

Designing Secure Systems with Transport Layer Security

Anthony Vcherushniy, Google

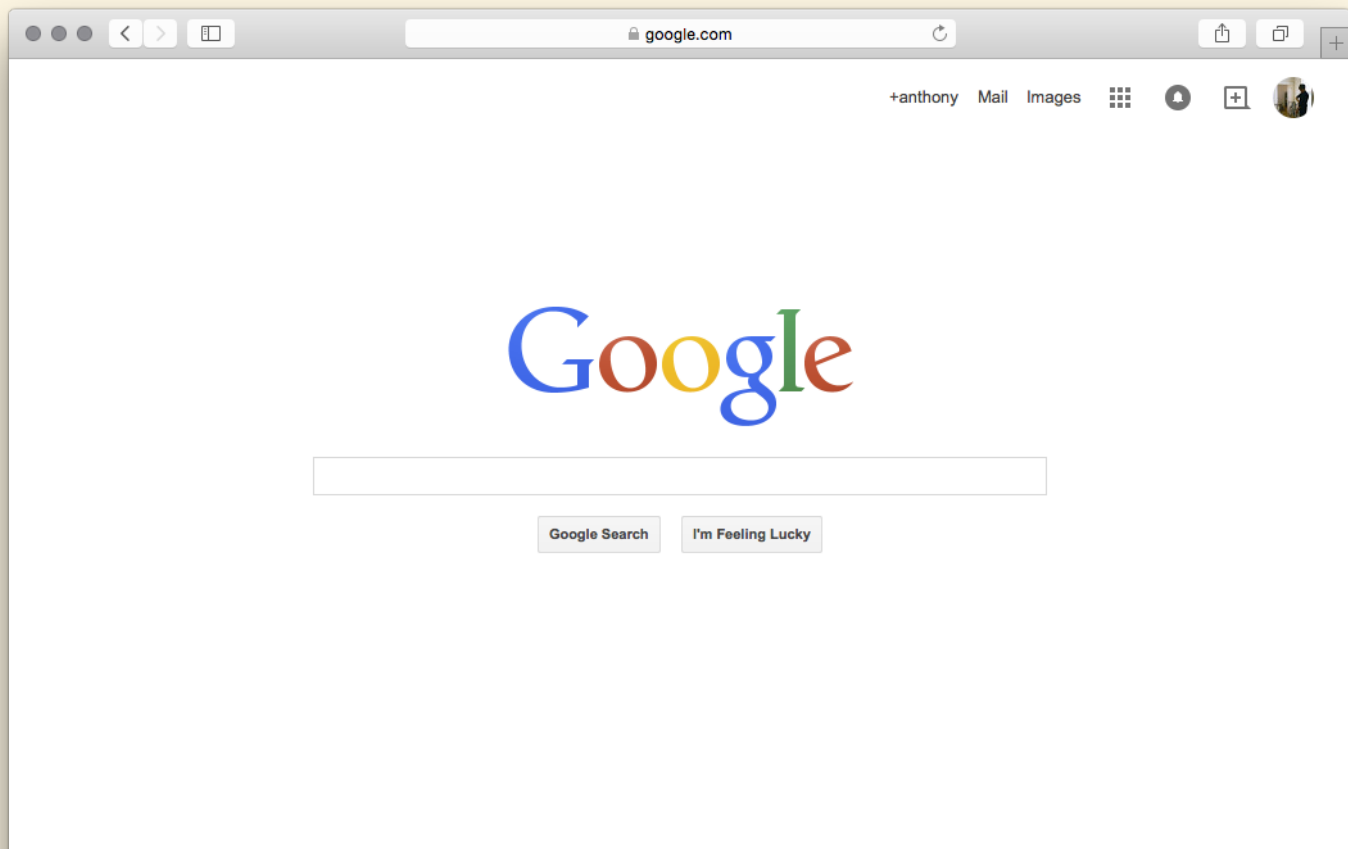
November 6, 2014

About our Macs

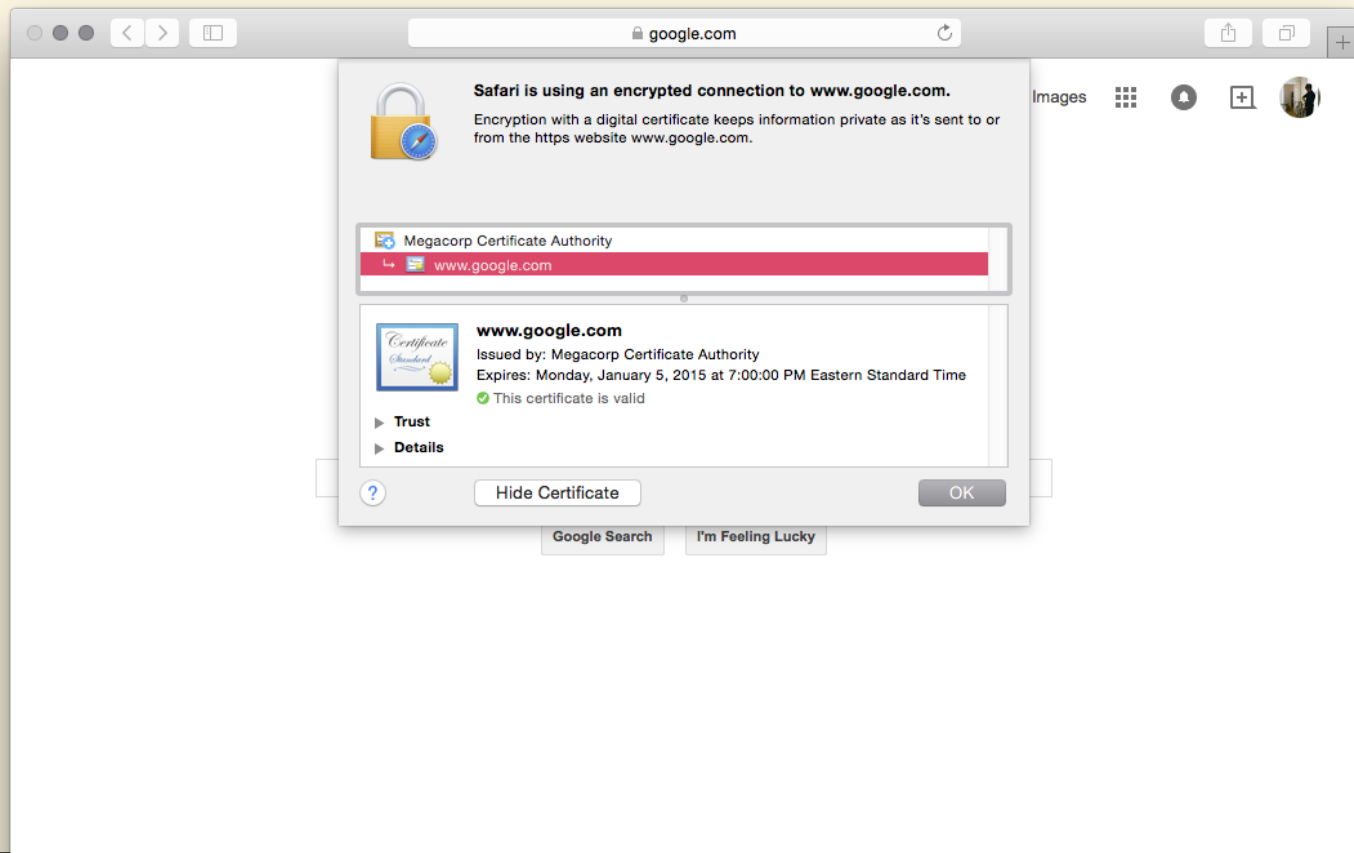
- 6 engineers - with open headcount!
- 58,000 Macs and counting
- Security is serious business
- Almost exclusively mobile fleet
- Puppet, Munki, Simian, Cauliflower Vest
- No dependence on privileged networks or VPNs

About this talk

- Cryptographic protocols
- Implementations and mistakes
- Open source code



It feels safe



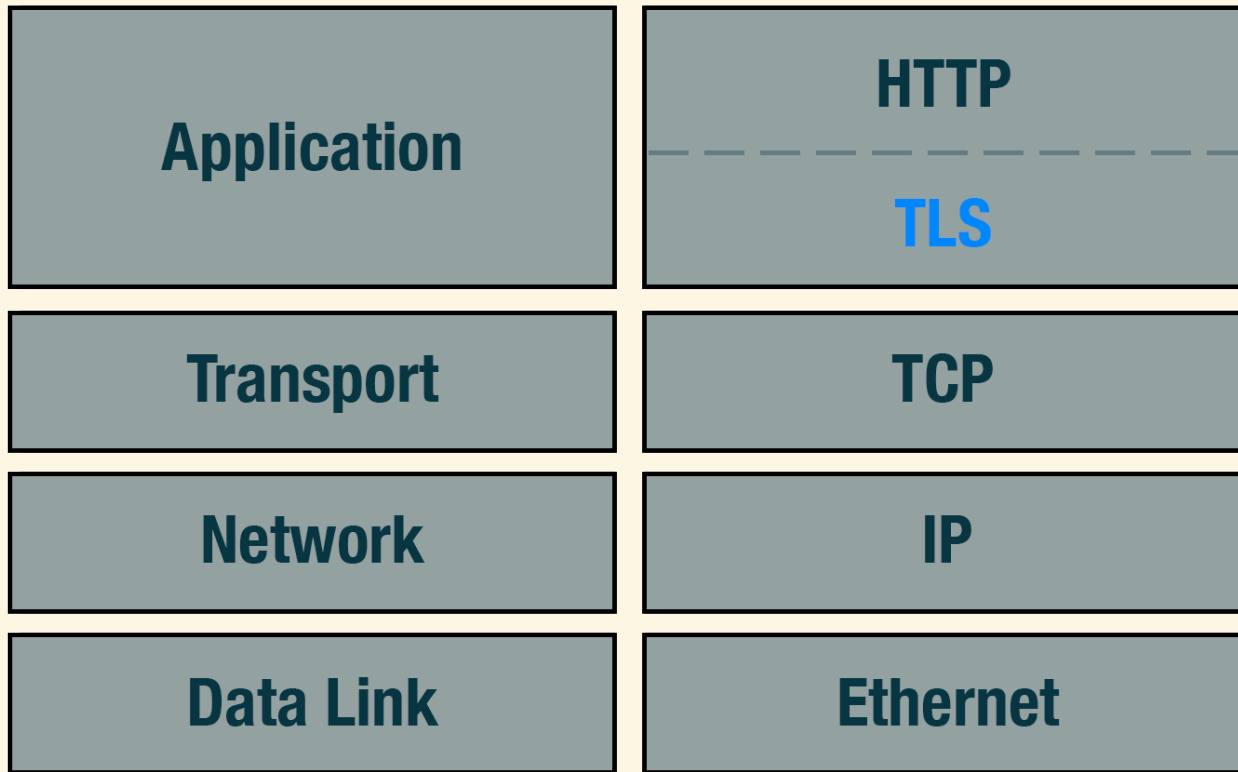
But is it?



Man in the middle

TLS, formerly known as SSL

- SSLv1
 - Netscape internal only
- SSLv2
 - 1995
 - Netscape Navigator
 - inherently insecure:
MD5 MACs, no cert chains
- SSLv3
 - over 15 years old
 - bit by POODLE
 - should not be used
- TLSv1.0
 - First IETF standard,
RFC 2246, 1999
 - essentially SSLv3.1
 - holds up with careful
configuration
- TLSv1.1
 - fixes CBC-related attacks
- TLSv1.2
 - best protocol to use today
- TLSv1.3
 - in draft
 - datatracker.ietf.org/wg/tls



HTTP over TLS = https

What TLS has to offer

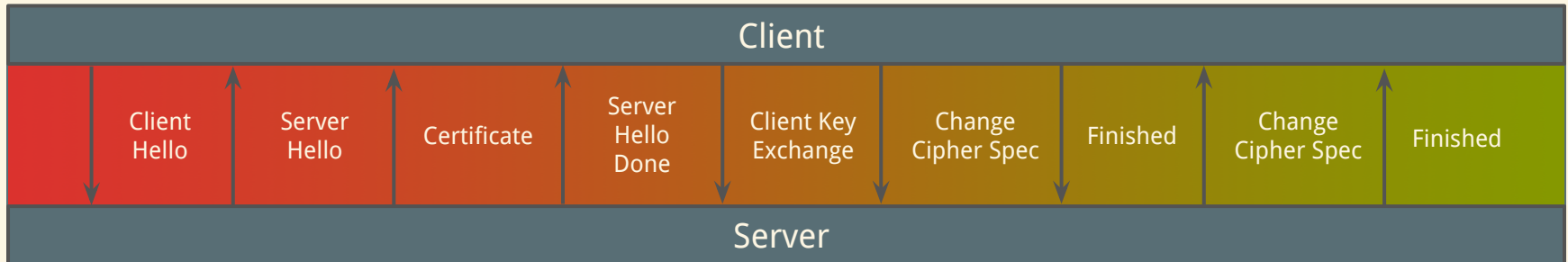
- Server and client authentication
 - web server proves its identity to the browser
- Negotiation of a secret key
 - protected from passive or active eavesdropper
- Privacy and confidentiality
 - shared secret is used to encrypt bytes on the wire
- Data integrity
 - signed, tamper proof message digests using message authentication codes (MAC)

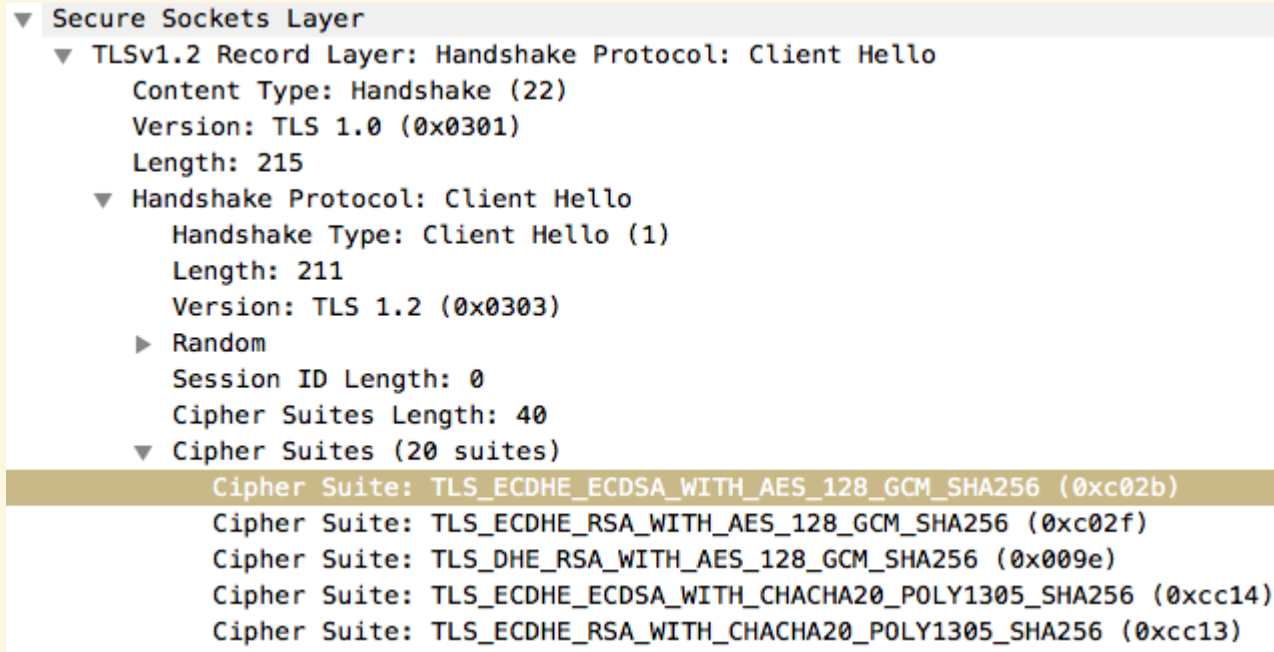
1	0.000	172.16.0.7	74.125.226.17	TCP	78 60126→http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=16 TSval=522238539 TSecr=0 SACK_PERM=1
2	0.002	74.125.226.17	172.16.0.7	TCP	74 http→60126 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=68110181 TSecr=522238539 WS=16
3	0.002	172.16.0.7	74.125.226.17	TCP	66 60126→http [ACK] Seq=1 Ack=1 Win=131760 Len=0 TSval=522238541 TSecr=68110181
4	0.002	172.16.0.7	74.125.226.17	HTTP	363 GET / HTTP/1.1
5	0.003	74.125.226.17	172.16.0.7	TCP	66 http→60126 [ACK] Seq=1 Ack=298 Win=6864 Len=0 TSval=68110181 TSecr=522238541
6	0.007	74.125.226.17	172.16.0.7	HTTP	201 HTTP/1.1 302 Local Redirect from Privoxy
7	0.007	172.16.0.7	74.125.226.17	TCP	66 60126→http [ACK] Seq=298 Ack=136 Win=131632 Len=0 TSval=522238545 TSecr=68110181
8	0.007	172.16.0.7	74.125.226.17	TCP	78 60127→https [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=16 TSval=522238545 TSecr=0 SACK_PERM=1
9	0.048	74.125.226.17	172.16.0.7	TCP	74 https→60127 [SYN, ACK] Seq=0 Ack=1 Win=42540 Len=0 MSS=1430 SACK_PERM=1 TSval=2131823103 TSecr=522238545 WS=128
10	0.048	172.16.0.7	74.125.226.17	TCP	66 60127→https [ACK] Seq=1 Ack=1 Win=131872 Len=0 TSval=522238586 TSecr=2131823103
11	0.048	172.16.0.7	74.125.226.17	TLSv1.2	233 Client Hello
12	0.088	74.125.226.17	172.16.0.7	TCP	66 https→60127 [ACK] Seq=1 Ack=168 Win=43648 Len=0 TSval=2131823146 TSecr=522238586
13	0.091	74.125.226.17	172.16.0.7	TLSv1.2	1484 Server Hello
14	0.091	74.125.226.17	172.16.0.7	TCP	1484 [TCP segment of a reassembled PDU]
15	0.091	74.125.226.17	172.16.0.7	TLSv1.2	767 Certificate
16	0.091	172.16.0.7	74.125.226.17	TCP	66 60127→https [ACK] Seq=168 Ack=2837 Win=129648 Len=0 TSval=522238628 TSecr=2131823148
17	0.091	172.16.0.7	74.125.226.17	TCP	66 60127→https [ACK] Seq=168 Ack=3538 Win=128944 Len=0 TSval=522238628 TSecr=2131823148
18	0.094	172.16.0.7	74.125.226.17	TLSv1.2	228 Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
19	0.094	172.16.0.7	74.125.226.17	TLSv1.2	131 Application Data
20	0.094	172.16.0.7	74.125.226.17	TLSv1.2	465 Application Data
21	0.122	74.125.226.17	172.16.0.7	TLSv1.2	312 New Session Ticket, Change Cipher Spec, Hello Request, Hello Request
22	0.122	172.16.0.7	74.125.226.17	TCP	66 60127→https [ACK] Seq=794 Ack=3784 Win=130816 Len=0 TSval=522238657 TSecr=2131823179
23	0.122	74.125.226.17	172.16.0.7	TLSv1.2	123 Application Data
24	0.122	74.125.226.17	172.16.0.7	TLSv1.2	111 Application Data
25	0.122	172.16.0.7	74.125.226.17	TCP	66 60127→https [ACK] Seq=794 Ack=3841 Win=131008 Len=0 TSval=522238657 TSecr=2131823179
26	0.122	172.16.0.7	74.125.226.17	TCP	66 60127→https [ACK] Seq=794 Ack=3886 Win=130960 Len=0 TSval=522238657 TSecr=2131823179
27	0.132	74.125.226.17	172.16.0.7	TCP	66 https→60127 [ACK] Seq=3886 Ack=794 Win=45824 Len=0 TSval=2131823189 TSecr=522238631

Secure session established with www.google.com

Establishing a secure session

1. Client
 - Generates random numbers
 - Initiates contact & version
 - Sends preferred ciphers
2. Server
 - Picks most secure cipher supported and sets version
 - Generates random numbers
 - Sends certificate(s)
3. Client
 - Evaluates certificate(s)
 - Generates session key
 - Encrypts session key with server's public key
4. Server
 - Can decrypt or derive secret session key and start encrypted communications





ECDHE - Elliptic Curve Ephemeral Diffie-Hellman, the key exchange mechanism

ECDSA - Elliptic Curve Digital Signature Algorithm, the key authentication mechanism

AES_128_GCM - AES with 128-bit key in counter mode, the cipher used for encryption (with GHASH MAC)

SHA256 - Secure Hash Algorithm with 256-bit digest, used in pseudorandom function (PRF)

TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

Key exchange

- need to exchange secret key over an insecure channel, like the Internet
- using asymmetric encryption or key exchange algorithm
- in which a public key is used for encryption, private key for decryption
- much more computationally expensive than symmetric encryption

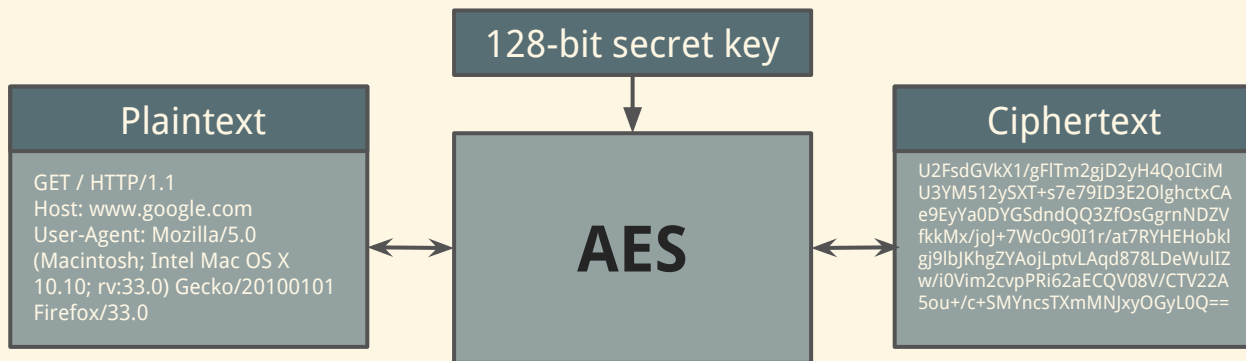
Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2
RSA	Yes	Yes	Yes	Yes	Yes
DH-RSA	No	Yes	Yes	Yes	Yes
DHE-RSA (forward secrecy)		Yes	Yes	Yes	Yes
ECDH-RSA	No	No	Yes	Yes	Yes
ECDHE-RSA (forward secrecy)			Yes	Yes	Yes
DH-DSS	No	Yes	Yes	Yes	Yes
DHE-DSS (forward secrecy)		Yes	Yes	Yes	Yes
ECDH-ECDSA	No	No	Yes	Yes	Yes
ECDHE-ECDSA (forward secrecy)			Yes	Yes	Yes
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes

Key exchange

- typically using RSA
 - shared secret is encrypted and sent
 - private key used for authentication and encryption
 - based on integer factorization: find prime factors of a huge number
- or a variant of Ephemeral Diffie-Hellman
 - slower, but fast with elliptic curves, supports Forward Secrecy
 - private key used for authentication only
 - based on discrete logarithm: find g^{xy} given g , g^x and g^y
- Forward Secrecy
 - peers discard keys used for encryption after the conversation is done
 - encrypted communications can be recorded for several years
 - without FS, recovering a server's private key can decrypt all conversations later

Symmetric encryption

- uses the same key for encryption and decryption, unlike asymmetric crypto
- client and server have shared secret key from key exchange
- rest of the conversation is encrypted using a block or stream cipher
- most well-researched cipher is AES, operates on fixed-length blocks
- stream ciphers like ChaCha20 are much faster, operate on bit stream

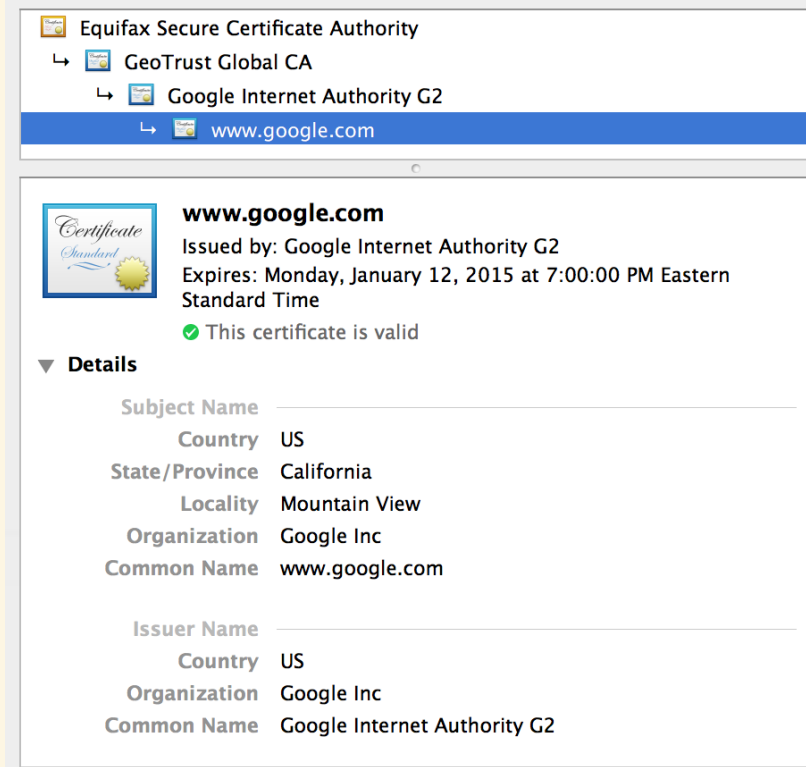


Symmetric encryption

Cipher			Protocol version				
Type ⇅	Algorithm ⇅	Strength (bits) ⇅	SSL 2.0 ⇅	SSL 3.0 [note 1][note 2][note 3][note 4] ⇅	TLS 1.0 [note 1][note 3] ⇅	TLS 1.1 [note 1] ⇅	TLS 1.2 [note 1] ⇅
Block cipher with mode of operation	AES CBC ^[note 5]	128, 256	N/A	N/A	Depends on mitigations	Secure	Secure
	AES GCM ^{[22][note 6]}		N/A	N/A	N/A	N/A	Secure
	AES CCM ^{[23][note 6]}		N/A	N/A	N/A	N/A	Secure
	CAMELLIA CBC ^{[24][note 5]}	128, 256	N/A	N/A	Depends on mitigations	Secure	Secure
	CAMELLIA GCM ^{[25][note 6]}		N/A	N/A	N/A	N/A	Secure
	SEED CBC ^{[26][note 5]}	128	N/A	N/A	Depends on mitigations	Secure	Secure
	ARIA CBC ^{[27][note 5]}	128, 256	N/A	N/A	Depends on mitigations	Secure	Secure
	ARIA GCM ^{[27][note 6]}		N/A	N/A	N/A	N/A	Secure
	IDEA CBC ^{[note 5][note 7]}	128	Insecure	Insecure	Depends on mitigations	Secure	N/A
	3DES EDE CBC ^[note 5]	112 ^[note 8]	Insecure	Insecure	Low strength, Depends on mitigations	Low strength	Low strength
	DES CBC ^{[note 5][note 7]}	56	Insecure	Insecure	Insecure	Insecure	N/A
		40 ^[note 9]	Insecure	Insecure	Insecure	N/A	N/A
	RC2 CBC ^[note 5]	40 ^[note 9]	Insecure	Insecure	Insecure	N/A	N/A
Stream cipher	CHACHA20+POLY1305 ^{[31][note 6]}	256	N/A	N/A	N/A	N/A	Secure
	RC4 ^[note 10]	128	Insecure	Insecure	Insecure	Insecure	Insecure
		40 ^[note 9]	Insecure	Insecure	Insecure	N/A	N/A
None	NULL ^[note 11]	-	N/A	Insecure	Insecure	Insecure	Insecure

X.509 certificates

- contain digitally signed identity information
- have a trust path to a trusted root certificate authority
- hierarchical authority system, not like PGP's web of trust
- certificate contents digested by a crypto hash function
- making them unique
- its up to the implementation in software to actually verify them




Equifax Secure Certificate Authority

↳ GeoTrust Global CA

↳ Google Internet Authority G2

↳ **www.google.com**

 **www.google.com**

Issued by: Google Internet Authority G2

Expires: Monday, January 12, 2015 at 7:00:00 PM Eastern Standard Time

✔ This certificate is valid

▼ Details

Subject Name	
Country	US
State/Province	California
Locality	Mountain View
Organization	Google Inc
Common Name	www.google.com

Issuer Name	
Country	US
Organization	Google Inc
Common Name	Google Internet Authority G2

Private key

- size matters, minimum of 2048-bit recommended
- protect with a password
- renew frequently, at least once a year
- random number generator is important
- OS X uses Yarrow algorithm, inherited from FreeBSD
 - from the *random* manual page:

LIMITATIONS AND WARNINGS

Yarrow is a fairly resilient algorithm, and is believed to be resistant to non-root. The quality of its output is however dependent on regular addition of appropriate entropy. If the SecurityServer system daemon fails for any reason, output quality will suffer over time without any explicit indication from the random device itself.



Certificate authorities

- Certificate authority is essentially a trusted self-signed certificate
- All trusted authorities can issue certificates for any domain
- A single CA undermines the security of the entire PKI
- Preinstalled OS X Trust Store
 - <http://support.apple.com/kb/HT6005>
 - 220 (!) trusted root certificates in Yosemite
 - mostly for-profit corporations like Apple, Verisign, Thawte, Digicert
 - 15 government agencies from China, Japan, Netherlands, U.S., etc.
 - far too many 1024-bit keys
- CAs can be managed through either Keychain Access or *security*
- OS X trust store allows restricting certificates to code signing, X.509, S/MIME, etc only, but not actually done for any roots



Fun CA facts

- Distribution among certificate authorities is heavily skewed towards a handful of large authorities
- Three organizations control 75% of all trusted certificates
- Compromise of the private key used by one particular intermediate certificate would require 26% of HTTPS websites to obtain new certs
- A single vulnerable, malicious or coercible CA undermines security of all
- Google's Certificate Transparency project makes TLS certificates public knowledge to hold CAs accountable for all certificates issued

www.certificate-transparency.org



CA compromise

- DigiNotar
 - July 2011: internal systems compromise discovered
 - 247 fake certificates issued over 27 CNs
- TurkTrust
 - August 2011: issues two intermediate certificates instead of regular ones
- Comodo
 - compromised by Iranian hackers
- French and Indian governments both issued fake certificates for google.com this year

WIRED GEAR SCIENCE ENTERTAINMENT BUSINESS SECURITY DESIGN OPINION MAGAZINE

THREAT LEVEL | cybersecurity | DigiNotar | Hacks and Cracks | ssl

DigiNotar Files for Bankruptcy in Wake of Devastating Hack

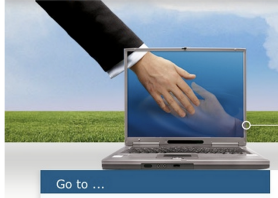

BY KIM ZETTER 09.20.11 | 3:05 PM | PERMALINK

0 | 8+1 | 17

A Dutch certificate authority that suffered a major hack attack this summer has been unable to recover from the blow and filed for bankruptcy this week.

DigiNotar, which is owned by Illinois-based Vasco Data Security and was the primary provider of digital security certificates for domains owned by the Dutch government, was breached in early June due to lax security.

The breach allowed the intruder to trick DigiNotar's system into issuing him more than 500 fraudulent digital certificates for top internet companies like Google, Mozilla, and Skype. This meant that users who went to a supposedly secure page such as <https://google.com> were at risk of having a malicious third party who possessed the Google certificate pose as the legitimate site and trick the user into entering his username and password into the impostor site.



Server and client certificates

- Check “not before” and “not after” dates
- Check the digital signature of the signing CA
- Check the certificate chain up to a trusted root CA
- Check the server’s hostname
 - would you trust a certificate for www.xcz.cn when accessing email?
 - must match either Common Name (CN) or subjectAltName (SAN)
 - must match the user’s requested hostname, not DNS
- See RFC 6125 for more
- Clients can also present certificates for authentication
 - private key material safely stored in OS X keychain



Simple public key infrastructure



Implementations

- OpenSSL
 - well known, widely audited and available on most platforms
 - unstable API between major versions
 - outdated (version 0.9.8za) and considered deprecated by Apple
- Secure Transport aka DarwinSSL aka libsecurity_ssl
 - Apple's native solution for iOS and OS X
- OpenSSL forks
 - BoringSSL: being deployed in Chromium
 - OpenBSD's LibreSSL
- Network Security Services (NSS)
 - used by Mozilla products, AIM, others
- GnuTLS
- Microsoft's Secure Channel

TLS in Python

- Mostly bad; many libraries do not properly implement TLS
 - urllib2 documentation starts out with

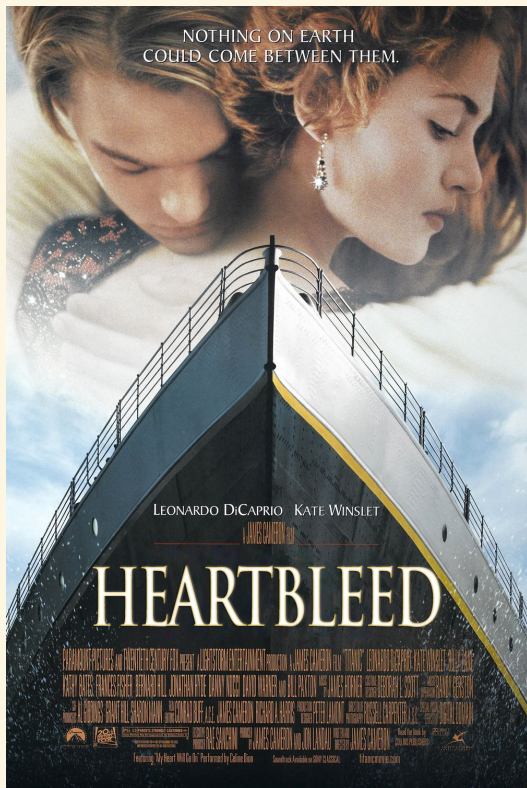
```
urllib2.urlopen(url[, data][, timeout])
```

Open the URL *url*, which can be either a string or a `Request` object.

Warning: HTTPS requests do not do any verification of the server's certificate.

- When in doubt, do what modern browsers do
 - Requests: <http://docs.python-requests.org/>
- If you *really* know what you're doing
 - M2Crypto: <https://pypi.python.org/pypi/M2Crypto>
- Hard mode
 - Python 3, ssl standard library and PEP 466

Mistakes



Heartbleed (CVE-2014-0160)

- programming mistake in OpenSSL v1.0.1-1.0.1f implementation of TLS/DTLS heartbeat extension
- reveals memory contents, like a private key
- vulnerable server responds with up to 64k of memory adjacent to request buffer
- leaves no trace

POODLE (CVE--2014--3566)

- possible downgrade to SSLv3 for client interoperability
- side channel attack on nondeterministic padding
- Security Update 2014-005 disables CBC in SSLv3
- SSLv3 is unsafe and obsolete, do not enable it on servers!

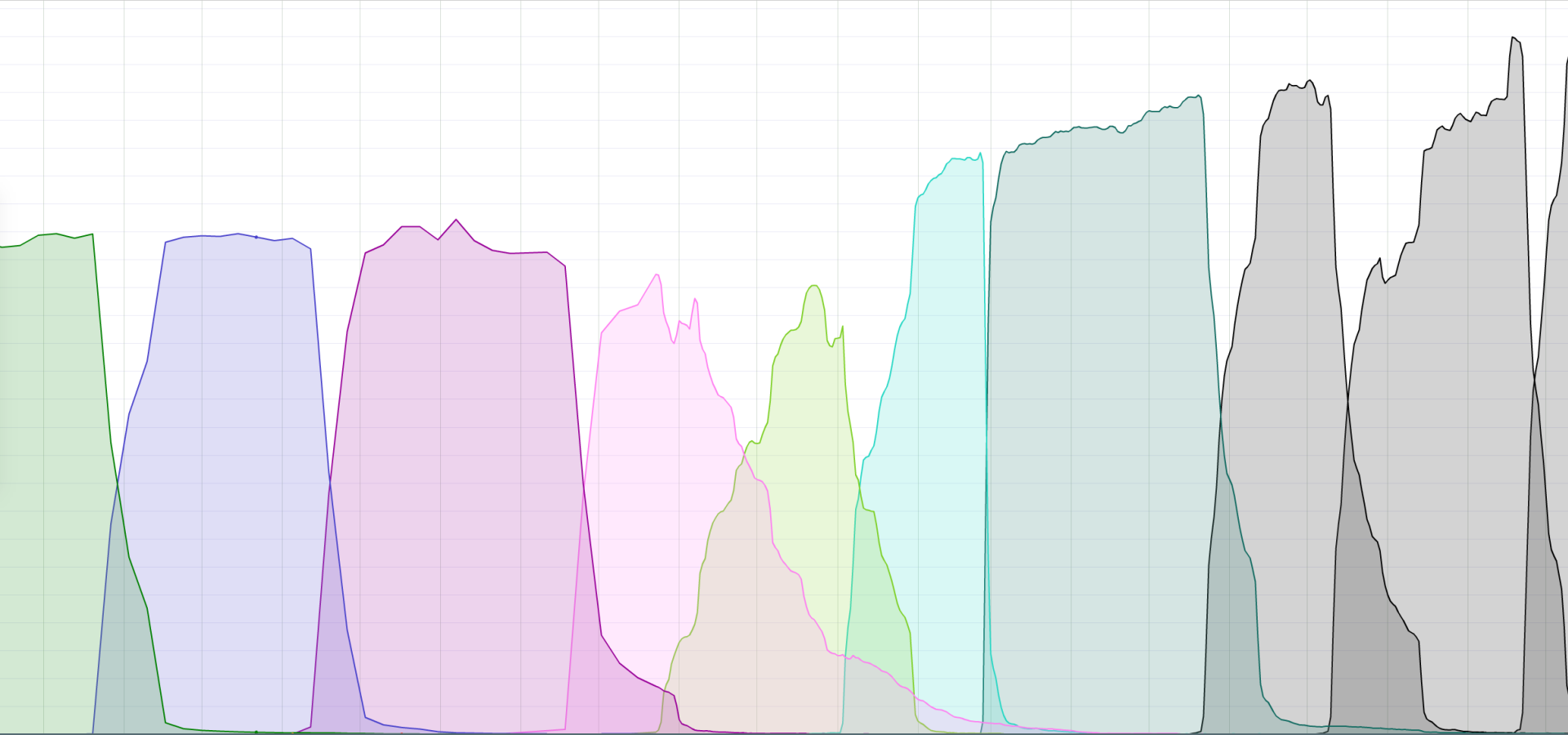
Both vulnerabilities were disclosed by Googlers

Mistakes

goto fail (CVE-2014-1266)

- affected SecureTransport on iOS before 7.0.6 and OS X before 10.9.2
- erroneous “goto” statement circumvented signature verification
- allowing impersonation of any server unnoticed
- Safari (not Chrome or Firefox), Mail, App Store, etc vulnerable to MITM!
- copy paste error, merge conflict, aliens?!
- patch for iOS on Friday, no patch for OS X until the following week.

```
@@ -627,6 +628,7 @@
    goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
+   goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
```



OS X patching this year

Even more mistakes

- cURL on Mavericks: curl.haxx.se/mail/archive-2013-10/0036.html
 - TLS handled by Secure Transport, no longer OpenSSL
 - new engine supports TLS 1.1 and 1.2, new cipher suites not supported by the old version of OpenSSL shipped in OS X
 - but cacert, capath, crlf, ciphers options silently ignored
 - requires certificates and key material to be stored in Keychain

```
vch ~ otool -L /usr/bin/curl
/usr/bin/curl:
    /usr/lib/libcurl.4.dylib (compatibility version 7.0.0, current version 8.0.0)
    /usr/lib/libbz.1.dylib (compatibility version 1.0.0, current version 1.2.5)
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1213.0.0)
vch ~ otool -L /usr/lib/libcurl.4.dylib
/usr/lib/libcurl.4.dylib:
    /usr/lib/libcurl.4.dylib (compatibility version 7.0.0, current version 8.0.0)
    /System/Library/Frameworks/Security.framework/Versions/A/Security (compatibility version 1.0.0, current version 57031.1.27)
    /System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation (compatibility version 150.0.0, current version 1151.14.0)
    /System/Library/Frameworks/LDAP.framework/Versions/A/LDAP (compatibility version 1.0.0, current version 2.4.0)
    /System/Library/Frameworks/Kerberos.framework/Versions/A/Kerberos (compatibility version 5.0.0, current version 6.0.0)
    /usr/lib/libbz.1.dylib (compatibility version 1.0.0, current version 1.2.5)
    /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1213.0.0)
```

Is my personal information safe?

Yes! Safeway e-Auction uses 1024 bit nanotech encryption. It's the same stuff NASA uses.



Rest assured that when you shop with us your details are secure

128 BIT SSL SECURE SITE
100% Safe and Secure Processing



Verified Secure



Credit Card Protection

Card Check

Has your credit card number been **STOLEN** on the Internet?

/

card number expires



TLS does not equal security

Parting words

- Easy to deploy and easy to make mistakes
 - it just works ... until it doesn't
- No way around ultimately having to implicitly trust somebody
 - so trust, but verify
- Cryptography is easily defeated by bad implementations of good math
- Read the documentation
 - RFC 5246 (TLS 1.2), RFC 3280, RFC 5280
 - `man {openssl,rsa,dsa,req,ca,x509}` to start
 - Apple's Cryptographic Services Guide in the developer library
- Crypto must be audited and tested for years before considered safe
 - if you're typing 'RSA' or 'AES' in code, STOP and talk to a cryptographer
- Defense in depth

Plan B

- Host remediation program
- Securely download DMGs and install PKGs
- Written in Objective-C to be bulletproof
- Connection only trusts pinned certificates
- Uses client certificate authentication
- Source available today
 - <https://github.com/google/macops>
- Security workshop this afternoon

Thanks!

vch@google.com