SSL/TLS

Vitaly Shmatikov
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 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?
  • 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

C

ClientHello

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

S

[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

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
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)

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

```c
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 Handshake

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

\text{version}\_s = 3.0, suite\_s, N\_s, certificate for PK\_s, “ServerHelloDone”

\{Secret\_c\}_{PKs} \text{ if using RSA}

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

\text{switch to keys derived from secret\_c, N\_c, N\_s}

\text{Finished}

\text{Finished}
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
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, N_s,
certificate for PK_s,
“ServerHelloDone”

{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)
C, version_c=3.0, suites_c, N_c

version_s=3.0, suite_s, N_s, certificate for PK_s, “ServerHelloDone”

“Embed” version number into secret

{version_c, secret_c}_{PK_s}

C and S share secret key material secret_c at this point

switch to key derived from secret_c, N_c, N_s

switch to key derived from secret_c, N_c, N_s

Check that received version is equal to the version in ClientHello
TLS Version Rollback

C, version = 3.0, suites_c, N_c

Server is fooled into thinking he is communicating with a client who supports only SSL 3.0

Version_s = 3.0, suite_s, N_s, certificate for PK_s, "ServerHelloDone"

C and S end up communicating using SSL 3.0 (deprecated but supported by everyone for backward compatibility)

Attack exploits "padding oracle" in CBC encryption mode as used by SSL 3.0 to infer the value of encrypted cookies

Many "padding oracle" attacks over the years: BEAST, CRIME, ...

POODLE attack (October 2014)
“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)
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

Use symmetric keys established in the handshake protocol
TLS Heartbeat

A way to keep TLS connection alive without constantly transferring data

If you are alive, send me this 5-letter word: “xyzzy”

Per RFC 6520:

```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

OpenSSL omitted to check that this value matches the actual length of the heartbeat message.
Heartbleed Consequences

- Attacker can obtain chunks of server memory
  - Passwords, contents of other users’ communications, even the server’s private RSA key
- Assisted by a custom allocator that does not zero out malloc’d memory (for “performance,” natch!)
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
- 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

- Security Information window: This page contains both secure and nonsecure items. Do you want to display the nonsecure items?
- Internet Explorer window: This page has an unspecified potential security flaw. Would you like to continue?
- iGoogle - Mozilla Firefox window: https://www.google.com/ig
### 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?
Distribution of Public Keys

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

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

◆ 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

![Certificate Manager window with Trusted Certificate Authorities listed](image)

- 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
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”
    – $\text{sig}_{\text{Verisign}}(\text{“UT Austin”, PK}_\text{UT})$, $\text{sig}_{\text{UT}}(\text{“Vitaly S.”, PK}_{\text{Vitaly}})$
◆ 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
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

Internet Explorer 9

[Embedded link]

Firefox 4

[Embedded link]

Chrome 8

[Embedded link]

Safari 4

[Embedded link]

What is the difference?

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

Means what?
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?
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
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. “MD5 Considered Harmful Today: Creating a Rogue CA Certificate”]
### Colliding Certificates

[Sotirov et al.]

<table>
<thead>
<tr>
<th>set by the CA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>serial number</td>
<td>serial number</td>
<td>serial number</td>
</tr>
<tr>
<td>validity period</td>
<td>validity period</td>
<td></td>
</tr>
<tr>
<td>real cert</td>
<td>chosen prefix (difference)</td>
<td>rogue cert</td>
</tr>
<tr>
<td>domain name</td>
<td></td>
<td>domain name</td>
</tr>
<tr>
<td>real cert</td>
<td>collision bits (computed)</td>
<td></td>
</tr>
<tr>
<td>RSA key</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X.509 extensions</td>
<td>identical bytes (copied from real cert)</td>
<td>X.509 extensions</td>
</tr>
<tr>
<td>signature</td>
<td>signature</td>
<td></td>
</tr>
</tbody>
</table>

Hash to the same MD5 value!

Valid for both certificates!
Generating Collisions [Sotirov et al.]

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

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

<table>
<thead>
<tr>
<th>Serial number</th>
<th>Validity period</th>
<th>Real cert domain name</th>
<th>Rogue CA cert</th>
<th>CA bit!</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Real cert RSA key</td>
<td>Rogue CA RSA key</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collision bits (computed)</td>
<td>Rogue CA X.509 extensions</td>
<td>NetscapeComment Extension (contents ignored by browsers)</td>
<td>We are now an intermediate CA. W00T!</td>
</tr>
<tr>
<td></td>
<td>identical bytes (copied from real cert)</td>
<td>Signature</td>
<td>Signature</td>
<td></td>
</tr>
</tbody>
</table>
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

Certificate

General | Details | Certification Path

Certification path:
- Equifax Secure Global eBusiness CA-1
- MD5 Collisions Inc. (http://www.phreedom.org/md5)
- i.broke.the.internet.and.all.i.got.was.this.t-shirt.phreedom

Certificate status:
This certificate is OK.
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

Validate the certificate
SSL/TLS Handshake

Hello

I am Chase.com
Here is my certificate

Issued by GoDaddy to
AllYourSSLAreBelongTo.us

Ok!
“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)
Testing Certificate Validation Code

Generating Test Certificates

◆ Requirements
  • Must generate “semantically bad” certificates
  • Should be syntactically correct, otherwise will fail during parsing and won’t exercise most of the certificate validation code
  • Must scale to millions of certificates

◆ Idea
  • X.509 certificates contain structured data, can we exploit this?
X.509 Certificate Structure

- Multilayered structured data
- **Syntactic constraints** for each piece
  - Ex: Version must be an integer
- **Semantic constraints** for individual piece or across multiple pieces
  - Ex: Version must be 0, 1, or 2
  - Ex: if version!=2, extensions must be NULL

Version
Serial Number
Signature
Algorithm
Identifier
Issuer Name
Validity Period
Subject Name
Public Key
Information
Issuer Unique ID
Subject Unique ID
Extensions
X.509 Standards... Ugh!
Idea: Random Re-assembly

Create X.509 certs using randomly picked syntactically valid pieces

Likely to violate some semantic constraints and will thus generate “bad” test certs just as we wanted

Wait, how can we generate a large set of such syntactically valid pieces without reading X.509 specs?
1. Scan the Internet

Collect 243,246 X.509 server certificates
2. Extract Syntactically Valid Pieces

- keyUsage extension from cert3
- keyUsage extension from cert2
- version from cert 1
- ExtendedkeyUsage extension from cert4
3. Frankencerts

Generate 8 million **frankencerts** from random combinations of certificate pieces.
Differential Testing

- Multiple implementations of SSL/TLS should implement the same certificate validation logic

- If a certificate is accepted by some and rejected by others, what does this mean?
No false positives, although some discrepancies might be due to different interpretations of X.509
Results of Differential Testing

- 14 different SSL/TLS implementations
- 208 discrepancies due to 15 root causes
- Multiple bugs
  - Accepting fake and unauthorized intermediate certificate authorities
  - Attacker can impersonate any website!
  - Accepting certificates not authorized for use in SSL or not valid for server authentication
  - Several other issues
## Results Summary

<table>
<thead>
<tr>
<th>Problem</th>
<th>OpenSSL</th>
<th>PolarSSL</th>
<th>Gnu/TLS</th>
<th>CyaSSL</th>
<th>MatrixSSL</th>
<th>NSS</th>
<th>OpenJDK, Bouncy Castle</th>
<th>Browsers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrusted version 1 intermediate CA certificate</td>
<td>No</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Untrusted version 2 intermediate CA certificate</td>
<td>No</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Version 1 certificate with valid basic constraints</td>
<td>No</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>Firefox: reject Opera, Chrome: accept</td>
</tr>
<tr>
<td>Intermediate CA not authorized to issue further intermediate CA certificates, but followed in the chain by an intermediate CA certificate...Followed by a leaf CA certificate</td>
<td>No</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Intermediate CA not authorized to issue certificates for server’s hostname</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate not yet valid</td>
<td>Yes</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate expired in its timezone</td>
<td>Yes</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>CA certificate not authorized for signing other certificates</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Server certificate not authorized for use in SSL/TLS handshake</td>
<td>Yes</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Server certificate not authorized for server authentication</td>
<td>Yes</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate with unknown critical extension</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate with malformed extension value</td>
<td>No</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Certificate with the same issuer and subject and a valid chain of trust</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Issuer name does not match AKI</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
<tr>
<td>Issuer serial number does not match AKI</td>
<td>No</td>
<td>reject</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>accept</td>
<td>reject</td>
<td>reject</td>
</tr>
</tbody>
</table>
Version 1 CA certificates

If an SSL/TLS implementation encounters a version 1 (v1) CA certificate that cannot be validated out of band, it must reject it.

RFC 5280 Section 6.1.4(k)

v1 CA certificates do not support the CA bit: anybody with a valid v1 certificate can pretend to be a CA.
/* Disable V1 CA flag to prevent version 1 certificates in a supplied chain. */
flags &= ~(GNUTLS_VERIFY_ALLOW_X509_V1_CA_CRT);
ret = _gnutls_verify_certificate2 (flags,..))

int _gnutls_verify_certificate2(flags, ..)
{
    if (!(flags & GNUTLS_VERIFY_DISABLE_CA_SIGN) &&
        ((flags & GNUTLS_VERIFY_DO_NOT_ALLOW_X509_V1_CA_CRT)
         || issuer_version != 1))
    {
/*check the CA bit */
    }
}
Exhibit 2: Google Chrome

OK to click through?
Exhibit 2: Google Chrome

The site is on a certificate

You attempt to access a site available to the Internet.
Chrome cannot verify the certificate.
Certificate's common name should be www.google.com.
You should not proceed.

Certificate Viewer: www.google.com

This certificate has been verified for the following usages:

Issued To
Common Name (CN): www.google.com
Organization (O): Google Inc.
Organizational Unit (OU): <Not Part Of Certificate>
Serial Number: 00:BC:BA:57:5A:51:B4:D5:31

Issued By
Common Name (CN): www.foobar.com
Organization (O): Foobar Inc.
Organizational Unit (OU): <Not Part Of Certificate>

Validity Period
Issued On: 2/5/12
Expires On: 2/5/14

Fingerprints
SHA-256 Fingerprint: B9 4B 94 80 9F 99 B3 90 CD DC CD BA FF 4F E4 06 8B 0E AC 26 81 A9 A2 04 15 0C 18 22 71 7E EB AD

untrusted CA
Exhibit 2: Root Cause

- Chrome on Linux uses a modified version of NSS.
- If a certificate is issued by an untrusted CA and is expired, the NSS certificate validation code returns only the “expired” error.
- Firefox uses a glue layer called Personal Security Manager (PSM) over NSS and thus is not affected.
Another Bad Warning

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
Goto Fail

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

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 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?
Comodo

◆ Comodo is one of the trusted root CAs
  • Its certificates for any website in the world are accepted by every browser

◆ Comodo accepts certificate orders submitted through resellers
  • Reseller uses a program to authenticate to Comodo and submit an order with a domain name and public key, Comodo automatically issues a certificate for this site
Comodo Break-In

◆ An Iranian hacker broke into instantSSL.it and globalTrust.it resellers, decompiled their certificate issuance program, learned the credentials of their reseller account and how to use Comodo API
  • username: gtadmin, password: globaltrust

◆ Wrote his own program for submitting orders and obtaining Comodo certificates

◆ On March 15, 2011, got Comodo to issue 9 rogue certificates for popular sites
  • mail.google.com, login.live.com, login.yahoo.com, login.skype.com, addons.mozilla.org, "global trustee"
Consequences

◆ Attacker needs to first divert users to an attacker-controlled site instead of Google, Yahoo, Skype, but then...
  • For example, use DNS to poison the mapping of mail.yahoo.com to an IP address
◆ ... “authenticate” as the real site
◆ ... decrypt all data sent by users
  • Email, phone conversations, Web browsing

Q: Does HTTPS help? How about EV certificates?
Message from the Attacker

I'm single hacker with experience of 1000 hacker, I'm single programmer with experience of 1000 programmer, I'm single planner/project manager with experience of 1000 project managers ...

When USA and Isarel could read my emails in Yahoo, Hotmail, Skype, Gmail, etc. without any simple little problem, when they can spy using Echelon, I can do anything I can. It's a simple rule. You do, I do, that's all. You stop, I stop. It's rule #1 ...

Rule#2: So why all the world got worried, internet shocked and all writers write about it, but nobody writes about Stuxnet anymore?... So nobody should write about SSL certificates.

Rule#3: I won't let anyone inside Iran, harm people of Iran, harm my country's Nuclear Scientists, harm my Leader (which nobody can), harm my President, as I live, you won't be able to do so. as I live, you don't have privacy in internet, you don't have security in digital world, just wait and see...
DigiNotar Break-In

In June 2011, the same “ComodoHacker” broke into a Dutch certificate authority, DigiNotar

- Message found in scripts used to generate fake certificates:
  "THERE IS NO ANY HARDWARE OR SOFTWARE IN THIS WORLD EXISTS WHICH COULD STOP MY HEAVY ATTACKS MY BRAIN OR MY SKILLS OR MY WILL OR MY EXPERTISE"

Security of DigiNotar servers

- All core certificate servers in a single Windows domain, controlled by a single admin password (Pr0d@dm1n)
- Software on public-facing servers out of date, unpatched
- Tools used in the attack would have been easily detected by an antivirus... if it had been present
Consequences of DigiNotar Hack

- Break-in not detected for a month
- Rogue certificates issued for *.google.com, Skype, Facebook, www.cia.gov, and 527 other domains
- 99% of revocation lookups for these certificates originated from Iran
  - Evidence that rogue certificates were being used, most likely by Iranian government or Iranian ISPs to intercept encrypted communications
    - Textbook man-in-the-middle attack
  - 300,000 users were served rogue certificates
Another Message from the Attacker

http://pastebin.com/u/ComodoHacker

Most sophisticated hack of all time ... I’m really sharp, powerful, dangerous and smart!

My country should have control over Google, Skype, Yahoo, etc. [...] I’m breaking all encryption algorithms and giving power to my country to control all of them.

You only heards Comodo (successfully issued 9 certs for me -thanks by the way-), DigiNotar (successfully generated 500+ code signing and SSL certs for me -thanks again-), StartCOM (got connection to HSM, was generating for twitter, google, etc. CEO was lucky enough, but I have ALL emails, database backups, customer data which I'll publish all via cryptome in near future), GlobalSign (I have access to their entire server, got DB backups, their linux / tar gzipped and downloaded, I even have private key of their OWN globalsign.com domain, hahahaa).... BUT YOU HAVE TO HEAR SO MUCH MORE! SO MUCH MORE! At least 3 more, AT LEAST!
In Feb 2012, admitted issuing an intermediate CA certificate to a corporate customer

- Purpose: “re-sign” certificates for “data loss prevention”
- Translation: forge certificates of third-party sites in order to spy on employees’ encrypted communications with the outside world

Customer can now forge certificates for any site in world... and they will be accepted by any browser!

- What if a “re-signed” certificate leaks out?

Do other CAs do this?
TurkTrust

- In Jan 2013, a rogue *.google.com certificate was issued by an intermediate CA that gained its authority from the Turkish root CA TurkTrust
  - TurkTrust accidentally issued intermediate CA certs to customers who requested regular certificates
  - Ankara transit authority used its certificate to issue a fake *.google.com certificate in order to filter SSL traffic from its network
- This rogue *.google.com certificate was trusted by every browser in the world