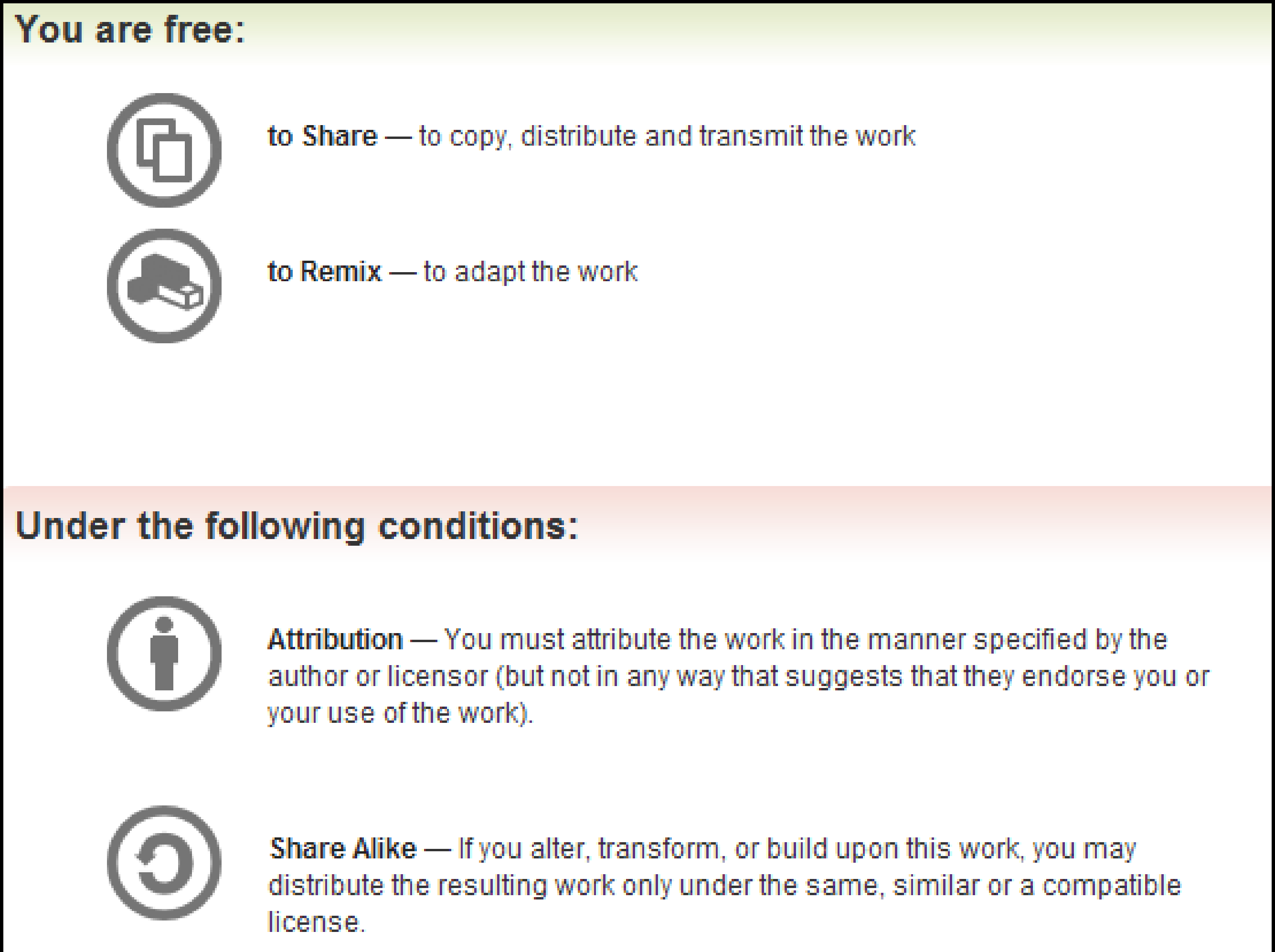
Hacking Techniques & Intrusion Detection

Ali Al-Shemery
arabnix [at] gmail

All materials is licensed under a Creative Commons "Share Alike" license.

- http://creativecommons.org/licenses/by-sa/3.0/



You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

# whoami

- Ali Al-Shemery
- Ph.D., MS.c., and BS.c., Jordan
- More than 14 years of Technical Background (mainly Linux/Unix and Infosec)
- Technical Instructor for more than 10 years (Infosec, and Linux Courses)
- Hold more than 15 well known Technical Certificates
- Infosec & Linux are my main Interests

Software Exploitation

Prepared by:

Dr. Ali Al-Shemery

Mr. Shadi Naif

Debugging Fundamentals for Pentesters

Outline – Part 2

- Debugger
  - GDB
  - Immunity Debugger
- Debuggers Offer?
- Popular Debuggers?
- Which to use?
- Example: Debugging auth.c using gdb

Debugger

- A computer program that lets you run your program, line by line and examine the values of variables or look at values passed into functions and let you figure out why it isn't running the way you expected it to.

Debuggers Offer?

- Debuggers offer sophisticated functions such as:
  - Running a program step by step (single-stepping mode),
  - Stopping (breaking) (pausing the program to examine the current state) at some event or specified instruction by means of a breakpoint,
  - Tracking the values of variables,
  - Tracking the values of CPU registers,
  - Attach to a process,
  - View the process's Memory map,
  - Load memory dump (post-mortem debugging),
  - Disassemble program instructions,
  - Change values at runtime,
  - Continue execution at a different location in the program to bypass a crash or logical error.

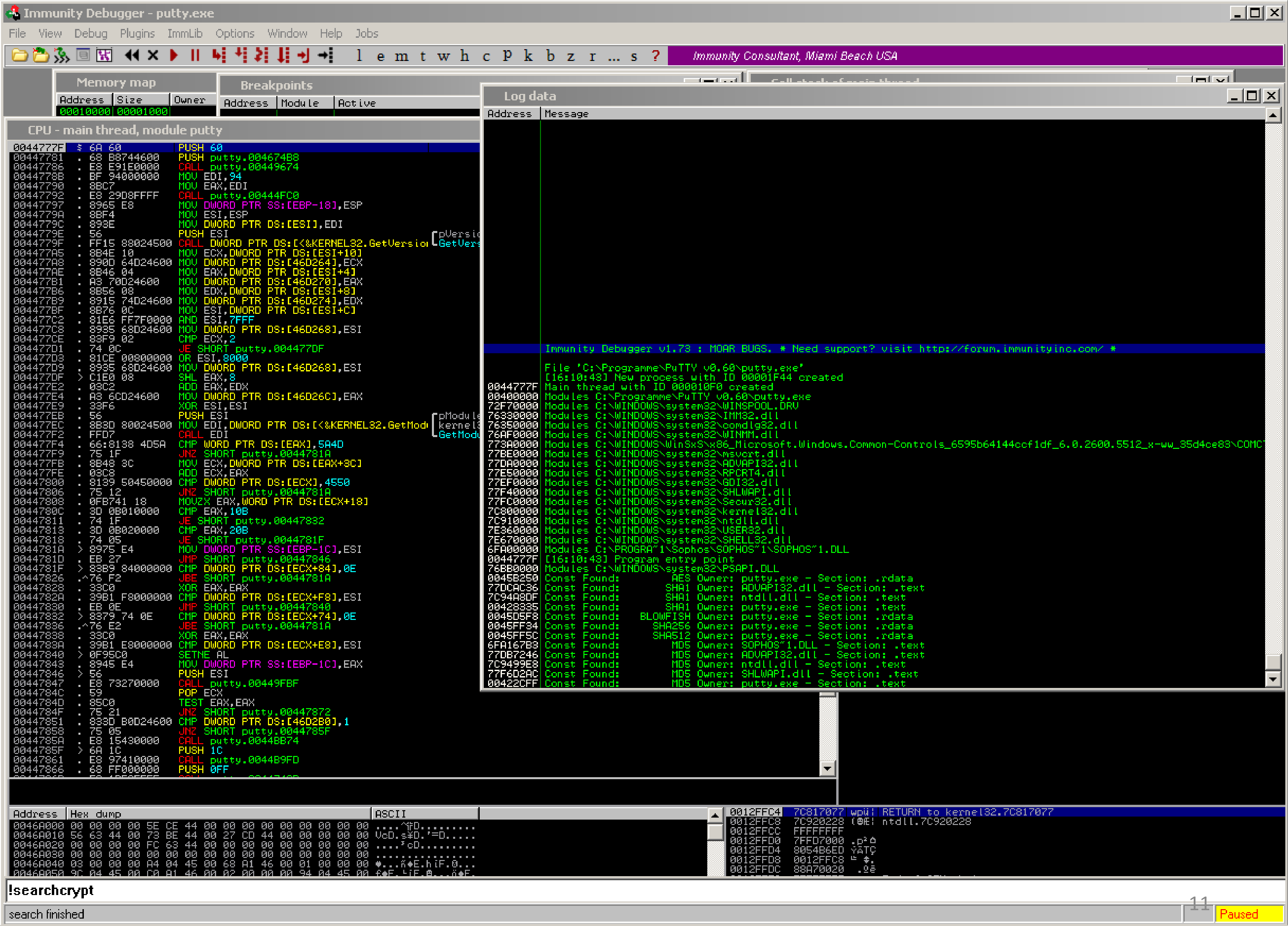
Popular Debuggers?

- GNU Debugger (GDB)
- Microsoft Windows Debugger (Windbg)
- OllyDbg
- Immunity Debugger
- Microsoft Visual Studio Debugger
- Interactive Disassembler (IDA Pro)

Immunity Debugger

- A powerful new way to write exploits, analyze malware, and reverse engineer binary files.
- It builds on a solid user interface with function graphing, and a large and well supported Python API for easy extensibility.

Did you read that? **Python** ☺


Immunity Debugger - putty.exe
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
Immunity Consultant, Miami Beach USA
Memory map
Breakpoints
Log data
CPU - main thread, module putty
Paused

```
Address  Size     Owner
00010000 00001000
```

```
Address  Module  Active
```

```
0044777F  $ 6A 60          PUSH 60
00447781  . 68 B8744600    PUSH putty.004674B8
00447786  . E8 E91E0000    CALL putty.00449674
0044778B  . BF 94000000    MOV EDI,94
00447790  . 8BC7           MOV EAX,EDI
00447792  . E8 29D8FFFF    CALL putty.00444FC0
00447797  . 8965 E8        MOV DWORD PTR SS:[EBP-18],ESP
0044779A  . 8BF4           MOV ESI,ESP
0044779C  . 893E           MOV DWORD PTR DS:[ESI],EDI
0044779E  . 56             PUSH ESI                                  [pVersio
0044779F  . FF15 88024500  CALL DWORD PTR DS:[<&KERNEL32.GetVersio   [GetVers
004477A5  . 8B4E 10        MOV ECX,DWORD PTR DS:[ESI+10]
004477A8  . 890D 64D24600  MOV DWORD PTR DS:[46D264],ECX
004477AE  . 8B46 04        MOV EAX,DWORD PTR DS:[ESI+4]
004477B1  . A3 70D24600    MOV DWORD PTR DS:[46D270],EAX
004477B6  . 8B56 08        MOV EDX,DWORD PTR DS:[ESI+8]
004477B9  . 8915 74D24600  MOV DWORD PTR DS:[46D274],EDX
004477BF  . 8B76 0C        MOV ESI,DWORD PTR DS:[ESI+C]
004477C2  . 81E6 FF7F0000  AND ESI,7FFF
004477C8  . 8935 68D24600  MOV DWORD PTR DS:[46D268],ESI
004477CE  . 83F9 02        CMP ECX,2
004477D1  . 74 0C          JE SHORT putty.004477DF
004477D3  . 81CE 00800000  OR ESI,8000
004477D9  . 8935 68D24600  MOV DWORD PTR DS:[46D268],ESI
004477DF  > C1E0 08        SHL EAX,8
004477E2  . 03C2           ADD EAX,EDX
004477E4  . A3 6CD24600    MOV DWORD PTR DS:[46D26C],EAX
004477E9  . 33F6           XOR ESI,ESI
004477EB  . 56             PUSH ESI                                  [pModule
004477EC  . 8B3D 80024500  MOV EDI,DWORD PTR DS:[<&KERNEL32.GetMod   |kernel3
004477F2  . FFD7           CALL EDI                                  [GetModu
004477F4  . 66:8138 4D5A   CMP WORD PTR DS:[EAX],5A4D
004477F9  . 75 1F          JNZ SHORT putty.0044781A
004477FB  . 8B48 3C        MOV ECX,DWORD PTR DS:[EAX+3C]
004477FE  . 03C8           ADD ECX,EAX
00447800  . 8139 50450000  CMP DWORD PTR DS:[ECX],4550
00447806  . 75 12          JNZ SHORT putty.0044781A
00447808  . 0FB741 18      MOVZX EAX,WORD PTR DS:[ECX+18]
0044780C  . 3D 0B010000    CMP EAX,10B
00447811  . 74 1F          JE SHORT putty.00447832
00447813  . 3D 0B020000    CMP EAX,20B
00447818  . 74 05          JE SHORT putty.0044781F
0044781A  > 8975 E4        MOV DWORD PTR SS:[EBP-1C],ESI
0044781D  . EB 27          JMP SHORT putty.00447846
0044781F  > 83B9 84000000  CMP DWORD PTR DS:[ECX+84],0E
00447826  .^76 F2          JBE SHORT putty.0044781A
00447828  . 33C0           XOR EAX,EAX
0044782A  . 39B1 F8000000  CMP DWORD PTR DS:[ECX+F8],ESI
00447830  . EB 0E          JMP SHORT putty.00447840
00447832  > 8379 74 0E     CMP DWORD PTR DS:[ECX+74],0E
00447836  .^76 E2          JBE SHORT putty.0044781A
00447838  . 33C0           XOR EAX,EAX
0044783A  . 39B1 E8000000  CMP DWORD PTR DS:[ECX+E8],ESI
00447840  > 0F95C0         SETNE AL
00447843  . 8945 E4        MOV DWORD PTR SS:[EBP-1C],EAX
00447846  > 56             PUSH ESI
00447847  . E8 73270000    CALL putty.00449FBF
0044784C  . 59             POP ECX
0044784D  . 85C0           TEST EAX,EAX
0044784F  . 75 21          JNZ SHORT putty.00447872
00447851  . 833D B0D24600  CMP DWORD PTR DS:[46D2B0],1
00447858  . 75 05          JNZ SHORT putty.0044785F
0044785A  . E8 15430000    CALL putty.0044BB74
0044785F  > 6A 1C          PUSH 1C
00447861  . E8 97410000    CALL putty.0044B9FD
00447866  . 68 FF000000    PUSH 0FF
```

```
Address  Message
         Immunity Debugger v1.73 : MOAR BUGS. * Need support? visit http://forum.immunityinc.com/ *

         File 'C:\Programme\PuTTY v0.60\putty.exe'
         [16:10:43] New process with ID 00001F44 created
0044777F Main thread with ID 000010F0 created
00400000 Modules C:\Programme\PuTTY v0.60\putty.exe
72F70000 Modules C:\WINDOWS\system32\WINSPOOL.DRV
76330000 Modules C:\WINDOWS\system32\IMM32.dll
76350000 Modules C:\WINDOWS\system32\comdlg32.dll
76AF0000 Modules C:\WINDOWS\system32\WINMM.dll
773A0000 Modules C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\COMC
77BE0000 Modules C:\WINDOWS\system32\msvcrt.dll
77DA0000 Modules C:\WINDOWS\system32\ADVAPI32.dll
77E50000 Modules C:\WINDOWS\system32\RPCRT4.dll
77EF0000 Modules C:\WINDOWS\system32\GDI32.dll
77F40000 Modules C:\WINDOWS\system32\SHLWAPI.dll
77FC0000 Modules C:\WINDOWS\system32\Secur32.dll
7C800000 Modules C:\WINDOWS\system32\kernel32.dll
7C910000 Modules C:\WINDOWS\system32\ntdll.dll
7E360000 Modules C:\WINDOWS\system32\USER32.dll
7E670000 Modules C:\WINDOWS\system32\SHELL32.dll
6FA00000 Modules C:\PROGRA~1\Sophos\SOPHOS~1\SOPHOS~1.DLL
0044777F [16:10:43] Program entry point
76BB0000 Modules C:\WINDOWS\system32\PSAPI.DLL
0045B250 Const Found:      AES Owner: putty.exe - Section: .rdata
77DCAC36 Const Found:     SHA1 Owner: ADVAPI32.dll - Section: .text
7C94A8DF Const Found:     SHA1 Owner: ntdll.dll - Section: .text
00428335 Const Found:     SHA1 Owner: putty.exe - Section: .text
0045D5F8 Const Found: BLOWFISH Owner: putty.exe - Section: .rdata
0045FF34 Const Found:   SHA256 Owner: putty.exe - Section: .rdata
0045FF5C Const Found:   SHA512 Owner: putty.exe - Section: .rdata
6FA167B3 Const Found:      MD5 Owner: SOPHOS~1.DLL - Section: .text
77DB7246 Const Found:      MD5 Owner: ADVAPI32.dll - Section: .text
7C9499E8 Const Found:      MD5 Owner: ntdll.dll - Section: .text
77F6D2AC Const Found:      MD5 Owner: SHLWAPI.dll - Section: .text
00422CFF Const Found:      MD5 Owner: putty.exe - Section: .text
```

```
Address  Hex dump                                         ASCII
0046A000 00 00 00 00 5E CE 44 00 00 00 00 00 00 00 00 00 ....^╬D.........
0046A010 56 63 44 00 73 BE 44 00 27 CD 44 00 00 00 00 00 VcD.s¥D.'=D.....
0046A020 00 00 00 00 FC 63 44 00 00 00 00 00 00 00 00 00 ....³cD.........
0046A030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0046A040 03 00 00 00 A4 04 45 00 68 A1 46 00 01 00 00 00 ♥...ñ♦E.hiF.☺...
```

```
0012FFC4  7C817077  wpü|  RETURN to kernel32.7C817077
0012FFC8  7C920228  (☻Æ|  ntdll.7C920228
0012FFCC  FFFFFFFF
0012FFD0  7FFD7000  .p²⌂
0012FFD4  8054B6ED  ÝÂTÇ
0012FFD8  0012FFC8  ╚ ↕.
0012FFDC  88A70020   .ºê
```

```
!searchcrypt
```

search finished

Which to use?

- IMO there is no exact answer to this question, it’s a matter of comfort!
- Choose the debugger comfortable for you and helps you with your debugging process.

Example – Auth.c

- What does auth.c do?
  - It takes the first argument from the command line,
  - It then passes this argument to a basic authentication function for checking,
  - If the argument is the correct password, it prints a success message,
  - If the argument isn't the correct password, it prints a failure message.
- There is a bug in the code!
- Let's try to discover it.

Auth.c using gdb

- gdb is a command line debugger, not very user friendly, but very powerful.
- First we need to compile auth.c, then run auth from within gdb.
- Use gcc:
  - gcc –ggdb –O0 auth.c -o auth

Auth.c using gdb - Cont.

- Start auth from within gdb:
  - gdb auth
- Run it with no arguments

  (gdb) run
- This will give us a Segmentation fault.
- The program now crashes!
- Let’s find what made the program crash.

Auth.c using gdb - Cont.

- We need to reconstruct the frames on the stack.
- The frames will show us the function calling sequence.
- Use the gdb command “backtrace”

```
(gdb) backtrace
```

- If you examine the output of the command you will find that the crash happened after calling the auth() function (frame #1)!

Auth.c using gdb - Cont.

- We need to check the instructions in the code where it has crashed.
- EIP points to the last instruction executed.
- We need to examine the memory and EIP:
- To do that we will use the "x" to display memory contents:

```
(gdb) x/5i $eip
```

- What does all that do????

Auth.c using gdb - Cont.

- “x” is used to display memory content in various formats,
- “i” is used for displaying instructions (disassembly),
- “5” is the number of instructions to display.

Check next slide for “x” formats.

“x” – Examine Memory

x / <count> <format> <unit>

Format	Description
x	hexadecimal
d	decimal
o	octal
t	binary
i	instructions
s	string
c	character
u	unsigned

Unit	Description
b	bytes
w	words (4 bytes)

Auth.c using gdb - Cont.

- The fault occurred at this instruction:

```
(gdb) x/10i $eip
cmp al, BYTE PTR [edx]
```

- cmp al, BYTE PTR [edx] compares al with the byte at the memory address stored within edx.
- There doesn’t seem to be an error here!
- Wait, let’s inspect the register edx and see what does it hold?

Auth.c using gdb - Cont.

- Let’s inspect the local variables and arguments.
- We can use the gdb “info locals” and “info args” commands:

```
(gdb) info locals
No symbol table info availabe

(gdb) info args
No symbol table info availabe
```

Auth.c using gdb - Cont.

- That means there is no debugging information. (Re-compile to resolve!)
- Quit gdb:

```
(gdb) q
```

- Recompile with debugging information enabled:

```
gcc –g auth.c –o auth
```

- The –g informs the compile to include symbolic debugging information within the compiled binary.

Auth.c using gdb - Cont.

- Let’s load auth in gdb again:

  ```
  $ gdb auth
  ```

- Now we can list the program code which is available from the debugging information.
- For that we use the gdb “list” command:

  ```
  (gdb) list
  ```

  - Press Enter if not all the code is shown.

Auth.c using gdb - Cont.

- If you remember the program crashed when calling the auth() function.
- Let us setup a break point. We can use the gdb "break" command:
  - (gdb) break 13
- Now run the program:
  - (gdb) run
- The process execution is suspended when it reaches our breakpoint. This is how we made gdb control the execution process!

Auth.c using gdb - Cont.

- Let us check the arguments values.
- We can use the gdb “print” command for inspecting variables.
  - (gdb) print argv[1]
- argv[1] is the argument passed to the auth function. And as you can see it’s value is 0x0 which is a NULL pointer!
- Continue the execution with the gdb command “continue”:
  - (gdb) continue

Auth.c using gdb - Cont.

- Now if we inspect the registers using the gdb command “info registers” we see that edx is holding 0x0 (the NULL pointer).
  - (gdb) info registers
  - (gdb) x/5i $eip

- This is what is causing the crash, as the program is comparing to a NULL pointer!

Auth.c using gdb – Summary

- Using gdb we managed to discover the bug in our code.
- All we need to do to solve this problem is check for the number of given arguments before calling the auth() function!

as simple as that!

Load Configurations

- Tired of always setting your GDB configurations?
- Use the -x file
- Add your configurations to a file such as gdb.config and then:
  - gdb –x gdb.config auth

Quit GDB Debugging

- Just press ‘q’ !

References (1)

- Papers/Presentations/Links:
  - ShellCode, http://www.blackhatlibrary.net/Shellcode
  - Introduction to win32 shellcoding, Corelan, http://www.corelan.be/index.php/2010/02/25/exploit-writing-tutorial-part-9-introduction-to-win32-shellcodeing/
  - Hacking/Shellcode/Alphanumeric/x64 printable opcodes, http://skypher.com/wiki/index.php/Hacking/Shellcode/Alphanumeric/x64_printable_opcodes
  - Learning Assembly Through Writing Shellcode, http://www.patternsinthevoid.net/blog/2011/09/learning-assembly-through-writing-shellcode/
  - Shellcoding for Linux and Windows Tutorial, http://www.vividmachines.com/shellcode/shellcode.html
  - Unix Assembly Codes Development, http://pentest.cryptocity.net/files/exploitation/asmcodes-1.0.2.pdf
  - Win32 Assembly Components, http://pentest.cryptocity.net/files/exploitation/winasm-1.0.1.pdf

References (2)

- Papers/Presentations/Links:
  - 64-bit Linux Shellcode, http://blog.markloiseau.com/2012/06/64-bit-linux-shellcode/
  - Writing shellcode for Linux and *BSD, http://www.kernel-panic.it/security/shellcode/index.html
  - Understanding Windows's Shellcode (Matt Miller's, aka skape)
  - Metasploit's Meterpreter (Matt Miller, aka skape)
  - Syscall Proxying fun and applications, csk @ uberwall.org
  - X86 Opcode and Instruction Reference, http://ref.x86asm.net/
  - Shellcode: the assembly cocktail, by Samy Bahra, http://www.infosecwriters.com/hhworld/shellcode.txt

References (3)

- Books:
  - Grayhat Hacking: The Ethical Hacker's Handbook, 3rd Edition
  - The Shellcoders Handbook,
  - The Art of Exploitation, 2nd Edition,
- Shellcode Repositories:
  - Exploit-DB: http://www.exploit-db.com/shellcodes/
  - Shell Storm: http://www.shell-storm.org/shellcode/
- Tools:
  - BETA3 - Multi-format shellcode encoding tool, http://code.google.com/p/beta3/
  - X86 Opcode and Instruction Reference, http://ref.x86asm.net/
  - bin2shell, http://blog.markloiseau.com/wp-content/uploads/2012/06/bin2shell.tar.gz