Secure HTTP (HTTPS)

Pat Morin
COMP 2405
Contents

• Symmetric encryption
• Public-Key encrypt
• Public-Key Infrastructure
• HTTPS
HTTP and Security

• Most users believe that HTTP transactions take place between clients and servers with no third party involved
• This isn't at all true
**Attackers Could Be Everywhere**

- Any host on the traceroute path can
  - Monitor traffic between the client and server
  - Modify traffic between the client and server
  - Pretend to be the server (to the client)
  - Pretend to be the client (to the server)

- Anyone on your local network can
  - Monitor all traffic on the local network (including requests and responses)

- The local network administrator can
  - Monitor all traffic
  - Provide bad DNS information to reroute requests to servers
Attacker Listens to Conversation

Client

\[\text{Attacker}\]
\((\text{Listening to conversation})\)

Server
Attacker Listens to Conversation

• An attacker listening in on an HTTP exchange can
  – Record passwords for later use
  – Record cookies (almost as good as passwords) for later use
  – Record any personal data sent to or retrieved from a server

• myspace.com has a real problem with this right now (Mar. 20, 2008)
  – Student in residence logs in to myspace.com
  – Attacker (another student living in residence) collects student's myspace password and login (usually a gmail address)
  – Attacker logs in to student's gmail account (which uses the same password) and searches for 'password' and 'login'
  – Attacker now has many of student's passwords and logins for online services (banking, shopping, credit cards, Paypal, ...)

Encryption

• An encryption function takes a key $k$ and a message $M$ and computes an encrypted text $k(M)$
  – Knowing only $k(M)$ it is difficult to compute $M$
• $M$ is called the plaintext
• $k(M)$ is called the encrypted text or cyphertext
• A decryption function takes a key $k$ and an encrypted message $k(M)$ and computes $k^{-1}(k(M)) = M$
An Encrypted HTTP Exchange

- To avoid an attacker listening in, the client and server can use encryption

- The client and server should encrypt their traffic using a secret key that they both know
  - Client and server share a secret key \( \text{sec} \)
  - Client has a request \( R \) but sends \( \text{sec}(R) \) to the server
  - Server receives \( \text{sec}(R) \) and computes \( \text{sec}^{-1}(\text{sec}(R))=R \)
  - Server computes response \( M \) and send \( \text{sec}(M) \) to the client
  - Client receives \( \text{sec}(M) \) and recovers \( \text{sec}^{-1}(\text{sec}(M))=M \)

- The attacker sees only \( \text{sec}(R) \) and \( \text{sec}(M) \), which is not enough to know \( R \) or \( M \)
Encrypted Conversation

Client

483sks8asis0

Server

? Attacker ?
(Listening, sees gibberish)
Encryption

- Encryption requires that the client and server share a piece of information \(sec\) (the key) that only they know
- One of them (the client) needs to select \(sec\) and send it to the other one (the server)
- Oops: The attacker might be listening in and receive the key
An Attacker with the Secret Key

Client

Attacker
(Listening and decrypting)

Server

483sks8asis0
Public-Key Cryptography

• Public-key cryptography offers a solution
  – The server has a key (priv, pub) with two parts
  – The private part (priv) is a secret that only the server knows
  – The public part (pub) is not secret at all
    • The server announces pub to anyone who will listen
  – For a message $M$
    • $pub(M)$ is an encrypted version of $M$
    • it's difficult to recover $M$ from $pub(M)$ without knowing $priv$
    • $priv(pub(M)) = M$

• Just like encryption, but encryption and decryption are done with different keys
Request-Response with PKE

- When a client wants to send a request $R$ to a server with key pair $(priv, pub)$
  - Client chooses a random secret key $sec$
  - Client sends $(pub(sec), sec(R))$
  - Server decrypts $sec$ (because $priv(pub(sec)) = sec$
  - Server computes $sec^{-1}(sec(R)) = R$
  - Server processes $R$ and decides on a response $M$
  - Server sends $sec(M)$ to the client
  - Client decrypts $M$ using $sec^{-1}(sec(M)) = M$

- We use the secret key to encrypt $R$ and $M$ because it's faster than using PKE
  - symmetric encryption (with one key) is faster than public-key encryption (with two keys)
Attacker's View of PKE

• Attacker sees
  – $pub(R)$ and $sec(R)$
  – $sec(M)$

• Attacker knows $pub$

• None of this is enough to recover $R$ or $M$
  – The client and server have effectively hidden the request and the response from an attacker who is listening in

• Problem solved, but...
  – How does the client learn the server's public key, $pub$?
  – The server will send it to anyone, so the client can just ask for it in an unencrypted request
**Impersonation**

- An attacker can pretend to be the server if they can cut off client's access to the server and redirect it to their own server
  - Can be done with a physical disconnection
  - The attacker might be the client's network admin
- Consider wireless internet access
  - The attacker might be the DNS admin
DNS Attack

• Using DNS tricks, an attacker can intercept requests to a server

• When sending a request to the aaa.bbb.com
  – Client send a request to name server to find IP address of aaa.bbb.com
  – Nameserver responds with an IP address (10.233.23.8)
  – Client then starts communicating with 10.233.23.8

• The communication between the client and the DNS server is not authenticated
  – Anyone between the client and the DNS server can intercept and change the response
  – The DNS server may not be trusted (consider wireless browsing)
Active Attacks

• If the attacker can get between the client and server they can do much more than listen in
  – The attacker can impersonate the server
  – The attacker can form a bridge between the server and the client
Attacker Pretends to be Server

Client

Attacker
(Masquerading as Server)

Server
**Man-in-the-Middle Attack**

- **Client**
- **Server**
- **Attacker**
  - (Masquerading as Server)
  - (Masquerading as Client)
**Man-In-the-Middle**

- **Client (to server)**
  - "What is your public key?"

- **Attacker (pretending to be server)**
  - "my public key is 'pubfake"

- **Client (to server):**
  - "here's my request: '(pubfake(sec), sec(R)"

- **Attacker**
  - computes privfake(pubfake(sec)) = sec and decodes R
  - Send (pub(sec'), sec'(R)) to real server and receives response sec'(M) and decodes M

- **Attacker (to client, pretending to be server)**
  - "here's your response: 'sec(M)"
The Public Key Infrastructure

• Man-in-the-middle attacks are possible because the client doesn't know the server's public key in advance
  – client has to ask the server, but has no way of knowing that he's really talking to the server
• The PKI infrastructure attempts to address this
  – when the server responds with their public key the present a certificate for it
  – the certificate is signed by someone that everyone trusts
• Requires a digital signature algorithm
Digital Signatures

- Some public key encryption algorithms have the property of being commutative
  - $priv(pub(M)) = M$
  - $pub(priv(M)) = M$

- This allows the algorithm to be used for digital signing
  - A server takes a document $M$, computes $priv(M)$, and published $(M, priv(M))$
  - Anyone who knows $pub$ can check that $pub(priv(M)) = M$
    - They are convinced that the server examined M and signed it because only the server could compute $priv(M)$

- Actually, digital signatures are slightly more complicated
Public Key Infrastructure

• The PKI uses signed certificates that are signed by certification authorities

• Certificate includes:
  – The name of the server
  – The server's public key and encryption algorithm
  – A serial number
  – A period of validity
  – A digital signature of the above information using the certification authorities private key

• There are very few trusted CAs and their private keys are closely guarded
  – The list of public keys for all trusted certification authorities is broadly distributed
HTTPS

- HTTPS (HTTP Secure) is a protocol for securely accessing a web server
- How it works:
  - Client opens connection to server and requests a certificate
  - Server sends back certificate containing server's public key
  - The client checks the validity of the certificate
  - The client creates a session key, encrypts it with the server's public key and sends it to the server
  - The server decrypts the session key using its private key
  - All further communication between the client and server is encrypted using the session key (HTTP is now used over this secure channel)
**Certificates**

- Ideally, the certification authority is a third party, trusted by all.
- In practice, many certificates are signed by the same authority controlling the server.
  - In such cases, the web browser usually informs the user.
- In practice, many certificates are completely invalid, or out of date.
  - In such cases, the web browser usually informs the user.
- This is one area where IE is better than Firefox.
What HTTPS Provides

• If the certificate is valid then
  – The client can be sure it is communicating with the server it is supposed to be
  – The client and server can be sure that all HTTP traffic between them is not visible to any third party

• Third parties can still
  – Determine that the client is communicating with the server
  – Measure the amount of information exchanged between the client and server
  – Measure the duration and frequency of exchanges

• This security relies on the assumption that the client will not accept an invalid certificate