SSL/TLS

*Slides borrowed from Vitaly Shmatikov
Optional Reading

◆ Kaufman. Chapters 15.1-7 and 19.
What Is SSL/TLS?

◆ Secure Sockets Layer and Transport Layer Security 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
  - Attacker completely owns the network: 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 rooted 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?
  - Lost in the mists of time
- 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
Handshake Protocol Structure

ClientHello

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

[Certificate], ClientKeyExchange, [CertificateVerify]

switch to negotiated cipher

Finished

Record of all sent and received handshake messages

switch to negotiated cipher

Finished
ClientHello

Client announces (in plaintext):
- Protocol version he is running
- Cryptographic algorithms he supports
- Fresh, random number
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello

- Highest version of the protocol supported by the client
- Session id (if the client wants to resume an old session)
- Set of cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)
ServerHello

ServerHello

C, version$_C$, suites$_C$, N$_C$

Server responds (in plaintext) with:
- Highest protocol version supported by both the client and the server
- Strongest cryptographic suite selected from those offered by the client
- Fresh, random number
ServerKeyExchange

Server sends his public-key certificate containing either his RSA, or his Diffie-Hellman public key (depending on chosen crypto suite)
ClientKeyExchange

The client generates secret key material and sends it to the server encrypted with the server’s public key (if using RSA)
ClientKeyExchange (RFC)

struct {
    select (KeyExchangeAlgorithm) {
        case rsa: EncryptedPreMasterSecret;
        case diffie_hellman: ClientDiffieHellmanPublic;
    } exchange_keys
} ClientKeyExchange

struct {
    ProtocolVersion client_version;
    opaque random[46];
} PreMasterSecret

Where do random bits come from?

Random bits from which symmetric keys will be derived (by hashing them with nonces)
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: **all** 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”
“Core” SSL 3.0 Handshake

C, version$_c$=3.0, suites$_c$, N$_c$

version$_s$=3.0, suite$_s$, N$_s$,
certificate for PK$_s$,
“ServerHelloDone”

{Secret$_c$}$_{PKs}$ if using RSA

C and S share
secret key material (secret$_c$) at this point

switch to keys derived
from secret$_c$, N$_c$, N$_s$

Finished

Finished
Version Rollback Attack

C, version = 2.0, suites\(c\), \(N_c\)

Server is fooled into thinking he is communicating with a client who supports only SSL 2.0

version\(s\) = 2.0, suite\(s\)\(r\), \(N_{sr}\), certificate for PK\(s\), “ServerHelloDone”

\(\{\text{Secret}_c\}\)\(PK_s\)

C and S end up communicating using SSL 2.0 (weaker earlier version of the protocol that does not include “Finished” messages)
SSL 2.0 Weaknesses (Fixed in 3.0)

- Cipher suite preferences are not authenticated
  - “Cipher suite rollback” attack is possible
- Weak MAC construction, MAC hash uses only 40 bits in export mode
- SSL 2.0 uses padding when computing MAC in block cipher modes, but padding length field is not authenticated
  - Attacker can delete bytes from the end of messages
- No support for certificate chains or non-RSA algorithms
“Chosen-Protocol” Attacks

◆ Why do people release new versions of security protocols? Because the old version got broken!
◆ New version must be backward-compatible
  • Not everybody upgrades right away
◆ Attacker can fool someone into using the old, broken version and exploit known vulnerabilities
  • Similar: fool victim into using weak crypto algorithms
◆ Defense is hard: must authenticate version early
◆ Many protocols had “version rollback” attacks
  • SSL, SSH, GSM (cell phones)
Version Check in SSL 3.0

C, version\textsubscript{C}=3.0, suites\textsubscript{C}, N\textsubscript{C}

version\textsubscript{S}=3.0, suite\textsubscript{S}, N\textsubscript{S}, certificate for PK\textsubscript{S}, “ServerHelloDone”

“Embed” version number into secret

\{version\textsubscript{C}, secret\textsubscript{C}\}\textsubscript{PKS}

C and S share secret key material secret\textsubscript{C} at this point

switch to key derived from secret\textsubscript{C}, N\textsubscript{C}, N\textsubscript{S}

Check that received version is equal to the version in ClientHello

switch to key derived from secret\textsubscript{C}, N\textsubscript{C}, N\textsubscript{S}
Exploiting SSL for Denial of Service

2 simple commands in 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

- Application Data
- Fragment
- Compress
- Add MAC
- Encrypt
- Append SSL Record Header

Use symmetric keys established in the handshake protocol
Most Common Use of SSL/TLS
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

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

◆ Goal: identify secure connection
  • SSL/TLS is being used between client and server to protect against active network attacker

◆ Lock icon should only be shown when the page is secure against network attacker
  • Semantics subtle and not widely understood by users
  • Problem in user interface design
HTTPS Security Guarantees

- The origin of the page is what it says in the address bar
  - User must interpret what he sees - remember amazonaccounts.com?

- Contents of the page have not been viewed or modified by a network attacker
Evolution of the Lock in Firefox

How about Firefox 4?
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: UI Challenges
Mixed Content and Network Attacks

- Banks: after login, all content served over HTTPS
- Developer error: somewhere on bank site write
  
  ```html
  <script src=http://www.site.com/script.js>  </script>
  ```
  
  - Active network attacker can now hijack any session (how?)
- Better way to include content:
  
  ```html
  <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

- 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?

⇒

Clever favicon inserted by network attacker
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?
Distribution of Public Keys

◆ Public announcement or public directory
  • Risks: forgery and tampering

◆ Public-key certificate
  • Signed statement specifying the key and identity
    \[ \text{sig}_{\text{Alice}}(\text{``Bob'', PK}_B) \]

◆ Common approach: certificate authority (CA)
  • An agency responsible for certifying public keys
  • Browsers are pre-configured with 100+ of trusted CAs
  • A public key for any website in the world will be accepted by the browser if certified by one of these CAs
Trusted Certificate Authorities

You have certificates on file that identify these certificate authorities:

- TDC
- TDC Internet
- Thawte
- Thawte Consulting
- Thawte Consulting cc
- thawte, Inc.
- The Go Daddy Group, Inc.
- The USERTRUST Network
- TÜRKTRUST Bilgi İletişim ve Bilişim Güvenliği Hizmetleri A.Ş. ...
- Unizeto Sp. z o.o.
- ValiCert, Inc.
- VeriSign, Inc.
- VISA
- Wells Fargo
- Wells Fargo WellsSecure
- XRamp Security Services Inc

[Image of Certificate Manager window showing trusted certificate authorities]
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”
    - $\text{sig}_{\text{Verisign}}(\text{“UT Austin”}, \text{PK}_{\text{UT}})$, $\text{sig}_{\text{UT}}(\text{“Vitaly S.”}, \text{PK}_{\text{Vitaly}})$
- 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

- **Certificate Signature Algorithm**
- **Issuer**
- **Validity**
  - Not Before
  - Not After
- **Subject**
- **Subject Public Key Info**
  - Subject Public Key Algorithm
- **Subject’s Public Key**
- **Extensions**

**Field Value**

- Modulus (1024 bits):
  - ac 73 14 97 b4 10 a3 aa f4 c1 15 ed cf 92 f3 9a
  - 97 26 9a cf 1b e4 1b dc d2 c9 37 2f d2 e6 07 1d
  - ad b2 3e f7 8c 2f fa a1 b7 9e e3 54 40 34 3f gb
  - e2 1c 12 8a 30 6b 0c fa 30 6a 01 61 e9 7c b1 98
  - 2d 0d c6 38 03 b4 55 33 7f 10 40 45 c5 c3 e4 d6
  - 6b 9c 0d d0 8e 4f 39 0d 2b d2 e9 88 cb 2d 21 a3
  - f1 84 61 3c 3a aa 80 18 27 e6 7e f7 b8 6a 0a 75
  - e1 bb 14 72 95 cb 64 78 06 84 81 eb 7b 07 8d 49

**Certificate Viewer:**

- **This certificate has been verified for the following uses:**
  - SSL Server Certificate

**Issued To**
- Common Name (CN): *.gmail.com
- Organization (O): Google Inc
- Organizational Unit (OU): <Not Part Of Certificate>

**Serial Number:**

**Issued By**
- Common Name (CN): Thawte Premium Server CA
- Organization (O): Thawte Consulting cc
- Organizational Unit (OU): Certification Services Division

**Validity**
- Issued On: 9/25/2008
- Expires On: 9/25/2010

**Fingerprints**
Common Name

- **Explicit name**: www.foo.com
- **Wildcard**: *.foo.com or www*.foo.com
- **Matching rules**
  - Firefox 3: * matches anything
  - Internet Explorer 7: * must occur in the leftmost component, does not match ‘.’
    - *.foo.com matches a.foo.com, but not a.b.foo.com
International Domain Names

- Rendered using international character set
- Chinese character set contains characters that look like / ? = .
  - What could go wrong?

- Can buy a certificate for *.foo.cn, create any number of domain names that look like
  www.bank.com/accounts/login.php?q=me.foo.cn
  - What does the user see?
  - *.foo.cn certificate works for all of them!
Example

[Moxie Marlinspike]
Meaning of Color

What is the difference?

Domain Validation (DV) certificate vs. Extended Validation (EV) certificate

Means what?
Mobile Browsing

Windows Phone 7: same behavior
... but only when URL bar present
... landscape mode: no URL bar

Same lock for DV and EV

Extended Validation (EV) Certificates

Certificate request must be approved by a human lawyer at the certificate authority.
Questions about EV Certificates

◆ What does EV certificate mean?
◆ What is the difference between an HTTPS connection that uses a regular certificate and an HTTPS connection that uses an EV certificate?
◆ If an attacker has somehow obtained a non-EV certificate for bank.com, can he inject a script into https://bank.com content?
  • What is the origin of the script? Can it access or modify content that arrived from actual bank.com via HTTPS?
◆ What would the browser show – blue or green?
When Should The Lock Be Shown?

- All elements on the page fetched using HTTPS
- For all elements:
  - HTTPS certificate is issued by a certificate authority (CA) trusted by the browser
  - HTTPS certificate is valid — means what?
  - Common Name in the certificate matches domain name in the URL
X.509 Authentication Service

- Internet standard (1988-2000)
- Specifies certificate format
  - X.509 certificates are used in IPsec and SSL/TLS
- Specifies certificate directory service
  - For retrieving other users’ CA-certified public keys
- Specifies a set of authentication protocols
  - For proving identity using public-key signatures
- Can use with any digital signature scheme and hash function, but must hash before signing

Remember MD5?
X.509 Certificate

Added in X.509 versions 2 and 3 to address usability and security problems.
Back in 2008

Many CAs still used MD5

- RapidSSL, FreeSSL, TrustCenter, RSA Data Security, Thawte, verisign.co.jp

Sotirov et al. collected 30,000 website certificates

- 9,000 of them were signed using MD5 hash
- 97% of those were issued by RapidSSL

[Sotirov et al. “Rogue Certificates”]
Colliding Certificates

[Slide 51]

<table>
<thead>
<tr>
<th>Serial number</th>
<th>validation period</th>
<th>real cert</th>
<th>domain name</th>
<th>chosen prefix (difference)</th>
<th>Hash to the same MD5 value!</th>
</tr>
</thead>
<tbody>
<tr>
<td>set by the CA</td>
<td></td>
<td>real cert</td>
<td>RSA key</td>
<td>identical bytes (computed)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X.509 extensions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>signature</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>serial number</td>
<td>validation period</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rogue cert</td>
<td>domain name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>X.509 extensions</td>
<td></td>
<td>collided from real cert</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>signature</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Valid for both certificates!

[Sotirov et al. “Rogue Certificates”]

Hash to the same MD5 value!

Valid for both certificates!
Generating Collisions

1-2 days on a cluster of 200 PlayStation 3’s

Equivalent to 8000 desktop CPU cores or $20,000 on Amazon EC2

[Sotirov et al. “Rogue Certificates”]
Generating Colliding Certificates

[Sotirov et al. “Rogue Certificates”]

◆ RapidSSL uses a fully automated system
  • $69 for a certificate, issued in 6 seconds
  • Sequential serial numbers

◆ Technique for generating colliding certificates
  • Get a certificate with serial number S
  • Predict time T when RapidSSL’s counter goes to S+1000
  • Generate the collision part of the certificate
  • Shortly before time T buy enough (non-colliding) certificates to increment the counter to S+999
  • Send colliding request at time T and get serial number S+1000
Creating a Fake Intermediate CA

[Sotirov et al. “Rogue Certificates”]

<table>
<thead>
<tr>
<th>Serial number</th>
<th>Real cert domain name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validity period</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Choosing prefix (difference)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision bits (computed)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Identical bytes (copied from real cert)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Rogue CA cert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rogue CA RSA key</td>
</tr>
<tr>
<td>Rogue CA X.509 extensions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Netscape Comment Extension (contents ignored by browsers)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Signature</th>
</tr>
</thead>
</table>

We are now an intermediate CA. W00T!

CA bit!
Result: Perfect Man-in-the-Middle

This is a “skeleton key” certificate: it can issue fully trusted certificates for any site (why?)

To take advantage, need a network attack
  • Insecure wireless, DNS poisoning, proxy auto-discovery, hacked routers, etc.
A Rogue Certificate
Remember Flame?

- Cyber-espionage virus (2010-2012)
- Signed with a fake intermediate CA certificate that appears to be issued by Microsoft and thus accepted by any Windows Update service
  - Fake intermediate CA certificate was created using an MD5 chosen-prefix collision against an obscure Microsoft Terminal Server Licensing Service certificate that was enabled for code signing and still used MD5
- MD5 collision technique possibly pre-dates Sotirov et al.’s work
  - Evidence of state-level cryptanalysis?
SSL/TLS Handshake

Hello

Here is my certificate

C

Validate the certificate

S
SSL/TLS Handshake

Hello

I am Chase.com
Here is my certificate

Issued by GoDaddy to
AllYourSSLAreBelongTo.us

Ok!
Failing to Check Hostname

“Researchers at the University of Texas at Austin and Stanford University have discovered that poorly designed APIs used in SSL implementations are to blame for vulnerabilities in many critical non-browser software packages. Serious security vulnerabilities were found in programs such as Amazon’s EC2 Java library, Amazon’s and PayPal’s merchant SDKs, Trillian and AIM instant messaging software, popular integrated shopping cart software packages, Chase mobile banking software, and several Android applications and libraries. SSL connections from these programs and many others are vulnerable to a man in the middle attack…”

- Threatpost (Oct 2012)
What Happens After Validation?

Hello

I am PayPal.com
(or whoever you want me to be)
Here is PayPal’s certificate for its RSA signing key
And here is my signed Diffie-Hellman value

Validate the certificate
... then verify the signature on the DH value using the public key from the certificate
Here is PayPal’s certificate
And here is my signed Diffie-Hellman value

... verify the signature on the DH value using the public key from the certificate

```c
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
goto fail;
goto fail;  // ??
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
goto fail;  ...
err = sslRawVerify(...);  // Signature is verified here
...  ...
fail: ... return err ...
```
Complete Fail Against MITM

- Discovered in February 2014
- All OS X and iOS software vulnerable to man-in-the-middle attacks
  - Broken TLS implementation provides no protection against the very attack it was supposed to prevent
- What does this tell you about quality control for security-critical software?
Certificate Revocation

◆ Revocation is very important
◆ Many valid reasons to revoke a certificate
  • Private key corresponding to the certified public key has been compromised
  • User stopped paying his certification fee to the CA and the CA no longer wishes to certify him
  • CA’s certificate has been compromised!
◆ Expiration is a form of revocation, too
  • Many deployed systems don’t bother with revocation
  • Re-issuance of certificates is a big revenue source for certificate authorities
Certificate Revocation Mechanisms

◆ Online revocation service
  • When a certificate is presented, recipient goes to a special online service to verify whether it is still valid

◆ Certificate revocation list (CRL)
  • CA periodically issues a signed list of revoked certificates
  • Can issue a “delta CRL” containing only updates

Q: Does revocation protect against forged certificates?