Networks IV: SSL/TLS, Certificates, and Heartbleed

CS 1660: Introduction to Computer Systems Security

(recap) Linux Firewall Stateful example

Iptables block ping requests (ICMP echo-requests) from a specific IP **after 3 pings in 20 seconds**

- **automatically reset** after 20 seconds of no activity from that IP

# 1. Track each incoming ping (ICMP echo-request)

- iptables -A INPUT -p icmp --icmp-type echo-request -m recent --name ping_limit --set

# 2. Drop pings if the IP sent more than 3 in the past 20 seconds

- iptables -A INPUT -p icmp --icmp-type echo-request -m recent --name ping_limit --update --seconds 20 --hitcount 4 -j REJECT   (or DROP)

Iptables rule order is extremely important

- The firewall processes rules sequentially and stops at the first match:
  - Once a packet matches a rule, the corresponding action is applied and no further rules are evaluated.

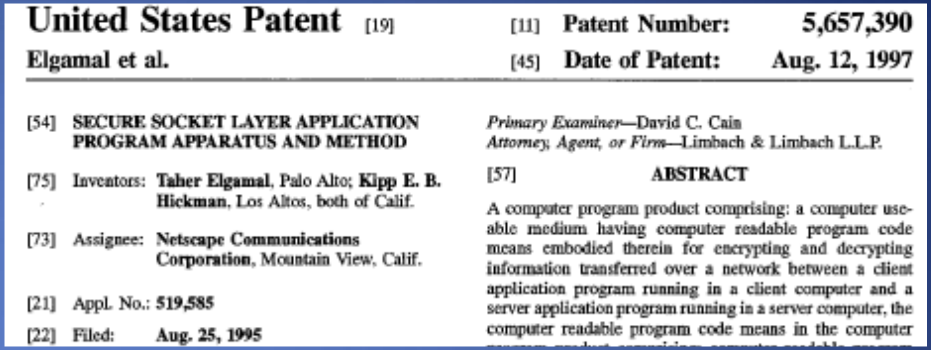
SSL/TLS Overview

Overview

- Why is it important to use HTTPS instead of HTTP?
- What is the difference between SSL and TLS?
- What are the goals of SSL and TLS?

SSL and TLS

- Secure Socket Layer (SSL)
  - Early protocol for securing web connections
  - Developed in the 90s by team led by Taher Elgamal at Netscape
- Transport Layer Security (TLS)
  - Evolution of SSL
  - Standardized by IETF
  - TLS 1.0 RFC 2246 (1999)
  - TLS 1.2 RFC 5246 (2008)
  - TLS 1.3 RFC 8446 (2018)



United States Patent [19] [11] **Patent Number: 5,657,390**

Elgamal et al. [45] **Date of Patent: Aug. 12, 1997**

[54] **SECURE SOCKET LAYER APPLICATION PROGRAM APPARATUS AND METHOD**

[75] Inventors: **Taher Elgamal**, Palo Alto; **Kipp E. B. Hickman**, Los Altos, both of Calif.

[73] Assignee: **Netscape Communications Corporation**, Mountain View, Calif.

[21] Appl. No.: **519,585**

[22] Filed: **Aug. 25, 1995**

Primary Examiner—David C. Cain
Attorney, Agent, or Firm—Limbach & Limbach L.L.P.

[57] **ABSTRACT**

A computer program product comprising: a computer useable medium having computer readable program code means embodied therein for encrypting and decrypting information transferred over a network between a client application program running in a client computer and a server application program running in a server computer, the computer readable program code means in the computer

- Patent issued in 1997
  - … method of encrypting and decrypting information transferred over a network between a client … and a server …

Taher Elgamal

Image source: Alexander Klink via Wikipedia

Goals of SSL/TLS

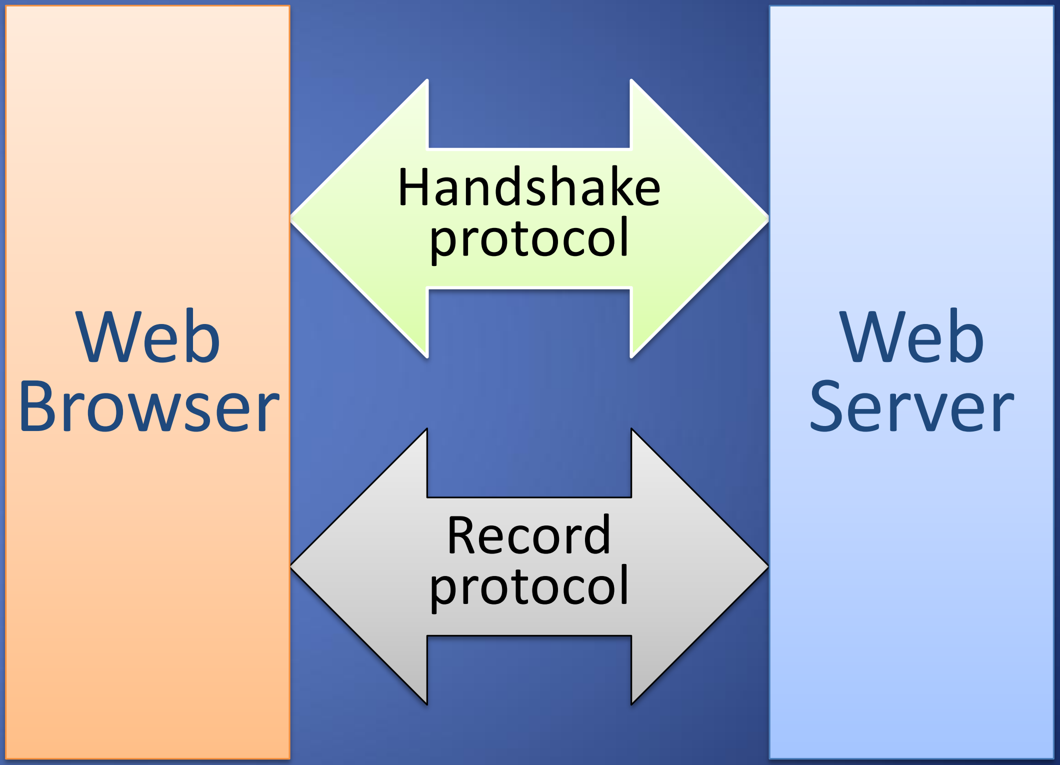
- End-to End Confidentiality
  - Encrypt communication between client and server applications
- End-to-End Integrity
  - Detect corruption of communication between client and server applications
- Required server Authentication
  - Identity of server always proved to client
- Optional client authentication
  - Identity of client optionally proved to server
- Modular deployment
  - Intermediate layer between application and transport layers
  - Handles encryption, integrity, and authentication on behalf of client and server applications

TLS Building Blocks

	Confidentiality	Integrity	Authentication
Setup	Public-key encryption (e.g, RSA)	Public-key digital signature (e.g., RSA)	Public-key digital signature (e.g., RSA)
Data transmission	Symmetric encryption (e.g., AES)	Cryptographic hashing (e.g., SHA256)	

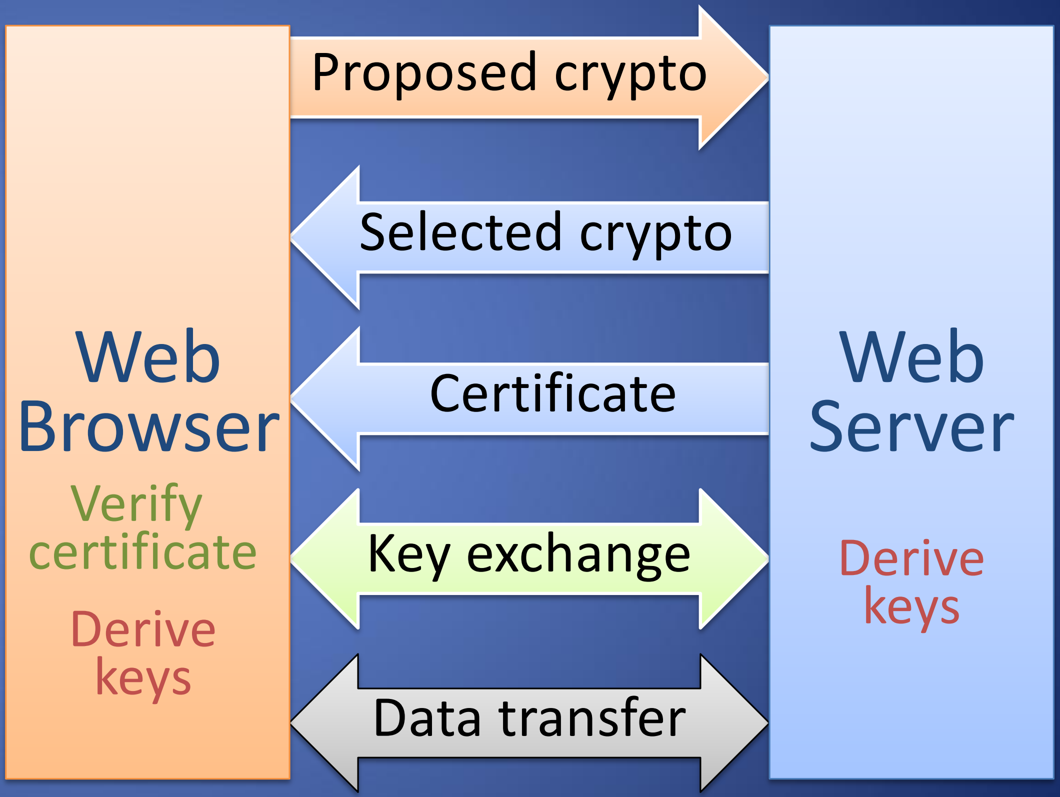
TLS Overview

- Handshake protocol
  - Client authenticates server
  - [Server authenticates client]
  - Client and server agree on crypto algorithms
  - Client and server establish session keys
- Record protocol
  - Encrypt and add integrity protection before sending data
  - Verify integrity and decrypt after receiving data



TLS Overview

- Browser sends supported crypto algorithms (aka cipher suite)
- Server picks strongest algorithms it supports
- Server sends certificate (chain)
- Client verifies certificate (chain)
- Client and server agree on secret value by exchanging messages
- Secret value is used to derive keys for symmetric encryption and hash-based authentication of subsequent data transfer



Example of Cipher Suite

TLS_RSA_WITH_AES_128_GCM_SHA256

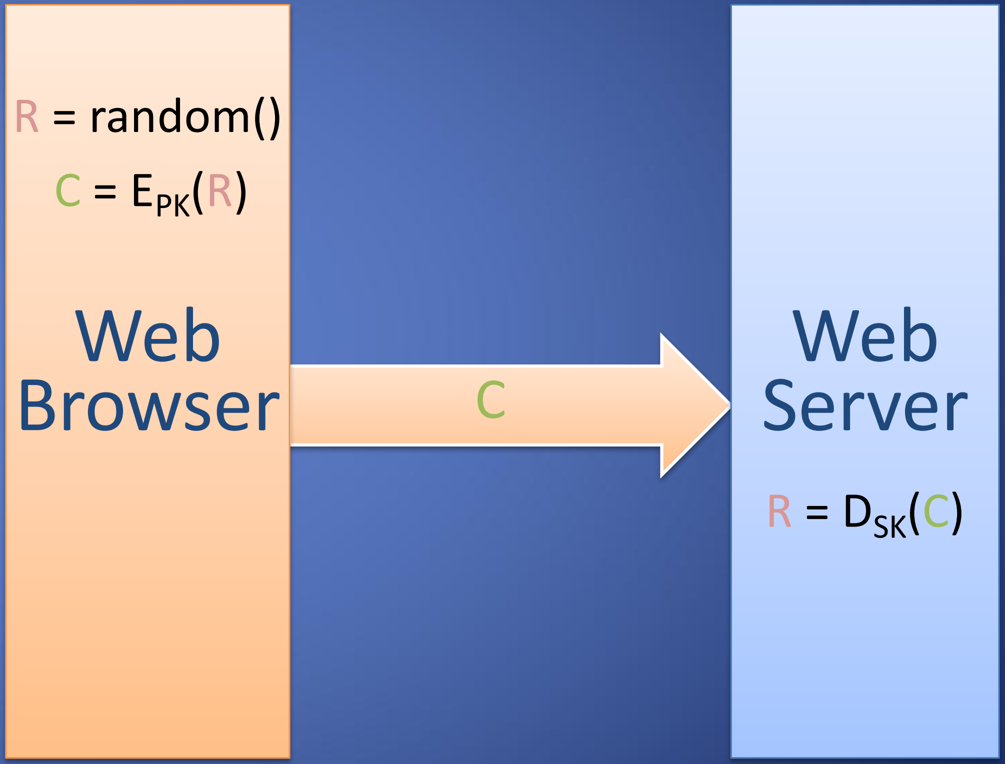
- **TLS** defines the protocol
- **RSA** specifies the key exchange algorithm
- **AES_128_GCM** indicates the cipher being used to encrypt the message stream
- **SHA256** identifies the hash algorithm used to authenticate messages

SSL/TLS analysis with Wireshark
https://tls12.xargs.org/
First part Cipher suite selection

Key Exchange and Forward Secrecy

Basic Key Exchange

- Called RSA key exchange for historical reasons
- Client generates random secret value R
- Client encrypts R with public key, PK, of server: $C = E_{PK}(R)$
- Client sends C to server
- Server decrypts C with private key, SK, of server: $R = D_{SK}(C)$

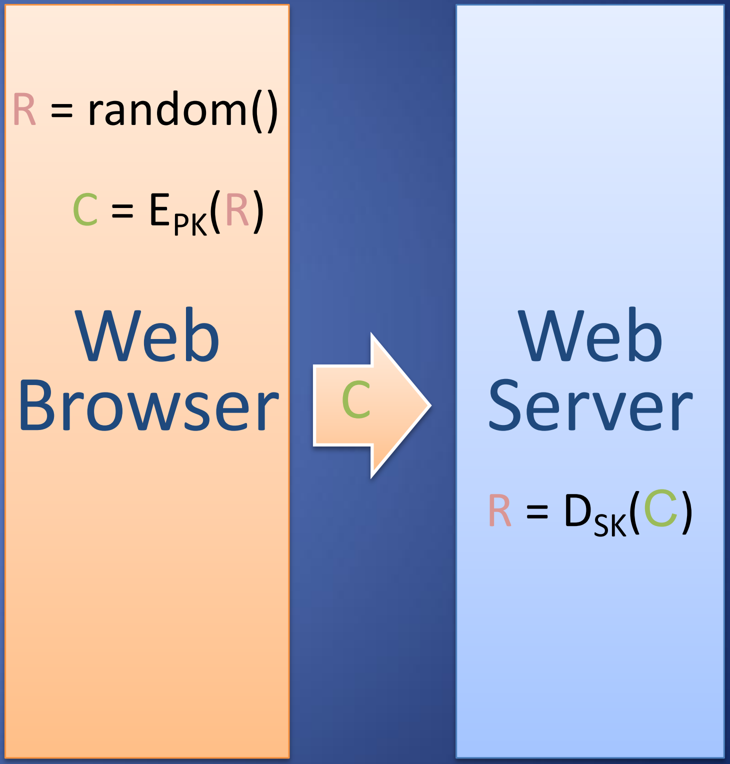


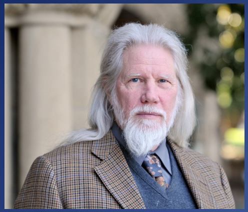
Forward Secrecy (FS) or Perfect Forward Secrecy (PFS)

- General concept
    - Compromise of public-key encryption private keys does not break confidentiality of past encrypted messages
- Forward secrecy in the context of TLS
    - Compromise of server's private key (associated with public key in certificate) does not break confidentiality of past TLS sessions
- TLS with basic key exchange (aka RSA key exchange) does not provide forward secrecy

Forward Secrecy

- Compromise of public-key encryption private keys does not break confidentiality of past messages
- TLS with basic key exchange does not provide forward secrecy
  - Attacker eavesdrop and stores all TLS communication
  - If server's private key, SK, is compromised, attacker recovers secret value R in key exchange and derives from R encryption key used in subsequent encrypted TLS communication

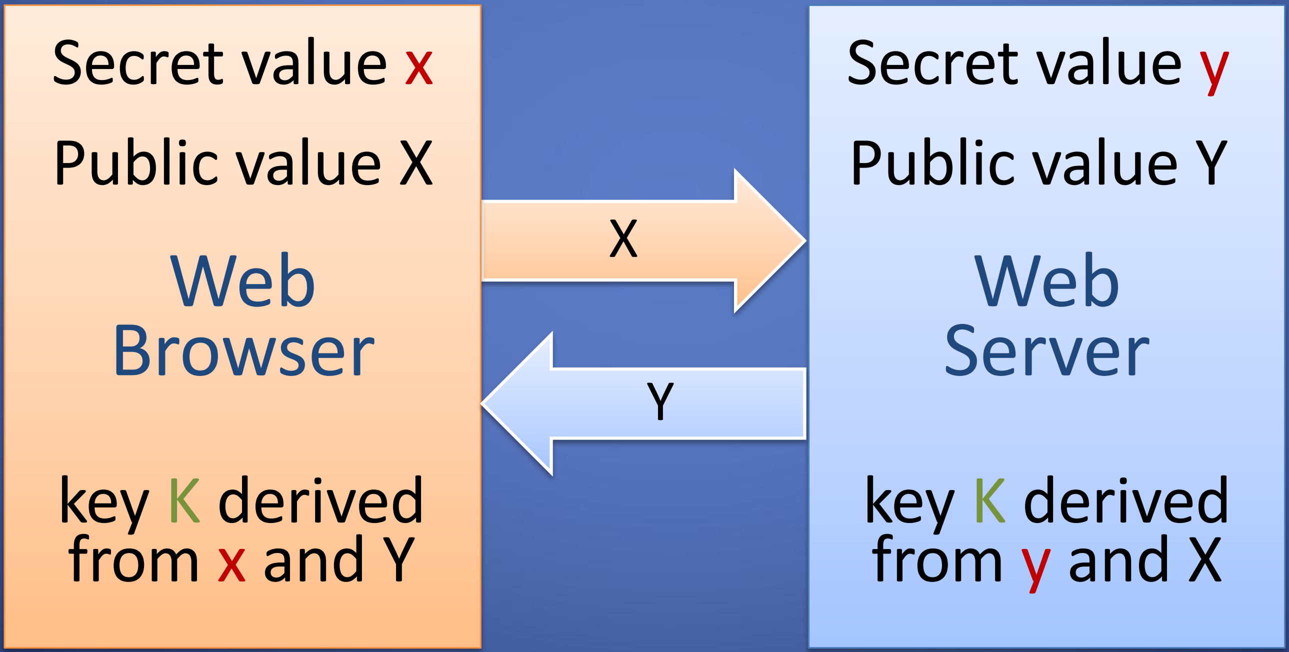


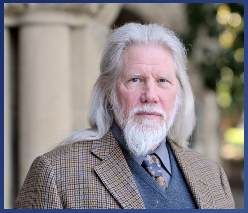

Source: ACM

Diffie Hellman Key Exchange

Source: ACM

Achieves forward secrecy



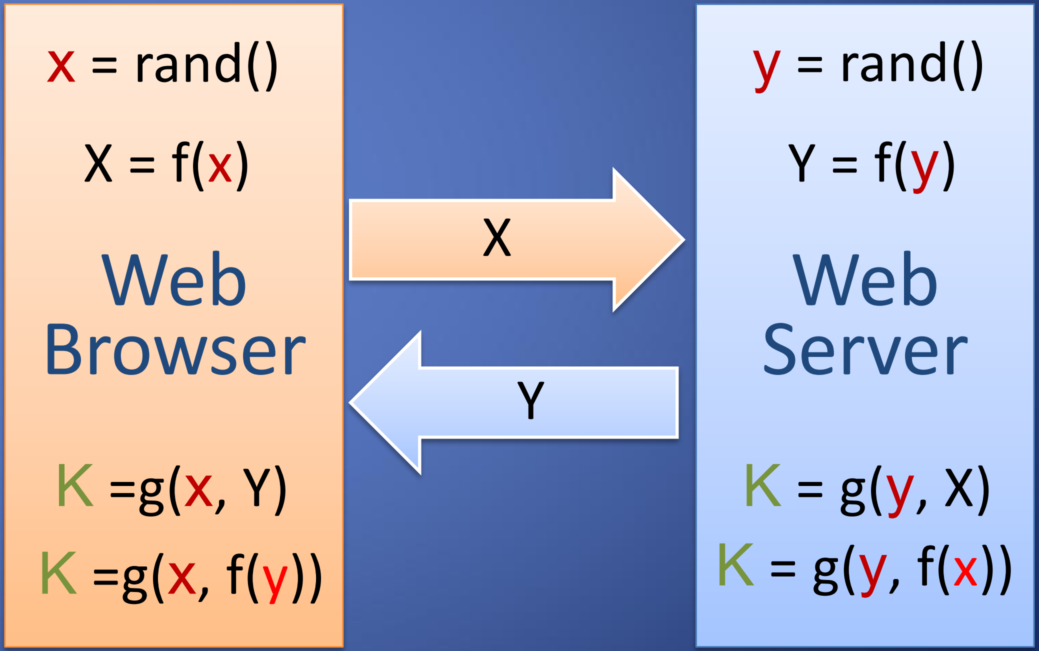

Source: ACM

Diffie Hellman Key Exchange

Source: ACM

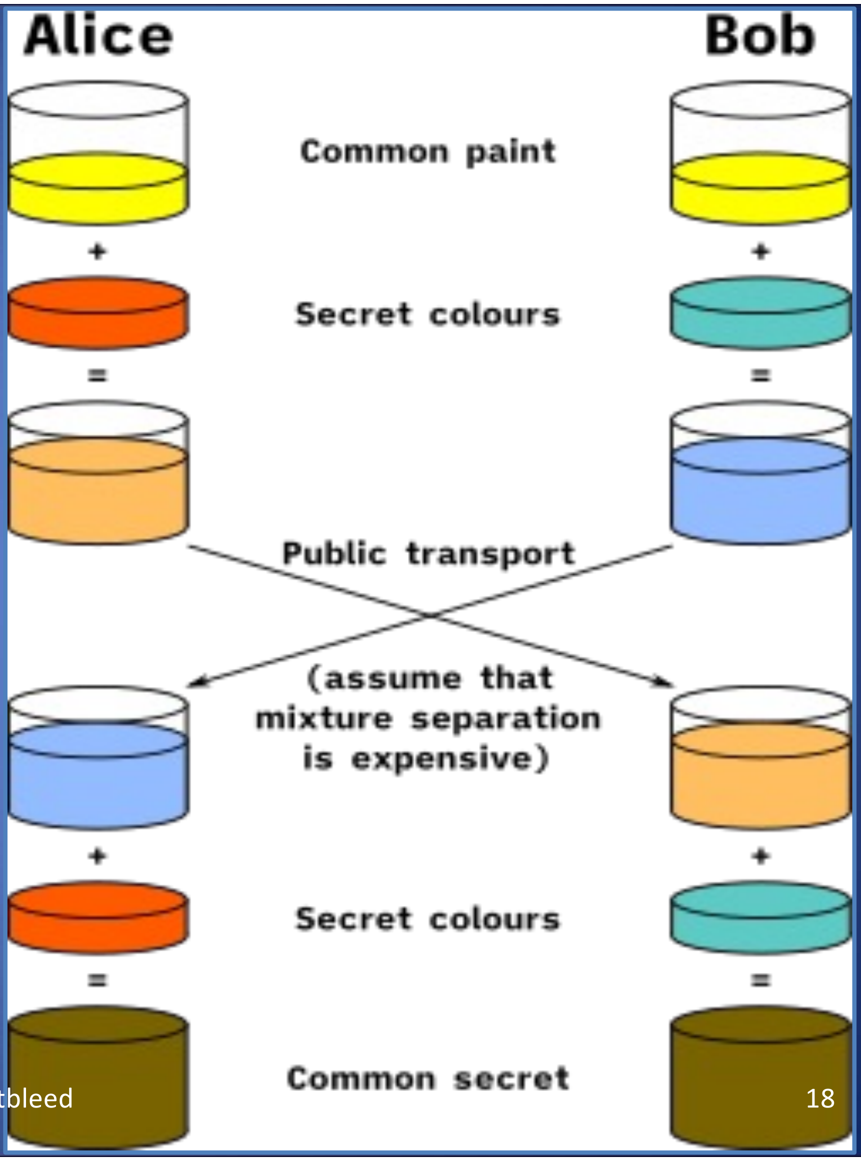
Achieves forward secrecy

- Client randomly generates x and derives public value X
- Server randomly generates y and derives public value Y
- Client and server exchange values X and Y
- Client derives key K from x and Y
- Server derives key K from y and X
- Attacker who captures X and Y cannot reconstruct K



Diffie Hellman Key Exchange

- An intuitive example
- The function is mixing colors
- We assume that is practically impossible to separate e mixture



Source: upload.wikimedia.org/wikipedia/commons/4/46/Diffie-Hellman_Key_Exchange.svg

Modular Arithmetic

- mod function
  - x mod n is the remainder of the division of x by n
  - x mod n has values between 0 and $n - 1$
- Examples
  - $14 \bmod 13 = 1$
  - $29 \bmod 13 = 3$
  - $13 \bmod 13 = 0$
  - $-1 \bmod 13 = 12$

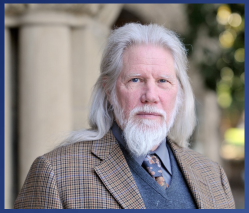
- Modular arithmetic has properties similar to standard arithmetic
  - E.g., associative and commutative
- Several cryptographic functions are based on modular arithmetic
  - E.g., RSA cryptosystem

Power of a Power Property

- Standard arithmetic
  - $a^{xy} = (a^x)^y = (a^y)^x$
  - Example: $2^{2 \cdot 3} = (2^2)^3 = (2^3)^2 = 64$
- Modular arithmetic
  - $a^{xy} \bmod n = (a^x)^y \bmod n = (a^y)^x \bmod n$

Discrete Logarithm Problem

- Modular power and logarithm
  - $y = a^x \bmod n$
  - Assume a and n are fixed public parameters
  - x is the logarithm of y in base a modulo n
- Modular power is easy
  - There is an efficient algorithm to compute y given x
- Modular logarithm is hard
  - No efficient algorithm is known to compute x given y


Source: ACM

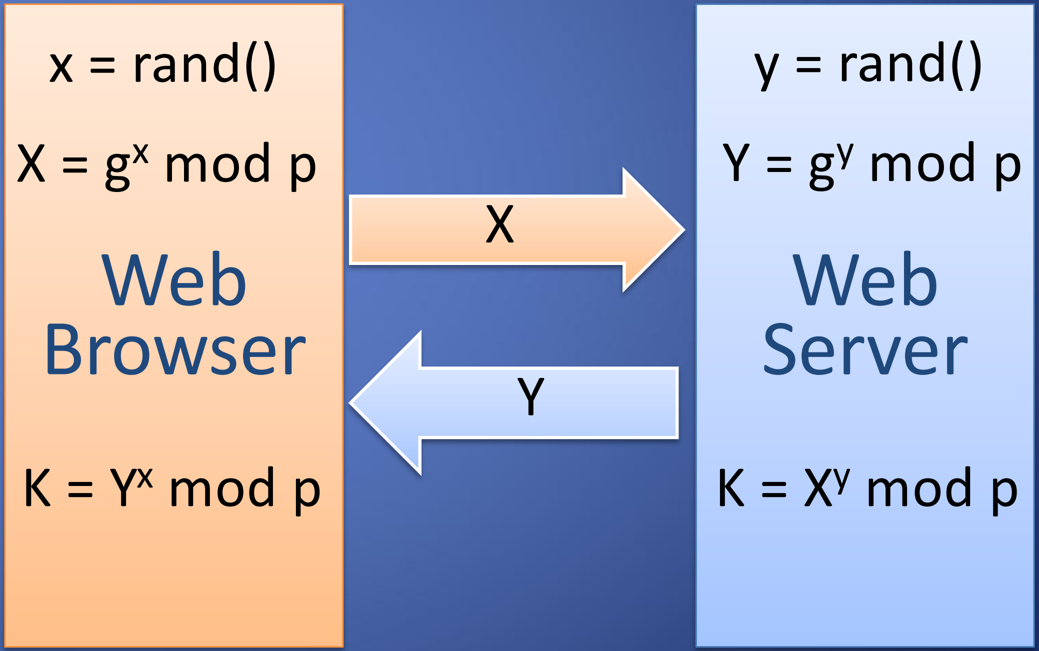
DH Key Exchange Details

Source: ACM

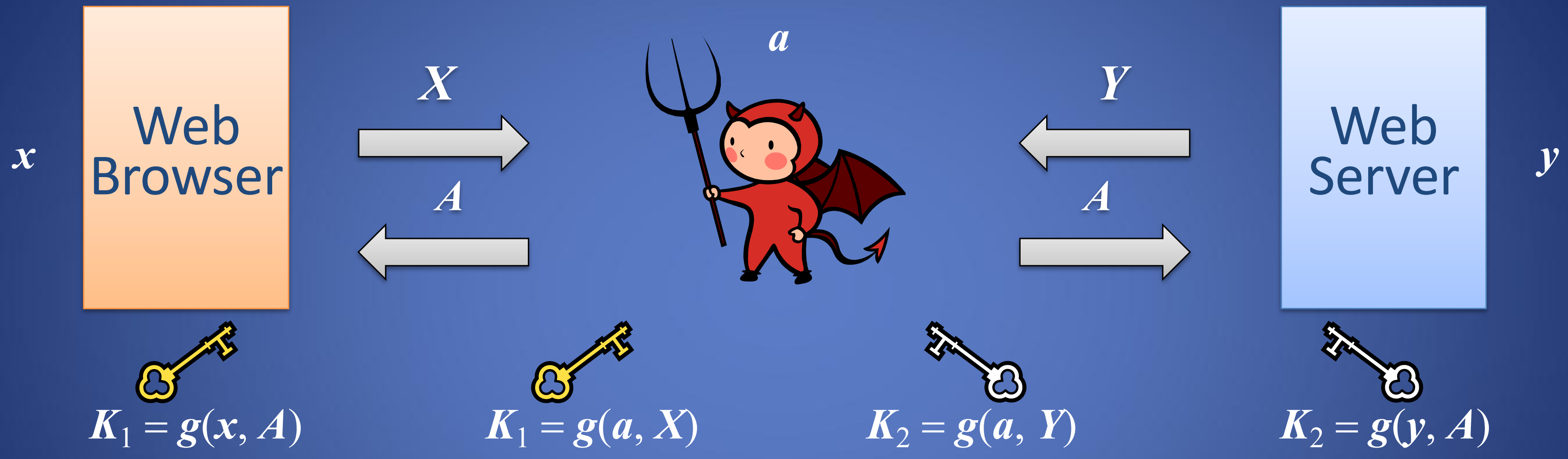
Achieves forward secrecy

- Public parameters: prime p and generator g
- Client generates random x and computes $X = g^x \bmod p$
- Server generates random y and computes $Y = g^y \bmod p$
- Client sends X to server
- Server sends Y to client
- Client and server compute

$$K = g^{xy} \bmod p$$



Injection Attack



An authentication problem the solution is with the use of certificates:

- Browser and server send signed X and Y respectively
- Requires each to know the public key of the other
- Optional for browser as it usually does not have certificate

BREAK!

RESTART!

5 4 3 2 1

Certificates and PKI

TLS Goals

- Confidentiality
- Integrity
- Authentication

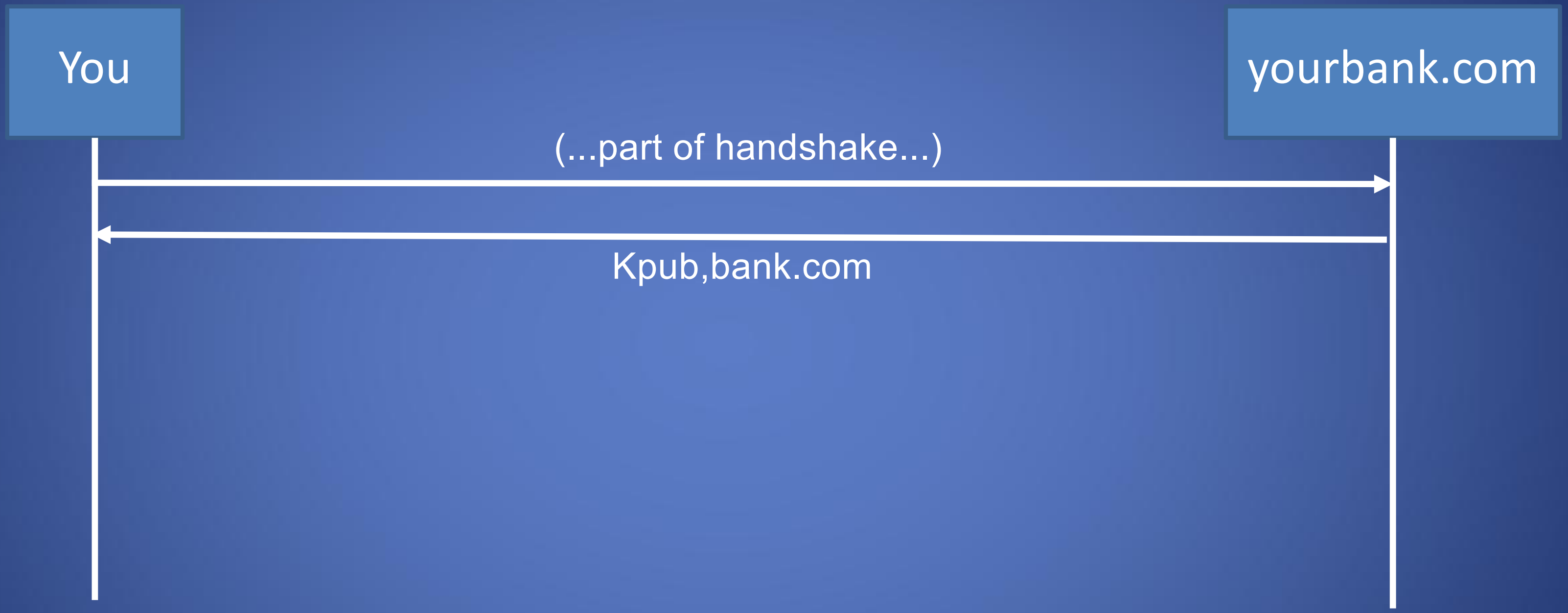
TLS Goals

Authentication: verifying that the entity on the other end of the connection is who they claim to be

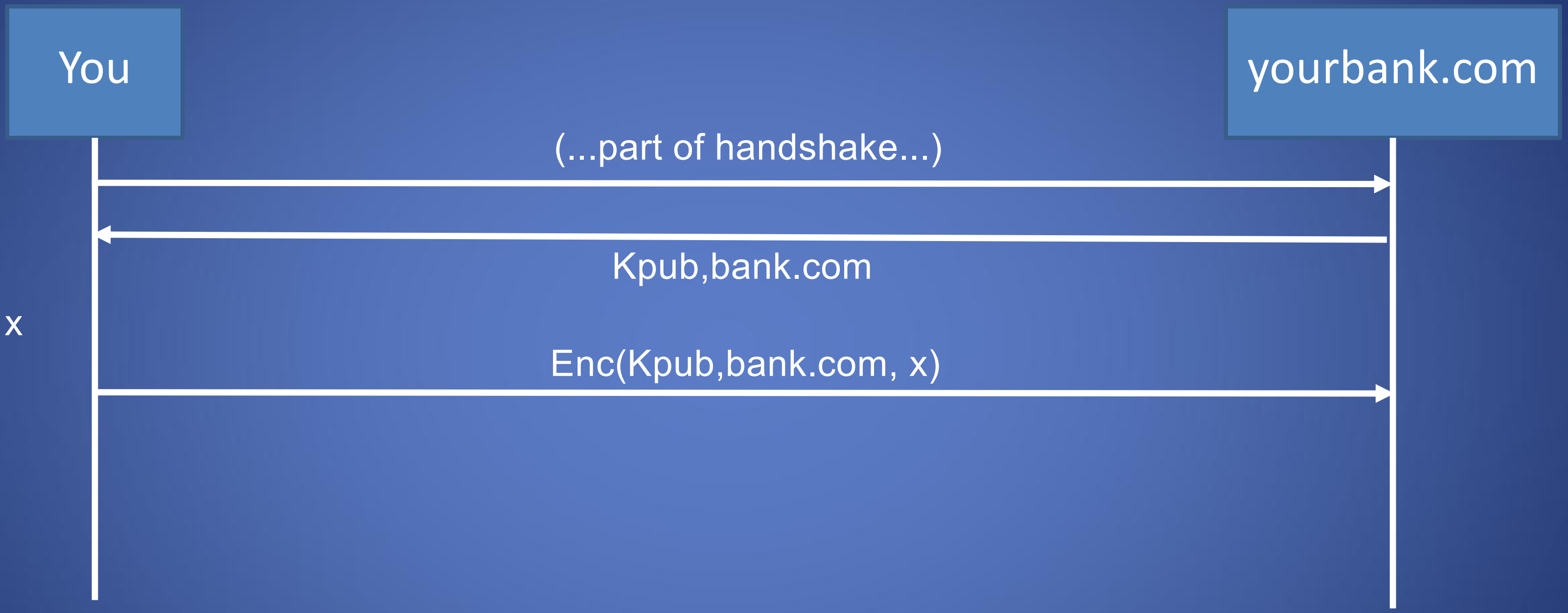
- Technical aspects: crypto
- Social aspects
  - How to distribute keys to entities
  - What to do when things go wrong

TLS: relies on Public Key Infrastructure (PKI) via certificates

The Challenge

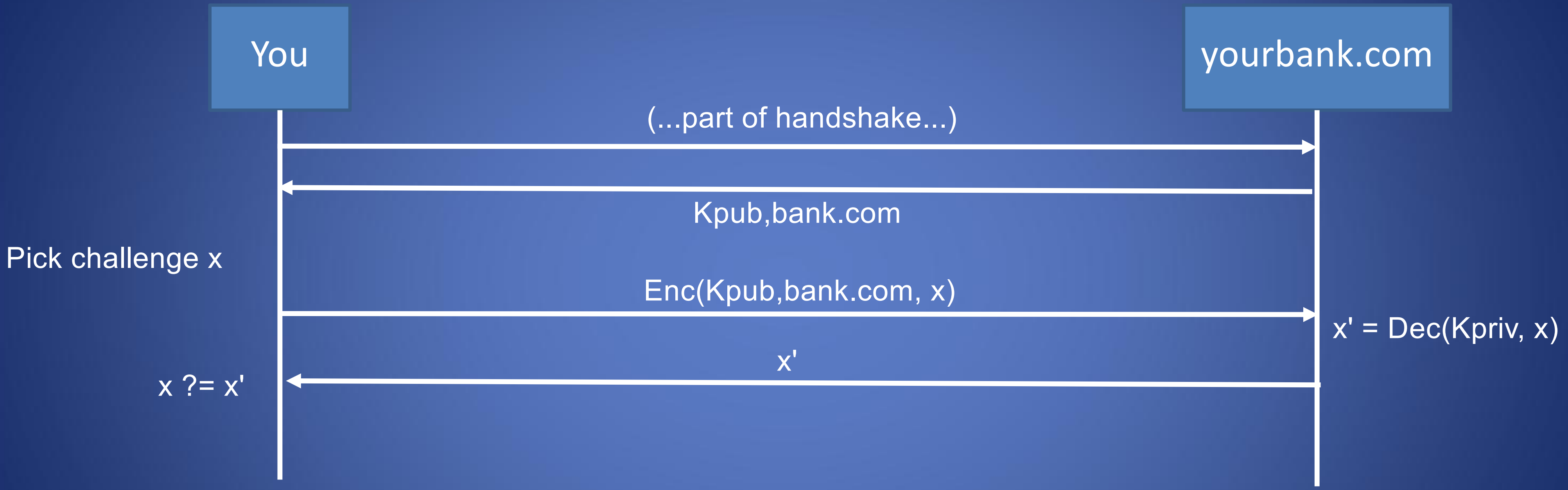


The Challenge



Pick challenge x

The Challenge



<u>What does this prove?</u>

Authentication challenges

- Challenge proves that the server at yourbank.com holds Kpriv
- Does NOT prove the server belongs to YourBank, the real-life bank that holds your money

"But I'm visiting yourbank.com!"

- DNS can be spoofed
- Possible active network attacker (redirecting your IP traffic to malicious server)
- Domain names can expire and be re-registered...

Problem: distributing trust

How can we trust Kpub is Your Bank's public key?

Problem: Trust distribution

- Hard to verify real-world identities
- Hard to scale to the whole Internet

Different protocols have different mechanisms

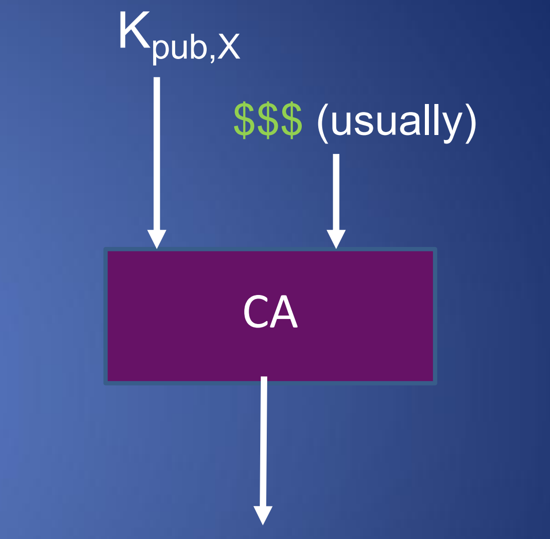
=> TLS (and others): Public Key Infrastructure (PKI) with certificates

PKI: The main idea

Public keys managed by Certificate Authorities (CAs)

- Everyone knows public key for some root CAs
  - Pre-installed into browser/OS
- If X wants a public key, request from CA
  - CA validates X's identity, then signs X's public key
  - Generates certificate
- Client can verify $K_{pub,X}$ from CA's signature:

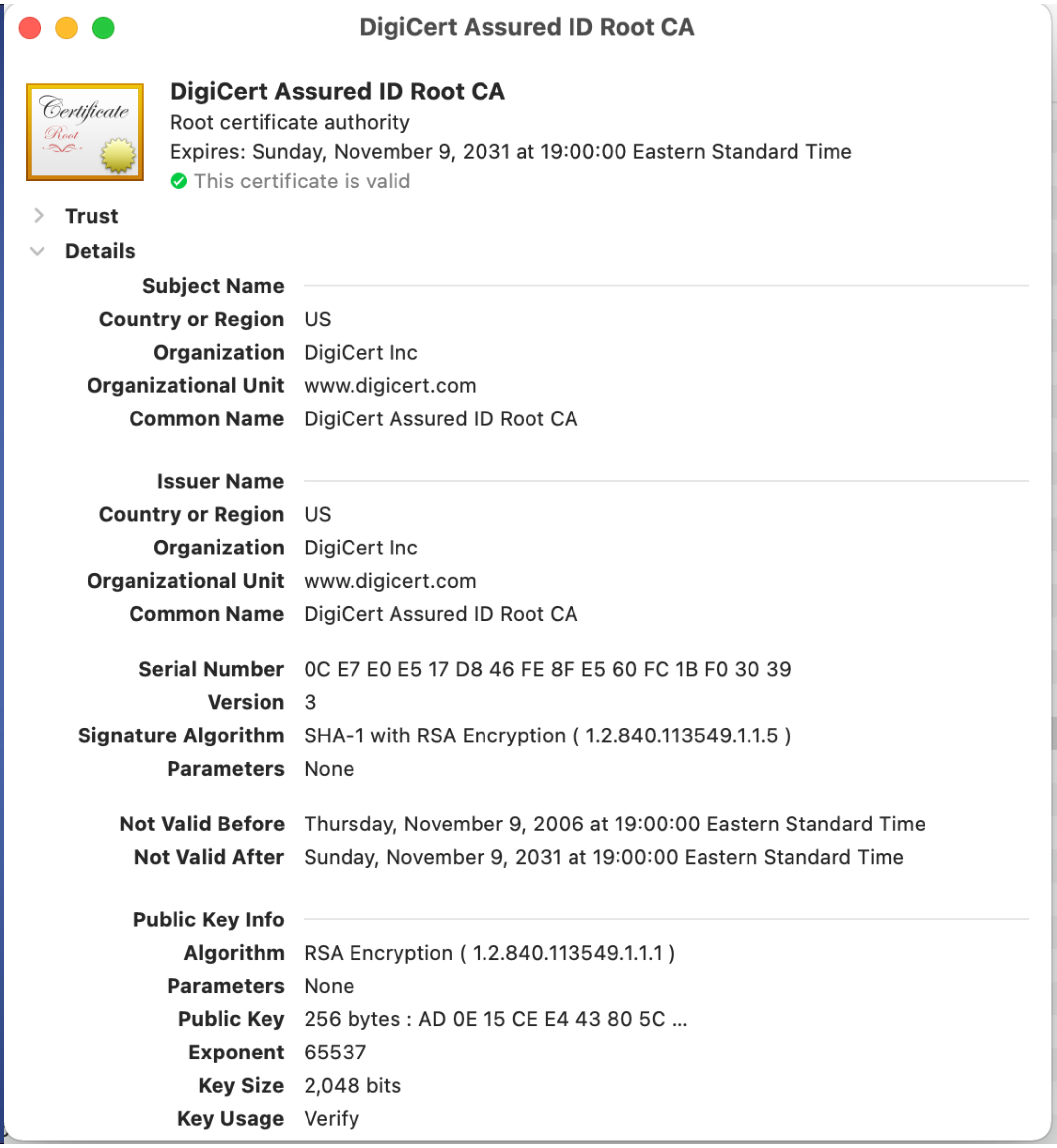
  Verify($K_{pub,CA}$ Cert) => True/False



s = Sign($K_{priv,CA}$, {$K_{pub,X}$, ... })

Cert = {$K_{pub,X}$, metadata, s}

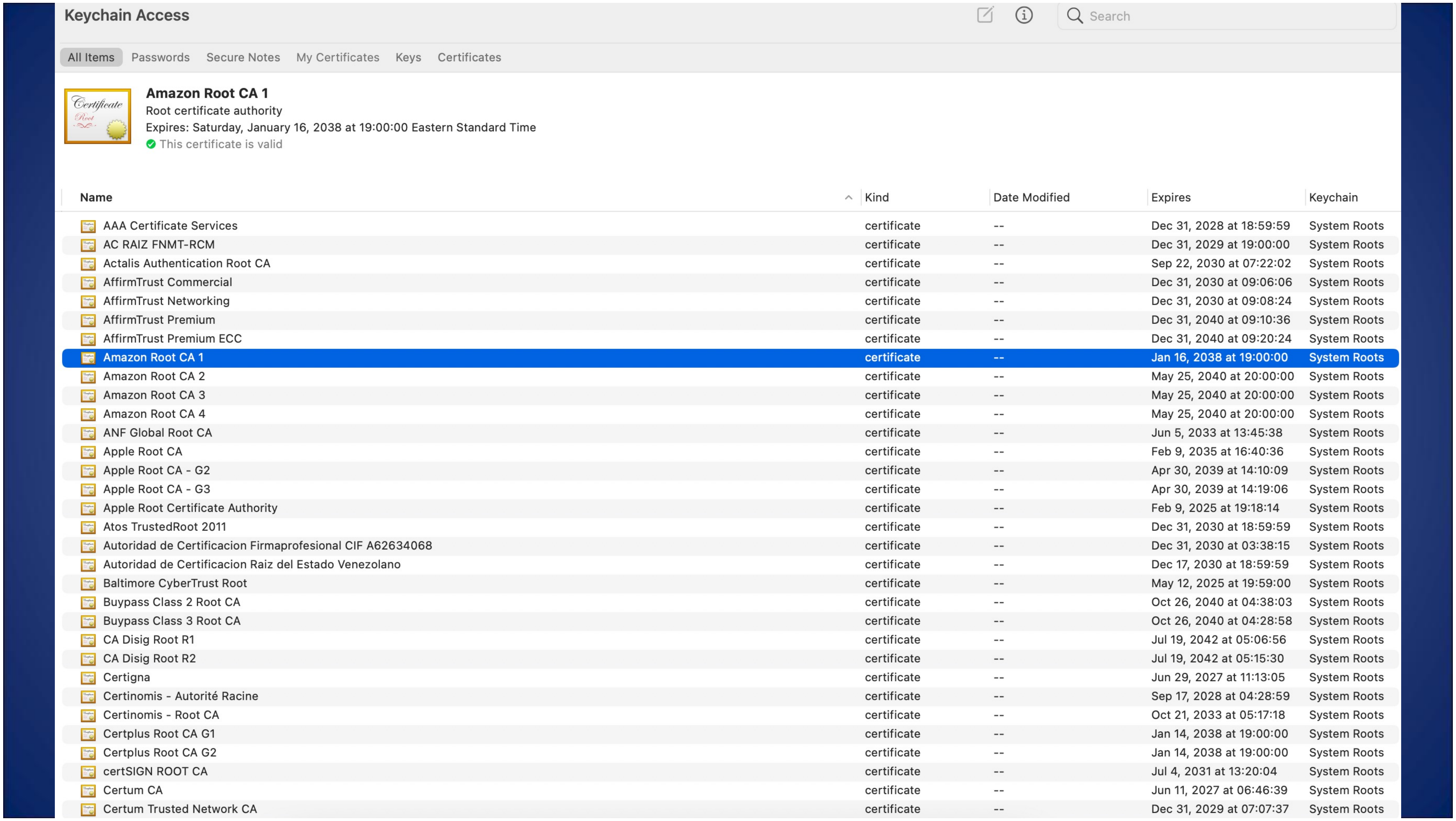
=> Delegates trust for individual entity to a more trusted authority


DigiCert Assured ID Root CA

DigiCert Assured ID Root CA
Root certificate authority
Expires: Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time
✓ This certificate is valid

Trust
Details

Subject Name	
Country or Region	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert Assured ID Root CA
Issuer Name	
Country or Region	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert Assured ID Root CA
Serial Number	0C E7 E0 E5 17 D8 46 FE 8F E5 60 FC 1B F0 30 39
Version	3
Signature Algorithm	SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)
Parameters	None
Not Valid Before	Thursday, November 9, 2006 at 19:00:00 Eastern Standard Time
Not Valid After	Sunday, November 9, 2031 at 19:00:00 Eastern Standard Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes : AD 0E 15 CE E4 43 80 5C ...
Exponent	65537
Key Size	2,048 bits
Key Usage	Verify



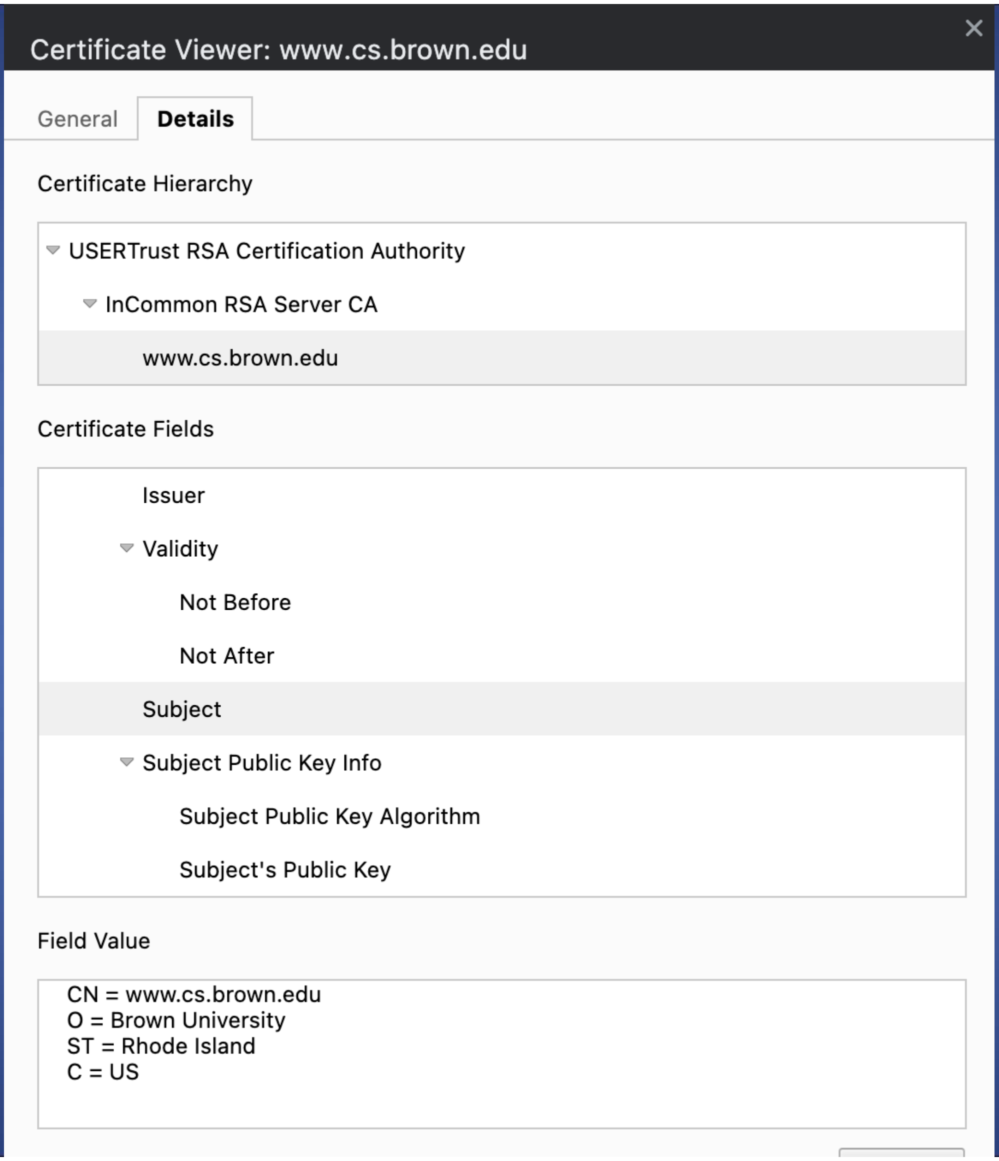
Amazon Root CA 1
Root certificate authority
Expires: Saturday, January 16, 2038 at 19:00:00 Eastern Standard Time
This certificate is valid

Name	Kind	Date Modified	Expires	Keychain
AAA Certificate Services	certificate	--	Dec 31, 2028 at 18:59:59	System Roots
AC RAIZ FNMT-RCM	certificate	--	Dec 31, 2029 at 19:00:00	System Roots
Actalis Authentication Root CA	certificate	--	Sep 22, 2030 at 07:22:02	System Roots
AffirmTrust Commercial	certificate	--	Dec 31, 2030 at 09:06:06	System Roots
AffirmTrust Networking	certificate	--	Dec 31, 2030 at 09:08:24	System Roots
AffirmTrust Premium	certificate	--	Dec 31, 2040 at 09:10:36	System Roots
AffirmTrust Premium ECC	certificate	--	Dec 31, 2040 at 09:20:24	System Roots
Amazon Root CA 1	certificate	--	Jan 16, 2038 at 19:00:00	System Roots
Amazon Root CA 2	certificate	--	May 25, 2040 at 20:00:00	System Roots
Amazon Root CA 3	certificate	--	May 25, 2040 at 20:00:00	System Roots
Amazon Root CA 4	certificate	--	May 25, 2040 at 20:00:00	System Roots
ANF Global Root CA	certificate	--	Jun 5, 2033 at 13:45:38	System Roots
Apple Root CA	certificate	--	Feb 9, 2035 at 16:40:36	System Roots
Apple Root CA - G2	certificate	--	Apr 30, 2039 at 14:10:09	System Roots
Apple Root CA - G3	certificate	--	Apr 30, 2039 at 14:19:06	System Roots
Apple Root Certificate Authority	certificate	--	Feb 9, 2025 at 19:18:14	System Roots
Atos TrustedRoot 2011	certificate	--	Dec 31, 2030 at 18:59:59	System Roots
Autoridad de Certificacion Firmaprofesional CIF A62634068	certificate	--	Dec 31, 2030 at 03:38:15	System Roots
Autoridad de Certificacion Raiz del Estado Venezolano	certificate	--	Dec 17, 2030 at 18:59:59	System Roots
Baltimore CyberTrust Root	certificate	--	May 12, 2025 at 19:59:00	System Roots
Buypass Class 2 Root CA	certificate	--	Oct 26, 2040 at 04:38:03	System Roots
Buypass Class 3 Root CA	certificate	--	Oct 26, 2040 at 04:28:58	System Roots
CA Disig Root R1	certificate	--	Jul 19, 2042 at 05:06:56	System Roots
CA Disig Root R2	certificate	--	Jul 19, 2042 at 05:15:30	System Roots
Certigna	certificate	--	Jun 29, 2027 at 11:13:05	System Roots
Certinomis - Autorité Racine	certificate	--	Sep 17, 2028 at 04:28:59	System Roots
Certinomis - Root CA	certificate	--	Oct 21, 2033 at 05:17:18	System Roots
Certplus Root CA G1	certificate	--	Jan 14, 2038 at 19:00:00	System Roots
Certplus Root CA G2	certificate	--	Jan 14, 2038 at 19:00:00	System Roots
certSIGN ROOT CA	certificate	--	Jul 4, 2031 at 13:20:04	System Roots
Certum CA	certificate	--	Jun 11, 2027 at 06:46:39	System Roots
Certum Trusted Network CA	certificate	--	Dec 31, 2029 at 07:07:37	System Roots

What's in a certificate?

- Public key of entity (eg. yourbank.com)
- Common name: DNS name of server (yourbank.com)
- Contact info for organization
- Validity dates (start date, expire date)
- URL of *revocation center* to check if key has been revoked

All of this is part of the data signed by the CA
=> Critical to check all parts during TLS startup!



Certificate Viewer: www.cs.brown.edu

Certificate Hierarchy

- USERTrust RSA Certification Authority
  - InCommon RSA Server CA
    - www.cs.brown.edu

Certificate Fields

- Issuer
- Validity
  - Not Before
  - Not After
- Subject
- Subject Public Key Info
  - Subject Public Key Algorithm
  - Subject's Public Key

Field Value

```
CN = www.cs.brown.edu
O = Brown University
ST = Rhode Island
C = US
```

PKI hierarchy

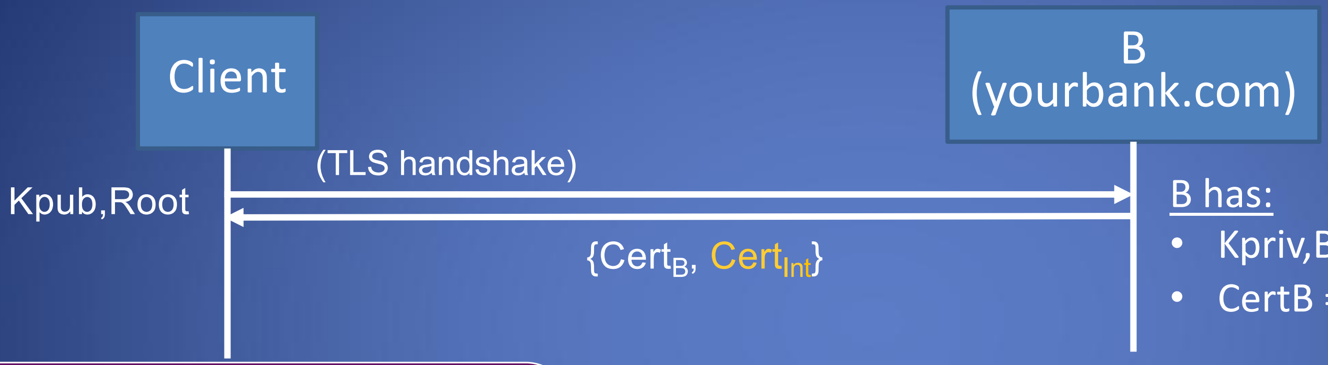
In reality, PKI creates a hierarchy of trust:

- Root CAs: k_{pub} stored in virtually every browser, OS
  - Private keys protected by most stringent security measures (software, hardware, physical)
- Intermediate CAs: k_{pub} signed by root CA
  - Sign certificates for general use (ie, regular websites)
  - Doesn't require same protections as root
- General-use certificates: for a specific webserver

What happens if a root is compromised?

How the hierarchy works

Ex. Server has certificate from Intermediate CA_{Int}

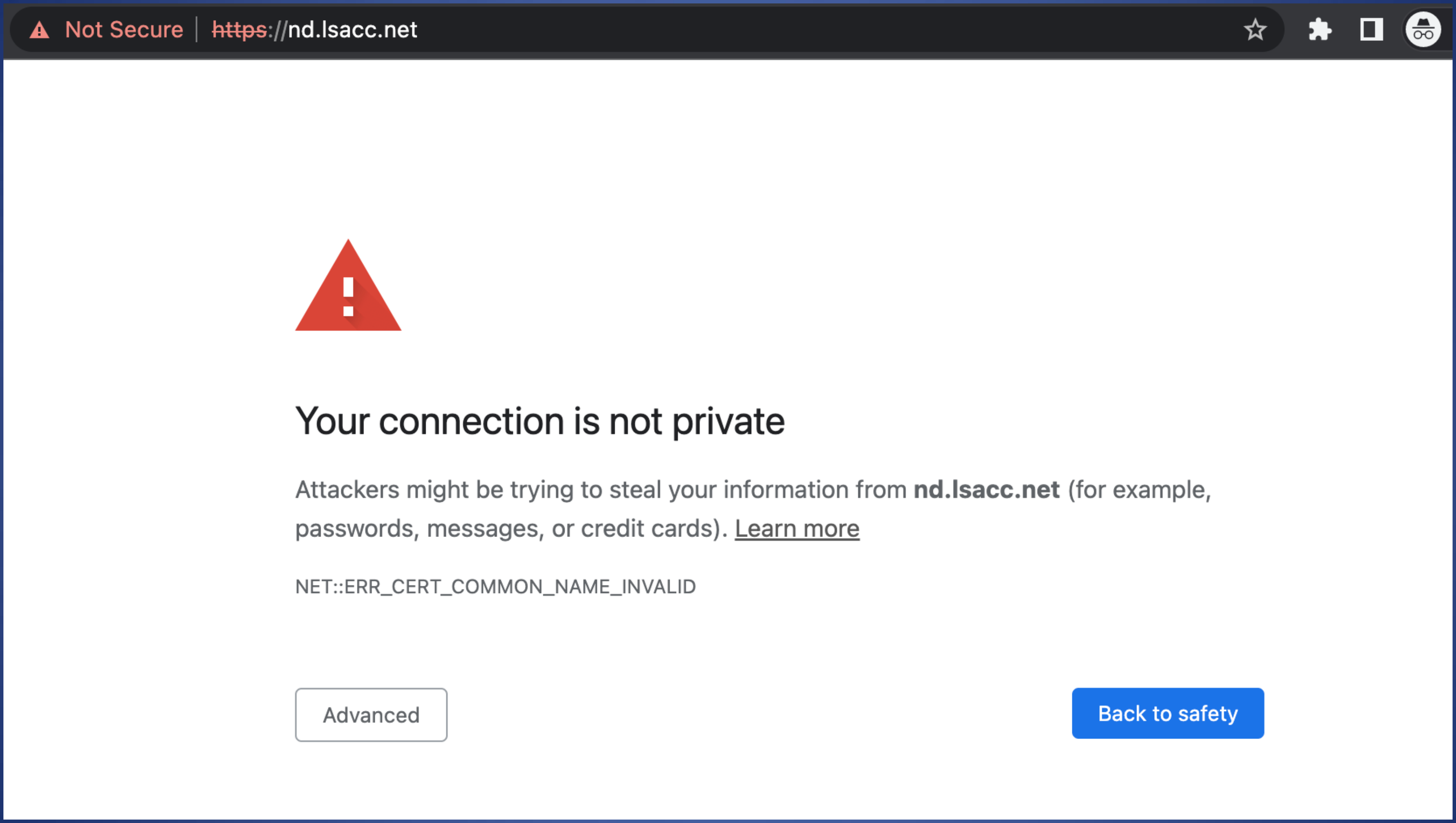


B has:
- Kpriv,B
- CertB = $\{ K_{pub,B}, Sign(K_{pub,B}, K_{priv,Int}), \ldots \}$

Client's workflow:
- Checks metadata ✅
- $Verify(Cert_B, K_{pub,Int})$ ✅
- $Verify(Cert_{Int}, K_{pub,Root})$ ✅

=> To verify integrity, need to verify certificates back to (trusted) root certificate

=> OK if verification passes and metadata correct: 🔒



Not Secure | ~~https~~://nd.lsacc.net

Your connection is not private

Attackers might be trying to steal your information from **nd.lsacc.net** (for example, passwords, messages, or credit cards). Learn more

NET::ERR_CERT_COMMON_NAME_INVALID

Most common TLS errors you might see

- Common name invalid
- Self-signed
- Certificate expired

When is it okay to click "proceed"? What happens if you do?

=> Might occur if webserver configured improperly, or if you're setting up a system

Rogue Certificates?

- In 2011, DigiNotar, a Dutch root certificate authority, was compromised
- The attacker created rogue certificates for popular domains like google.com and yahoo.com
- DigiNotar was distrusted by browsers and filed for bankruptcy
- See the incident investigation report by European Agency for Cybersecurity (ENISA)

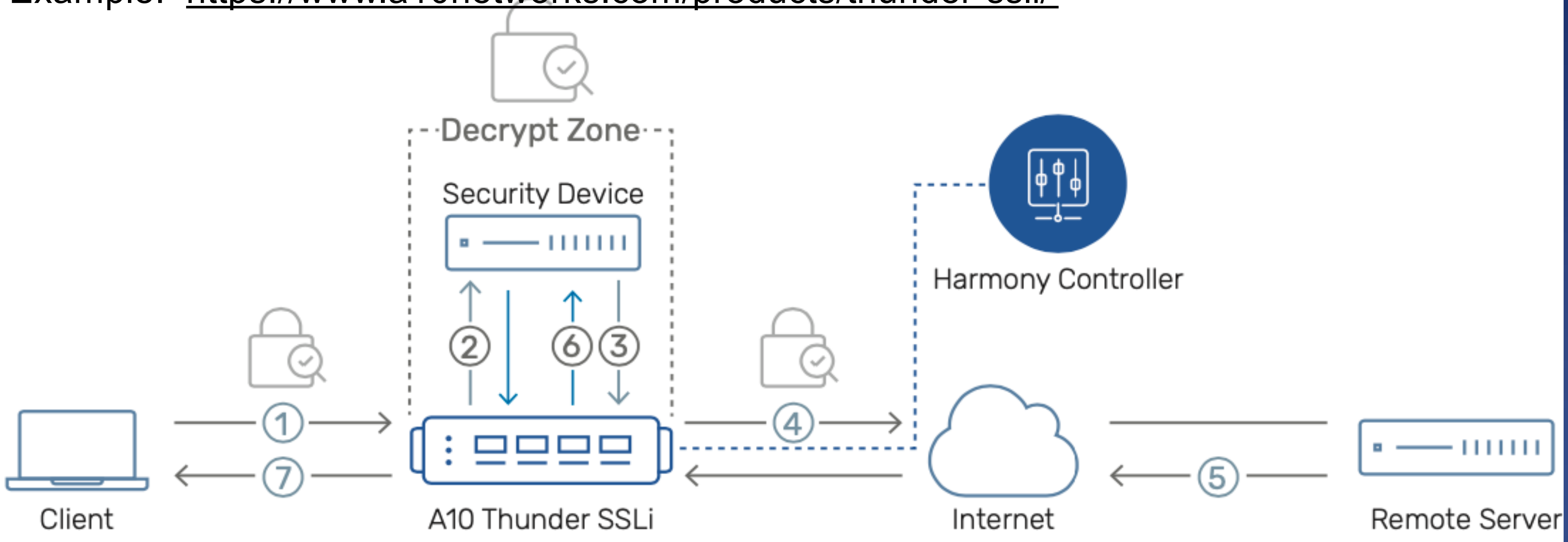
Another issue

- In 2017, Google questioned the certificate issuance policies and practices of Symantec
- Google's Chrome would start distrusting Symantec's certificates unless certain remediation steps were taken
- See back and forth between Ryan Sleevi (Chromium team) and Symantec
- The matter was settled with DigiCert acquiring Symantec's certificate business
- Certificate Transparency is an initiative that helps to mitigate this issue

TLS decryption

What happens when an organization wants to view TLS traffic on its network?

Example: https://www.a10networks.com/products/thunder-ssli/



1. Encrypted traffic from the client is intercepted by Thunder SSLi and decrypted.
2. Thunder SSLi sends the decrypted traffic to a security device, which inspects it in clear-text.
3. The security device, after inspection, sends the traffic back to Thunder SSLi, which intercepts and re-encrypts it.
4. Thunder SSLi sends the re-encrypted traffic to the server.
5. The server processes the request and sends an encrypted response to Thunder SSLi.
6. Thunder SSLi decrypts the response traffic and forwards it to the same security device for inspection.
7. Thunder SSLi receives the traffic from the security device, re-encrypts it and sends it to the client.

View SSL Certificates

- Browser can show certificate chain
- View OS's certificate keystore
  - MacOS: Keychain Access app or through Finder
- Linux tools: openssl
  - E.g., inspect the brown.edu certificate

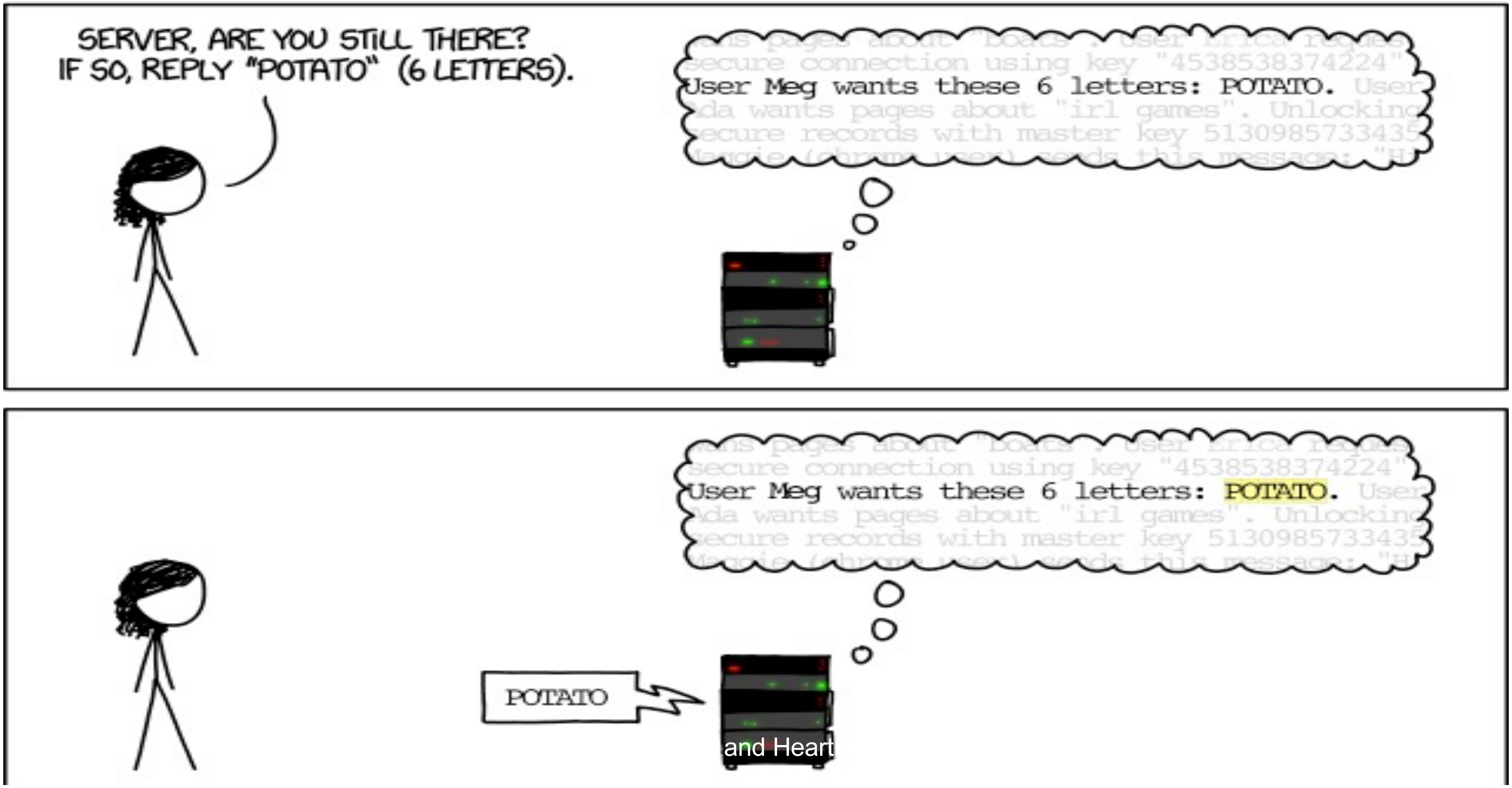
```
openssl s_client -connect cs.brown.edu:443
```

Heart Bleed

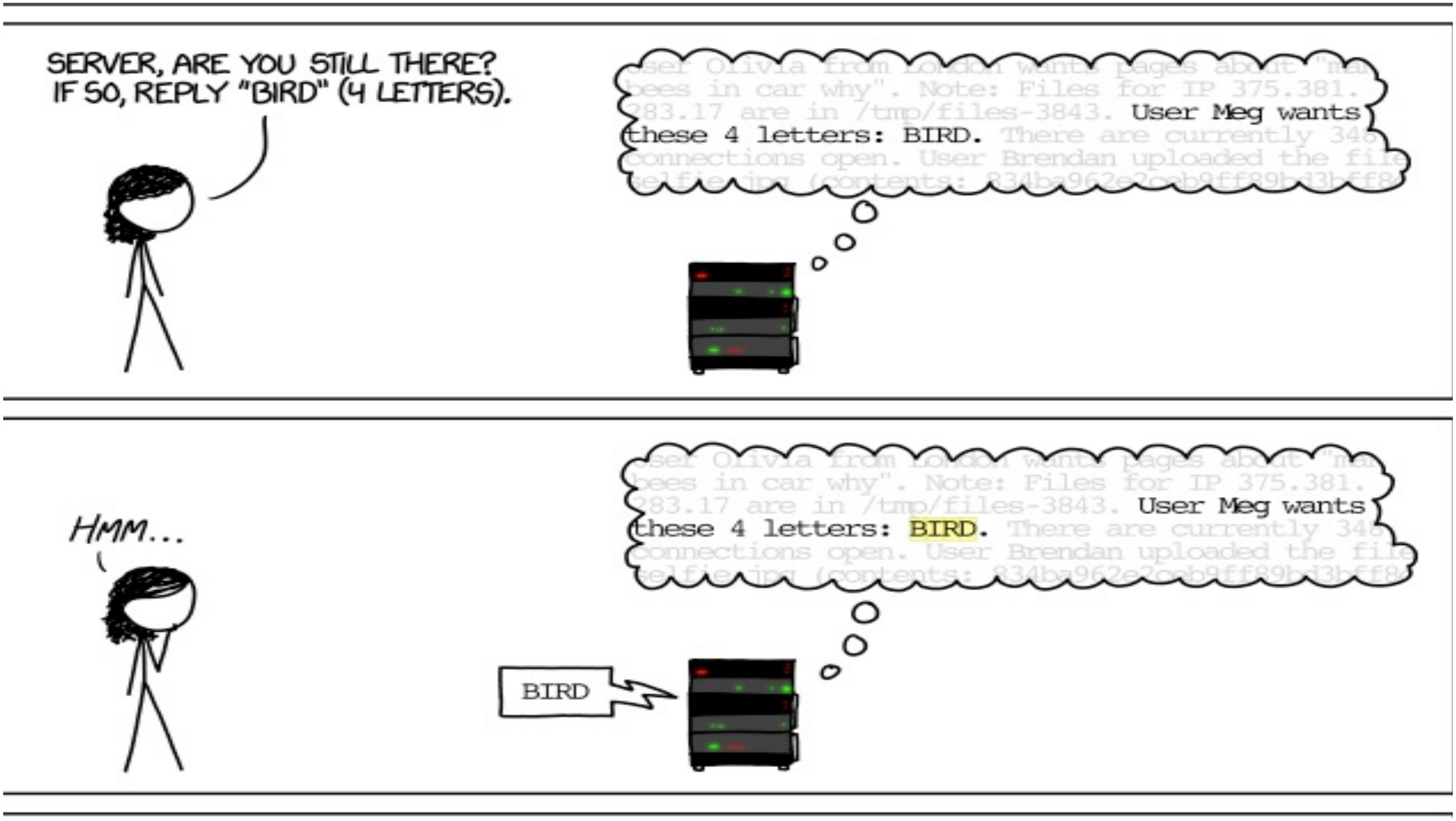
Heartbleed.com

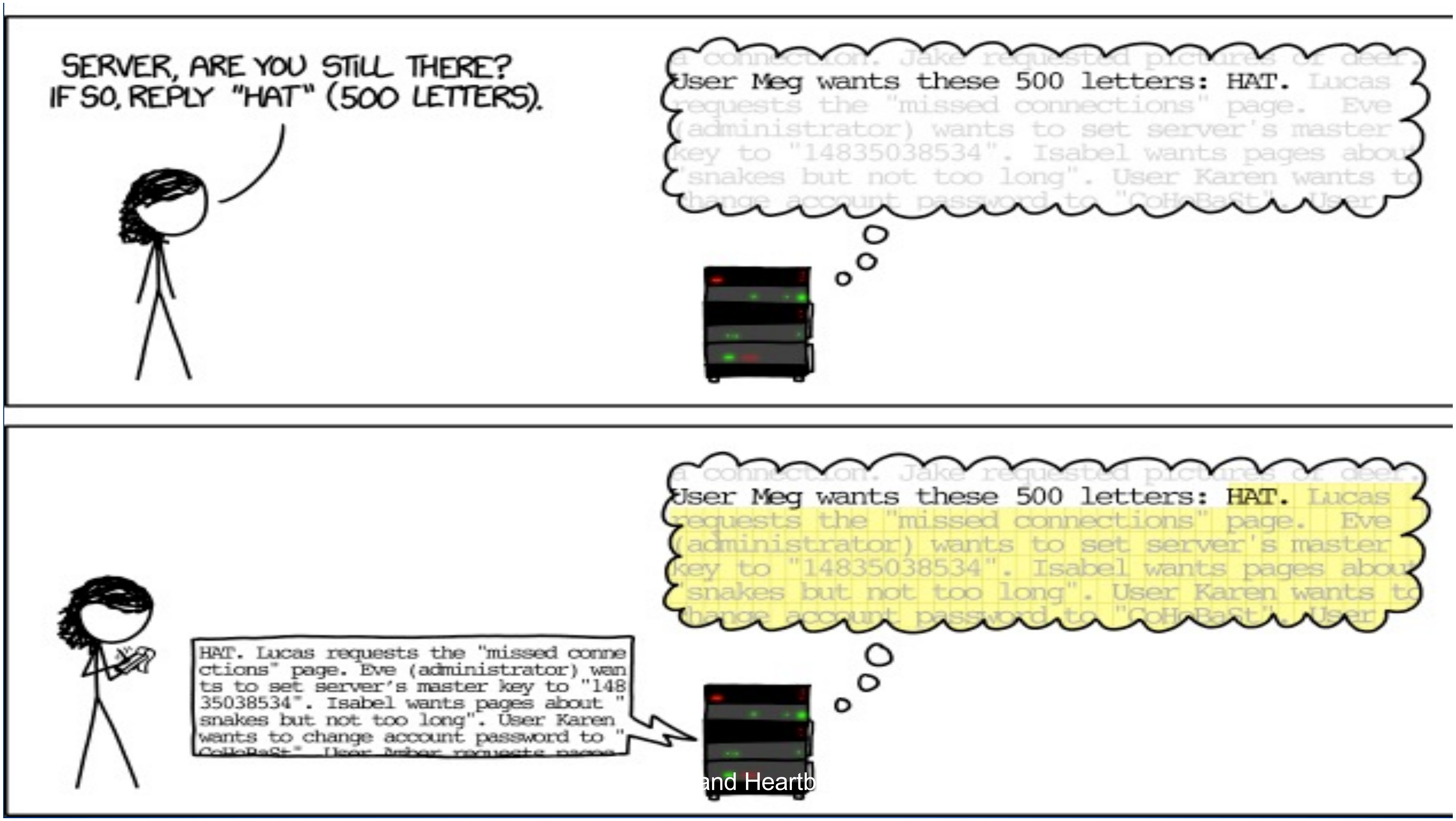
- Heartbleed Bug is a vulnerability in OpenSSL
- This weakness allows stealing the information protected by the opensource SSL/TLS encryption software (e.g. HTTPS)
- Heartbleed bug allows anyone on the Internet to read the memory of the vulnerable systems
  - Sensitive information in the memory: session identifiers, usernames, passwords, tokens, and in particular condition server's private cryptographic keys

HOW THE HEARTBLEED BUG WORKS:


SERVER, ARE YOU STILL THERE? IF SO, REPLY "POTATO" (6 LETTERS).
User Meg wants these 6 letters: POTATO.
POTATO

and Heart


SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).
User Olivia from London wants pages about "ma
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8
HMM...
BIRD
User Olivia from London wants pages about "ma
bees in car why". Note: Files for IP 375.381.
283.17 are in /tmp/files-3843. User Meg wants
these 4 letters: BIRD. There are currently 346
connections open. User Brendan uploaded the file
selfie.jpg (contents: 834ba962e2ceb9ff89bd3bff8


SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User
HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User Amber requests pages

and Heartb

Heartbleed disclosure

- 4/7/14 - Public disclosure of the Heartbleed bug
  - Full Disclosure mailing list managed by Fyodor http://seclists.org/fulldisclosure/
    - **[FD] heartbleed OpenSSL bug CVE-2014-0160**
- Heartbleed.com
- Several websites and news agencies started to publish information

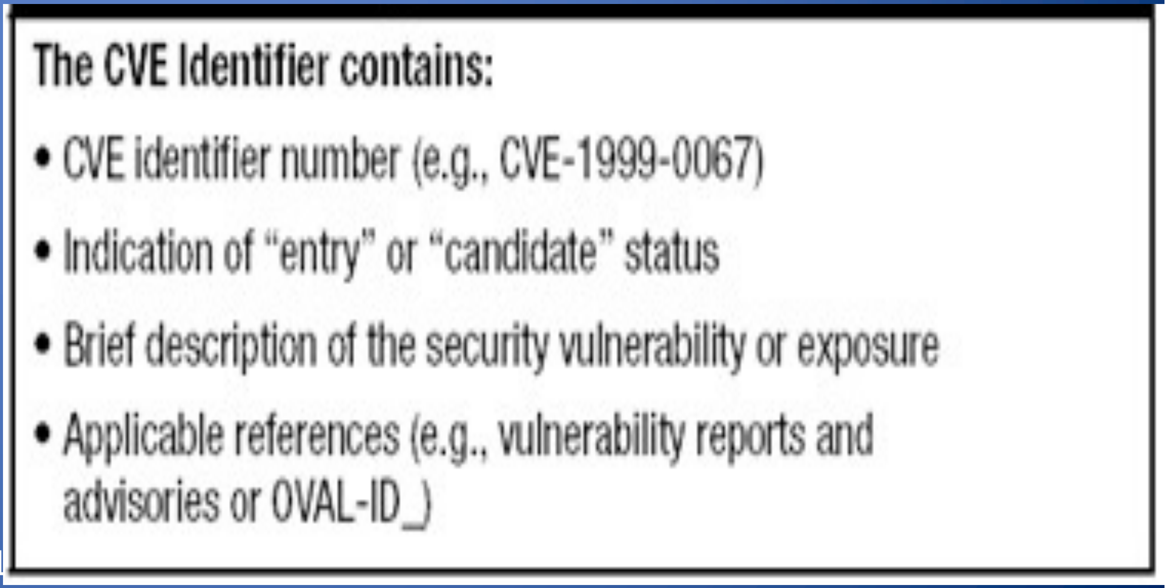
Different Vulnerability Disclosure

- **Responsible** disclosure (White hat)
  - Discovered vulnerability first reported to vendor
  - Disclosed to CERT later (usually 2 weeks)
    - CERT = Computer Emergency Response Team
  - Full disclosure to the public much later
- **Quick** disclosure (Grey hat)
  - Discovered vulnerability immediately (or quickly) disclosed publically
- **No disclosure** (Black hat)
  - Remains a “zero-day” attack until someone else finds it

CVE-2014-0160

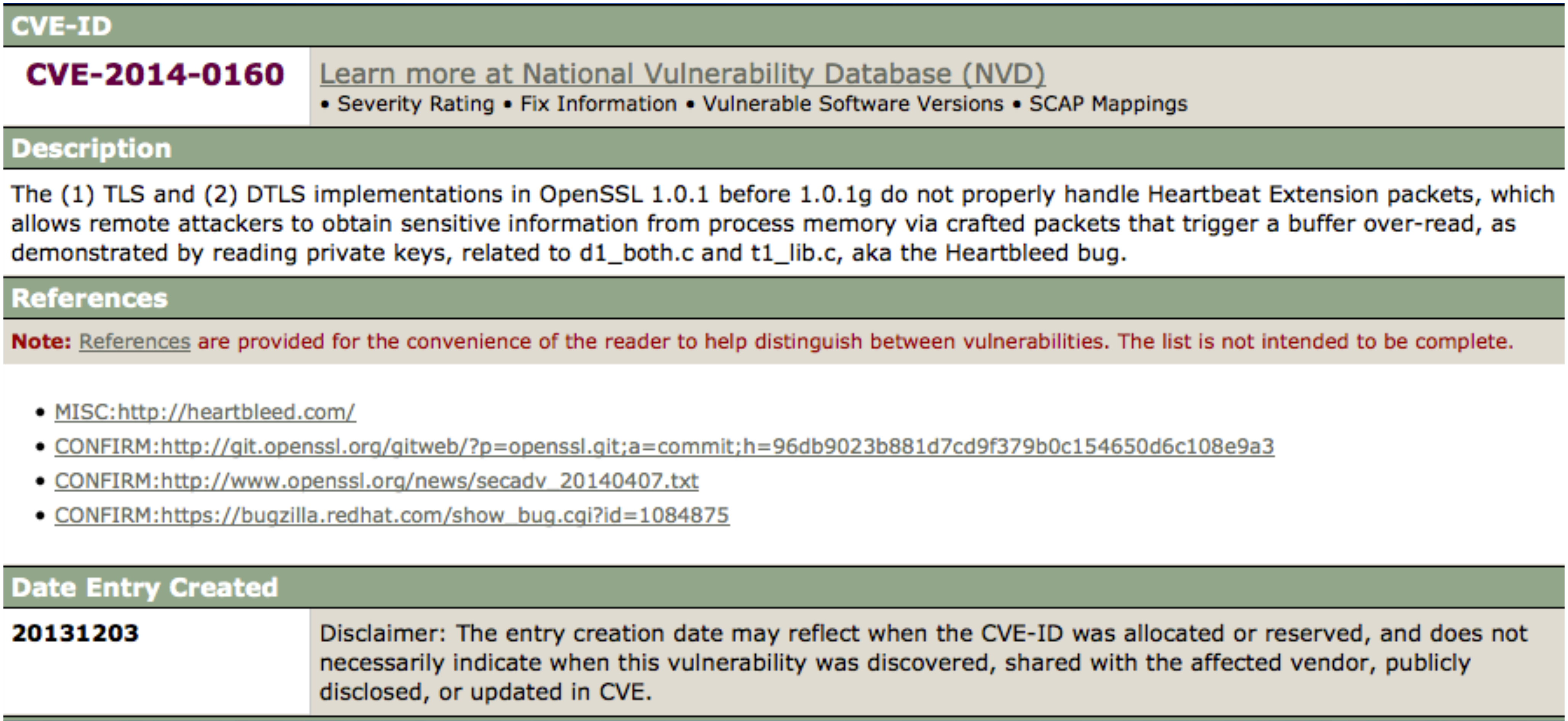
cve.mitre.org

- Common Vulnerabilities and Exposures (cve.mitre.org)
- MITRE is a FFRDC — federally funded research and development center
- Captures *specific* vulnerabilities
  - Standard name
  - Cross-reference to CERT, etc.
- Entry has three parts
  - Unique ID
  - Description
  - References
- OVAL
  - Open Vulnerability
  Assessment Language



The CVE Identifier contains:

- CVE identifier number (e.g., CVE-1999-0067)
- Indication of "entry" or "candidate" status
- Brief description of the security vulnerability or exposure
- Applicable references (e.g., vulnerability reports and advisories or OVAL-ID_)



CVE-ID

CVE-2014-0160 | Learn more at National Vulnerability Database (NVD)
• Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings

Description

The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MISC:http://heartbleed.com/
- CONFIRM:http://git.openssl.org/gitweb/?p=openssl.git;a=commit;h=96db9023b881d7cd9f379b0c154650d6c108e9a3
- CONFIRM:http://www.openssl.org/news/secadv_20140407.txt
- CONFIRM:https://bugzilla.redhat.com/show_bug.cgi?id=1084875

Date Entry Created

20131203 | Disclaimer: The entry creation date may reflect when the CVE-ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160

Sponsored by
DHS National Cyber Security Division/US-CERT

NIST
National Institute of Standards and Technology

National Vulnerability Database

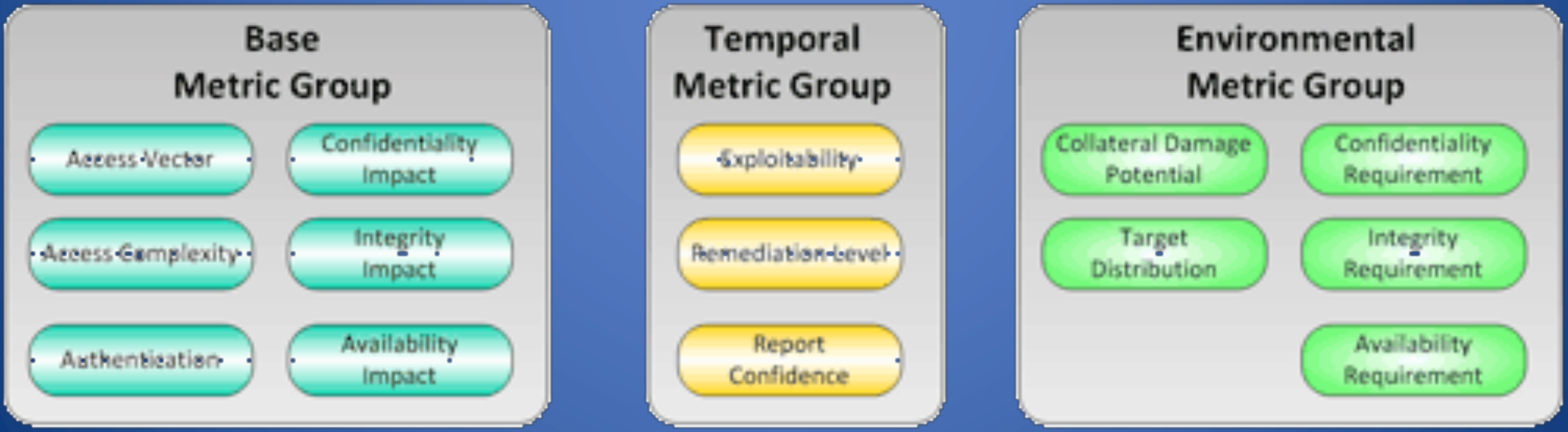
automating vulnerability management, security measurement, and compliance checking

- NVD is the U.S. government repository of standards-based vulnerability .
- NVD includes:
  - Original release date
  - Impact: CVSS Severity and Metrics
  - References to Advisories, Solutions, and Tools
    - Links external to NVD
  - Vulnerable software and versions
  - Technical Detail: Vulnerability Type and CVE link

http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160

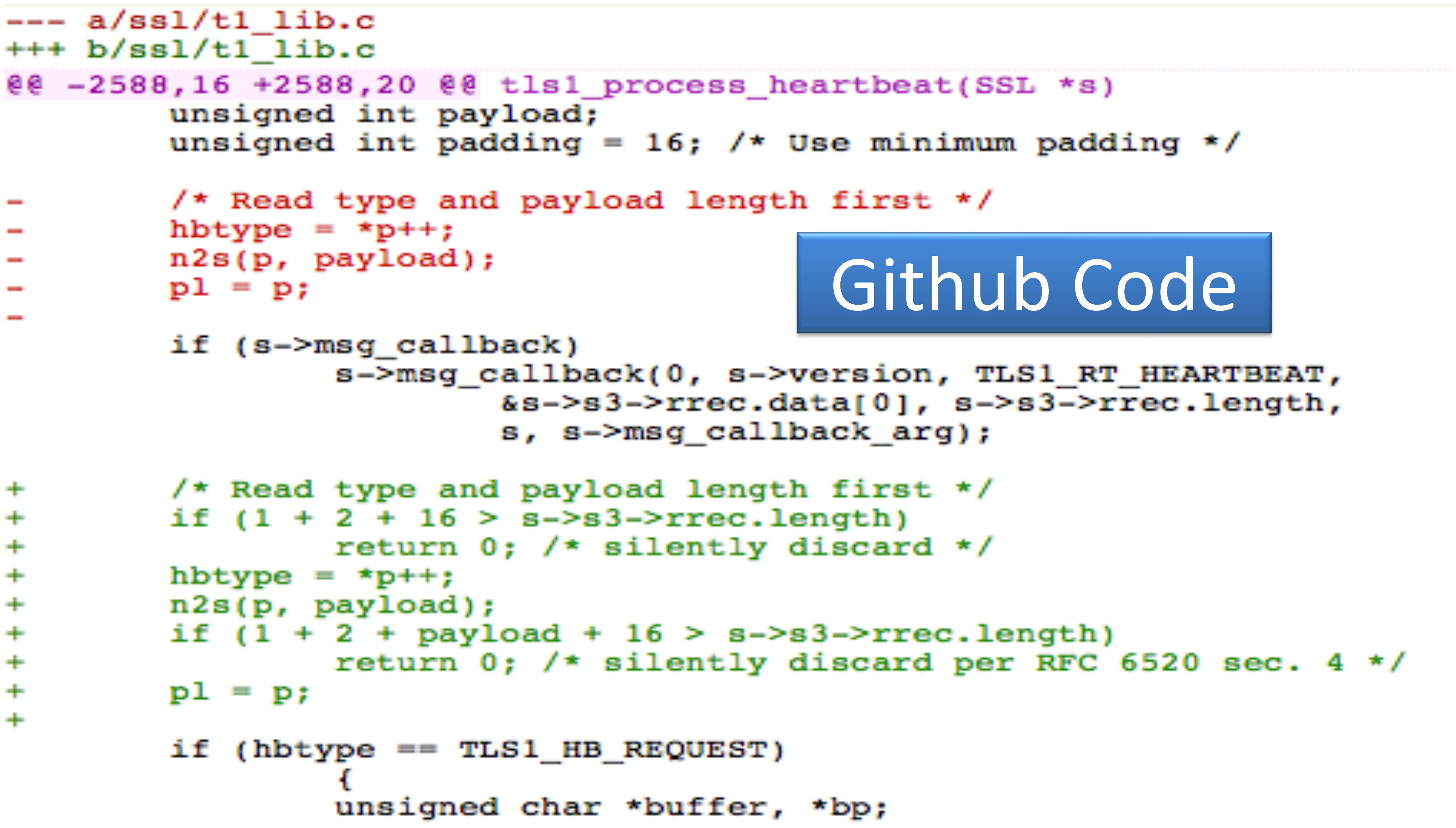
CVSS = Common Vulnerability Scoring System

- 3 values: Base, Temporal, Environmental
- Each ranges from 0 to 10
- Each value calculated from a formula based on criteria



References to Advisories, Solutions, and Tools

- Bugzilla
  - A Web-based general-purpose bugtracker
    - https://bugzilla.redhat.com/show_bug.cgi?id=1084875
    - https://bugzilla.redhat.com/attachment.cgi?id=883475&action=diff
- Github
  - A Web-based hosting service that uses Git revision control
    - http://git.openssl.org/gitweb/?p=openssl.git;a=commit;h=96db9023b881d7cd9f379b0c154650d6c108e9a3
    - Here we can find that the bug was fixed on April 5

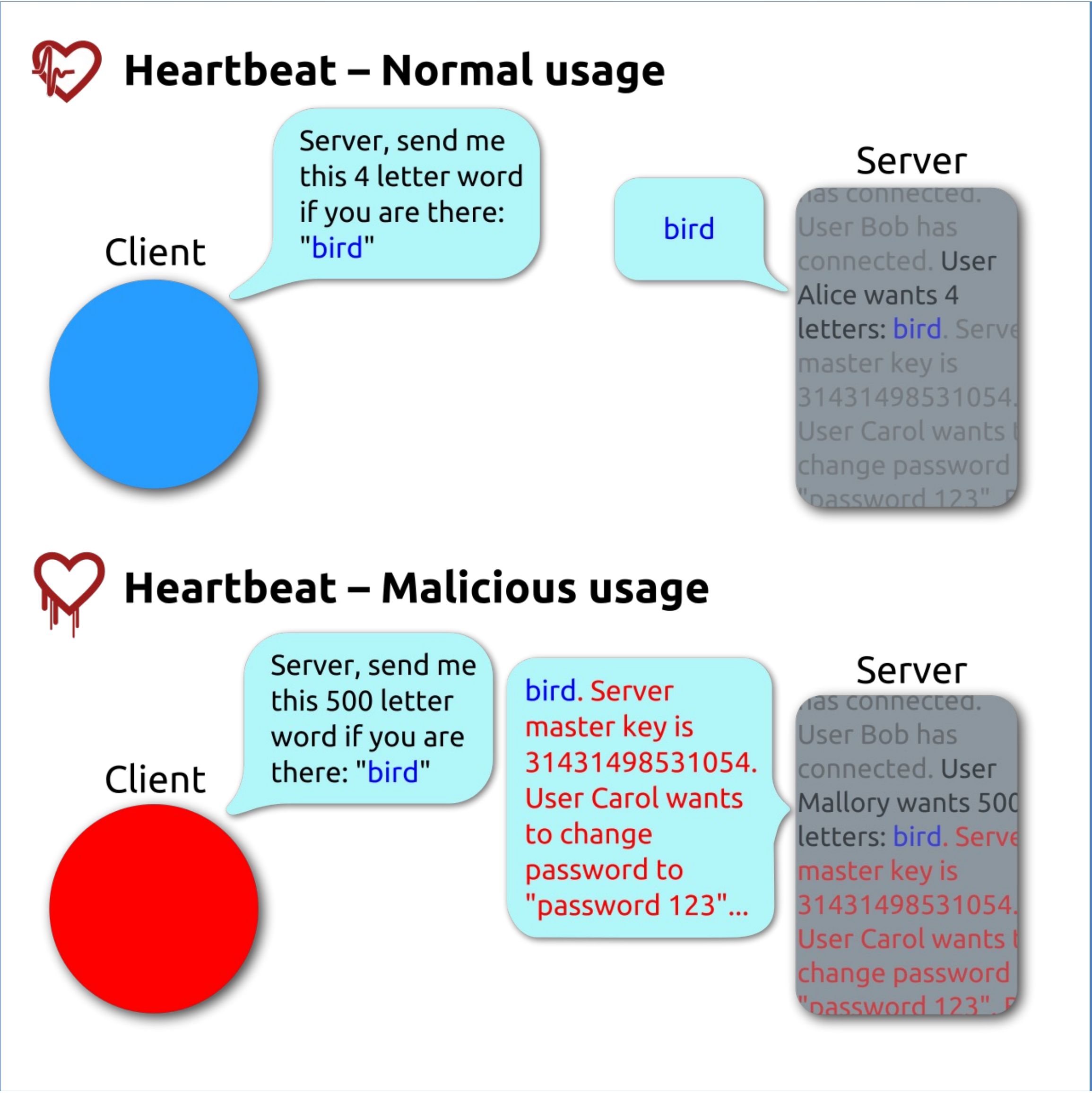


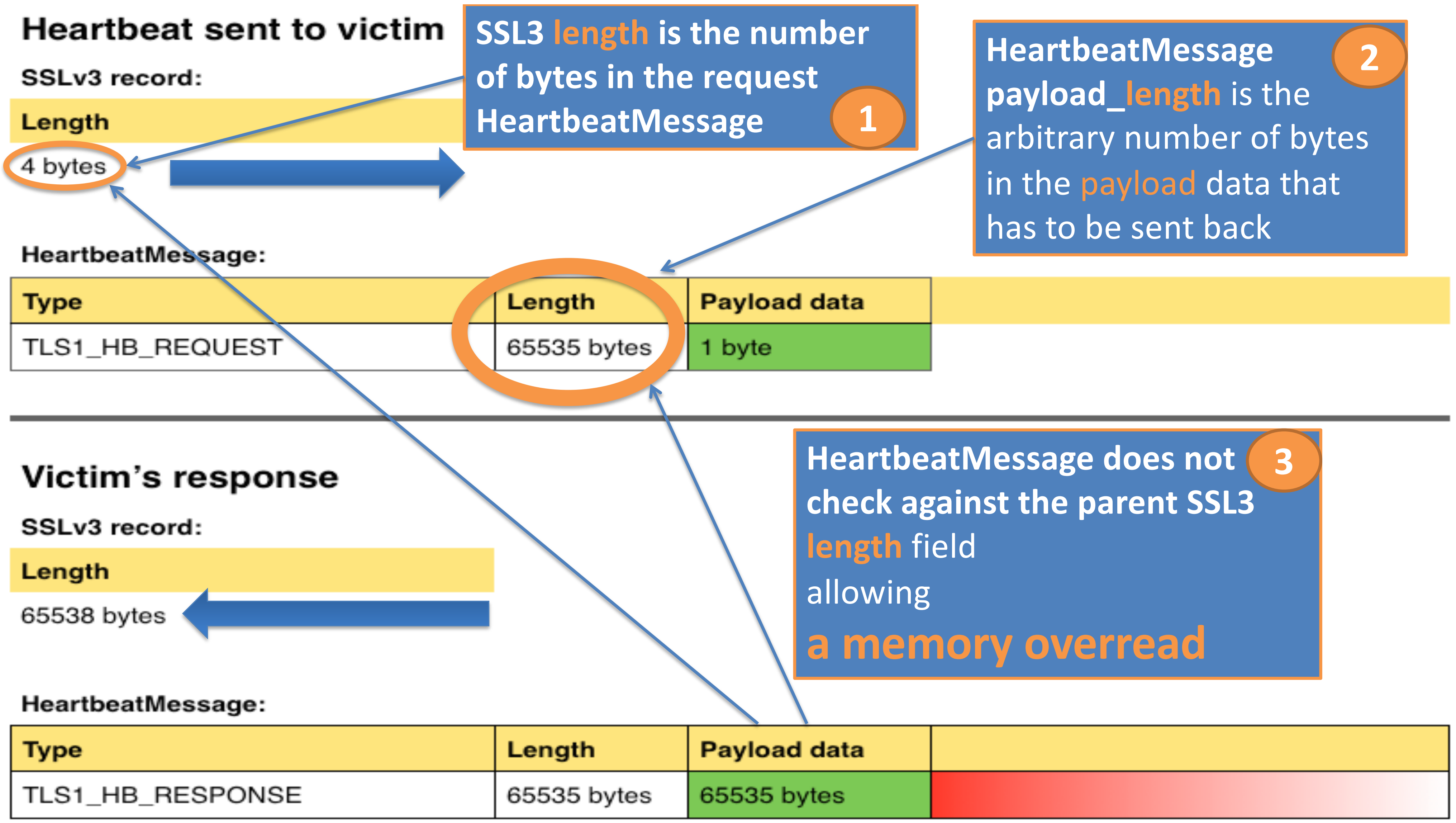
```diff
--- a/ssl/t1_lib.c
+++ b/ssl/t1_lib.c
@@ -2588,16 +2588,20 @@ tls1_process_heartbeat(SSL *s)
        unsigned int payload;
        unsigned int padding = 16; /* Use minimum padding */

-       /* Read type and payload length first */
-       hbtype = *p++;
-       n2s(p, payload);
-       pl = p;
-
        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

+       /* Read type and payload length first */
+       if (1 + 2 + 16 > s->s3->rrec.length)
+               return 0; /* silently discard */
+       hbtype = *p++;
+       n2s(p, payload);
+       if (1 + 2 + payload + 16 > s->s3->rrec.length)
+               return 0; /* silently discard per RFC 6520 sec. 4 */
+       pl = p;
+
        if (hbtype == TLS1_HB_REQUEST)
                {
                unsigned char *buffer, *bp;
```

Github Code

What is an SSL heartbeat?

- IETF RFC 6520 February 2012
- The Heartbeat Extension provides a new protocol for SSL (TLS/DTLS) allowing the usage of keep-alive functionality without performing a renegotiation


Heartbeat – Normal usage
Client
Server, send me this 4 letter word if you are there: "bird"
bird
Server
User Bob has connected. User Alice wants 4 letters: bird. Serve master key is 31431498531054. User Carol wants t change password "password 123".
Heartbeat – Malicious usage
Client
Server, send me this 500 letter word if you are there: "bird"
bird. Server master key is 31431498531054. User Carol wants to change password to "password 123"...
Server
User Bob has connected. User Mallory wants 500 letters: bird. Serve master key is 31431498531054. User Carol wants t change password "password 123".



Heartbeat sent to victim

SSLv3 record:

Length
4 bytes

HeartbeatMessage:

Type	**Length**	**Payload data**
TLS1_HB_REQUEST	65535 bytes	1 byte

SSL3 length is the number of bytes in the request HeartbeatMessage 1

HeartbeatMessage payload_length is the arbitrary number of bytes in the payload data that has to be sent back 2

Victim's response

SSLv3 record:

Length
65538 bytes

HeartbeatMessage:

Type	**Length**	**Payload data**	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	🟥

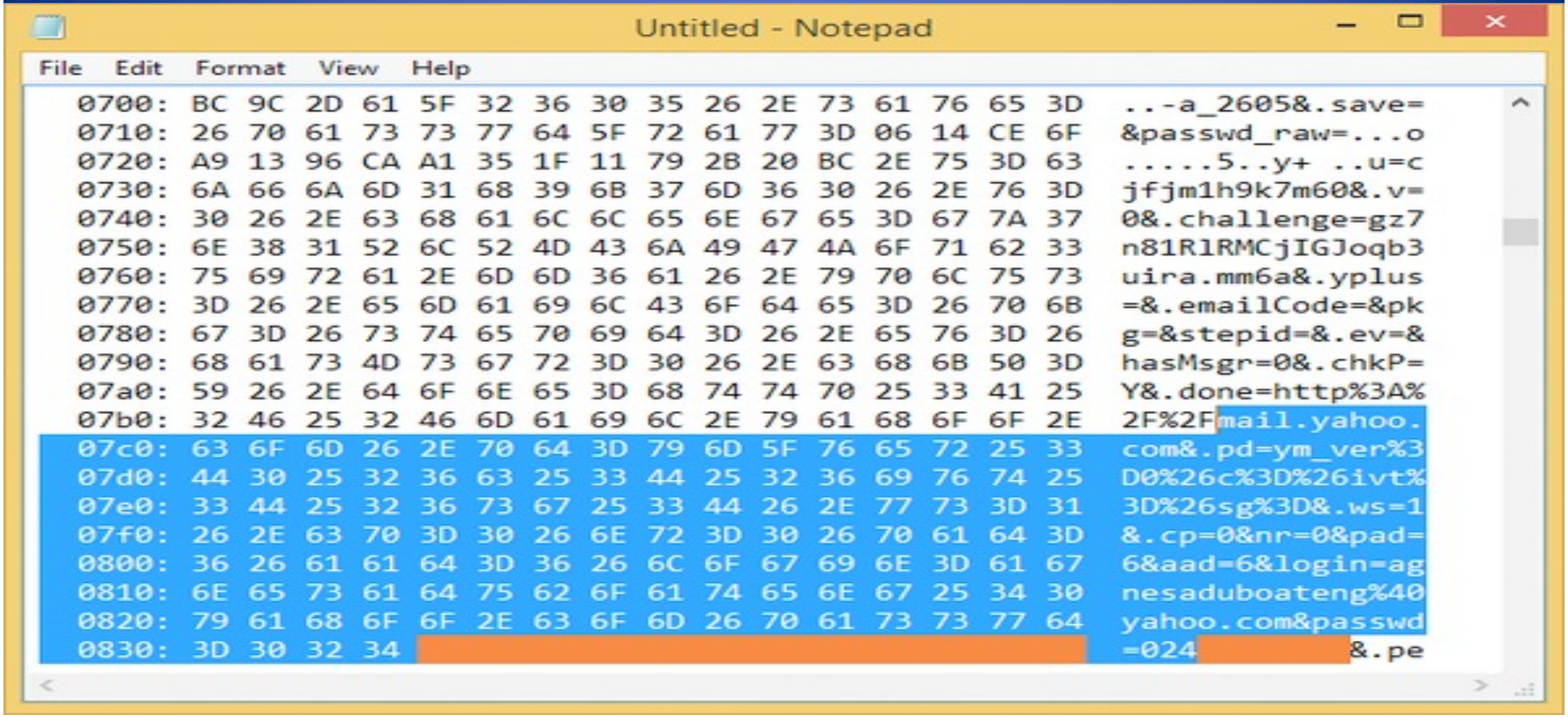
HeartbeatMessage does not check against the parent SSL3 length field allowing 3

a memory overread

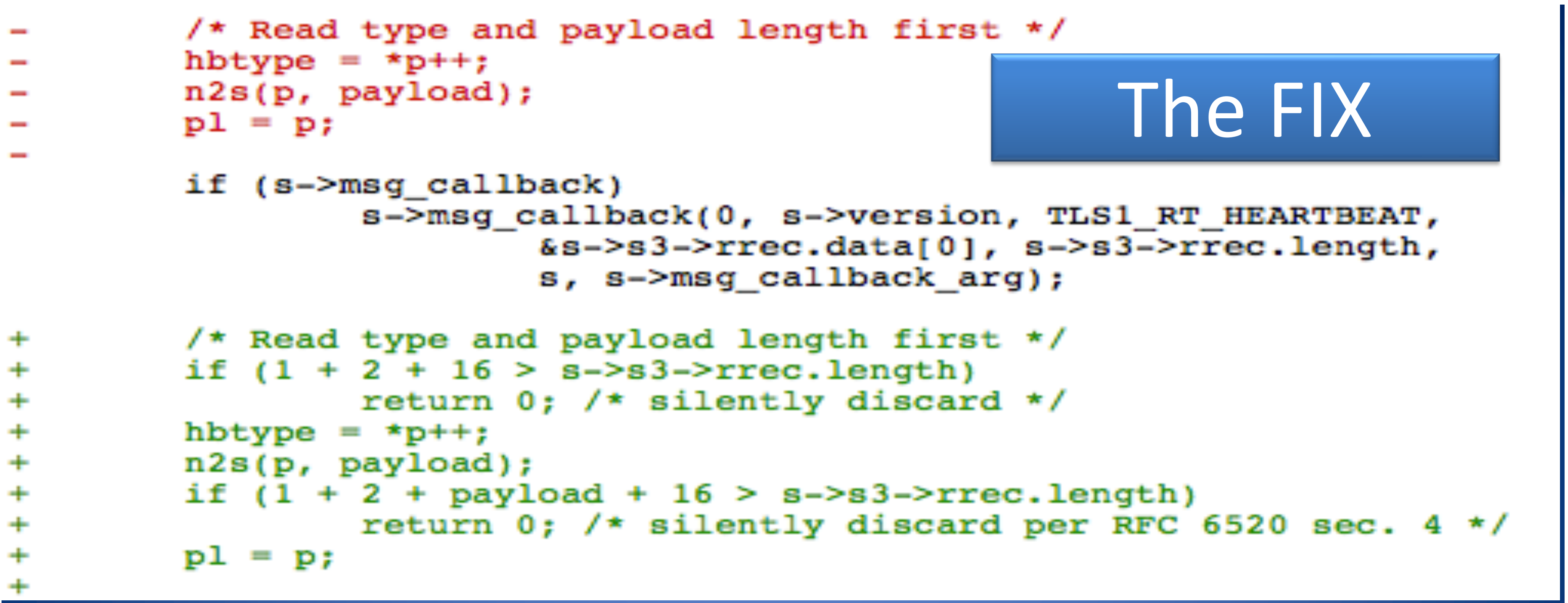
Heartbeat -> Heartbleed

- **payload** is controlled by the attacker, and it's quite large at 64KB
- **HeartbeatMessage** sent by the attacker only has a payload of one byte, and its **payload_length** is a lie
- **memcpy()** will read beyond the end of the received and creates a **HeartbleedMessage** that reads from the victim main memory
- In memory you can find passwords or decrypted messages from other users.
- Sending another heartbeat message leaks another 64KB and so on…
- Nmap script for Heartbleed: svn.nmap.org/nmap/scripts/ssl-heartbleed.nse

Password leaks on Yahoo



```
0700: BC 9C 2D 61 5F 32 36 30 35 26 2E 73 61 76 65 3D   ..-a_2605&.save=
0710: 26 70 61 73 73 77 64 5F 72 61 77 3D 06 14 CE 6F   &passwd_raw=...o
0720: A9 13 96 CA A1 35 1F 11 79 2B 20 BC 2E 75 3D 63   .....5..y+ ..u=c
0730: 6A 66 6A 6D 31 68 39 6B 37 6D 36 30 26 2E 76 3D   jfjm1h9k7m60&.v=
0740: 30 26 2E 63 68 61 6C 6C 65 6E 67 65 3D 67 7A 37   0&.challenge=gz7
0750: 6E 38 31 52 6C 52 4D 43 6A 49 47 4A 6F 71 62 33   n81RlRMCjIGJoqb3
0760: 75 69 72 61 2E 6D 6D 36 61 26 2E 79 70 6C 75 73   uira.mm6a&.yplus
0770: 3D 26 2E 65 6D 61 69 6C 43 6F 64 65 3D 26 70 6B   =&.emailCode=&pk
0780: 67 3D 26 73 74 65 70 69 64 3D 26 2E 65 76 3D 26   g=&stepid=&.ev=&
0790: 68 61 73 4D 73 67 72 3D 30 26 2E 63 68 6B 50 3D   hasMsgr=0&.chkP=
07a0: 59 26 2E 64 6F 6E 65 3D 68 74 74 70 25 33 41 25   Y&.done=http%3A%
07b0: 32 46 25 32 46 6D 61 69 6C 2E 79 61 68 6F 6F 2E   2F%2Fmail.yahoo.
07c0: 63 6F 6D 26 2E 70 64 3D 79 6D 5F 76 65 72 25 33   com&.pd=ym_ver%3
07d0: 44 30 25 32 36 63 25 33 44 25 32 36 69 76 74 25   D0%26c%3D%26ivt%
07e0: 33 44 25 32 36 73 67 25 33 44 26 2E 77 73 3D 31   3D%26sg%3D&.ws=1
07f0: 26 2E 63 70 3D 30 26 6E 72 3D 30 26 70 61 64 3D   &.cp=0&nr=0&pad=
0800: 36 26 61 61 64 3D 36 26 6C 6F 67 69 6E 3D 61 67   6&aad=6&login=ag
0810: 6E 65 73 61 64 75 62 6F 61 74 65 6E 67 25 34 30   nesaduboateng%40
0820: 79 61 68 6F 6F 2E 63 6F 6D 26 70 61 73 73 77 64   yahoo.com&passwd
0830: 3D 30 32 34 [redacted]                              =024[redacted]&.pe
```



```
-       /* Read type and payload length first */
-       hbtype = *p++;
-       n2s(p, payload);
-       pl = p;
-
        if (s->msg_callback)
                s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                        &s->s3->rrec.data[0], s->s3->rrec.length,
                        s, s->msg_callback_arg);

+       /* Read type and payload length first */
+       if (1 + 2 + 16 > s->s3->rrec.length)
+               return 0; /* silently discard */
+       hbtype = *p++;
+       n2s(p, payload);
+       if (1 + 2 + payload + 16 > s->s3->rrec.length)
+               return 0; /* silently discard per RFC 6520 sec. 4 */
+       pl = p;
+
```

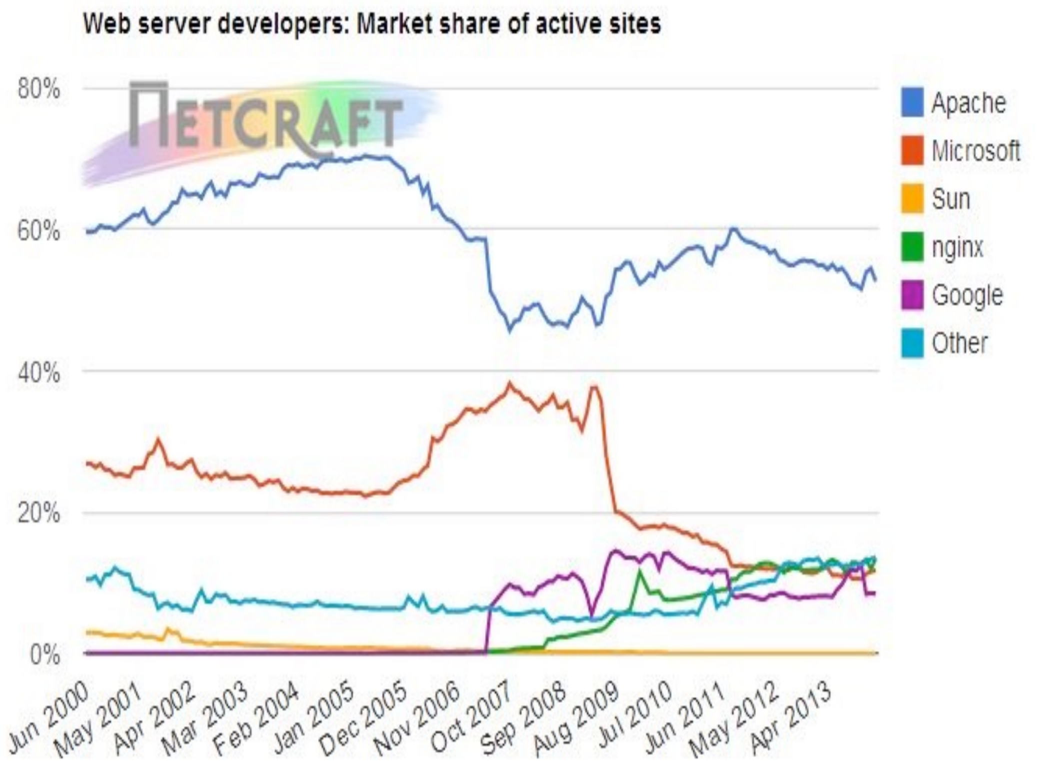
The FIX

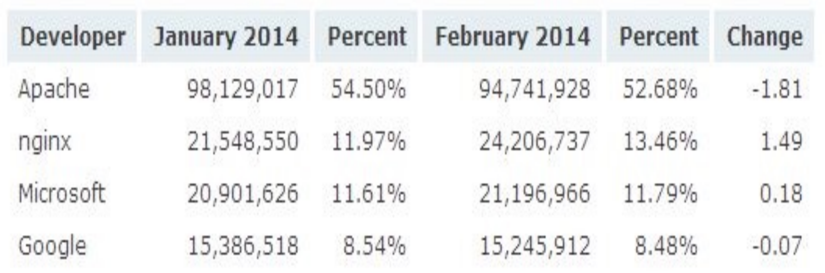
OpenSSL 1.0.1g implements a bounds check that discards:

- A HeartbeatMessage Packet smaller than with a payload of zero byte
- A **HeartbeatMessage**Packet with a payload greater than SSL3 structure (**s3->rrec**)

Why so dangerous?

- It was on the wild for two year in an open source software
- Possibly two-thirds of the server in the world can use this kind of software





Developer	January 2014	Percent	February 2014	Percent	Change
Apache	98,129,017	54.50%	94,741,928	52.68%	-1.81
nginx	21,548,550	11.97%	24,206,737	13.46%	1.49
Microsoft	20,901,626	11.61%	21,196,966	11.79%	0.18
Google	15,386,518	8.54%	15,245,912	8.48%	-0.07

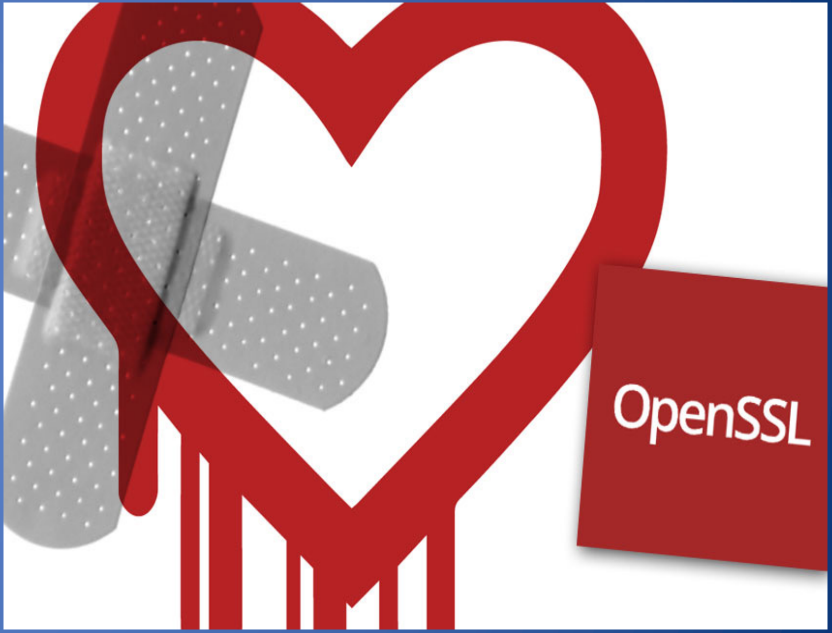
Class Discussion on Heartbleed

If you are Heartbleed vulnerable:

- Is it possible to sniff https traffic?
- And what about the former https traffic?
- Is it possible to spoof the server certificate?
- Which kind of protection is possible to perform for the final users to protect?

After Heartbleed…

- Open source seems to be just potentially more secure than proprietary software
- Core Infrastructure Initiative
  a multi-million dollar project funded by major IT companies
  - www.coreinfrastructure.org
  - From 2020 moved in:
    openssf.org
- Previously OpenSSL received about $2,000 per year in donations



https://threatpost.com/openssl-operating-with-renewed-vision-two-years-after-heartbleed/116567/

What We Have Learned

- Goals of the SSL/TLS protocol
- Overview of the SSL protocol
- Perfect forward secrecy
- SSL certificates, chain of trust, and revocation
- Heartbleed attack