Network Security 4

Cryptocurrency somehow combines everything we love about religious fanatics with everything we love about Ponzi schemes.

7:08 AM - 23 Oct 2017

68 Retweets 128 Likes

matt blaze @mattblaze

Tweet your reply

matt blaze @mattblaze · 18m

Replying to @mattblaze

There's a lot to dislike about the world we're in, but at least Ayn Rand didn't have bitcoin to write about.
“Best Effort” is Lame! What to do?

- It’s the job of our Transport (layer 4) protocols to build data delivery services that our apps need out of IP’s modest layer-3 service
- #1 workhorse: **TCP** (Transmission Control Protocol)
- Service provided by TCP:
  - **Connection oriented** (explicit set-up / tear-down)
    - End hosts (processes) can have multiple concurrent long-lived communication
  - **Reliable**, in-order, *byte-stream* delivery
    - Robust detection & retransmission of lost data
TCP “Bytestream” Service

Processes don’t ever see packet boundaries, lost or corrupted packets, retransmissions, etc.
Bidirectional communication:

There are two separate **bytestreams**, one in each direction.
TCP

Source port | Destination port
---|---
Sequence number
Acknowledgment
HdrLen | Flags | Advertised window
Checksum | Urgent pointer
Options (variable)
Data
TCP

These plus IP addresses define a given connection

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Flags</th>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Options (variable)

Data
Suppose our browser used port 23144 for our connection, and Google’s server used 443.

Then our connection will be fully specified by the single tuple 
<216.97.19.132, 23144, 172.217.6.78, 443, TCP>
TCP

Used to order data in the connection: client program receives data *in order*

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sequence number</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>0</th>
<th>Flags</th>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Data |
|------|---|

Sequence number assigned to start of byte stream is picked when connection begins; *doesn’t* start at 0
TCP

Used to say how much data has been received

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sequence number

<table>
<thead>
<tr>
<th>Acknowledgment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HdrLen</th>
<th>Flags</th>
<th>Advertised window</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checksum</th>
<th>Urgent pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options (variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

Data

Acknowledgment gives seq # just beyond highest seq. received in order.

If sender successfully sends N bytestream bytes starting at seq S then “ack” for that will be S+N.
Sequence Numbers

Host A

ISN (initial sequence number)

Sequence number from A = 1st byte of data

TCP Data

TCP HDR

Host B

TCP Data

ACK sequence number from B = next expected byte
TCP

<table>
<thead>
<tr>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment</td>
<td></td>
</tr>
<tr>
<td>HdrLen 0</td>
<td>Flags</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent pointer</td>
</tr>
<tr>
<td>Options (variable)</td>
<td></td>
</tr>
</tbody>
</table>

Data

---

Flags have different meaning:

- **SYN**: Synchronize, used to initiate a connection
- **ACK**: Acknowledge, used to indicate acknowledgement of data
- **FIN**: Finish, used to indicate no more data will be sent (but can still receive and acknowledge data)
- **RST**: Reset, used to terminate the connection completely
TCP Conn. Setup & Data Exchange

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

- **SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80, SYN, Seq = x**
- **SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=3344, SYN+ACK, Seq = y, Ack = x+1**
- **SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80, ACK, Seq = x+1, Ack = y+1**
  - **SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80, ACK, Seq = x+1, Ack = y+1, Data="GET /login.html"**
- **SrcA=9.8.7.6, SrcP=80, DstA=1.2.1.2, DstP=3344, ACK, Seq = y+1, Ack = x+16, Data="200 OK … <html> …"**
 Abrupt Termination

• A sends a TCP packet with RESET (RST) flag to B
  • E.g., because app. process on A crashed
  • (Could instead be that B sends a RST to A)
• Assuming that the sequence numbers in the RST fit with what B expects, That’s It:
  • B’s user-level process receives: ECONNRESET
  • No further communication on connection is possible
Disruption

• Normally, TCP finishes (“closes”) a connection by each side sending a FIN control message
  – Reliably delivered, since other side must ack

• But: if a TCP endpoint finds unable to continue (process dies; info from other “peer” is inconsistent), it abruptly terminates by sending a RST control message
  – Unilateral
  – Takes effect immediately (no ack needed)
  – Only accepted by peer if has correct* sequence number
TCP Threat: Data Injection

• If attacker knows ports & sequence numbers (e.g., on-path attacker), attacker can inject data into any TCP connection
  • Receiver B is *none the wiser!*

• Termed TCP *connection hijacking* (or *“session hijacking”*)
  • A general means to take over an already-established connection!

• *We are toast if an attacker can see our TCP traffic!*
  • Because then they immediately know the port & sequence numbers
TCP Data Injection

Client (initiator)
IP address 1.2.1.2, port 3344

Server
IP address 9.8.7.6, port 80

Attacker (AirPwn, QUANTUM, etc)
IP address 6.6.6.6, port N/A

Client dutifully processes as server’s response

SrcA=1.2.1.2, SrcP=3344, DstA=9.8.7.6, DstP=80,
ACK, Seq=x+1, Ack = y+1, Data="GET /login.html"

SrcA=9.8.7.6, SrcP=80,
DstA=1.2.1.2, DstP=3344,
ACK, Seq = y+1, Ack = x+16
Data="200 OK ... <poison> ...",
TCP Data Injection

Client (initiator)
IP address 1.2.1.2, port 3344

....

Server
IP address 9.8.7.6, port 80

Attacker
IP address 6.6.6.6, port N/A

Client ignores since already processed that part of bytestream: the network can duplicate packets so only pay attention to the first version in sequence.
TCP Threat: Disruption aka RST injection

- The attacker can also inject RST packets instead of payloads
  - TCP clients must respect RST packets and stop all communication
    - Because its a real world error recovery mechanism
    - So "just ignore RSTs don't work"

- Who uses this?
  - China: The Great Firewall does this to TCP requests
  - A long time ago: Comcast, to block BitTorrent uploads
  - Some intrusion detection systems: To hopefully mitigate an attack in progress
TCP Threat: Blind Hijacking

• Is it possible for an off-path attacker to inject into a TCP connection even if they can’t see our traffic?
• YES: if somehow they can infer or guess the port and sequence numbers
TCP Threat: Blind Spoofing

- Is it possible for an off-path attacker to create a fake TCP connection, even if they can’t see responses?
- YES: if somehow they can infer or guess the TCP initial sequence numbers

- Why would an attacker want to do this?
  - Perhaps to leverage a server’s trust of a given client as identified by its IP address
  - Perhaps to frame a given client so the attacker’s actions during the connections can’t be traced back to the attacker
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port N/A

Server
IP address 9.8.7.6, port 80

Blind Attacker

Attacker’s goal:

- Source: 1.2.1.2, Source Port: 5566, Destination: 9.8.7.6, Destination Port: 80, SYN, Sequence: z

- Source: 9.8.7.6, Source Port: 80, Destination: 1.2.1.2, Destination Port: 5566, SYN+ACK, Sequence: y, Acknowledgment: z+1

- Source: 1.2.1.2, Source Port: 5566, Destination: 9.8.7.6, Destination Port: 80, ACK, Sequence: z+1, Acknowledgment: y+1

- Source: 1.2.1.2, Source Port: 5566, Destination: 9.8.7.6, Destination Port: 80, ACK, Sequence: z+1, Acknowledgment: y+1, Data: “GET /transfer-money.html”
Blind Spoofing on TCP Handshake

**Alleged Client (not actual)**
- IP address 1.2.1.2, port NA

**Server**
- IP address 9.8.7.6, port 80

**Blind Attacker**
- SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, SYN, Seq = z

Small Note #1: if alleged client receives this, will be confused ⇒ send a RST back to server … … So attacker may need to hurry!
But firewalls may inadvertently stop this reply to the alleged client so it never sends the RST🤔
Blind Spoofing on TCP Handshake

Alleged Client (not actual)
IP address 1.2.1.2, port NA

Server
IP address 9.8.7.6, port 80

Blind Attacker
SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6,
DstP=80, SYN, Seq = z

SrcA=9.8.7.6, SrcP=80,
DstA=1.2.1.2, DstP=5566, SYN+ACK, Seq = y, Ack = z+1

Big Note #2: attacker doesn’t get to see this packet!
**Alleged Client (not actual)**
IP address 1.2.1.2, port N/A

**Server**
IP address 9.8.7.6, port 80

**Blind Attacker**

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, SYN, Seq = \( z \)

So how can the attacker figure out what value of \( y \) to use for their ACK?

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = \( z+1 \), ACK = \( y+1 \)

SrcA=1.2.1.2, SrcP=5566, DstA=9.8.7.6, DstP=80, ACK, Seq = \( z+1 \), ACK = \( y+1 \), Data = “GET /transfer-money.html"
Reminder: Establishing a TCP Connection

Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

Hmm, any way for the attacker to know this?

Sure – make a non-spoofed connection *first*, and see what server used for ISN y then!

How Do We Fix This?

Use a (Pseudo)-Random ISN
Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully terminate by forging a RST packet
  - Inject (spoof) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - Remains a major threat today
Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully terminate by forging a RST packet
  - Inject (spoof) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - Remains a major threat today

- If attacker could predict the ISN chosen by a server, could “blind spoof” a connection to the server
  - Makes it appear that host ABC has connected, and has sent data of the attacker’s choosing, when in fact it hasn’t
  - Undermines any security based on trusting ABC’s IP address
  - Allows attacker to “frame” ABC or otherwise avoid detection
  - Fixed (mostly) today by choosing random ISNs
But wasn't fixed completely...

- CVE-2016-5696
  - "Off-Path TCP Exploits: Global Rate Limit Considered Dangerous" Usenix Security 2016
  - https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cao

- Key idea:
  - RFC 5961 added some global rate limits that acted as an information leak:
    - Could determine if two clients were communicating on a given port
    - Could determine if you could correctly guess the sequence #s for this communication
      - Required a third host to probe this and at the same time spoof packets
    - Once you get the sequence #s, you can then inject arbitrary content into the TCP stream (d'oh)
And the Firewall...

- Attackers can't attack what they can't talk to!
  - If you don't accept *any* communication from an attacker, you can't be exploited
- The firewall is a network device (or software filter on the end host) that restricts communication
  - Primarily just by IP/Port or network/Port
- Default deny:
  - By default, disallow any contact to this host on any port
- Default allow:
  - By default, allow any contact to this host on any port
- More when we discuss Intrusion Detection next week
Theme of The Rest Of This Lecture...

"Trust does not scale because trust is not reducible to math."

- Taylor Swift
But Trust Can Be Delegated...

"Trust does not scale because trust is not reducible to math."

- Taylor Swift
The Rest of Today's Lecture:

- Applying crypto technology in practice
- Two simple abstractions cover 80% of the use cases for crypto:
  - "Sealed blob": Data that is encrypted and authenticated under a particular key
  - Secure channel: Communication channel that can’t be eavesdropped on or tampered with
- Today: TLS – a secure channel
  - In network parlance, this is an “application layer” protocol but…
  - designed to have any application over it, so really “layer 6.5” is a better description
Building Secure End-to-End Channels

- End-to-end = communication protections achieved all the way from originating client to intended server
  - With no need to trust intermediaries

- Dealing with threats:
  - Eavesdropping?
    - Encryption (including session keys)
  - Manipulation (injection, MITM)?
    - Integrity (use of a MAC); replay protection
  - Impersonation?
    - Signatures

What’s missing?

Availability …
Building A Secure End-to-End Channel: SSL/TLS

- **SSL =** Secure Sockets Layer (predecessor)
- **TLS =** Transport Layer Security (standard)
  - Both terms used interchangeably
- **Security for any application that uses TCP**
  - Secure = encryption/confidentiality + integrity + authentication (of server, but not of client)
- **Multiple uses**
  - Puts the ‘s’ in “https”
  - Secures mail sent between servers (STARTTLS)
  - Virtual Private Networks
An “Insecure” Web Page
A “Secure” Web Page

Lock Icon means:

“Your communication between your computer and the site is encrypted and authenticated”

“Some other third party attests that this site belongs to Amazon”

“These properties hold not just for the main page, but any image or script is also fetched from a site with attestation and encryption”

People *think* lock icon means “Hey, I can trust this site” (no matter where the lock icon itself actually appears).
Basic idea

- Browser (client) picks some symmetric keys for encryption + authentication
- Client sends them to server, encrypted using RSA public-key encryption
- Both sides send MACs
- Now they use these keys to encrypt and authenticate all subsequent messages, using symmetric-key crypto
HTTPS Connection (SSL / TLS)

- Browser (client) connects via TCP to Amazon’s HTTPS server
- Client picks 256-bit random number $R_B$, sends over list of crypto protocols it supports
- Server picks 256-bit random number $R_S$, selects protocols to use for this session
- Server sends over its certificate
  - (all of this is in the clear)
- Client now validates cert

![Diagram of the HTTPS connection process](image)
HTTPS Connection (SSL / TLS), cont.

- For RSA, browser constructs “Premaster Secret” PS
- Browser sends PS encrypted using Amazon’s public RSA key $K_{Amazon}$
- Using PS, $R_B$, and $R_S$, browser & server derive symmetric cipher keys $(C_B, C_S)$ & MAC integrity keys $(I_B, I_S)$
  - One pair to use in each direction
  - Done by seeding a pRNG in common between the browser and the server:
    Repeated calls to the pRNG then create the common keys
HTTPS Connection (SSL / TLS), cont.

- For RSA, browser constructs “Premaster Secret” PS
- Browser sends PS encrypted using Amazon’s public RSA key \( K_{Amazon} \)
- Using PS, \( R_B \), and \( R_S \), browser & server derive symm. cipher keys (\( C_B, C_S \)) & MAC integrity keys (\( I_B, I_S \))
  - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displays ⛔
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
  - Sequence #’s thwart replay attacks

Here’s my cert
~2-3 KB of data

\( \{ PS \}_{K_{Amazon}} \)

\( \{ M_1, MAC(M_1, I_B) \}_{C_B} \)

\( \{ M_2, MAC(M_2, I_S) \}_{C_S} \)
Alternative: Ephemeral Key Exchange via Diffie-Hellman

- For Diffie-Hellman, server generates random $a$, sends public parameters and $g^a \mod p$
  - Signed with server’s private key
- Browser verifies signature
- Browser generates random $b$, computes $PS = g^{ab} \mod p$, sends $g^b \mod p$ to server
- Server also computes $PS = g^{ab} \mod p$
- Remainder is as before: from $PS, R_B$, and $R_S$, browser & server derive symm. cipher keys ($C_B, C_S$) and MAC integrity keys ($I_B, I_S$), etc…

Here’s my cert
~2-3 KB of data

$\{g, p, g^a \mod p\} K^{-1}_{Amazon}$

$g^b \mod p$

MAC(dialog, $I_B$)

MAC(dialog, $I_S$)

$\{M_1, MAC(M_1, I_B)\}_{C_B}$

...
Big Changes for TLS 1.3
Diffie/Hellman and ECDHE only

- The RSA key exchange has a substantial vulnerability
  - If the attacker is ever able to compromise the server and obtain its RSA key…
    the attacker can decrypt any traffic captured
- RSA lacks forward secrecy
- So TLS 1.3 uses DHE/ECDHE only
- TLS 1.3 also speeds things up:
  - In the client hello, the client includes \( g^b \mod p \) for preferred parameters
    - If the server finds it suitable, the server returns \( g^a \mod p \)
  - Saves a round-trip time
But What About that “Certificate Validation”

- Certificate validation is used to establish a chain of “trust”
  - It actually is an attempt to build a scalable trust framework
- This is commonly known as a Public Key Infrastructure (PKI)
  - Your browser is trusting the “Certificate Authority” to be responsible…

"Trust does not scale because trust is not reducible to math."
- Taylor Swift
Certificates

- Cert = signed statement about someone’s public key
  - Note that a cert does not say anything about the identity of who gives you the cert
  - It simply states a given public key $K_{Bob}$ belongs to Bob …
    - … and backs up this statement with a digital signature made using a different public/private key pair, say from Verisign (a “Certificate Authority”)

- Bob then can prove his identity to you by you sending him something encrypted with $K_{Bob}$ …
  - … which he then demonstrates he can read

- … or by signing something he demonstrably uses
- Works provided you trust that you have a valid copy of Verisign’s public key …
  - … and you trust Verisign to use prudence when she signs other people’s keys
Validating Amazon’s Identity

- Browser compares domain name in cert w/ URL
  - Note: this provides an **end-to-end** property (as opposed to say a cert associated with an IP address)

- Browser accesses separate cert belonging to issuer
  - These are hardwired into the browser – **and trusted**!
  - There could be a chain of these …

- Browser applies issuer’s public key to verify signature $S$, obtaining the hash of what the issuer signed
  - Compares with its own SHA-1 hash of Amazon’s cert

- Assuming hashes match, now have high confidence it’s indeed Amazon’s public key …
  - assuming signatory is trustworthy, didn’t lose private key, wasn’t tricked into signing someone else’s certificate, and that Amazon didn’t lose their key either…
End-to-End ⇒ Powerful Protections

- Attacker runs a sniffer to capture our WiFi session?
  - But: encrypted communication is unreadable
    - No problem!
- DNS cache poisoning?
  - Client goes to wrong server
  - But: detects impersonation
    - No problem!
- Attacker hijacks our connection, injects new traffic
  - But: data receiver rejects it due to failed integrity check since all communication has a mac on it
    - No problem!
- Only thing a **full man-in-the-middle** attacker can do is inject RSTs, inject invalid packets, or drop packets: limited to a **denial of service**
Validating Amazon’s Identity, cont.

- Browser retrieves cert belonging to the issuer
  - These are hardwired into the browser – and trusted!
- But what if the browser can’t find a cert for the issuer?
This Connection is Untrusted

You have asked Firefox to connect securely to www.mikestoolbox.org, but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

[Get me out of here]

▶ Technical Details

www.mikestoolbox.org uses a certificate issued by a trusted certificate authority.

The certificate is not trusted by Firefox.

(Error code: sec_error_untrusted_cert_issuer)

▶ I Understand the Risk:

[Safari can't verify the identity of the website “www.mikestoolbox.org”.

The certificate for this website was signed by an unknown certifying authority. You might be connecting to a website that is pretending to be “www.mikestoolbox.org”, which could put your confidential information at risk. Would you like to connect to the website anyway?

[Show Certificate] [Cancel] [Continue]
Validating Amazon’s Identity, cont.

• Browser retrieves cert belonging to the issuer
  • These are hardwired into the browser – and trusted!

• What if browser can’t find a cert for the issuer?
• If it can’t find the cert, then warns the user that site has not been verified
  • Can still proceed, just without authentication

• Q: Which end-to-end security properties do we lose if we incorrectly trust that the site is whom we think?
• A: All of them!
  • Goodbye confidentiality, integrity, authentication
  • Active attacker can read everything, modify, impersonate
SSL / TLS Limitations

- Properly used, SSL / TLS provides powerful end-to-end protections
- So why not use it for everything??

Issues:
- Cost of public-key crypto (fairly minor)
  - Takes non-trivial CPU processing (but today a minor issue)
  - Note: symmetric key crypto on modern hardware is effectively free
- Hassle of buying/maintaining certs (fairly minor)
  - LetsEncrypt makes this almost automatic
- Integrating with other sites that don’t use HTTPS
  - Namely, you can’t: Non-HTTPS content won’t load!
- Latency: extra round trips ⇒ 1st page slower to load
SSL / TLS Limitations, cont.

- Problems that SSL / TLS does not take care of?
- Censorship:
  - The censor sees the certificate in the clear, so knows who the client is talking to
  - Optional Server Name Identification (SNI) is also sent in the clear
  - The censor can then inject RSTs or block the communication
- SQL injection/XSS/CSRF/server-side coding/logic flaws
- Vulnerabilities introduced by server inconsistencies
SSL/TLS Problem: Revocation

• A site screws up and an attacker steals the private key associated with a certificate, what now?
  • Certificates have a timestamp and are only good for a specified time
    • But this time is measured in years!?!?

• Two mitigations:
  • Certificate revocation lists
    • Your browser occasionally calls back to get a list of "no longer accepted" certificates
  • OSCP
“sslstrip”
(Amazon FINALLY fixed this recently)

Regular web surfing: http: URL

So *no integrity* - a MITM attacker can alter pages returned by server ...

And when we click here ... ... attacker has changed the corresponding link so that it’s ordinary http rather than https!

We never get a chance to use TLS’s protections! :-(
SSL / TLS Limitations, cont.

- Problems that SSL / TLS does not take care of?
- Censorship
- SQL injection / XSS / server-side coding/logic flaws
- Vulnerabilities introduced by server inconsistencies
- Browser and server bugs
- Bad passwords
- What about the trust?
TLS/SSL Trust Issues

- User has to make correct trust decisions ...
Please confirm your identity.

Please answer security question:

Select your secret question:

Answer the secret question you provided:

What is your other eBay user ID or another name:

What email used to be associated with this account:

Have you ever sold something on eBay?

- No
- Yes

Certificate Information:

This certificate is intended for the following purpose(s):
- Ensures the identity of a remote computer

* Refer to the certification authority's statement for details.

Issued to: reвер. ebay.com

Issued by: VeriSign Class 3 Secure Server CA - GO

Valid from: 10/22/2010 to 12/1/2012
Please confirm your identity.

For security reasons, please answer the following question:

Select your secret question:

Answer the secret question you provided.

What is your other eBay user ID or another email used to be associated with this account?

Have you ever sold something on eBay?
- No
- Yes

Insert the value of the following field:

- Field: Version
  - Value: V3

- Field: Serial number
  - Value: 4d ab c5 as 0a 30 20 57 1d 2c ...

- Field: Signature algorithm
  - Value: sha1RSA

- Field: Issuer
  - Value: VeriSign Class 3 Secure Server...

- Field: Valid from
  - Value: Friday, October 22, 2010 0:00...

- Field: Valid to
  - Value: Saturday, December 01, 2012...

- Field: Subject
  - Value: seller.ebay.com, Site Operations...

- Field: Public key
  - Value: RSA (1024 bits)
Please confirm your identity.

Please answer security question:

Select your secret question:

Answer the secret question you provided:

What is your other eBay user ID or another email used to be associated with this account?

Have you ever sold something on eBay?
- No
- Yes

Certificate

OK
The equivalent as seen by most Internet users:

(note: an actual Windows error message!)
TLS/SSL Trust Issues, cont.

- “Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.”
- Matt Blaze, circa 2001
- So how many CAs do we have to worry about, anyway?
Click to unlock the System Roots keychain.

<table>
<thead>
<tr>
<th>Name</th>
<th>Kind</th>
<th>Date Modified</th>
<th>Expires</th>
<th>Keychain</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA Certificate Services</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2026, 3:59:59 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Actails Authentication Root CA</td>
<td>certificate</td>
<td>--</td>
<td>Sep 22, 2030, 4:22:02 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust Class 1 CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020, 3:38:31 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AddTrust External CA Root</td>
<td>certificate</td>
<td>--</td>
<td>May 30, 2020, 3:48:38 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Admin-Root-CA</td>
<td>certificate</td>
<td>--</td>
<td>Nov 9, 2021, 11:51:07 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Commercial</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 6:06:06 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Networking</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 6:08:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2040, 6:10:36 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>AffirmTrust Premium ECC</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2040, 6:20:24 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ANF Global Root CA</td>
<td>certificate</td>
<td>--</td>
<td>Jun 5, 2033, 10:45:38 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA</td>
<td>certificate</td>
<td>--</td>
<td>Feb 9, 2036, 1:40:36 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA - G2</td>
<td>certificate</td>
<td>--</td>
<td>Apr 30, 2039, 11:10:09 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root CA - G3</td>
<td>certificate</td>
<td>--</td>
<td>Apr 30, 2039, 11:19:06 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Apple Root Certificate Authority</td>
<td>certificate</td>
<td>--</td>
<td>Feb 9, 2026, 4:18:14 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ApplicationCA</td>
<td>certificate</td>
<td>--</td>
<td>Dec 12, 2017, 7:00:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>ApplicationCA2 Root</td>
<td>certificate</td>
<td>--</td>
<td>Mar 12, 2033, 7:00:00 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Atos TrustedRoot 2011</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 3:59:59 PM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Autoridad de...nal CIF A62634068</td>
<td>certificate</td>
<td>--</td>
<td>Dec 31, 2030, 12:38:15 AM</td>
<td>System Roots</td>
</tr>
<tr>
<td>Autoridad de...Estado Venezolano</td>
<td>certificate</td>
<td>--</td>
<td>Dec 17, 2030, 3:59:59 PM</td>
<td>System Roots</td>
</tr>
</tbody>
</table>

168 items
TLS/SSL Trust Issues

• “Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.”
  • Matt Blaze, circa 2001
• So how many CAs do we have to worry about, anyway?
• Of course, it’s not just their greed that matters …
News

Solo Iranian hacker takes credit for Comodo certificate attack

Security researchers split on whether 'ComodoHacker' is the real deal

By Gregg Keizer
March 27, 2011 08:39 PM ET

Computerworld - A solo Iranian hacker on Saturday claimed responsibility for stealing multiple SSL certificates belonging to some of the Web's biggest sites, including Google, Microsoft, Skype and Yahoo.

Early reaction from security experts was mixed, with some believing the hacker's claim, while others were dubious.
Fraudulent Google certificate points to Internet attack

Is Iran behind a fraudulent Google.com digital certificate? The situation is similar to one that happened in March in which spoofed certificates were traced back to Iran.

by Elinor Mills | August 29, 2011 1:22 PM PDT

A Dutch company appears to have issued a digital certificate for Google.com to someone other than Google, who may be using it to try to re-direct traffic of users based in Iran.

Yesterday, someone reported on a Google support site that when attempting to log in to Gmail the browser issued a warning for the digital certificate used as proof that the site is legitimate, according to this thread on a Google support forum site.
This appears to be a fully valid cert using normal browser validation rules.

Only detected by Chrome due to its introduction of cert “pinning” – requiring that certs for certain domains must be signed by specific CAs rather than any generally trusted CA.
Final Report on DigiNotar Hack Shows Total Compromise of CA Servers

The attacker who penetrated the Dutch CA DigiNotar last year had complete control of all eight of the company’s certificate-issuing servers during the operation and he may also have issued some rogue certificates that have not yet been identified. The final report from a

Evidence Suggests DigiNotar, Who Issued Fraudulent Google Certificate, Was Hacked Years Ago

from the *diginot* dept

The big news in the security world, obviously, is the fact that a [fraudulent Google certificate made its way out into the wild](https://example.com), apparently targeting internet users in Iran. The Dutch company DigiNotar has put out a statement saying that [it discovered a breach](https://example.com) back on July 19th during a security audit, and that fraudulent certificates were generated for "several dozen" websites. The only one known to have gotten out into the wild is the Google one.
The DigiNotar Fallout

• The result was the “CA Death Sentence”:
  • Web browsers removed it from the trusted root certificate store
• This has just happened again with “WoSign”
  • A Chinese CA
• WoSign would allow an interesting attack
  • If I controlled nweaver.github.com…
  • WoSign would allow me to create a certificate for *.github.com!?!?
  • And a bunch of other shady shenanigans
TLS/SSL Trust Issues

- “Commercial certificate authorities protect you from anyone from whom they are unwilling to take money.”
  - Matt Blaze, circa 2001

- So how many CAs do we have to worry about, anyway?

- Of course, it’s not just their greed that matters …

- … and it’s not just their diligence & security that matters …

- “A decade ago, I observed that commercial certificate authorities protect you from anyone from whom they are unwilling to take money. That turns out to be wrong; they don't even do that much.” - Matt Blaze, circa 2010
So the Modern Solution: Invoke Ronald Reagan, “Trust, but Verify”

- Static Certificate Pinning:
  The chrome browser has a list of certificates or certificate authorities that it trusts for given sites
  - Now creating a fake certificate requires attacking a particular CA

- HPKP Certificate Pinning:
  The web server provides hashes of certificates that should be trusted
  - This is “Leap of Faith”: The first time you assume it is honest but you will catch future changes

- Transparency mechanisms:
  - Public logs provided by certificate authorities
  - Browser extensions (EFF’s TLS observatory)
  - Backbone monitors (ICSI’s TLS notary)