15. SSL/TLS and the Public Key Infrastructure

ENEE 757 | CMSC 818V

Prof. Tudor Dumitraș
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/757
https://www.facebook.com/SDSAtUMD

Today’s Lecture

• Where we’ve been
  – Authentication and access control
  – Security of Internet protocols – TCP/IP, BGP, DNS
  – Web security
  – Web authentication

• Where we’re going today
  – SSL/TLS
  – The public key infrastructure

• Where we’re going next
  – Patch deployment and certificate revocation
What Is SSL / TLS?

- Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols
  - Same protocol design, different crypto algorithms

- De facto standard for Internet security
  - “The primary goal of the TLS protocol is to provide privacy and data integrity between two communicating applications”

- Deployed in every Web browser; also VoIP, payment systems, distributed systems, etc.

SSL / TLS Guarantees

- End-to-end secure communications in the presence of a network attacker
  - Controls Wi-Fi, DNS, routers, his own websites
  - Can listen to any packet, modify packets in transit, inject his own packets into the network

- Scenario: you are reading your email from an Internet café connected via a r00ted Wi-Fi access point to a dodgy ISP in a hostile authoritarian country
History of the Protocol

- **SSL 1.0** – internal Netscape design (early 1994?)
  - Initial designer: Kipp Hickman
- **SSL 2.0** – Netscape, Nov 1994
  - Several weaknesses
- **SSL 3.0** – Netscape and Paul Kocher, Nov 1996
- **TLS 1.0** – Internet standard, Jan 1999
  - Based on SSL 3.0, but not interoperable (uses different cryptographic algorithms)
- **TLS 1.1** – Apr 2006
- **TLS 1.2** – Aug 2008

SSL Basics

- SSL consists of two protocols

  - **Handshake protocol**
    - Uses public-key cryptography to establish several shared secret keys between the client and the server

  - **Record protocol**
    - Uses the secret keys established in the handshake protocol to protect confidentiality, integrity, and authenticity of data exchange between the client and the server
SSL Handshake Protocol

• Runs between a client and a server
  – For example, client = Web browser, server = website

• Negotiate version of the protocol and the set of cryptographic algorithms to be used
  – Interoperability between different implementations

• Authenticate server and client (optional)
  – Use digital certificates to learn each other’s public keys and verify each other’s identity
  – Often only the server is authenticated

• Use public keys to establish a shared secret

Recall: Diffie-Hellman Key Exchange

• Alice and Bob never met and share no secrets
• Public info: p and g
  – p is a large prime number, g is a generator of $\mathbb{Z}_p^*$
    • $\mathbb{Z}_p^* =\{1, 2 \ldots p-1\}; \forall a \in \mathbb{Z}_p^* \exists i$ such that $a = g^i \mod p$

  \[ g^x \mod p \]
  \[ g^y \mod p \]

Pick secret, random X
Alice

Bob
Pick secret, random Y

Compute $k = (g^y)^x = g^{xy} \mod p$
Compute $k = (g^x)^y = g^{xy} \mod p$

Security based on hardness of discrete log: hard to extract $x$ from $g^x \mod p$
TLS Handshake Protocol Structure

ClientHello

ServerHello, [Certificate], [ServerKeyExchange], [CertificateRequest], ServerHelloDone

Certificate

ClientKeyExchange, [CertificateVerify]

Finished

switch to negotiated cipher

Finished

switch to negotiated cipher

Record of all sent and received handshake messages

Debian Linux (2006-08)

- A line of code commented out from md_rand
  - MD_Update(&m,buf,j); /* purify complains */

- Without this line, the seed for the pseudo-random generator is derived only from process ID
  - Default maximum on Linux = 32768

- Result: keys generated using Debian-based OpenSSL package in 2006-08 are predictable
  - “Affected keys include SSH keys, OpenVPN keys, DNSSEC keys, and key material for use in X.509 certificates and session keys used in SSL/TLS connections”
  - More on this in next lecture
Exploiting SSL for Denial of Service

https://www.thc.org/thc-ssl-dos/

2 simple commands in bash:

```bash
-----BASH SCRIPT BEGIN------
thc-ssl-dosit() { while ;; do (while ;; do echo R; done) | openssl s_client -connect 127.0.0.1:443 2>/dev/null; done }
for x in `seq 1 100`; do thc-ssl-dosit & done
-----BASH SCRIPT END-------
```

THC-SSL-DOS is a tool to verify the performance of SSL

Establishing a secure SSL connection requires 15x more processing power on the server than on the client

“THC-SSL-DOS exploits this asymmetric property by overloading the server and knocking it off the Internet”

SSL/TLS Record Protection

Use symmetric keys established in the handshake protocol
HTTPS and Its Adversary Model

- HTTPS: **end-to-end** secure protocol for Web
- Designed to be secure against network attackers, including man-in-the-middle (MITM) attacks

![Diagram of HTTPS](image)

- HTTPS provides encryption, authentication (usually for server only), and integrity checking

HTTPS Security Guarantees

- The origin of the page is what it says in the address bar
  - Example: amazonaccounts.com vs. amazon.com

- Contents of the page have not been viewed or modified by a network attacker
Combining HTTPS and HTTP

- Page served over HTTPS but contains HTTP
  - IE 7: no lock, “mixed content” warning
  - Firefox: “!” over lock, no warning by default
  - Safari: does not detect mixed content
- Flash does not trigger warning in IE7 and FF
- Network attacker can now inject scripts, hijack session

Mixed Content and Network Attacks

- Banks: after login, all content served over HTTPS
- Developer error: somewhere on bank site write
  `<script src=http://www.site.com/script.js> </script>`
  - Allows active network attacker to hijack session (how?)
- Better way to include content:
  `<script src=/www.site.com/script.js> </script>`
  - Served over the same protocol as embedding page
HTTP → HTTPS and Back

• Typical pattern: HTTPS upgrade
  – Come to site over HTTP, redirect to HTTPS for login
  – Browse site over HTTP, redirect to HTTPS for checkout
• sslstrip: network attacker downgrades connection

HTTP → sslstrip → HTTPS

– Rewrite `<a href=https://...>` to `<a href=http://...>`
– Redirect `Location: https://...` to `Location: http://...`
– Rewrite `<form action=https://...>` to `<form action=http://...>`

Can the server detect this attack?

Will You Notice?
[Moxie Marlinspike]

Clever favicon inserted by network attacker
Motivation

Whose public key is used to establish the secure session?

Authenticity of Public Keys

Problem: How does Alice know that the public key she received is really Bob’s public key?
Recall: Authenticating a Channel

- Reference monitor receives a request $C$ says action($s$)
- An access-control list usually specifies named principals
- Thus, reference monitor must collect credentials to prove that $A$ says action($s$) for some $A$ on the access control list
- Credentials typically correspond to statements of the form
  $$K_{CA} \text{ signed } (\text{key}(K_A) \text{ speaksfor } \text{key}(K_{CA}).A)$$
- If $K_{CA}$ is a public key, this statement is called a certificate

Using Certificates to Authenticate Web Servers

- As part of SSL/TLS, web server sends a certificate
  $$K_{CA} \text{ signed } (\text{key}(K_{www.foo.com}) \text{ speaksfor } \text{key}(K_{CA}).'www.foo.com')$$
  to browser
- Browser is shipped with public keys for numerous CAs:
  $$K_{CA1}, K_{CA2}, K_{CA3}, ...$$
  - Mozilla Firefox 23.0.1 ships with ~200 CA keys loaded
  - Reportedly these represent organizations from over 30 countries: AT, BE, BM, CH, CN, CO, DE, DK, EE, ES, EU, FI, FR, GB, GR, HK, HU, IE, IL, IT, JP, NL, NO, PL, RO, SE, SK, TR, TW, US, VE, ZA
- Should we really trust that key($K_{CA}$).'www.foo.com' is the “right” www.foo.com for all of these CAs?
Trusted Certificate Authorities

![Certificate Manager Image]

CA Hierarchy

- Browsers, operating systems, etc. have trusted root certificate authorities
  - Firefox 3 includes certificates of 135 trusted root CAs

- A Root CA signs certificates for intermediate CAs, they sign certificates for lower-level CAs, etc.
  - Certificate “chain of trust”

- CA is responsible for verifying the identities of certificate requestors, domain ownership
Certificate Hierarchy

What power do they have?

Who trusts their certificates?

Example of a Certificate

Important fields
Sources

• Various slides from Vitaly Shmatikov

Review of Lecture

• What did we learn?
  – SSL/TLS
  – The public key infrastructure

• Paper discussion: “Mining Your Ps and Qs”
  – Discussion lead: Ahmed
  – Scribe: Yupeng

• What’s next?
  – Patch deployment and certificate revocation