CT417 SOFTWARE ENGINEERING III

BUFFER OVERFLOW CASE STUDY – THE HEARTBLEED BUG

Dr. Michael Schukat

Ollscoil na Gaillimhe
University of Galway

A Bug with its own Website (heartbleed.com) and Icon

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

What leaks in practice?

We have tested some of our own services from attacker's perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

How to stop the leak?

As long as the vulnerable version of OpenSSL is in use it can be abused. Fixed OpenSSL has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.

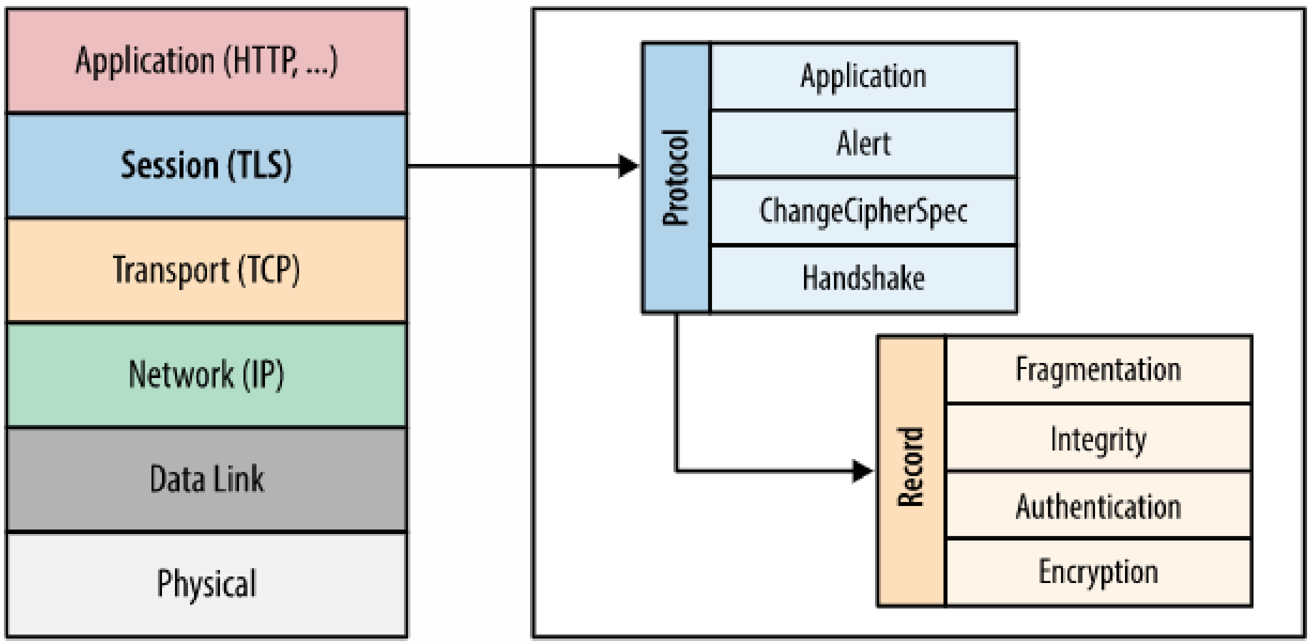
X X X

TLS Overview

- Based on the SSL protocol, which was originally developed in the 1990s to secure ecommerce transaction on the web, i.e.
  - encryption to protect customers' personal data
  - authentication and integrity check of transactions
- To achieve this, the SSL protocol was implemented at the application layer, directly on top of TCP, enabling (application layer) protocols above it (e.g. HTTP) to operate unchanged

TLS Overview

TLS location in protocol stack



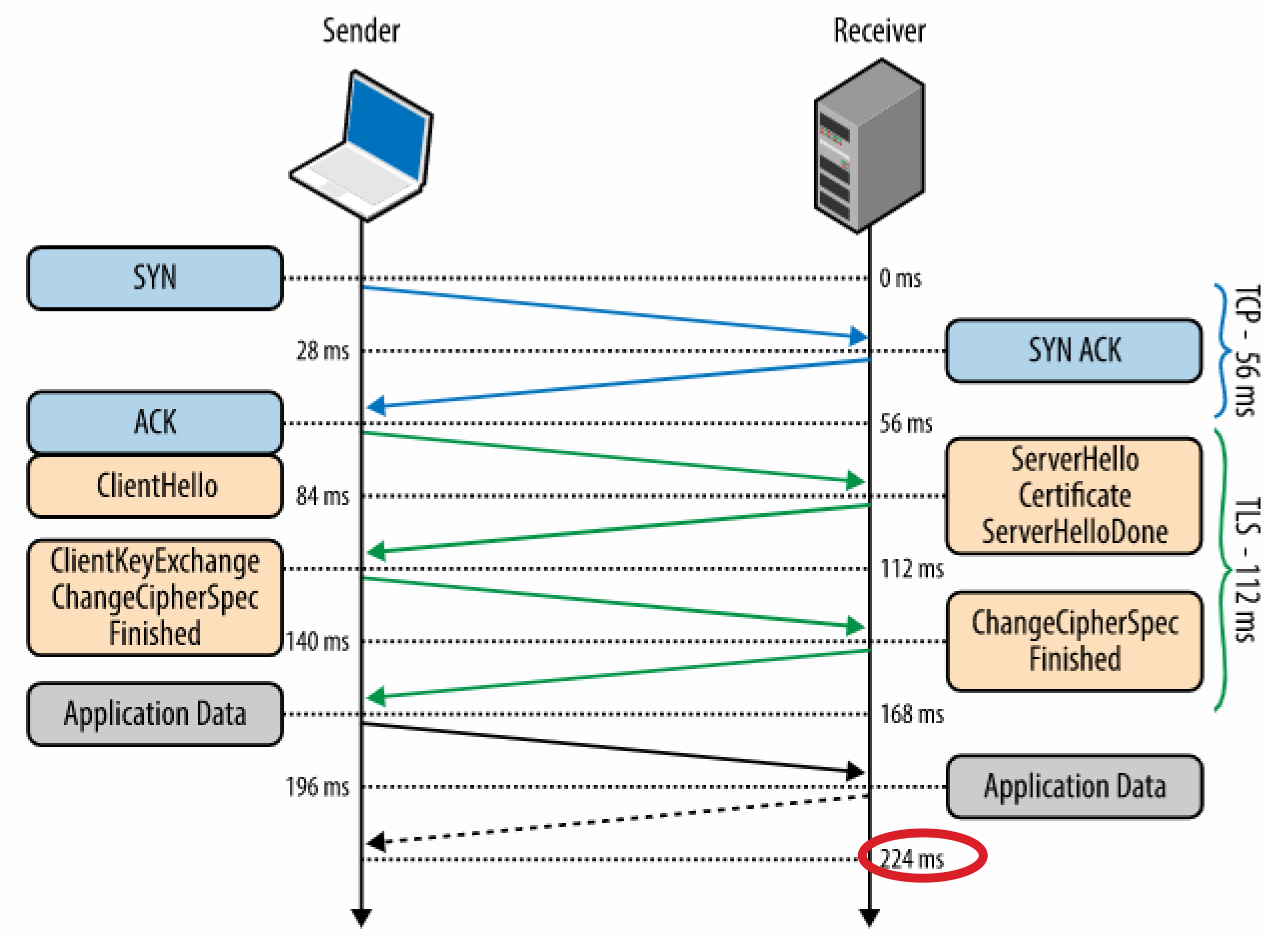
Encryption, Authentication and Integrity

- The TLS protocol provides three essential services to all application layer protocols running above it
- Encryption
  - A mechanism to obfuscate what is sent from one host to another (typically between a client and a server)
- Authentication
  - A mechanism to verify the validity of provided identification material (i.e. (mutual) authentication using digital certificates)
- Integrity
  - A mechanism to detect message tampering and forgery (messages cannot be manipulated in transit, and messages cannot be forged by a threat actor)

HTTPS

- "HTTP over TLS"
- HTTPS protects the integrity of the website
  - Encryption prevents intruders from tampering with transmitted data
- HTTPS protects the privacy and security of the user
  - Encryption prevents intruders from eavesdropping and abusing the exchanged data
- HTTPS enables new features on the web
  - Necessary to safely use new web platform features, such as accessing users geolocation, VoIP and videoconferencing

TLS Handshake



Overview Heartbleed

- Discovered in 2014
- Exploits a bug in the OpenSSL implementation of the TLS "heartbeat hello" extension
- Can affect both client and server side

X X

OpenSSL

- OpenSSL is an open-source library (→ GitHub) that contains routines / algorithms / protocol implementations / ciphers used for secure network communication
  - Including SSL (depreciated) and TLS implementations
- It is written in C and widely used in Linux distributions
  - Linux is a widely used server-side OS

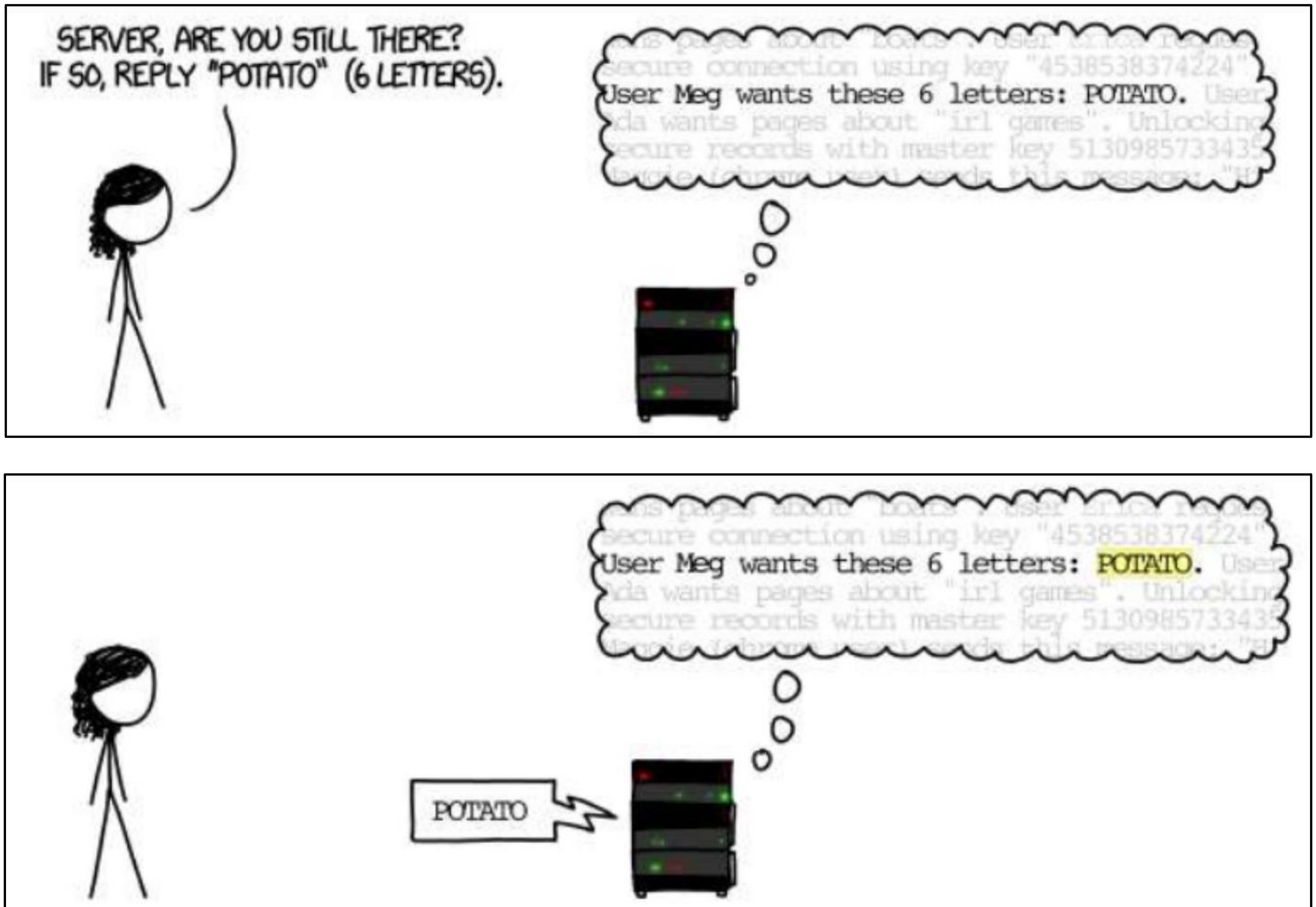
Heartbleed Impact

- Reported via CVE-2014-0160 (later)
- The following operating system distributions were potentially affected:
  - Debian Wheezy (stable)
  - Ubuntu 12.04.4 LTS
  - CentOS 6.5
  - Fedora 18
  - OpenBSD 5.3
  - FreeBSD 10.0
  - NetBSD 5.0.2
  - OpenSUSE 12.2

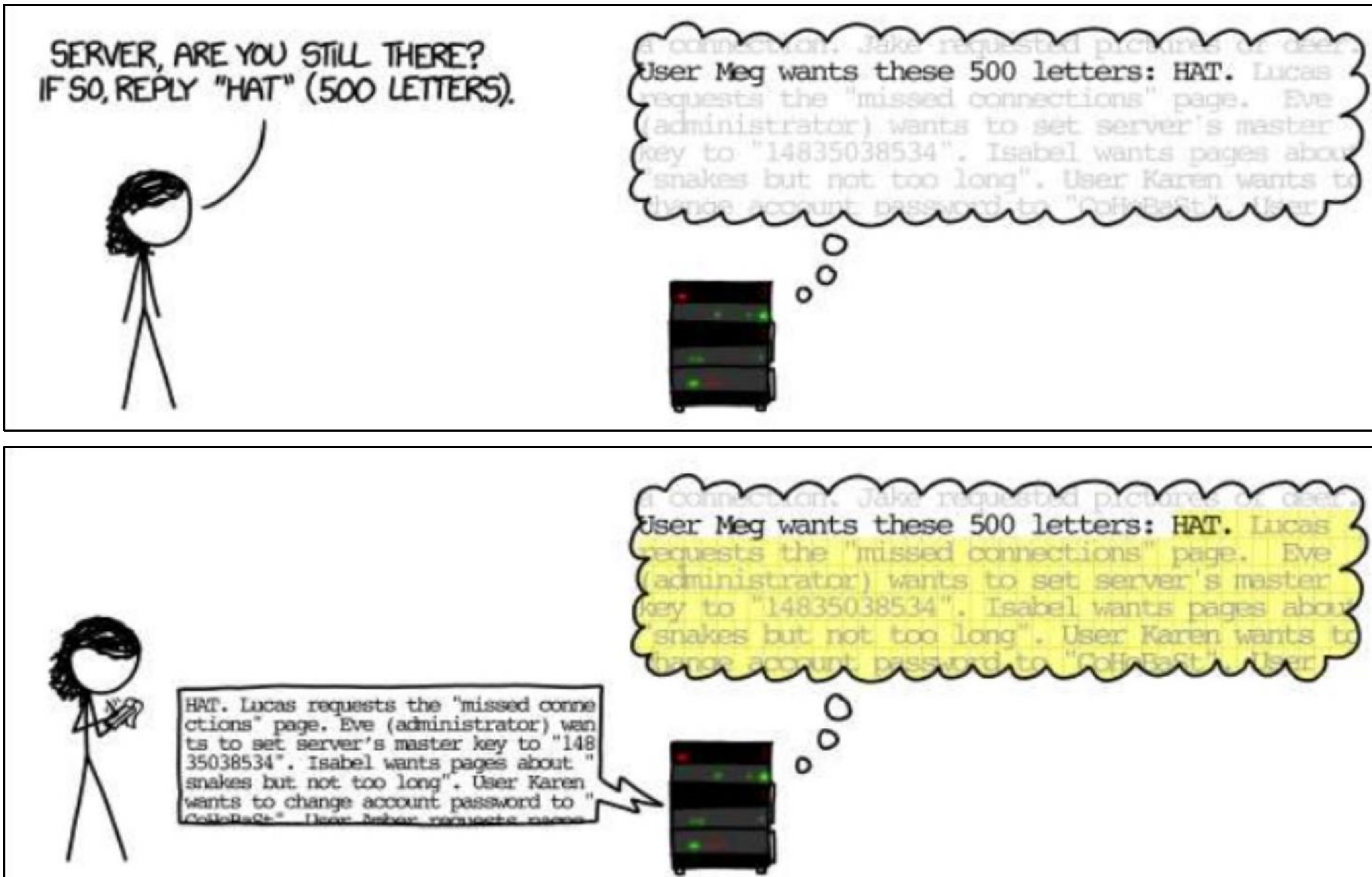
TLS Heartbeat Extension

- Originally TLS had no provisions to keep a client / server connection alive without continuous data transfer
  - Idle connections would timeout instead and a computationally expensive reconnect would have to take place (224 ms in example)
- The heartbeat extension provides a new protocol for "keep-alive" messages
  - One endpoint could send out a *HeartbeatRequest* message, which would be immediately responded with a *HeartbeatResponse* message

Heartbeat with incoming Message (correctly) buffered



The Heartbleed Attack



Heartbeat Request / Response Message

- The Heartbeat protocol messages consist of their type and an arbitrary payload and padding.

heartbeat_request or heartbeat_response

- struct {

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

16+ bytes of random content, ignored by receiver

- The sender composes a request message containing a payload with a specified length (i.e. *payload_length*)
- The receiver returns a response message containing a copy of the sender's payload (with length *payload_length*)
- "opaque" seems to be a typdef (i.e. unsigned char)

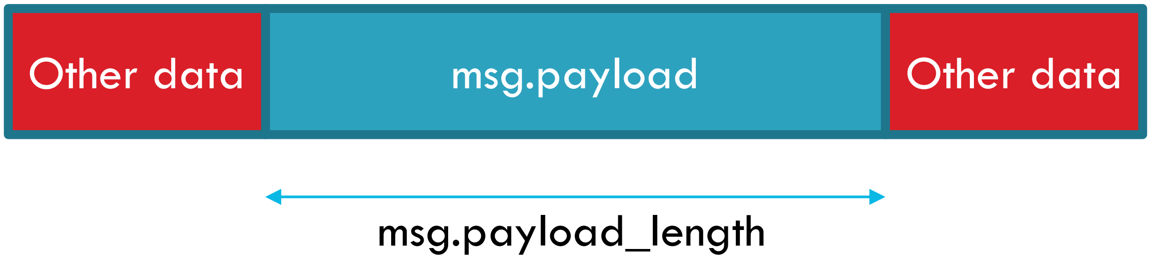
Pseudo-Code Example (correct)

Sender (constructs correct request message):

```
struct HeartbeatMessage msg;
msg.HeartbeatMessageType = heartbeat_request;
msg.payload_length = 2;
alloc(msg.payload, 2); // Note that the payload array is dynamically allocated
msg.payload = “AB”;
…
```

Receiver (receives above incoming msg) embedded in TCP/IP/TLS packet and constructs response s_msg:

```
struct HeartbeatMessage s_msg;
s_msg.HeartbeatMessageType = heartbeat_response;
s_msg.payload_length = msg.payload_length;
alloc (s_msg.payload, msg.payload_length);
memcpy(s_msg.payload, msg.payload, msg.payload_length);
…
```



Heartbleed Exploit

- The server receives a request message and stores in in (stack and heap) memory
  - Memory also contains other sessions-related information including tokens, keys, session IDs etc. from other sessions
- If (unint16) *payload_length* is actually larger than (opaque) *payload[..]*, the server will copy heap memory content beyond the payload array into the response message payload array (e.g. ret_*payload*), which is then sent back to the sender:
  memcpy(ret_payload, payload, payload_length);

memcpy(s_msg.payload, msg.payload, msg.payload_length);

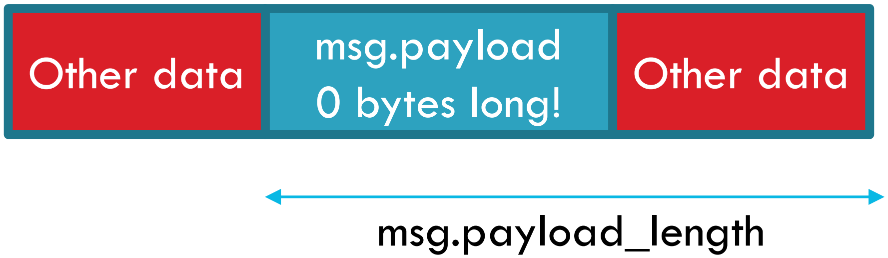
Pseudo-Code Example (Heartbleed Exploit)

Sender (constructs correct request message):

```
struct HeartbeatMessage msg;
msg.HeartbeatMessageType = heartbeat_request;
msg.payload_length = 0xFFFF;
msg.payload = “”;
…
```

Receiver (receives above incoming msg) embedded in TCP/IP/TLS packet and constructs response s_msg:

```
struct HeartbeatMessage s_msg;
s_msg.HeartbeatMessageType = heartbeat_response;
s_msg.payload_length = msg.payload_length;
alloc(s_msg.payload, msg.payload_length);
memcpy(s_msg.payload, msg.payload, msg.payload_length);
…
```



Heartbleed Exploit Extract (Python Code)

- https://gist.github.com/eelsivart/10174134

Heartbleed (CVE-2014-0160) Test & Exploit Python Script

heartbleed.py   Raw

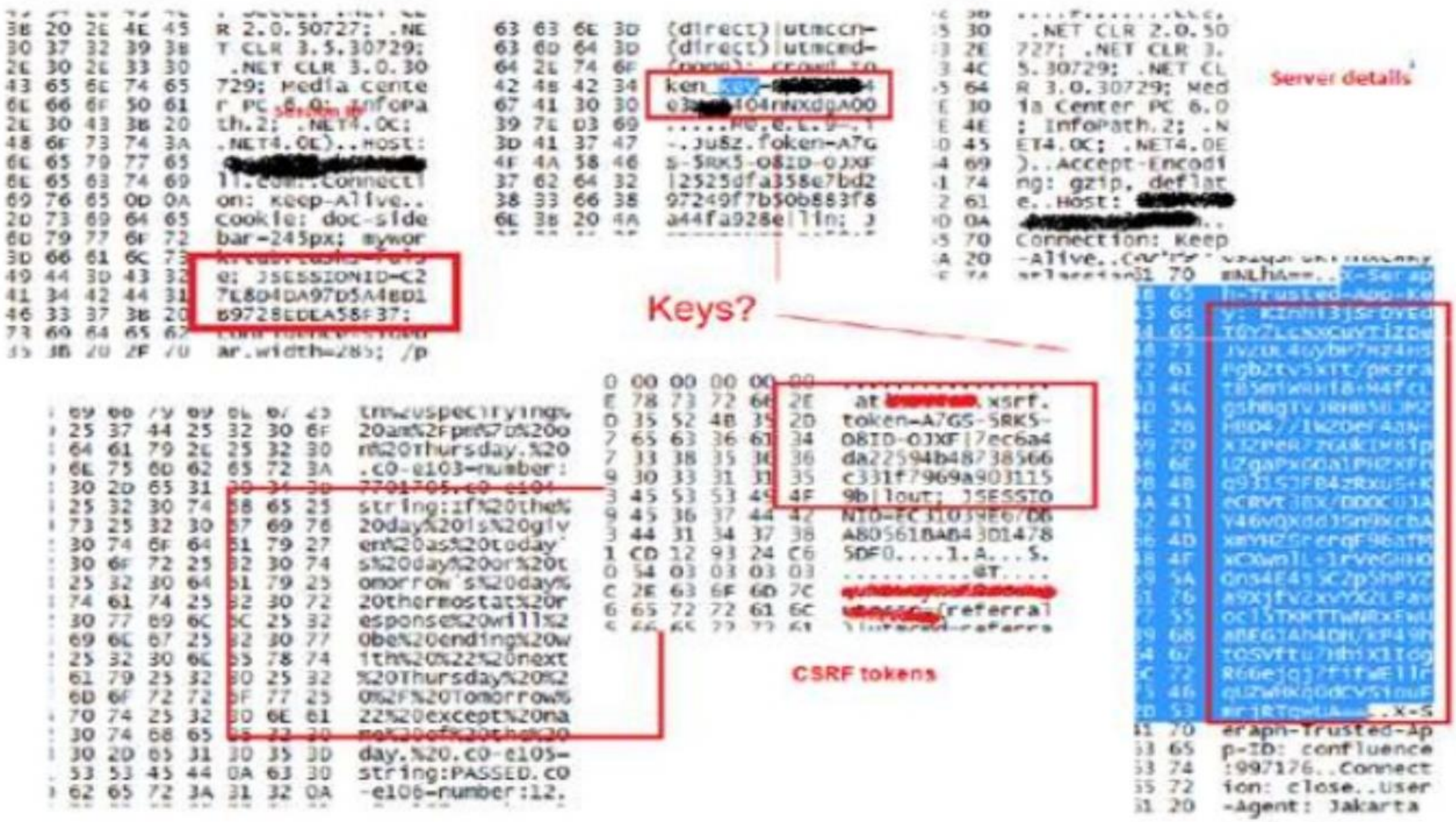
```
#!/usr/bin/python

# Modified by Travis Lee
# Last Updated: 4/21/14
# Version 1.16
#
# -changed output to display text only instead of hexdump and made it easier to read
# -added option to specify number of times to connect to server (to get more data)
# -added option to send STARTTLS command for use with SMTP/POP/IMAP/FTP/etc...
# -added option to specify an input file of multiple hosts, line delimited, with or without a port specified (host:port)
# -added option to have verbose output
# -added capability to automatically check if STARTTLS/STLS/AUTH TLS is supported when smtp/pop/imap/ftp ports are entered and automaticall
# -added option for hex output
# -added option to output raw data to a file
# -added option to output ascii data to a file
# -added option to not display returned data on screen (good if doing many iterations and outputting to a file)
# -added tls version auto-detection
# -added an extract rsa private key mode (orig code from epixoip. will exit script when found and enables -d (do not display returned data
#  -requires following modules: gmpy, pyasn1

# Quick and dirty demonstration of CVE-2014-0160 by Jared Stafford (jspenguin@jspenguin.org)
# The author disclaims copyright to this source code.

import sys
import struct
import socket
import time
import select
import re
```

What can be leaked?



What happened next?

- The Heartbleet bug was fixed (of course)
- Further checks and balances were added to validate that payload length was correct

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

==

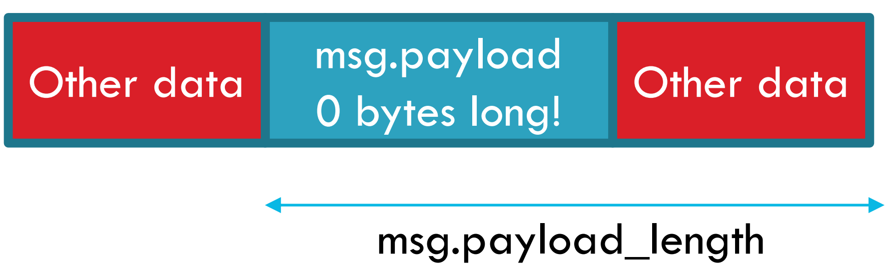
Pseudo-Code Example (Heartbleed Exploit Fixed)

Sender (constructs correct request message):

```
struct HeartbeatMessage msg;
msg.HeartbeatMessageType = heartbeat_request;
msg.payload_length = 0xFFFF;
msg.payload = “”;
…
```

Receiver (receives above incoming msg) embedded in TCP/IP/TLS packet and constructs response s_msg:

```
struct HeartbeatMessage s_msg;
int correctPayloadLen = len(msg.payload);
s_msg.HeartbeatMessageType = heartbeat_response;
s_msg.payload_length = correctPayloadLen;
alloc(s_msg.payload, correctPayloadLen);
memcpy(s_msg.payload, msg.payload, correctPayloadLen);
…
```



Recall (Menti Question): Attack (RFC2828, Internet Security Glossary)

- An assault on system security that derives from an intelligent threat, i.e. a deliberate attempt
- An "active attack" attempts to alter system resources or affect their operation
- A "passive attack“ attempts to learn or make use of information from the system, but does not affect system resources

```
+ - - - - - - - - - - - - +  + - - - - +  + - - - - - - - - - - -+
| An Attack:              |  |Counter- |  | A System Resource:   |
| i.e., A Threat Action   |  | measure |  | Target of the Attack |
| +----------+            |  |         |  | +-----------------+  |
| | Attacker |<==================||<=========                 |  |
| |   i.e.,  |   Passive  |  |         |  | |  Vulnerability  |  |
| | A Threat |<=================>||<========>                 |  |
| |  Agent   |  or Active |  |         |  | +-------|||-------+  |
| +----------+   Attack   |  |         |  |         VVV          |
|                         |  |         |  | Threat Consequences  |
+ - - - - - - - - - - - - +  + - - - - +  + - - - - - - - - - - -+
```

M

Lessons learnt

- OpenSSL core developer Ben Laurie claimed that a *security audit of OpenSSL would have caught Heartbleed*
- *Some other quotes from the security community:*
  - *"Think about it, OpenSSL only has two fulltime people to write, maintain, test, and review 500,000 lines of business critical code"*
  - *"The mystery is not that a few overworked volunteers missed this bug; the mystery is why it hasn't happened more often"*
  - *"There should be a continuous effort to simplify the code, because otherwise just adding capabilities will slowly increase the software complexity. The code should be refactored over time to make it simple and clear, not just constantly add new features. The goal should be code that is "obviously right", as opposed to code that is so complicated that "I can't see any problems"*