Networking Security (con’t)
Common Attacks & Countermeasures

- Finding a way into the network
  - Firewalls
- Exploiting software bugs, buffer overflows
  - Intrusion Detection Systems (IDS)
- TCP hijacking
  - IPSec
- Denial of Service
  - Ingress filtering, IDS
- Packet sniffing
  - Encryption (SSH, SSL, HTTPS)
- Social problems
  - Education
Denial-of-service

- DOS and DDOS
- Access over Internet must be unimpeded
  - Flooding attacks in which attackers try to overwhelm system resources
- Denial of service (DoS) attacks disrupt availability
  - Distributed DoS is a coordinated attack from multiple attackers
Independent Availability Zones

- Amazon Availability Zones: distinct physical locations of the same region
- Each availability zone runs on its own infrastructure, independent power, cooling, network
Availability disruption by malicious attacks

- Example: SYN flood
  - Problem: server cannot distinguish legitimate handshake from ones from attackers
  - Flood can overwhelm communication medium
    - Can’t do anything about this (except buy a bigger pipe)
  - Flood can overwhelm resources on legitimate system
Regular TCP handshake
Too many half-opened TCP connections at server creates a DoS attack

- Server is waiting client to finish the connection
- Buffer space is full and legitimate connection cannot be completed
Q: Can we just block the IP that sends too many SYN requests?
Prevention of SYN flood: SYN Cookies

- **SYN cookie**: server does not really keep the client’s state until the handshake is done

- **How it works**
  - Server embeds the state in the sequence number
  - When SYN received, server computes a sequence number to be function of source, destination, counter, and a secret
    - The function may be one-way hash function
    - The secret is known only by the server
    - Use as reply SYN sequence number
    - When reply ACK arrives, validate it
  - The sequence number must be hard to guess
**SYN Cookie**

Client cannot forge a sequence number without actually engaging in handshake.

$y = H(\text{source}, \text{destination}, \text{secret}, \text{counter})$

- $s$ is secret known only by server
- $H$ is a one-way hash function

Server no longer keeps client’s state, compute seq# using secret $s$

Server verifies sequence number with $s$
Alternative Prevention to SYN Flood: Adaptive Time-out

- Change time-out time as space available for pending connections decreases

**Example:**

- Time-out period shortened from 75 to 15 sec
- Formula for queueing pending connections changed:
  - Process allows up to \( b \) pending connections on port
  - \( a \) number of completed connections but awaiting process
  - \( p \) total number of pending connections
  - \( c \) tunable parameter
  - Whenever \( a + p > cb \), drop current SYN message
Ping of Death, Ping Flooding

- Ping of death
  - Attacker sends over-sized ping packets
    - 65,536 bytes as opposed to 56 bytes
  - Cause buffer overflow
  - Fixed already

- Ping flooding attack
  - Attacker sends many ping requests to victim
  - E.g., Attacker on a fast 100MB/s link, victim on a 10MB/s link
  - A DoS attack
Smurf Attack:

- attacker generates traffic to victim by broadcasting ping requests with spoofed source IP

Send your echo replies to me, I am 8.5.3.1
Flash crowd and DDoS

- Flash crowd –users visiting a website due to certain events
  - E.g., visited cnn.com after 911 tragedy
  - How to distinguish flash crowd and DDoS (e.g., CAPTCHA)

- DDoS attacks by bots, even hacked IoT devices

- Can attacker trick human users into launching DDoS?
  - Makes use of major hubs on online social network
  - E.g., post a victim’s URL in the comment of a popular facebook page
TLS/SSL → HTTPS

to prevent man-in-the-middle and packet sniffing
Man in the Middle

- C.I.A. when an adversary controls everything between the end points (i.e., client and server)
Challenges

- **Authentication**
  - The client must be able to verify that it is talking to the desired server

- **Confidentiality**
  - Data transmitted between the client and server must not be attacker visible

- **Integrity**
  - Data transmitted between client and server must not be attacker modifiable
Authentication - Certificates
Certificates

- This public key with SHA-256 hash (XXX) belongs to the site (name, e.g., Amazon.com)
  - Signed by a trusted authority (digital signature)
    - Called a Certificate Authority (CA)

- Your browser (e.g., Chrome) trusts a set of CAs as root CAs
  - Shipped with the public keys of the root CAs
SSL/TLS

- **SSL** (Secure Socket Layer) -- Netscape
  - Version 2.0 -- Broken, don't use (disabled by default in modern browsers)
  - Version 3.0 (older but still in use, http://disablessl3.com/)
- **TLS** (Transport Layer Security) -- IETF Standard
  - Version 1.0, 1.1, 1.2 (commonly used),
  - Version 1.3 (newer version)
SSL certificates

- A trusted authority vouches that a certain public key belongs to a particular site
- Browsers ship with CA public keys for a large number of trusted CAs
- Important fields:
  - Common Name (CN) [e.g., *.google.com]
  - Expiration Date [e.g., 2 years from now]
  - Subject’s Public Key
  - Issuer -- e.g., Verisign
  - Issuer’s signature
- Common Name field
  - Explicit name, e.g., cs.vt.edu
  - Or wildcard, e.g., *.vt.edu
Public-key cryptography

- Bob generates: $B_{Pub}$, $B_{Priv}$
- Alice can encrypt messages to Bob:
  - using $B_{Pub}$ to encrypt messages, only Bob can decrypt
- Bob can sign messages that Alice can verify:
  - using $B_{Priv}$ Bob signs message, anyone can verify
### X509 Certificates

<table>
<thead>
<tr>
<th><strong>Subject:</strong></th>
<th>C=US/O=Google Inc/CN=www.google.com</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Issuer:</strong></td>
<td>C=US/O=Google Inc/CN=Google Internet Authority</td>
</tr>
<tr>
<td><strong>Expiration Period:</strong></td>
<td>Jul 12 2010 - Jul 19 2012</td>
</tr>
<tr>
<td><strong>Public Key Algorithm:</strong></td>
<td>rsaEncryption</td>
</tr>
<tr>
<td><strong>Signature Algorithm:</strong></td>
<td>sha1WithRSAEncryption</td>
</tr>
</tbody>
</table>
How does Alice (web browser) obtain $B_{Pub}$?

1. Choose $(B_{Pub}, B_{Priv})$
   --- $B_{Pub}$ and proof he is "Bob" -->
   
2. Checks proof
   <--- Signs certificate with $CA_{Priv}$--
   "Bob's key is $B_{Pub}$-- Signed, CA"

3. Keeps cert on file

4. Goes to Bob.com
   <--- Sends cert to Alice ----
   "Bob's key is $B_{Pub}$-- Signed, CA"

5. Verifies signature on cert using $CA_{Pub}$
In Summary: Certificates

- How a website prove itself?
  - Goes to Certificate Authority (CA)
  - Obtain a Certificate signed by CA, certificate contains Bob’s public key

Browser Alice

- $PK_{CA}$
- verify Cert

Server Bob

- choose $(SK, PK)$
- $PK_{CA}$

CA

- check proof
- $SK_{CA}$

Bob

- Bob’s key is $PK$
- Bob’s key is $PK$

PK and proof “I am Bob”

issue Cert with $SK_{CA}$:
Certificate validation

- How is identity verification done?
  - Typically ’DV’ (**domain validation**) – just an email based challenge to the address in the domain registration records (Or some default email address); **minimally secure**.

- Extended validation (EV) – requires verification that a legal entity owns a particular domain name

- Cert has expiration date (e.g., one year ahead)
  - Why expire?
Avoiding Revoked Certificates

- Common reasons for revocation
  - Domain ownership change
  - Private key compromise

- Browsers periodically check for revoked certificates
  - Online Certificate Status Protocol (OCSP)

- Browsers get push updates of trusted CAs
Certificate chains

- CAs can delegate ability to generate certificates for certain names
  - Intermediate CA
  - Root CA signs "certificate issuing certificate" for delegated authority
- Delegated authority signs cert for "cs.vt.edu"
  - Delegated CA certificate: "pubkey=.... is allowed to sign certs for *.vt.edu"
- Browser peels off signatures until it gets to CA that it trusts
  - "Chain of trust"
- More than 1000 trusted parties today, can sign for any domain!
Self-signed certificates

- Issuer signs its own certificate
  - A loop in the owner and signer

- Avoid CA fees, useful for testing

- Browsers display warnings that users have to override
Certificate Summary

Certificates are digital signatures by trusted CAs

Each browser trusts a set of CAs
  CAs can sign certificates for new CAs
  CAs can sign certificates for any web site

If a single CA is compromised, then the entire system is compromised

We ultimately place our complete trust of the Internet in the weakest CA (and the browser)
Confidentiality and Integrity - SSL/TLS tunnel
HTTPS

- HTTPS is HTTP through an SSL/TLS encrypted, MAC’d tunnel
Where does TLS live?

- Application (HTTP)
- Transport (TCP)
- Network (IP)
- Data-Link (1gigE)
- Physical (copper)
SSL/TLS: a general-purpose tunnel

- Arguably the most important (and widely used) cryptographic protocol on the Internet

- Almost all encrypted protocols (minus SSH) use SSL/TLS for transport encryption

- HTTPS, POP3, IMAP, SMTP, FTP, NNTP, XMPP (Jabber), OpenVPN, SIP (VoIP), …
TLS

Confidentiality (Symmetric Crypto) ✔

Message Integrity (HMACs) ✔

Authentication (Public Key Crypto) ✔
Cryptographic Primitives

Symmetric Encryption
- RSA

HMAC Certificate
- PKI

Public Key
- RC4

Diffie-Hellman
- DSA

ECDSA
- Asymmetric Encryption

Typical HTTPS Connection
TLS: ciphers, key sharing, browsers

“the handshake”
Client Hello: Here’s what I support and a *random*
Client Hello: Here’s what I support and a random

Server Hello: Chosen Cipher, server’s random

Certificate: Here is my “X509 Certificate”

Here’s your random encrypted and/or signed
Client Hello: Here’s what I support and a random

Server Hello: Chosen Cipher

Certificate: Here is my “X509 Certificate”

Here’s your random encrypted and/or signed

Client Key Exchange: encrypted (secret)

Change Cipher Spec
HTTPS key exchange

1. RSA key exchange
   - Use RSA for encryption to achieve confidentiality

2. Ephemeral Diffie Hellman (EPH)
   - Use RSA for signature to achieve authentication

Which one to use?
- RSA is simpler, EPH is more work
- At the end of the exchange, a secret is used to generate 4 keys
Client Hello: Here’s what I support and a random

Server Hello: Chosen Cipher

Certificate: Here is my “X509 Certificate”

Here’s your random encrypted and/or signed

Client Key Exchange: encrypted(secret)

Change Cipher Spec

Change Cipher Spec
Brief overview of HTTPS

Browser → Client-Hello → Server-Hello + Server-Cert (PK)

Key Exchange (several options)

Client-Key-Exchange: E(PK, k)

Finished

HTTP data encrypted with k

Cert

Most common: server authentication only
HTTPS provides

- **Authentication**
  - Client verifies the server’s domain & public key based on certificate
    - Thus, no *man-in-the-middle* attack

- **Data confidentiality**
  - Communication is encrypted and can only be decrypted by server

- **Data integrity**
  - The use of unique nonce
    - Thus, no *replay* attacks
  - Message authentication code
    - Thus, no *tampering*
Attacking HTTPS

- Attack the weakest Certificate Authority
- Attack browser implementations
- Magically notice a bug in a key generation library that leads you to discovering all the private keys on the Internet
- Attack the cryptographic primitives
DNS poisoning

Slides credit to Ninghui Li
DNS Caching

- DNS responses are cached
  - Quick response for repeated translations
  - Useful for finding servers as well as addresses
    - NS records for domains

- Negative results are cached
  - Save time for nonexistent sites, e.g. misspelling

- Cached data periodically times out
  - Each record has a TTL field
DNS Cache Poisoning

- Attacker wants his IP address returned for a DNS query
- When the resolver asks ns1.google.com for www.google.com, the attacker could reply first, with his own IP
- What is supposed to prevent this?

- Transaction ID
  - 16-bit random number
  - The real server knows the number, because it was contained in the query
  - The attacker has to guess
Responding Before the Real Nameserver

- An attacker can guess when a DNS cache entry times out and a query has been sent, and provide a fake response.
- Successful fake response: the transaction ID need to match the query
- CERT 1997: sequential transaction ID and is easily predicted
- Fixed by using random transaction IDs
DNS cache poisoning (Vulnerability)

- Improve the chance of responding before the real nameserver
  - Have hundreds of clients send the same DNS request to the name server
    - Each generates a query
  - Send hundreds of reply with random transaction IDs at the same time
  - Due to the Birthday Paradox, the success probability can be close to 1

  - For a group of 23 people, the chance that at least two people have the same birthday $P > 0.5$
  - For any two people have different birthday $P_x = \frac{365 \times 364 \times \ldots \times 343}{365^{23}}$
  - $P = 1 - P_x = 0.51$

- Using a large number of queries and responses → high chance at least one pair matches
DNS Poisoning Defenses

• Difficulty to change the protocol
  ▪ Protocol stability (embedded devices)
  ▪ Backward compatible

• Long-term
  ▪ Cryptographic protections
    ▪ E.g., DNSSEC, DNSCurve
  ▪ Require changes to both recursive and authority servers

• Source port randomization
  • Add 16-bits entropy
  • resource intensive (select on a potentially large pool of ports)