

ISA 666 Internet Security Protocols

Lecture #9 Transport Layer Security SSL/TSL/SSH

ISA 666

By Dr. Xinyuan (Frank) Wang

1

SSL/TLS/SSH

- SSL/TLS overview and basic features
- SSL Record Protocol
- SSL Handshake Protocol
- Other SSL Protocols
- SSL and TLS differences
- SSL applications
- SSH overview
- SSH architecture
- SSH security
- Port forwarding with SSH
- SSH applications

ISE at George Mason University

ISA 666

By Dr. Xinyuan (Frank) Wang

2

SSL/TLS Overview

- SSL = Secure Sockets Layer.
 - Originally developed to secure http
 - unreleased v1, flawed but useful v2, good v3.
- TLS = Transport Layer Security.
 - TLS1.0 = SSL3.0 with minor tweaks ≈ SSL3.1
 - Defined in RFC 2246.
 - Open-source implementation at <http://www.openssl.org/>.
- SSL/TLS provides security ‘at TCP layer’.
 - Uses TCP to provide reliable, end-to-end transport.
 - Applications need some modification.
 - In fact, usually a thin layer between TCP and HTTP.

ISE at George Mason University

ISA 666

By Dr. Xinyuan (Frank) Wang

3

SSL/TLS Basic Features

- SSL/TLS widely used in Web browsers and servers to support ‘secure e-commerce’ over HTTP.
 - Built into Microsoft IE, Netscape, Mozilla, Apache, IIS,...
 - The (in)famous browser lock.
- SSL architecture provides two layers:
 - SSL Record Protocol
 - Provides secure, reliable channel to second layer.
 - Upper layer carrying:
 - SSL Handshake Protocol, Change Cipher Spec. Protocol, Alert Protocol, HTTP, any other application protocols.

ISE at George Mason University

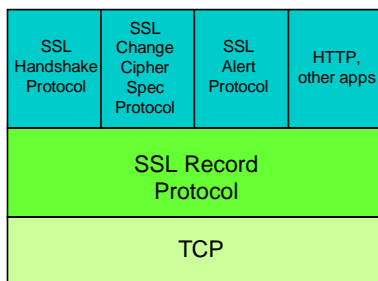
ISA 666

By Dr. Xinyuan (Frank) Wang

4

SSL Protocol Architecture

- A two-layer protocol



ISE at George Mason University

ISA 666

By Dr. Xinyuan (Frank) Wang

5

SSL Services

- Peer entity and data authentication
- Data confidentiality
- Data integrity
- Compression/decompression
- Generation/distribution of session keys
 - Integrated to protocol
 - A different approach from IPSec
- Security parameter negotiation.

ISE at George Mason University

ISA 666

By Dr. Xinyuan (Frank) Wang

6

SSL Connection and Session

- Each SSL session can be used for multiple SSL connections.
- SSL Session
 - An association between a client and a server.
 - Created by handshake protocol.
 - Are used to avoid negotiation of new security parameters for each connection.
- SSL Connection
 - A connection is a transport that provides a suitable type of service.
 - Peer-to-peer, transient
 - Each connection is associate with one session.

SSL Session

- We can view an SSL session as an SSL security association.
 - Created by handshake protocol
 - Defines set of cryptographic parameters (encryption and hash algorithm, master secret, certificates).
 - Carries multiple *connections* to avoid repeated use of expensive handshake protocol.
- A SSL session consists of
 - Session ID
 - X.509 public-key certificate of peer (could be null)
 - Compression algorithm
 - Cipher spec:
 - Encryption algorithm, message digest algorithm, etc.
 - Master secret: 48 byte secret shared between the client and server
 - Is reusable

SSL Connection

- SSL Connection concept
 - State defined by nonces, secret keys for MAC and encryption, IVs, sequence numbers.
 - Keys for many connections derived from single master secret created during handshake protocol.
- An SSL Connection consists of
 - Server and client random numbers
 - Server write MAC secret
 - Client write MAC secret
 - Server write key
 - Client write key
 - Server IV
 - Client IV
 - Sequence number

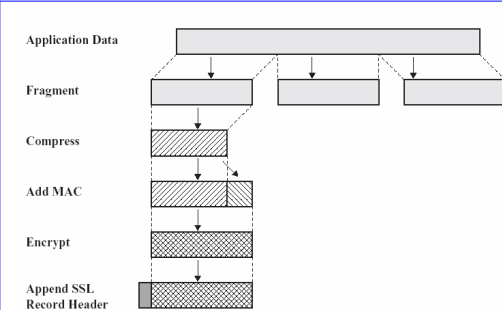
SSL Record Protocol

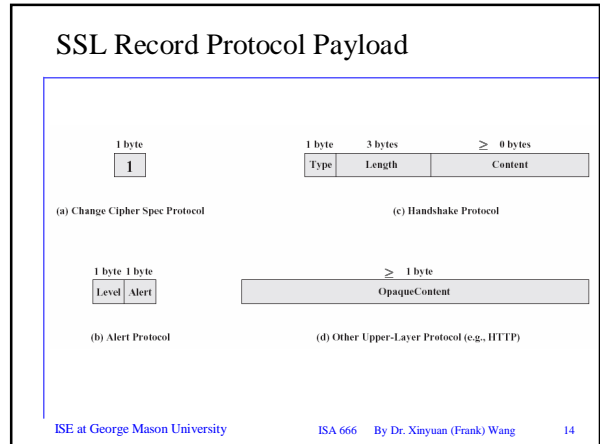
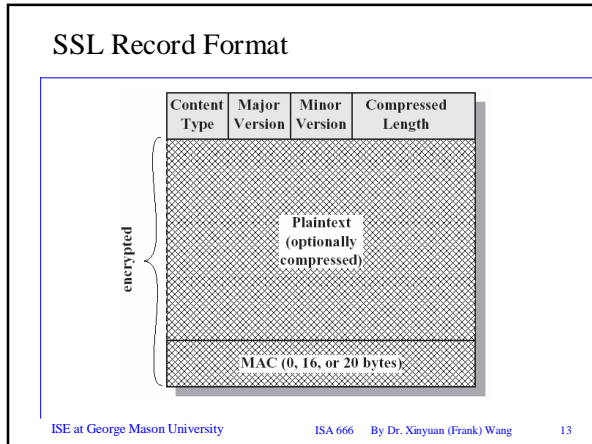
- SSL Record Protocol provides secure, reliable channel to second layer :
 - Data origin authentication and integrity.
 - MAC using algorithm similar to HMAC.
 - Based on MD-5 or SHA-1 hash algorithms.
 - MAC protects 64 bit sequence number for anti-replay.
 - Confidentiality.
 - Bulk encryption using symmetric algorithm.
 - IDEA, RC2-40, DES-40 (exportable), DES, 3DES,...
 - RC4-40 and RC4-128.
- Carries application data and SSL 'management' data.

SSL Record Protocol

- Data from application/upper layer SSL protocol partitioned into fragments (max size 2^{14} bytes).
- Optional compression
- MAC
- then pad (if needed)
- finally encrypt.
- Prepend header.
 - Content type, version, length of fragment.
- Submit to TCP.

SSL Record Protocol Operation

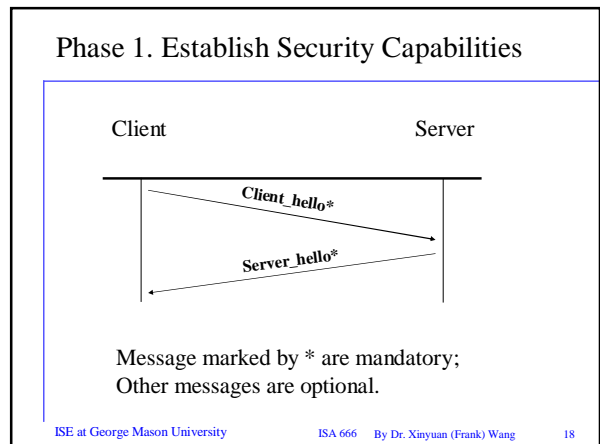




- ### SSL Handshake Protocol
- Like IPsec, SSL needs symmetric keys:
 - MAC and encryption at Record Layer.
 - Different keys in each direction.
 - These keys are established as part of the SSL Handshake Protocol.
 - As with IKE in IPsec, the SSL Handshake Protocol is a complex protocol with many options...
- ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 15

- ### SSL Handshake Protocol Security Goals
- Secure ciphersuite negotiation.
 - Encryption and hash algorithms
 - Authentication and key establishment methods.
 - Entity authentication of participating parties.
 - Participants are 'client' and 'server'.
 - Server nearly always authenticated, client more rarely.
 - Appropriate for most e-commerce applications.
 - Establishment of a fresh, shared secret.
 - Shared secret used to derive further keys.
 - For confidentiality and authentication in SSL Record Protocol.
- ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 16

- ### SSL Handshake Protocol
- Initially SSL session has null compression and encryption algorithm.
 - Both are set by the handshake protocol at the beginning of session.
 - Handshake protocol may be repeated during the session.
 - Four phases
 - Establish Security Capabilities
 - Server Authentication and Key Exchange
 - Client Authentication and Key Exchange
 - Finish
- ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 17



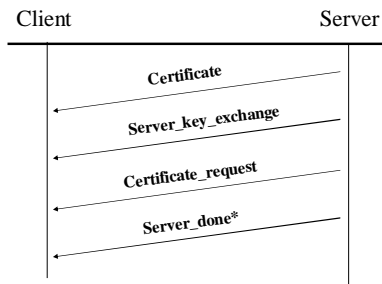
Phase 1 (Cont'd)

- Client_hello
 - Version: The highest SSL version understood by the client
 - Random: 4-byte timestamp + 28-byte random number.
 - Session ID: zero for new session, non-zero for a previous session
 - CipherSuite: list of supported algorithms
 - Compression Method: list of supported compression methods

Phase 1 (Cont'd)

- Server_hello
 - Version: min (client_hello version, highest version supported by the server)
 - Random: 4-byte timestamp + 28-byte random number.
 - Generated by the server
 - Session ID:
 - CipherSuite: selected from the client's list by the server
 - Compression method: selected from the client's list by the server

Phase 2: Server Authentication and Key Exchange



Certificate is almost always used.

Server_key_exchange message

- Not required if
 - The server has sent a certificate with fixed D-H parameters, or
 - RSA key exchange is to be used.
- Needed for
 - Anonymous D-H
 - Ephemeral D-H
 - RSA key exchange, in which the server is using RSA but has a signature-only RSA key.
 - Fortezza

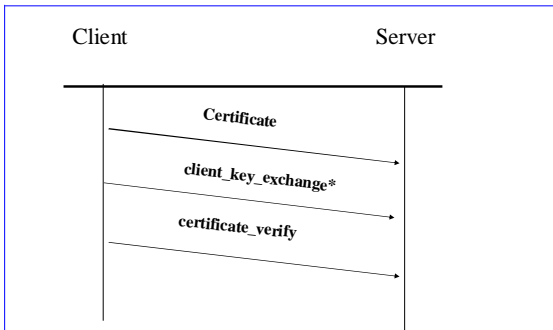
Certificate_request message

- Request a certificate from the client
- Two parameters
 - Certificate_type
 - RSA, signature only
 - DSS, signature only
 - ...
 - Certificate_authorities

Server_done message

- Indicate the end of server hello and associated messages.

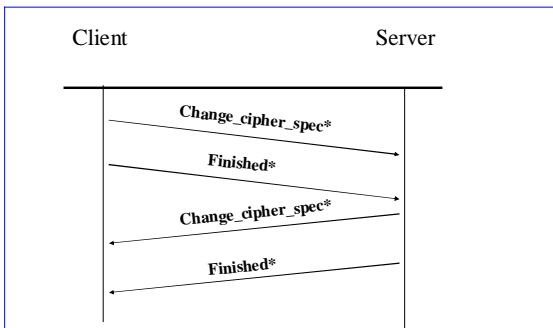
Phase 3. Client Authentication and Key Exchange



Phase 3. Client Authentication and Key Exchange

- Certificate
 - One or a chain of certificates.
- Client_key_exchange
 - RSA: encrypted pre-master secret with the server's public key.
 - D-H: client's public key.
- Certificate_verify
 - Only sent following any client certificate that has signing capability
 - Proves the client is the valid owner of the certificate.

Phase 4. Finish



Master Secret Creation

- The master secret is a one-time 48-byte value.
 - Pre-master secret: by RSA or D-H
 - Master secret is computed from the pre-master secret, client random and server random.

Generation of Cryptographic Parameters

- Generated from the master secret, client random, and server random.
 - Client write MAC secret
 - Server write MAC secret
 - Client write key
 - Server write key
 - Client write IV
 - Server write IV

SSL Handshake Protocol – Key Exchange

- SSL supports several key establishment mechanisms.
- Most common is RSA encryption
 - Client chooses `pre_master_secret`, encrypts using public RSA key of server, sends to server.
- Can also create `pre_master_secret` from:
 - Fixed Diffie-Hellman
 - Server (and possibly Client) certificate contains DH parameters.
 - Ephemeral Diffie-Hellman
 - Server and Client choose fresh Diffie-Hellman components.
 - Anonymous Diffie-Hellman
 - Each side sends Diffie-Hellman values, but no authentication.
 - Vulnerable to man-in-middle attacks.

SSL Handshake Protocol – Entity Authentication

- SSL supports several different entity authentication mechanisms.
- Most common based on RSA.
 - Ability to decrypt `pre_master_secret` and generate correct MAC in `finished` message using keys derived from `pre_master_secret` authenticates server to client
- DSS or RSA signatures on nonces (and other fields, e.g. Diffie-Hellman values).

ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 31

SSL Key Derivation

- Keys used for MAC and encryption in Record Layer derived from `pre_master_secret`:
 - Derive `master_secret` from `pre_master_secret` and client/server nonces using MD5 and SHA-1 hash functions.
 - Derive key material from `master_secret` and client/server nonces, by repeated use of hash functions.
 - Split key material up into MAC and encryption keys for Record Protocol as needed.

ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 32

SSL Handshake Protocol Run

- An illustrative protocol run follows.
- We choose the most common use of SSL.
 - No client authentication.
 - Client sends `pre_master_secret` using Server's public encryption key from Server certificate.
 - Server authenticated by ability to decrypt to obtain `pre_master_secret`, and construct correct `finished` message.
- Other protocol runs are similar.

ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 33

SSL Handshake Protocol Run

M1: C → S: ClientHello

- Client initiates connection.
- Sends client version number.
 - 3.1 for TLS.
- Sends `ClientNonce`.
 - 28 random bytes plus 4 bytes of time.
- Offers list of ciphersuites.
 - key exchange and authentication options, encryption algorithms, hash functions.
 - E.g. `TLS_RSA_WITH_3DES_EDE_CBC_SHA`.

ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 34

SSL Handshake Protocol Run

M2: S → C: ServerHello, ServerCertChain, ServerHelloDone

- Sends server version number.
- Sends `ServerNonce` and `SessionID`.
- Selects single ciphersuite from list offered by client.
 - E.g. `TLS_RSA_WITH_3DES_EDE_CBC_SHA`.
- Sends `ServerCertChain` message.
 - Allows client to validate server's public key back to acceptable root of trust.
- (optional) `CertRequest` message.
 - Omitted in this protocol run – no client authentication.
- Finally, `ServerHelloDone`.

ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 35

SSL Handshake Protocol Run

M3: C → S: ClientKeyExchange, ChangeCipherSpec, ClientFinished

- `ClientKeyExchange` contains encryption of `pre_master_secret` under server's public key.
- `ChangeCipherSpec` indicates that client is updating cipher suite to be used on this session.
 - Sent using SSL Change Cipher Spec. Protocol.
- (optional) `ClientCertificate`, `ClientCertificateVerify` messages.
 - Only when client is authenticated.
- Finally, `ClientFinished` message.
 - A MAC on all messages sent so far (both sides).
 - MAC computed using `master_secret`.

ISE at George Mason University ISA 666 By Dr. Xinyuan (Frank) Wang 36

SSL Handshake Protocol Run

M4: S → C: ChangeCipherSpec, ServerFinished

- ChangeCipherSpec indicates that server is updating cipher suite to be used on this session.
 - Sent using SSL Change Cipher Spec. Protocol.
- Finally, ServerFinished message.
 - A MAC on all messages sent so far (both sides).
 - MAC computed using master_secret.
 - Server can only compute MAC if it can decrypt pre_master_secret in M3.

SSL Handshake Protocol Run

Summary:

M1: C → S: ClientHello

M2: S → C: ServerHello, ServerCertChain, ServerHelloDone

M3: C → S: ClientKeyExchange, ChangeCipherSpec, ClientFinished

M4: S → C: ChangeCipherSpec, ServerFinished

SSL Handshake Protocol Run

1. Is the client authenticated to the server in this protocol run?
2. Can an adversary learn the value of pre_master_secret?
3. Is the server authenticated to the client?

1. No!
2. No! Client has validated server's public key; Only holder of private key can decrypt ClientKeyExchange to learn pre_master_secret.
3. Yes! ServerFinished includes MAC on nonces computed using key derived from pre_master_secret.

Other SSL Handshake Protocol Runs

- Many optional/situation-dependent protocol messages:
 - M2 (S→C) can include:
 - ServerKeyExchange (e.g. for DH key exchange).
 - CertRequest (for client authentication).
 - M3 (C→S) can include:
 - ClientCert (for client authentication),
 - ClientCertVerify (for client authentication).
- For details, see RFC 2246 (TLS).

SSL Handshake Protocol – Additional Features

- SSL Handshake Protocol supports *session resumption* and *ciphersuite re-negotiation*.
 - Allows authentication and shared secrets to be reused across multiple connections.
 - Eg. next webpage from same website.
 - Allows re-keying of current connection using fresh nonces.
 - Allows change of ciphersuite during session.
 - ClientHello quotes old SessionID.
 - Both sides contribute new nonces, update master_secret and key_block.
 - All protected by existing Record Protocol.

Other SSL Protocols

- Alert protocol.
 - Management of SSL session, error messages.
 - Fatal errors and warnings.
- Change cipher spec protocol.
 - Not part of SSL Handshake Protocol.
 - Used to indicate that entity is changing to recently agreed ciphersuite.
- Both protocols run over Record Protocol (so peers of Handshake Protocol).

Alert Protocol

- Convey SSL related alerts to the peer.
- Compressed and encrypted.
- Two types of alerts
 - Fatal
 - SSL immediately terminates the connection.
 - Examples
 - Unexpected message
 - Bad_record_mac
 - Warning
 - Examples
 - Close_notify
 - No_certificate

Change Cipher Spec Protocol

- Session State
 - Current state
 - The session state in effect
 - Pending state
 - The session being negotiated.
- Change Cipher Spec Protocol
 - Cause the pending state to be copied into the current state.

Application Ports Used with SSL

- https 443
- smtps 465
- nntps 563
- ldaps 636
- pop3s 995
- ftp-datas 889
- ftps 990
- imaps 991

SSL and TLS

- TLS1.0 = SSL3.0 with minor differences.
- TLS signalled by version number 3.1.
- Use of HMAC for MAC algorithm.
- Different method for deriving keying material (*master-secret* and *key-block*).
 - Pseudo-random function based on HMAC with MD5 and SHA-1.
- Additional alert codes.
- More client certificate types.
- Variable length padding.
 - Can be used to hide lengths of short messages and so frustrate traffic analysis.
- And more

SSL/TLS Applications

- Secure e-commerce using SSL/TLS.
 - Client authentication not needed until client decides to buy something.
 - SSL provides secure channel for sending credit card information.
 - Client authenticated using credit card information, merchant bears (most of) risk.
 - Wildly successful (amazon.com, on-line supermarkets,...)
 - Some famous disasters (boo.com, webvan), nothing to do with security though.

SSL/TLS Applications

- Secure e-commerce: some issues.
 - No guarantees about what happens to client data (including credit card details) after session: may be stored on insecure server.
 - Does client understand meaning of certificate expiry and other security warnings?
 - Does client software *actually* check complete certificate chain?
 - Does the name in certificate match the URL of e-commerce site? Does the user check this?
 - Is the site the one the client thinks it is?
 - Is the client software proposing appropriate ciphersuites?

SSL/TLS Applications

- Secure electronic banking.
 - Client authentication may be enabled using client certificates.
 - Issues of registration, secure storage of private keys, revocation and re-issue.
 - Otherwise, SSL provides secure channel for sending username, password, mother's maiden name, ...
 - What else does client use same password for?
 - Does client understand meaning of certificate expiry and other security warnings?
 - Is client software proposing appropriate ciphersuites?
 - Enforce from server.

Some SSL/TLS Security Flaws

- (Historical) flaws in random number generation for SSL.
 - Low quality RNG leads to predictable session keys.
 - Goldberg and Wagner, Dr. Dobbs's Journal, Jan. 1996.
 - <http://www.ddj.com/documents/s=965/ddj9601h/>
- Flaws in error reporting.
 - (differing response times by server in event of padding failure and MAC failure) + (analysis of padding method for CBC-mode) = recovery of SSL plaintext.
 - Canvel, Hiltgen, Vaudenay and Vuagnoux, Crypto2003.
 - http://lasecwww.epfl.ch/php_code/publications/search.php?ref=CHVV03
- Timing attacks.
 - analysis of OpenSSL server response times allows attacker in same LAN segment to derive server's private key!
 - Boneh and Brumley, 12th Usenix Security Symposium.
 - <http://crypto.stanford.edu/~dabo/abstracts/ssl-timing.html>

SSH

- SSH overview
- SSH architecture
- SSH security
- Port forwarding with SSH
- SSH applications

SSH Overview

- SSH = Secure Shell.
 - Initially designed to replace insecure rsh, telnet utilities.
 - Secure remote administration (mostly of Unix systems).
 - Extended to support secure file transfer and e-mail.
 - Latterly, provide a general secure channel for network applications.
 - SSH-1 flawed, SSH-2 better security (and different architecture).
- SSH provides security at Application layer.
 - Only covers traffic explicitly protected.
 - Applications need modification, but port-forwarding eases some of this (see later).
 - Built on top of TCP, reliable transport layer protocol.

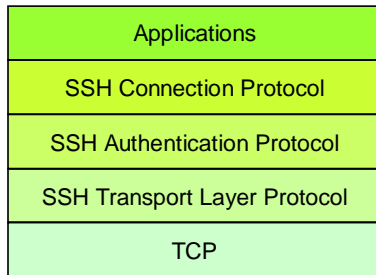
SSH Overview

- SSH Communications Security (SCS).
 - www.ssh.com.
 - Founded by Tatu Ylonen, writer of SSH-1.
 - SSH is a trademark of SCS.
- Open source version from OpenSSH.
- IETF Secure Shell (SECSH) working group.
 - Standard for SSH in preparation.
 - www.ietf.org/html.charters/secsh-charter.html.

SSH-2 Architecture

- SSH-2 adopts a three layer architecture:
- SSH Transport Layer Protocol.
 - Initial connection.
 - Server authentication (almost always).
 - Sets up secure channel between client and server.
 - SSH Authentication Protocol
 - Client authentication over secure transport layer channel.
 - SSH Connection Protocol
 - Supports multiple connections over a single transport layer protocol secure channel.
 - Efficiency (session re-use).

SSH-2 Architecture



SSH-2 Security Goals

- Server (nearly) always authenticated in transport layer protocol.
- Client (nearly) always authenticated in authentication protocol.
 - By public key (DSS, RSA, SPKI, OpenPGP).
 - Or simple password for particular application over secure channel.
- Establishment of a fresh, shared secret.
 - Shared secret used to derive further keys, similar to SSL/IPSec.
 - For confidentiality and authentication in SSH transport layer protocol.
- Secure ciphersuite negotiation.
 - Encryption, MAC, and compression algorithms.
 - Server authentication and key exchange methods.

SSH-2 Algorithms

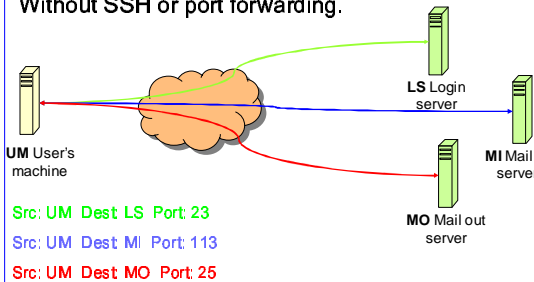
- Key establishment through Diffie-Hellman key exchange.
 - Variety of groups supported.
- Server authentication via RSA or DSS signatures on nonces (and other fields).
- HMAC-SHA1 or HMAC-MD5 for MAC algorithm.
- 3DES, RC4, or AES finalists (Rijndael/Serpent).
- Pseudo-random function for key derivation.
- Small number of 'official' algorithms with simple DNS-based naming of 'private' methods.

SSH-1 versus SSH-2

- Many vulnerabilities have been found in SSH-1 .
 - SSH-1 Insertion attack exploiting weak integrity mechanism (CRC-32) and unprotected packet length field.
 - SSHv1.5 session key retrieval attack (theoretical).
 - Man-in-the-middle attacks (using e.g. dnstf).
 - DoS attacks.
 - Overload server with connection requests.
 - Buffer overflows.
- But SSH-1 widely deployed.
- And SSH-1 supports:
 - Wider range of client authentication methods (.rhosts and Kerberos).
 - Wider range of platforms.

SSH Port Forwarding

Without SSH or port forwarding.

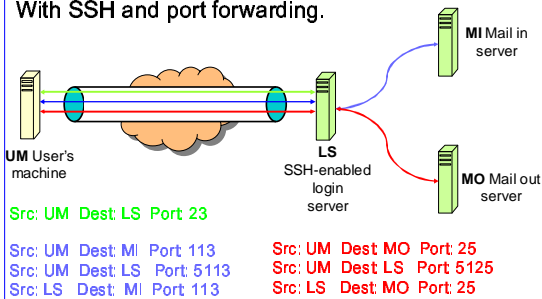


SSH Port Forwarding

- Recall: port number 'identifies' application.
- SSH on local machine:
 - Intercepts traffic bound for server.
 - Translates standard TCP port numbers.
 - E.g. port 113 → port 5113.
 - Sends packets to SSH-enabled server through SSH secure channel.
- SSH-enabled server:
 - Receives traffic.
 - Re-translates port numbers.
 - E.g. port 5113 → port 113.
 - Forwards traffic to appropriate server on network.

SSH Port Forwarding

With SSH and port forwarding.



SSH Applications

- Anonymous ftp for software updates, patches...
 - No client authentication needed, but clients want to be sure of origin and integrity of software.
- Secure ftp.
 - E.g. upload of webpages to webserver using sftp.
 - Server now needs to authenticate clients.
 - Username and password sufficient, transmitted over secure SSH transport layer protocol.
- Secure remote administration.
 - SysAdmin (client) sets up terminal on remote machine.
 - SysAdmin password protected by SSH transport layer protocol.
- Guerilla Virtual Private Network.
 - E.g. use SSH + port forwarding to secure e-mail communications.

6.3 Comparing IPsec, SSL/TLS, SSH

- All three have initial (authenticated) key establishment then key derivation.
 - IKE in IPsec
 - Handshake Protocol in SSL/TLS (can be unauthenticated!)
 - Authentication Protocol in SSH
- All protect ciphersuite negotiation.
- All three use keys established to build a 'secure channel'.

Comparing IPsec, SSL/TLS, SSH

- Operate at different network layers.
 - This brings pros and cons for each protocol suite.
 - Recall 'Where shall we put security?' discussion.
 - Naturally support different application types, can all be used to build VPNs.
- All practical, but not simple.
 - Complexity leads to vulnerabilities.
 - Complexity makes configuration and management harder.
 - Complexity can create computational bottlenecks.
 - Complexity necessary to give both flexibility and security.

Comparing IPsec, SSL/TLS, SSH

Security of all three undermined by:

- Implementation weaknesses.
- Weak server platform security.
 - Worms, malicious code, rootkits,...
- Weak user platform security.
 - Keystroke loggers, malware,...
- Limited deployment of certificates and infrastructure to support them.
 - Especially client certificates.
- Lack of user awareness and education.
 - Users click-through on certificate warnings.
 - Users fail to check URLs.
 - Users send sensitive account details to bogus websites ("phishing") in response to official-looking e-mail.