CSE 410 Fall 2025
Privacy-Enhancing Technologies

Marina Blanton

Department of Computer Science and Engineering
University at Buffalo

Lecture 16: Secure Communication Applications

Key Exchange Applications

Applications building secure communication channels

- Internet Security (IPsec) and Internet Key Exchange (IKE)
    - provides authentication and encryption mechanisms at low network layer
- Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols
    - public-key based authentication technique with wide use on the web
- Secure Shell (SSH)
    - public-key authentication protocol suite for secure remote login

Internet Security (IPsec)

We often think of techniques for protecting messages transmitted over networks as used at the application level

Protection at low level, however, can be effective for securing communication over the Internet

- if we can protect packet addressing information, we can prevent manipulation of this information
- the suite of authentication protocols standardized for internet security is called Internet Key Exchange (IKE)
- IKE includes a variety of algorithms that can be used for integrity and confidentiality protection

Internet Security (IPsec)

Suppose two parties wish to conduct confidential communications by using end-to-end encryption

If end-to-end encryption operates at the application level, then information in the IP header is the subject of attacks

- e.g., IP spoofing (masquerading by using a fake source IP address)

Virtually all active attacks that we've seen so far require Mallory to perform manipulation at the IP level

- message interception, man-in-the-middle, etc.

Internet Security (IPsec)

Internet Protocol Security (IPsec) standard has been developed to add cryptographic protection to the IP header

- it stipulates a mandatory authentication protection for IP header
- it also allows for optional confidentiality protection for the endpoint-identity information

Thus, IPsec effectively prevents many active attacks since IP header manipulation can now be detected

Internet Security (IPsec)

To use IPsec for authentication, there will be an additional header called the Authentication Header (AH)

- it is positioned between the IP header and the higher-layer data
- it provides data integrity and data origin authentication and covers the IP header fields that don't change in transit and packet payload

Optional confidentiality protection can be added

- datagrams called Encapsulating Security Payloads (ESP) are specified for this purpose
- ESP follows AH in a packet and does not protect the packet header

SSL and TLS

The Secure Sockets Layer Protocol (SSL) is an important authentication protocol widely used on the web

- the term "sockets" refers to standard communication channels linking peer processes on client/server machines
- it runs under the application-layer protocols such as HTTP and above network layer protocols (e.g., TCP/IP)
- when socket-layer communications are secured, communications in all application-layer protocols will be secured in the same manner
- SSL was originally developed by Netscape Communications Corporation as an integral part of the web browser and web server
- it was later accepted by other developers

SSL and TLS

SSL eventually evolved into an internet standard called Transport Layer Security (TLS)

- TLS is based on SSL and is not drastically different from SSL
- we'll focus on the standard track TLS in this lecture (SSL is currently deprecated)

TLS is composed of two layered protocols

- the TLS Record Protocol
- the TLS Handshake Protocol

The Handshake Protocol runs on top of the Record Protocol

TLS

The TLS Record Protocol provides secure encapsulation of the communication channel by higher layer application protocols

- it partitions data to be transmitted into blocks
- optionally compresses the data
- applies symmetric key algorithms to achieve confidentiality and integrity
  - used to have separate encryption and MAC algorithms, while the current version TLS 1.3 instead uses authenticated encryption

TLS

The TLS Handshake Protocol is used to set up a secure session connection

- it allows the client and server to authenticate each other
- negotiate cryptographic algorithms and cryptographic keys (for integrity and confidentiality)

TLS

TLS Handshake Protocol

- it is stateful process running on the client and server machines
- a stateful connection is called a session, where the communication peers perform the following steps:
  - exchange hello messages to agree on algorithms, exchange random values, and check for session resumption
  - exchange cryptographic parameters to be able to agree on a secret (called master secret)
  - exchange certificates and cryptographic information to allow them to authenticate one to another
  - generate session secrets from exchanged information
  - verify that their peer calculated the same parameters at the end of the handshake without tampering by an attacker

TLS Handshake Protocol

- The established channel is then passed on to the TLS Record Protocol for processing higher level application communications
- Simplified version of the handshake protocol:
  - $C \rightarrow S$: ClientHello;
  - $S \rightarrow C$: Server Hello,
    ServerCertificate (optional),
    ServerKeyExchange (optional),
    Certificate Request (optional),
    ServerHelloDone;
  - $C \rightarrow S$: ClientCertificate (optional),
    ClientKeyExchange,
    CertificateVerify (optional),
    ClientFinished;
  - $S \rightarrow C$: ServerFinished.

TLS Handshake Protocol

Hello message exchange

- the server and the client exchange protocol versions, random numbers, session ID, cryptographic algorithms, and compression methods
- the client will provide a session ID, if the session is to be resumed

TLS Handshake Protocol

Server's certificate

- if the server's certificate is to be authenticated, the server will send it
- X.509.v3 certificates are used
- each certificate contains sufficient information about the certificate owner's name and public key and the issuing certificate authority

TLS Handshake Protocol

Server's key exchange material

- ServerKeyExchange contains the server's public key material matching the certificate list
- material for DH key agreement will be included here as (p, g, g^{x_S})
- the server that provides non-anonymous services can request client's authentication here as well

TLS Handshake Protocol

Client's response

- the content of ClientKeyExchange message will depend on the public key algorithm agreed between the server and the client
    - in the case of RSA, the client used to generate a 48-byte master key and encrypt it under the server's certified RSA public key
- client's certificate (if any) will be provided at this stage

TLS Handshake Protocol

Finished message exchange

- the client sends the ClientFinished message which used to include a keyed (under the master key) HMAC
- it allows the server to confirm the proper handshake executed at the client side
- the server then sends a similar ServerFinished message

Example

- consider a typical run of the Handshake Protocol where the client chooses to be anonymous
- such one-directional authentication is the most common in e-commerce applications

TLS Handshake Protocol

Example run of the TLS handshake protocol (older version)

- $C \rightarrow S$:
  ClientHello.protocol_version = "TLS Version 1.0",
  ClientHello.random = T_C, N_C,
  ClientHello.session_id = "NULL",
  ClientHello.crypto_suite = "RSA: Enc, SHA-1: HMAC",
  ClientHello.compression_method = "NULL";
- $S \rightarrow C$:
  ServerHello.protocol_version = "TLS Version 1.0",
  ServerHello.random = T_S, N_S,
  ServerHello.session_id = "abc123",
  ServerHello.crypto_suite = "RSA: Enc, SHA-1: HMAC",
  ServerHello.compression_method = "NULL",
  ServerCertificate = server's X.509 certificate,
  ServerHelloDone;

TLS Handshake Protocol

Example TLS handshake protocol (cont.)

- $C \rightarrow S$:
  ClientKeyExchange = RSA_Enc(master_secret),
  ClientFinished = SHA-1(master_secret$||C||N_C||N_S\ldots$);
- $S \rightarrow C$: ServerFinished =
  SHA-1(master_secret$||S||N_S||N_C\ldots$).

TLS Handshake Protocol

TLS 1.3 has significant changes to the Handshake Protocol from prior versions

- the handshake uses fewer interactions
  - encryption can be used as early as in the second message
- the specification mandates forward secrecy
  - this makes the use of many algorithms in previous versions unacceptable
- obsolete and insecure features from TLS 1.2 were removed
  - this includes MD5, SHA-1, RC4, DES, 3DES, etc.

TLS Protocols

Recent specifications of TLS also include other protocols

- Alert protocol
    - used to convey TLS-related alerts (based on pre-defined codes) to the peer entity
- Heartbeat protocol
    - ensures that the other party is still alive
    - generates activity (prevents firewall closure and enables the use of connectionless service)

SSL/TLS Security

Over the years, many attacks on and abuses of SSL and TLS have been discoreved

- these include attacks on the handshake and record protocols, certificate-related attacks, and others
- the largest attack was on OpenSSL's implementation of TLS's Heartbeat Protocol
    - the Heartbleed expoit was able to read memory of a remote server
    - the memory could store private keys, user ids and passwords, and other sensitive information

Secure Shell (SSH)

Secure Shell (SSH) is a public-key based authentication protocol suite

- it enables a user to securely login onto a remote server host machine from a client machine through an insecure network
- it also allows to securely execute commands on the remote host and move files from one host to another

SSH is in wide use today

- a server can be run on many operating systems
  - a server will work if the operating system supports interactive command sessions for remote users
- a client can be run on any operating system

Secure Shell (SSH)

The basic idea behind SSH protocol

- a user on a client machine downloads a public key of a remote server
- he establishes a secure channel between the client and the server using the downloaded key and the user's cryptographic credentials
- even if the user's credentials are only a password, they will be sent encrypted to the server

Secure Shell (SSH)

The SSH protocol suite consists of three major components

- SSH Transport Layer Protocol
  - provides server authentication to the client
  - the server host uses its public-private key pair and the client uses the host's public key
- SSH User Authentication Protocol
  - it runs over the unilateral authentication channel established by the transport layer protocol
  - it achieves entity authentication from a client-side user to the server
  - it can be based on a public-key, password, or other mechanisms
  - at the end of it a mutually authenticated secure channel is formed

Secure Shell (SSH)

The SSH protocol major components (cont.)

- SSH Connection Protocol
  - materializes an encrypted communication channel
  - tunnels it into several secure logical channels for various communication purposes

SSH-1 was discovered to have design flaws and should be avoided

SSH-2 provides better security and has been standardized by IETF

Secure Shell (SSH)

SSH Transport Layer Protocol

- a server host maintains a public-private key pair for each required signature algorithm
- a client must have a priori knowledge of the server's host public key
- SSH supports two trust models on servers' public keys:
    - the client has a local database of host names and the corresponding public keys
    - the host name and its public key is certified by a trusted certification authority

Secure Shell (SSH)

- The list of known hosts' public keys is stored in `$HOME/.ssh/known_hosts`
- An entry might look like this:

```
timberlake.cse.buffalo.edu,128.205.36.8 ssh-rsa AAAAB3
NzaC1yc2EAAAABIwAAAIEArov5ZnZlpAETHjEvmLk7J/1g65JYIHYq
r6lfYWTHlTT20+IxfcWGX4vtsfcYwwpzLxhwlWTjah7/fK2MwgU1Lo
/HDDcjDZrpCFXN4pTAosLUdmV5uqadwNFbbtDTAESrjxJ/beAEwYZ/
Gvy/V36rZRFFWeFBMrDUiTXirc0NP80=
```

Secure Shell (SSH)

- When a user connects to a machine, the public key of which is not stored locally or has changed, SSH will ask the user to verify the public key
- Such verification happens in the form of a fingerprint
  - fingerprint(host key)= h(host key), where h is an agreed upon cryptographic hash function
- To verify the key, you might
  - have an authenticated copy of the key on another machine
  - have generated the key (for your workstation) and know it
  - call the security administrator and verify the fingerprint over the phone

Secure Shell (SSH)

SSH key exchange protocol

- each key exchange is initiated by the client
  - the server listens on a specific port, commonly port 22
- SSH-2 uses Diffie-Hellman key exchange for session key agreement
- we'll use the following notation:
  - C is the client and S is the server
  - p is a large prime, and $g \in \mathbb{Z}_p^*$ is an element with necessary properties
  - V_C and V_S are the client's and the server's versions, respectively
  - pk_S is the server's public key
  - I_C and I_S are the client's and the server's initial messages exchanged before this part begins

Secure Shell (SSH)

SSH key exchange protocol

- C chooses a random number x_C, computes $y_C = g^{x_C}$ and sends it to S
- S chooses a random number x_S, computes

$$y_S = g^{x_S},\ k = y_C^{x_S} = g^{x_C x_S},$$
$$H = h(V_C || V_S || I_C || I_S || pk_S || y_C || y_S || k),\ \mathsf{sig}_{sk_S}(H)$$

  and sends pk_S, y_S and $\mathsf{sig}_{sk_S}(H)$ to C
- C verifies that pk_S is really the host key for S
- C then computes $k = y_S^{x_C} = g^{x_C x_S}$, H (as above), and verifies the signature $\mathsf{sig}_{sk_S}(H)$, and accepts if verification passes

Secure Shell (SSH)

After the key exchange, the parties use k to secure the communication

The next step is to authenticate the client using a suitable mechanism

Secure Shell (SSH)

One of the goals of SSH is to improve security on the internet in a progressive manner

- this is why any suitable method of verifying the server's key is permitted
- a variety of user authentication mechanisms is supported as well
- this allows for quick deployment and backward compatibility
- this is why it's been popularly implemented and widely used

Summary

Key establishment

- Diffie-Hellman based key agreement protocols

Applications and standards

- IPsec and IKE: IP packet level integrity and confidentiality
- SSL and TLS: socket level secure channel for all applications
- Secure Shell: secure remote login and file transfer