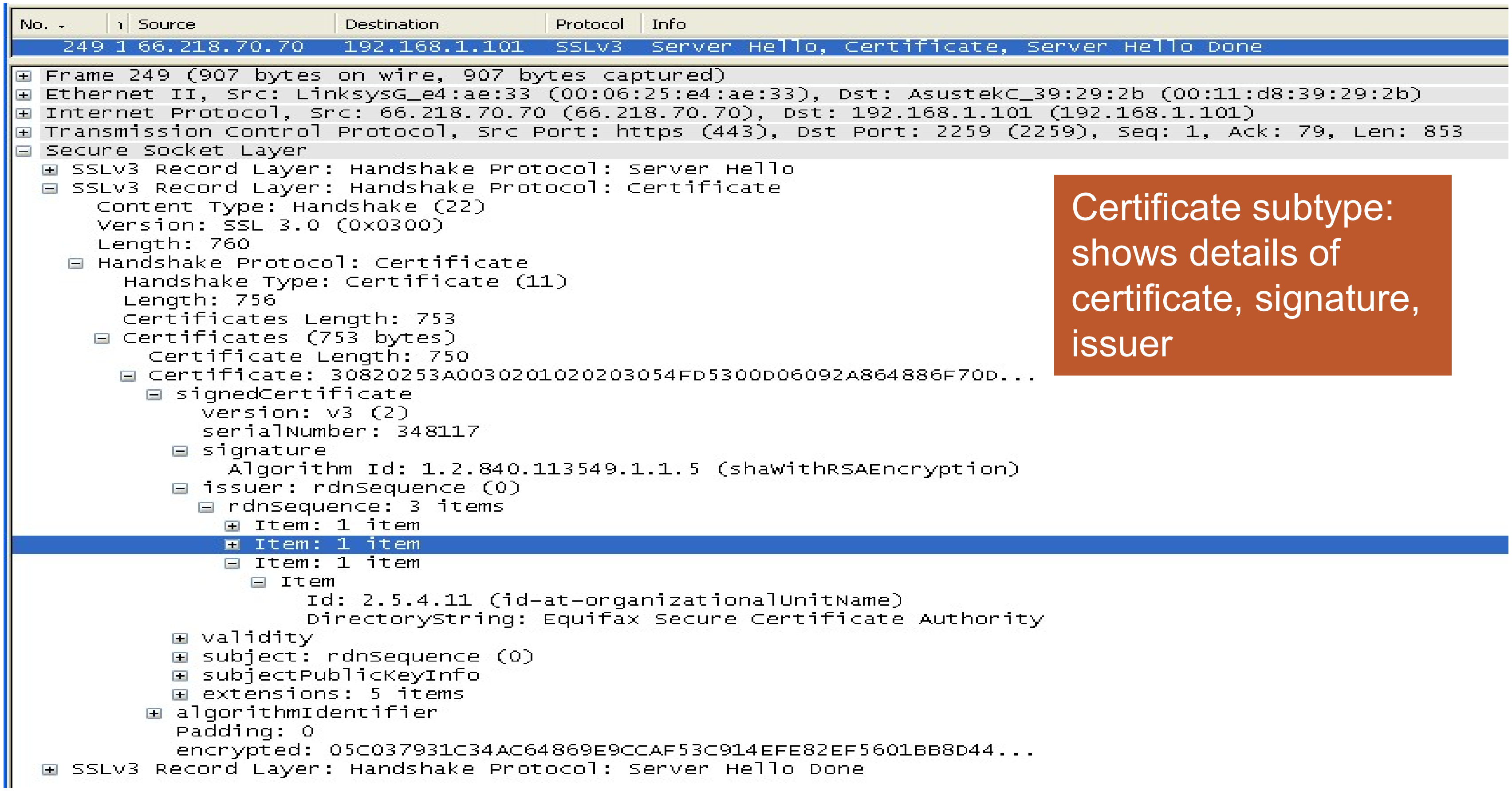
```
⊞ Frame 248 (132 bytes on wire, 132 bytes captured)
⊞ Ethernet II, Src: AsustekC_39:29:2b (00:11:d8:39:29:2b), Dst: LinksysG_e4:ae:33 (00:06:25:e4:ae:33)
⊞ Internet Protocol, Src: 192.168.1.101 (192.168.1.101), Dst: 66.218.70.70 (66.218.70.70)
⊞ Transmission Control Protocol, Src Port: 2259 (2259), Dst Port: https (443), Seq: 1, Ack: 1, Len: 78
⊟ Secure Socket Layer
  ⊟ SSLv2 Record Layer: Client Hello
      Length: 76
      Handshake Message Type: Client Hello (1)
      Version: SSL 3.0 (0x0300)
      Cipher Spec Length: 51
      Session ID Length: 0
      Challenge Length: 16
    ⊟ Cipher Specs (17 specs)
        Cipher Spec: TLS_RSA_WITH_RC4_128_MD5 (0x000004)
        Cipher Spec: TLS_RSA_WITH_RC4_128_SHA (0x000005)
        Cipher Spec: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x00000a)
        Cipher Spec: SSL2_RC4_128_WITH_MD5 (0x010080)
        Cipher Spec: SSL2_DES_192_EDE3_CBC_WITH_MD5 (0x0700c0)
        Cipher Spec: SSL2_RC2_CBC_128_CBC_WITH_MD5 (0x030080)
        Cipher Spec: TLS_RSA_WITH_DES_CBC_SHA (0x000009)
        Cipher Spec: SSL2_DES_64_CBC_WITH_MD5 (0x060040)
        Cipher Spec: TLS_RSA_EXPORT1024_WITH_RC4_56_SHA (0x000064)
        Cipher Spec: TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x000062)
        Cipher Spec: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x000003)
        Cipher Spec: TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x000006)
        Cipher Spec: SSL2_RC4_128_EXPORT40_WITH_MD5 (0x020080)
        Cipher Spec: SSL2_RC2_CBC_128_CBC_WITH_MD5 (0x040080)
        Cipher Spec: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x000013)
        Cipher Spec: TLS_DHE_DSS_WITH_DES_CBC_SHA (0x000012)
        Cipher Spec: TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA (0x000063)
      Challenge
```

TLS / SSL Handshake



Multiple subtypes for Server Hello: Server Hello, Certificate, Server Hello Done.

TLS / SSL Handshake



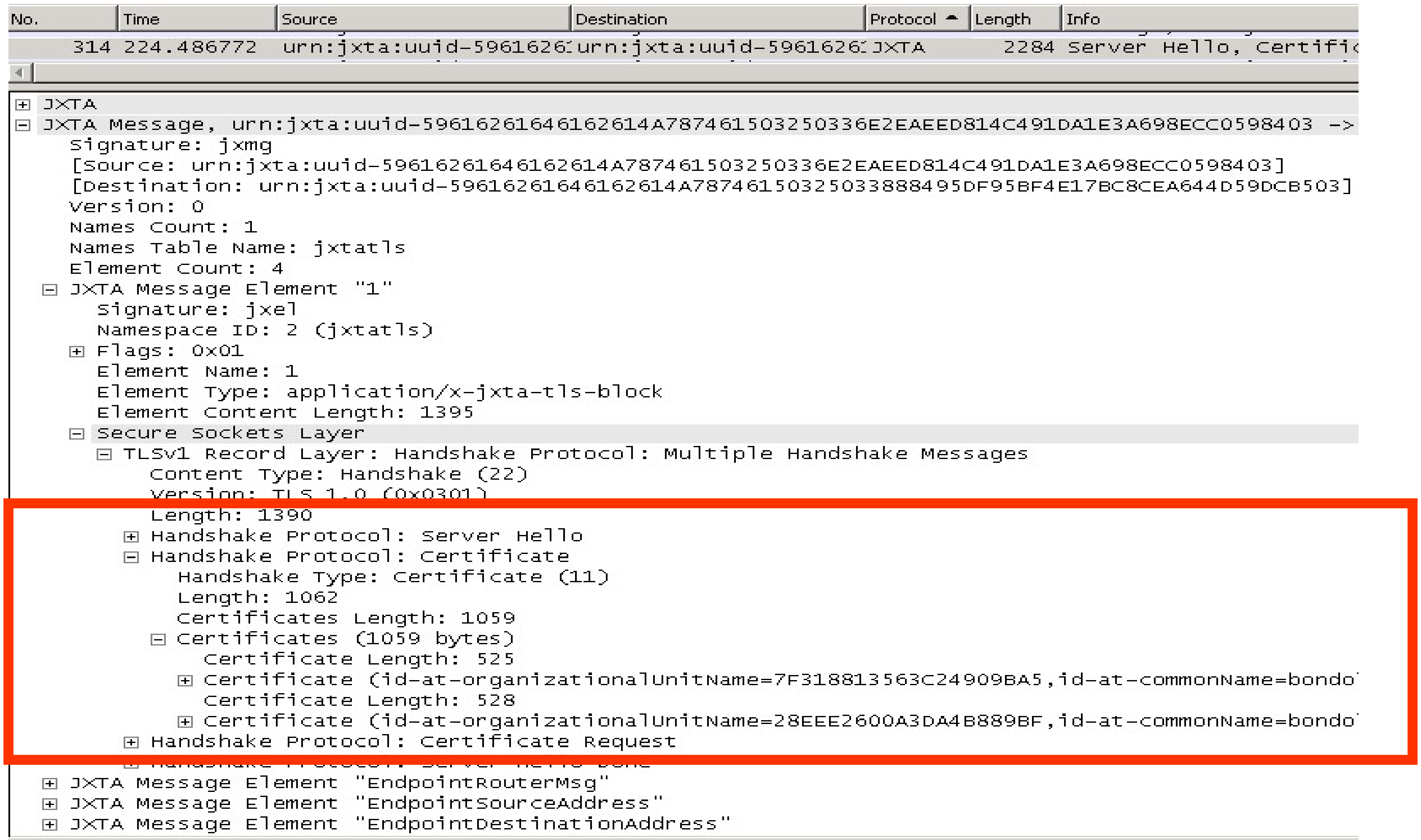
Certificate subtype: shows details of certificate, signature, issuer

TLS / SSL Handshake

```
⊞ Frame 314: 2284 bytes on wire (18272 bits), 2284 bytes captured (18272 bits)
⊞ Linux cooked capture
⊞ Internet Protocol Version 4, Src: 64.81.53.91 (64.81.53.91), Dst: 64.81.53.91 (64.81.53.91)
⊞ Transmission Control Protocol, Src Port: 32941 (32941), Dst Port: 8721 (8721), Seq: 159594, Ack: 9
⊞ JXTA
⊟ JXTA Message, urn:jxta:uuid-59616261646162614A787461503250336E2EAEED814C491DA1E3A698ECC0598403 ->
    Signature: jxmg
    [Source: urn:jxta:uuid-59616261646162614A787461503250336E2EAEED814C491DA1E3A698ECC0598403]
    [Destination: urn:jxta:uuid-59616261646162614A78746150325033888495DF95BF4E17BC8CEA644D59DCB503]
    Version: 0
    Names Count: 1
    Names Table Name: jxtatls
    Element Count: 4
  ⊟ JXTA Message Element "1"
      Signature: jxel
      Namespace ID: 2 (jxtatls)
    ⊞ Flags: 0x01
      Element Name: 1
      Element Type: application/x-jxta-tls-block
      Element Content Length: 1395
    ⊟ Secure Sockets Layer
      ⊟ TLSv1 Record Layer: Handshake Protocol: Multiple Handshake Messages
          Content Type: Handshake (22)
          Version: TLS 1.0 (0x0301)
          Length: 1390
        ⊞ Handshake Protocol: Server Hello
        ⊟ Handshake Protocol: Certificate
            Handshake Type: Certificate (11)
            Length: 1062
            Certificates Length: 1059
          ⊞ Certificates (1059 bytes)
        ⊞ Handshake Protocol: Certificate Request
        ⊞ Handshake Protocol: Server Hello Done
  ⊞ JXTA Message Element "EndpointRouterMsg"
  ⊞ JXTA Message Element "EndpointSourceAddress"
  ⊞ JXTA Message Element "EndpointDestinationAddress"
```

Server Certificate: Embedded in XML

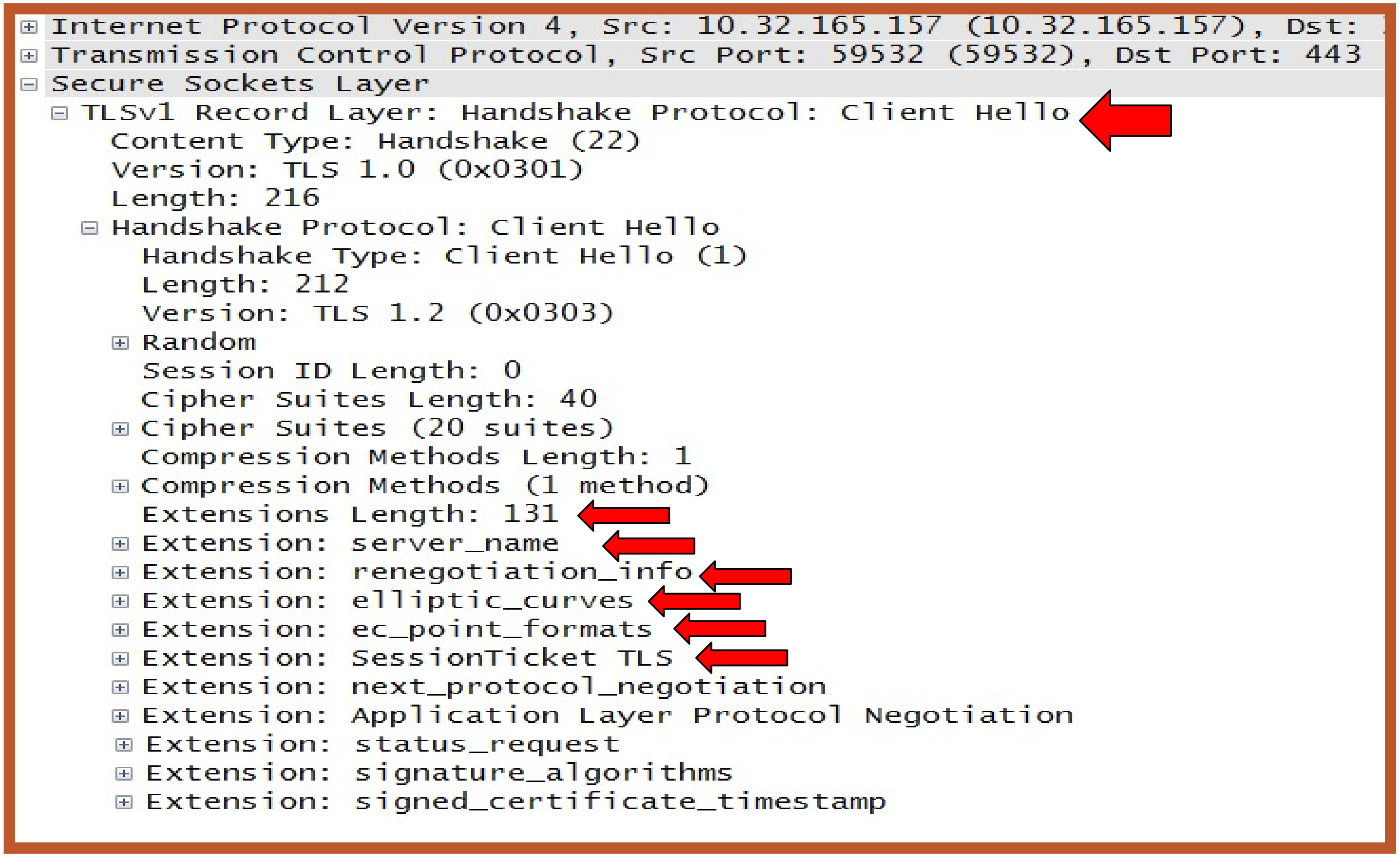
TLS / SSL Handshake



https://www.iiesoc.in/

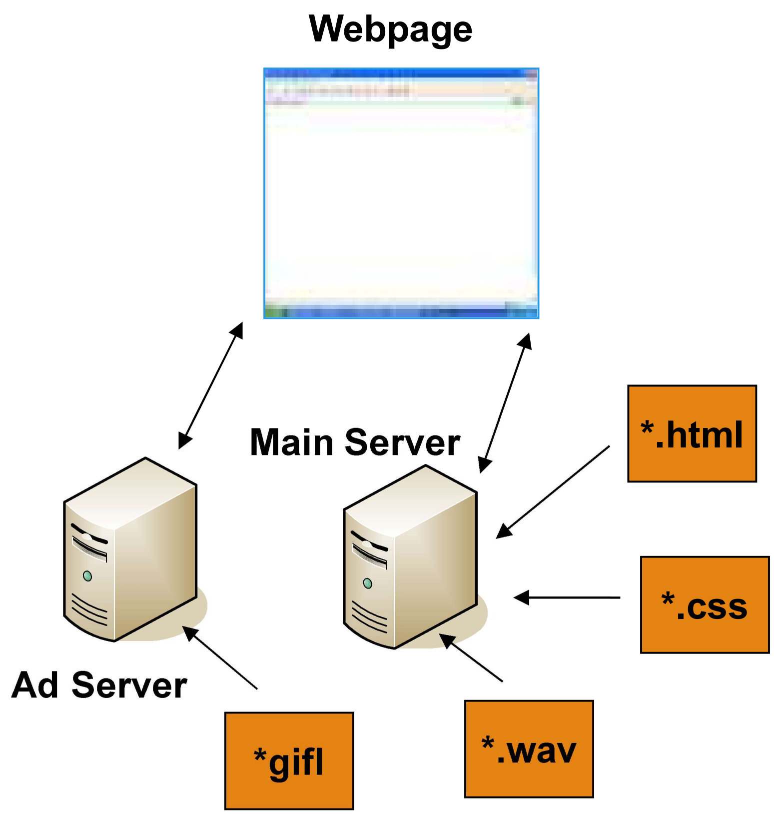
https://industrynetcouncil.org/

Client Hello Packet with Extensions



When to Handshake?

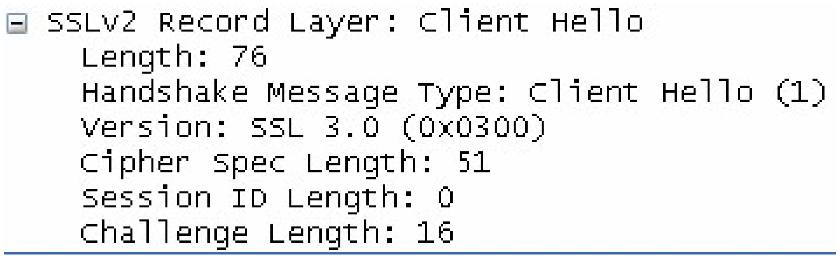
- When do TLS handshakes happen?
- How to reduce?
- Each time TCP connection is started

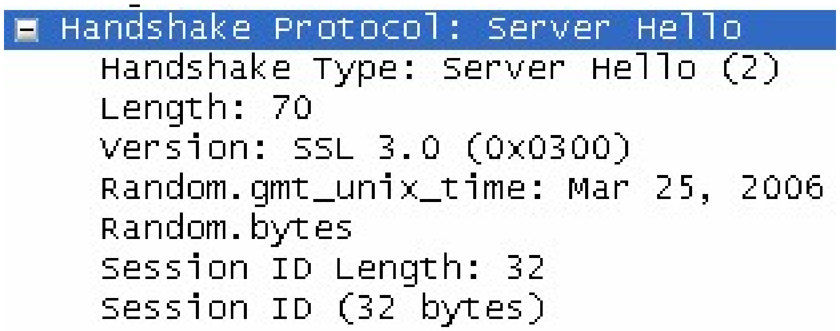


https://www.iiesoc.in/ https://industrynetcouncil.org/

Session ID Reuse: Older

- Built-in mechanism to reduce the number of handshakes: sessionID re-use
- TLS server set up to remember the pre-master secret with client and the corresponding sessionID.
- With new TLS session, sessionID presented by client
- If server remembers pre-master secret linked to this sessionID, then they can re-use the previous pre-master secret value.
- Handshake is smaller





TLS Time Out

- TLS parameter to remember the sessionID may be called SSLtimeOut
- Indicates how long the server should keep previous pre-master secret in its memory.
- Duration of SSLtimeOut is an installation trade-off between security and performance
- Even if pre-master secret value is reused, the derived keys are different every time

Session Tickets: RFC 5077

Transport Layer Security (TLS) Session Resumption without Server-Side State

Abstract

This document describes a mechanism that enables the Transport Layer Security (TLS) server to resume sessions and avoid keeping per-client session state. The TLS server encapsulates the session state into a ticket and forwards it to the client. The client can subsequently resume a session using the obtained ticket.

Renegotiation Indication Extension

RFC5746: Transport Layer Security (TLS) Renegotiation Indication Extension

Abstract

Secure Socket Layer (TLS) and Transport Layer Security (TLS) renegotiation are vulnerable to an attack in which the attacker forms a TLS connection with the target server, injects content of his choice, and then splices in a new TLS connection from a client. The server treats the client's initial TLS handshake as a renegotiation and thus believes that the initial data transmitted by the attacker is from the same entity as the subsequent client data. This specification defines a TLS extension to cryptographically tie renegotiations to the TLS connections they are being performed over, thus preventing this attack.

Why?

- TLS renegotiation Man-In-The-Middle attack

  (TLS Renego MITM)

- Problems can happen when a client is compromised or there is a Man-in-the-Middle.
- **At some point in the future, connectivity problems may occur because of server non-compliance with RFC 5746.**

TLS Renegotiation

Client Server

←--------------------Handshake-----------------------→

←============Protected Data=============→

←=============Handshake===============→

←============Protected Data=============→

- Initial Handshake Establishes a protected channel
- Re-negotiation is a new handshake run under the protection of the existing channel
- Upon completion the new channel replaces the old channel

ietf-76-tls-rengo.ppt : Joe Salowey/ Eric Rescorla

Renegotiation Attack

Client Attacker Server

←-------- Handshake-----------→

←===== Initial Traffic =====→

←----------------------Handshake=================→

←============= Client Traffic================ =→

- Initial traffic and client traffic are treated as originating under the same context
- Attacker injected traffic may be processed under clients context
- Attacker injected traffic may set up context under which client's traffic is processed
- Client handshake may use client certificates

ietf-76-tls-rengo.ppt : Joe Salowey/ Eric Rescorla

Vulnerability

- Attacker injects data that is processed under client's context
  - Process unauthenticated request under authenticated context
  - Attacker can inject data processed under client's authorization based on client certificate
- Attacker sets up context that discloses information in client's request
  - Client cert authentication not necessary for attack
- Complications
  - Renegotiation is often transparent to application
  - Client is not aware this is a renegotiation
  - Some HTTP servers support renegotiation to request client certs for a protected resource
- Other protocols may be vulnerable as well
  - IMAP, LDAP, XMPP, SIP, SMTP, …

ietf-76-tls-rengo.ppt : Joe Salowey/ Eric Rescorla

Server Name Indication Handshake

RFC6066 (and others)

a client indicates which hostname it is attempting to connect to at the start of the handshaking process. This allows a server to present multiple certificates on the same IP address and TCP port number and hence allows multiple secure (HTTPS) websites (or any other Service over TLS) to be served off the same IP address without requiring all those sites to use the same certificate.

```
⊟ Extension: server_name
    Type: server_name (0x0000)
    Length: 26
  ⊟ Server Name Indication extension
      Server Name list length: 24
      Server Name Type: host_name (0)
      Server Name length: 21
      Server Name: us-mg5.mail.yahoo.com
```

What Server Name?

- One certificate can cover multiple hostnames.
- subjectAltName field
- Only server names supported are DNS hostnames
- Other name types may be added in the future

ECC Extensions: RFC4492Bis

- Two new TLS extensions are defined:
  - (i) the Supported Elliptic Curves Extension, and
  - (ii) the Supported Point Formats Extension.

Brute Force Attacks

- Most basic method of attack is brute force — trying every possible key in turn
- Length of the key determines the number of possible keys

ECC Extensions: RFC4492Bis

- Two new TLS extensions are defined in this specification: (i) the Supported Elliptic Curves Extension, and (ii) the Supported Point Formats Extension. These allow negotiating the use of specific curves and point formats (e.g., compressed vs. uncompressed, respectively) during a handshake starting a new session.
- These extensions are especially relevant for constrained clients that may only support a limited number of curves or point formats. They follow the general approach outlined in [RFC4366]; message details are specified in Section 5. The client enumerates the curves it supports and the point formats it can parse by including the appropriate extensions in its ClientHello message. The server similarly enumerates the point formats it can parse by including an extension in its ServerHello message.

Simple Elliptical Curve Extension

```
⊟ Extension: elliptic_curves
    Type: elliptic_curves (0x000a)
    Length: 8
    Elliptic Curves Length: 6
  ⊟ Elliptic curves (3 curves)
      Elliptic curve: secp256r1 (0x0017)
      Elliptic curve: secp384r1 (0x0018)
      Elliptic curve: secp521r1 (0x0019)
```

SSLv2 Hellos

```
⊞ Transmission Control Protocol, Src Port: 59572 (59572), Dst Port: 443
⊟ Secure Sockets Layer
  ⊟ SSLv2 Record Layer: Client Hello
      [Version: SSL 2.0 (0x0002)]
      Length: 70
      Handshake Message Type: Client Hello (1)
      Version: TLS 1.0 (0x0301)
      Cipher Spec Length: 45
      Session ID Length: 0
      Challenge Length: 16
    ⊞ Cipher Specs (15 specs)
      Challenge
```

No extensions!

TLS1.3

The Transport Layer Security (TLS) Protocol Version 1.3
RFC8446

C.3 Backwards Compatibility Security Restrictions

Implementations MUST NOT send an TLS version 2.0 compatible CLIENT- HELLO. Implementations MUST NOT negotiate TLS 1.3 or later using an TLS version 2.0 compatible CLIENT-HELLO. Implementations are NOT RECOMMENDED to accept an TLS version 2.0 compatible CLIENT-HELLO in order to negotiate older versions of TLS.

Case Study

- Case Study
- TLS Application Concurrent Sessions

Problem Using TLS

- Large 4-year public university has a problem using a mainframe TCP application using SSL.
- At times when students registered for a dormitory room using their CICS web-enabled application, the CICS region became ‘hung’ and had to be restarted.
- Students were also unable to use the application and many calls were received at the Help Desk.

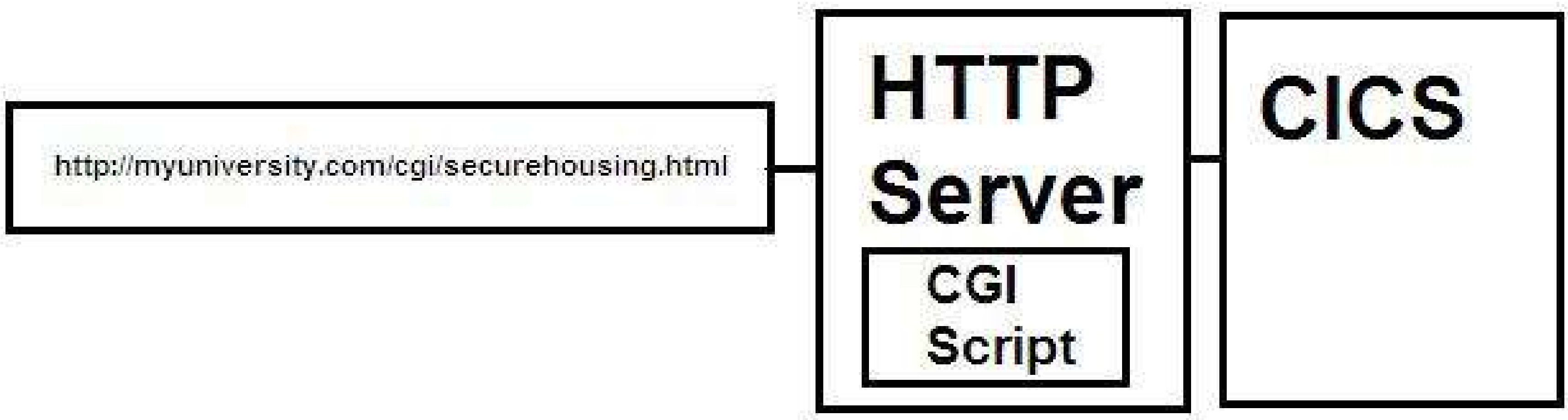
Staging Use of Application

- Application allowed students to select residence hall, roommate, meal plan and other related matters.
- Application was available for one week out of the year.
- Tried to stage the use of the application by having honors students and Seniors sign up the first day, Juniors, the following day etc.
- Application worked fine until the last day when students with poor grades, freshmen, and all others were allowed in.
- Then, chaos erupted.

https://www.iiesoc.in/

https://industrynetcouncil.org/

Application Architecture



- Student logs on to a web page such as http://myuniversity.com/cgi/securehousing.html.
- Web page is actually a CGI script running under the HTTP Web server on the mainframe.
- Port used was port 443 for secure sockets.
- CGI script initiated a connection to CICS to get data and pass it back.

Security Directives

We looked at the security directives in the HTTP WebServer. They had coded the Cipher Specs to go from highest strength to lowest. We found some documentation for another web server that led us to believe that this may make the TLS handshake longer.

```
SSLCipherSpec directive
# Specify the methods of encryption that an TLS connection will support. Each
# encoded cipher specification is tested in the order specified for
# compatibility with the requester. If the requester supports a method specified
# here, an TLS connection can be established. If not, the connection is refused.
SSLCipherSpec 39
SSLCipherSpec 27
SSLCipherSpec 21
TLSCipherSpec 23
SSLCipherSpec 26
SSLCipherSpec 22
TLSCipherSpec 24
SSLCipherSpec 3A
```

Cipher Specs

```
#        Syntax:   SSLCipherSpec <code>#  where <code> is one of:  SSL V2:
#
#                  Code    Meaning                 Note     Strength
#                  ====    =============           ====     ========
#                   21     RC4  (128 bit)            *      (weaker)
#                   22     RC4  (40 bit)
#                   23     RC2  (128 bit)            *          |
#                   24     RC2  (40 bit)                        V
#                   26     DES  (56 bit)             *
#                   27     Triple DES  (192 bit)     *     (stronger)
#
#                  SSL V3:
#                  Code    Meaning                 Note     Strength
#                  ====    =============           ====     ========
#                   33     RC4 MD5  (128 bit)               (weaker)
#                   34     RC4 MD5  (128 bit)        *
#                   35     RC4 SHA  (128 bit)        *          |
#                   36     RC2 MD5  (40 bit)                    V
#                   39     DES SHA  (56 bit)
#                   3A     Triple DES SHA (192 bit)  *     (stronger)
```

Cipher Strength

- Set Cipher Specs to go from lowest strength to highest instead of highest strength to lowest.
  - SSLCipherSpec 21
  - SSLCipherSpec 22
  - SSLCipherSpec 23
  - SSLCipherSpec 24
  - TLSCipherSpec 33
  - SSLCipherSpec 35 etc
- This fixed the problem a great deal.
- TLS often done in hardware
- Need to redo with new ciphers (possible grant!)

Questions?

Contact:

info@iiesoc.in

president@industrynetcouncil.org