18 - Security in Networks
<table>
<thead>
<tr>
<th>Target</th>
<th>Vulnerability</th>
</tr>
</thead>
</table>
| **Precursors to attack** | Port scan  
<p>|              | Social engineering                 |
|              | Reconnaissance                      |
|              | OS and application fingerprinting   |
| <strong>Authentication failures</strong> | Impersonation                         |
|              | Guessing                           |
|              | Eavesdropping                       |
|              | Spoofing                            |
|              | Session hijacking                   |
|              | Man-in-the-middle attack            |
| <strong>Programming flaws</strong> | Buffer overflow                     |
|              | Addressing errors                   |
|              | Parameter modification, time-of-check to time-of-use errors |
|              | Server-side include                 |
|              | Cookie                              |
|              | Malicious active code: Java, ActiveX |
|              | Malicious code: virus, worm, Trojan horse |
|              | Malicious typed code                |</p>
<table>
<thead>
<tr>
<th>Confidentiality</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Protocol flaw</td>
</tr>
<tr>
<td>- Eavesdropping</td>
</tr>
<tr>
<td>- Passive wiretap</td>
</tr>
<tr>
<td>- Misdelivery</td>
</tr>
<tr>
<td>- Exposure within the network</td>
</tr>
<tr>
<td>- Traffic flow analysis</td>
</tr>
<tr>
<td>- Cookie</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Protocol flaw</td>
</tr>
<tr>
<td>- Active wiretap</td>
</tr>
<tr>
<td>- Impersonation</td>
</tr>
<tr>
<td>- Falsification of message</td>
</tr>
<tr>
<td>- Noise</td>
</tr>
<tr>
<td>- Web site defacement</td>
</tr>
<tr>
<td>- DNS attack</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Protocol flaw</td>
</tr>
<tr>
<td>- Transmission or component failure</td>
</tr>
<tr>
<td>- Connection flooding, e.g., echo-chagen, ping of death, smurf, syn flood</td>
</tr>
<tr>
<td>- DNS attack</td>
</tr>
<tr>
<td>- Traffic redirection</td>
</tr>
<tr>
<td>- Distributed denial of service</td>
</tr>
</tbody>
</table>
Encryption

- Arguably most important/versatile tool for network security

- We have seen that it can be used for:
  - Confidentiality/Privacy
  - Authentication
  - Integrity
  - Limiting data access

- **Kinds** of encryption in networks:
  i. Link encryption vs. end-to-end (e2e) encryption
  ii. Virtual private network (VPN)
  iii. PKI and certificates
  iv. SSH protocol
  v. SSL protocol (a.k.a. TLS protocol)
  vi. IPsec protocol suite
  vii. Signed code
  viii. Encrypted e-mail
Link encryption

1) **Link encryption** = between 2 adjacent hosts
   - Data encrypted just before it is placed on physical communication links
     - At Layer 1 (or, perhaps, Layer 2)
   
   - Properties of link encryption
     - Packets **unprotected** inside S’s/R’s host
       - i.e., unprotected at layers 2-5 of S’s/R’s host (in plaintext)
     - Packets **protected** in transit between *pairs of* hosts
     - Packets **unprotected** inside intermediate hosts
       - i.e., unprotected at layers 2-3 of intermediate hosts
       => unprotected at data link and network layers at intermediate hosts (if link encryption at Layer 1)
         - Layers 2-3 provide **addressing and routing**
Figure 7-21  Link Encryption.
Link encryption

- Link encryption is transparent to users, their applications, and their OSs
  - Encryption service provided by physical (or data) layer
  - Can use encryption hardware (link encryption device)

- Message under link encryption
  - See which portions encrypted, which exposed
    - Only part of data link header & trailer created after encryption is exposed

- Link encryption is useful when transmission line is most vulnerable in a network
  - i.e., when S’s host, intermediate hosts, R’s host are reasonably secure (so messages at their Layers 2-5 can be exposed)
Figure 7-22  Message Under Link Encryption.
End-to-End encryption

2) End-to-end encryption = between 2 user applications
   ▪ Data encrypted as “close” to app as possible
     ▪ At Internet Protocol layer 5 (Application layer)

   ▪ Properties of e2e encryption
     ▪ Packets protected all the way once they “exit” S’s application
       & before they enter R’s application
       ▪ Packets protected (in ciphertext) inside S’s/R’s host
       ▪ Packets protected in transit between S’s & R’s hosts
         Including protection inside intermediate hosts
         ▪ i.e., protected at layers 1-3 of intermediate hosts
         Layers 1-3 provide physical connectivity, addressing and
         routing for packets
Figure 7-23 End-to-End Encryption.
End-to-End encryption

- E2e encryption is visible either to users or their apps
  - Encryption service provided by application or OS
    - Possibly provided only upon explicit user’s request => visible to user
  - Encryption by software

- Message under e2e encryption
  - See which portions encrypted, which exposed
    - Only user’s message (user’s data) encrypted
    - All headers & trailers exposed (all created after encryption)

- E2e encryption is useful when transmission lines and intermediate hosts are insecure
Figure 7-24  End-to-End Encrypted Message.
Link vs. end-to-end encryption

- **Comparison** of link vs. e2e encryption
  - Encryption of packets (whether link or e2e encryption) is no silver bullet
  - No guarantees of packet security

1) **Link encryption** — encrypts all traffic over physical link
   - Typically host H has one link into network
   - => link encryption encrypts all H’s traffic

   - Every H —including intermediate hosts— receiving traffic via link encryption must have decryption capabilities
     - Either (pairs of) hosts share symmetric key
     - OR
     - Hosts use asymmetric keys

   - All hosts along a path from S to R must provide link encryption to prevent (“partial”) packet exposure
   - link encryption may be provided on all network links
Link vs. end-to-end encryption

2) End-to-end (e2e) encryption — encrypts traffic only between 2 apps  (“virtual crypto channel between 2 applications”)
  ▪ Intermediate hosts don’t need to decrypt-encrypt pkts
    => intermediate hosts don’t need encryption facilities
      ▪ All intermediate hosts save time/processing
  ▪ Encrypts only some messages between 2 applications
    ▪ If no need to encrypt all messages => even S’s and R’s hosts save time/processing
      ▪ If needed, can encrypt all messages
  ▪ Using asymmetric keys requires fewer keys than using symmetric keys  (n key pairs vs. n*(n-1)/2 keys)
Link vs. end-to-end encryption

- **Comparison conclusions**
  - **Link** encryption:
    - Faster
    - **Uses fewer keys** (1 Key pair per host pair vs. 1 Key pair per app pair)
  - **End-to-end (e2e)** encryption:
    - More flexible
    - More selective (can select only some messages for encryption)
    - User-level, can be integrated with application

- **Optimize** whether link or e2e encryption better for you
  - If needed for higher security, use link *and* e2e encryption together
  - e.g., user not trusting network link encryption can use app with e2e encryption
Virtual private network (VPN)

- Virtual private network (VPN) = connection over public network giving its user impression of being on private network
  - It could be viewed as “logical link” encryption
    - Could be viewed as e2e encryption between client & server
    - Protecting remote user’s connection with her network

- Greatest risk for remote connection via public network:
  - Between user’s workstation (client) and perimeter of “home” network (with server)
  - Firewall protects network against external traffic
Figure 7-26 Establishing a Virtual Private Network.

1. Client authenticates to firewall
2. Firewall replies with encryption key
3. Client and server communicate via encrypted tunnel
Public key infrastructure (PKI) = enables use of public key cryptography (asymmetric cryptography)
- Usually in large & distributed environment

Elements of PKI:
1) Policies (higher level than procedures)
   - Define rules of operation
     - e.g., how to handle keys and sensitive info
     - e.g., how to match control level to risk level

2) Procedures (lower level than policies)
   - Dictate how keys should be generated, managed, used

3) Products
   - Implement policies and procedures
     - Generate, store, manage keys
PKI and certificates

**PKI services:**

1) PKI creates certificates
   - Certificate binds entity’s identity to entity’s public key
     - Entity = user or system or application or ...

2) PKI gives out certificates from its database

3) PKI signs certificates
   - Adding its credibility to certificate’s authenticity

4) PKI confirms/denies validity of a certificate
   - When queried about it

5) PKI invalidates certificates
   - For entities that are no longer certified by PKI
     OR
   - For entities whose private key has been exposed
PKI and certificates

- PKI sets up:
  1) Certificate authorities (CAs)
  2) Registration authority

1) Certificate authority (CA)

- CA can be in-house or external (commercial TTP = trusted third party)
- CA is trusted
  - Entities delegate to CA creation, issuance, acceptance, and revocation of their certificates
- CA actions:
  - Managing public key certificates (whole life cycle)
  - Issuing certificates by binding entity’s identity to its public key
    - Binding is done via CA’s digital signature
  - Determining expiration dates for certificates
  - Revoking certificates when necessary
    - By publishing revocation lists
PKI and certificates

- Example of CA analog: credit card company (CCC)
- Certificate analog: credit card (binds identity to account)
- Revocation list analog: lists of invalid credit cards

  - CCC is trusted
    - Customers delegate to CCC creation, issuance, acceptance, and revocation of their credit cards
  - CCC actions:
    - Managing credit cards (whole life cycle)
    - Issuing credit cards by binding customer’s identity to customer’s account
      - Binding is done via CCC’s protected databases
    - Determining expiration dates for credit cards
    - Revoking credit cards when necessary
      - By checking list of invalid credit cards
        (before computer-verification transaction era, CCC published booklets of invalidated credit cards)
PKI and certificates

2) Registration authority (RA) = interface between user and CA
   - Duties:
     - Capture and authenticate user’s identity
     - Submit certificate requests to appropriate CA
   - Analog: U.S. Citizen applying for passport and U.S. Postal Service (USPS)
     - Passport (official U.S. authentication) <-> certificate
     - USPS authenticates citizen
       - By verifies citizen's driver license + other proofs of identity
     - USPS submits passport request forms to appropriate passport office of the U.S. Gov’t
       - Passport office <-> CA
     - USPS brings passport to customer’s home
   - Note: Trustworthiness of USPS authentication determines level of trust that can be placed in passports
PKI and certificates

- Major PKI product vendors in the U.S.:
  - Baltimore Technologies
  - Northern Telecom/Entrust
  - Identrus
SSL protocol (a.k.a. TLS protocol)

- **SSL protocol (v3)** = (approx.) **TLS protocol** - interfaces between application (on client C) and TCP/IP protocols to provide server S authentication, optional C authentication, and encrypted communication channel between C and S for session between C and S
  - **SSL** = Secure Sockets Layer / **TLS** = Transport Layer Security
    - Originally defined by Netscape to protect browser-to-server communication

- Simple but effective – most widely used secure communication protocol on Internet (incl. WWW browsers/servers)

- Involves negotiation between C and S
  - Negotiate which encryption suite to use for session
    - e.g., DES? RC4 w/ 128-bit/40-bit key? RC2? Fortezza? [Bishop]
  - Negotiate which hashing technique to use for session
    - e.g., SHA1 or MD5?
SSL protocol (a.k.a. TLS protocol)

- SSL use scenario (handshake protocol)
  - C requests an SSL session by sending: Hello-C, Rand-C (random number), list of cipher (encryption) algorithms & hash algorithms known to C
    - Hash used to checksum messages
  - S responds with msgs including: Hello-S, Rand-S, cipher & hash algorithm selected by S (from C’s list), S’s certificate, $K_{PUB-S}$,
    [OPTIONAL: request for cert. fr. C]
    - C can use S’s certificate (X.509v3 cert.) to verify S’s authenticity
  - [OPTIONAL: C replies with: C’s certificate]
  - C returns “pre-master secret” encrypted under $K_{PUB-S}$
    - Pre-master secret - e.g., 48 random B if selected cipher is RSA [Bishop-CompSec-A&S, p.296]
SSL protocol (a.k.a. TLS protocol)

...continued...

- C and S calculate “master secret” using:
  - “Pre-master secret”
  - Constant strings ‘A’, ‘BB’ and ‘CCC’
  - Rand-C and Rand-S,
  - SHA hashing algorithm

- C and S switch to encrypted communication using “master secret” as session key

- C and S exchange application data for session duration (i.e., for as long as they stay connected)

- TLS is potentially vulnerable to MITM attacks

[ibid, p. 294]
[Conklin et al., p.163]
The Handshake

Client		Server

Client_hello

Server_hello

Certificate

Server_key_exchange

Certificate_request

Server_hello_done

Certificate

Client_key_exchange

Certificate_verify

Change_cipher_spec

Finished

Note: Dotted transfers are optional or situation-dependent messages that are not always sent.

Figure 1: The SSL Handshake

1
Figure 2: Microsoft Internet Explorer® List of Trusted Root Certification Authorities
Forcing weak cryptography in SSLv2

- When the cryptographic suite is being established, one could change the list of protocols and delete all strong protocols
  - The server is then forced to use a weak protocol which can then be exploited.
### HTTP Traffic Analysis

**HTTP Request**

- **Method**: GET
- **URL**: https://example.com
- **Header**: Accept: */*

**HTTP Response**

- **Status Code**: 200
- **Content Type**: text/html
- **Body**: HTML Content

### SSL/TLS Protocol Details

#### Handshake Protocol:

- **Version**: TLS 1.2
- **Cipher Suites**:
  - TLS_RSA_WITH_AES_128_GCM_SHA256
  - TLS_RSA_WITH_AES_256_GCM_SHA384

#### Key Exchange:

- **Elliptic Curve**: P-256
- **Key Size**: 256 bits

**Certificate Details**

- **Subject**: example.com
- **Issuer**: Let's Encrypt
- **Validity**: Valid for 3 years

### Security

- **SSL/TLS Version**: 3.3
- **Cipher Strength**: Good
- **SSL Protocol**: TLS 1.2

**Network Traffic**

<table>
<thead>
<tr>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000</td>
<td>192.168.1.102</td>
<td>192.168.1.102</td>
<td>TCP</td>
<td>1859 &gt; https [SYN] Seq=0 Ack=0 Win=1</td>
</tr>
<tr>
<td>0.000005</td>
<td>66.161.19.11</td>
<td>66.161.19.11</td>
<td>TCP</td>
<td>1859 &gt; https [SYN] Seq=0 Ack=0 Win=1</td>
</tr>
<tr>
<td>0.000010</td>
<td>192.168.1.102</td>
<td>192.168.1.102</td>
<td>TCP</td>
<td>1859 &gt; https [SYN] Seq=0 Ack=0 Win=1</td>
</tr>
</tbody>
</table>

**SSL/TLS Record Layer**

- **Length**: 76
- **Handshake Message Type**: Client Hello

**Cipher Suits**

- **TLS_RSA_WITH_AES_128_GCM_SHA256 (0x000004)
- **TLS_RSA_WITH_AES_256_GCM_SHA384 (0x000005)
- **TLS_RSA_WITH_AES_128_CBC_SHA (0x000004)
- **TLS_RSA_WITH_AES_256_CBC_SHA (0x000005)
- **SSL3_RC4_128_SHA (0x000003)
- **SSL3_TLSENCRYPT_SHA1 (0x000002)
Based on the testing, the site is assigned a grade. The following grading scheme is used to determine a letter grade:

<table>
<thead>
<tr>
<th>SSL Results</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only Excellent Security Ciphers supported</td>
<td>A+</td>
</tr>
<tr>
<td>Only Strong and Higher Security Ciphers supported</td>
<td>A</td>
</tr>
<tr>
<td>Any Weak Security Cipher supported</td>
<td>B</td>
</tr>
<tr>
<td>Only Weak Security Ciphers supported</td>
<td>C</td>
</tr>
<tr>
<td>Any No Security Cipher supported</td>
<td>D</td>
</tr>
<tr>
<td>Only Weak and No Security Ciphers supported</td>
<td>D</td>
</tr>
<tr>
<td>Only No Security Ciphers supported</td>
<td>F</td>
</tr>
</tbody>
</table>

Table 2: Grading Scheme
SSH protocol (SSH = Secure SHell)

- **SSH protocol** (newer: v.2) – provides authenticated and encrypted communication with shell/OS command interpreter
  - Originally defined for Unix
- Replaced insecure utilities for remote access
  - Such as Telnet / rlogin / rsh
- Protects against spoofing attacks (falsifying one end of communication, incl. masquerading, sesssion hijacking, MITM) & message modification / falsification
- Involves negotiation between local and remote sites
  - Negotiate which encryption algorithm to use
    - e.g., DES? IDEA? AES?
  - Negotiate which authentication technique to use
    - e.g., public key? Kerberos?