Dynamic Debugging

Hack Night - Fall 2015

Running Programs

- A series of bytes that your computer interprets as instructions
- Program is loaded into memory

What are debuggers?

- Help you get rid of bugs (duh)
- Latch onto your program and give you full control over memory of the program

GDB

- GNU DeBugger
- Most widely used debugger for Linux systems (also has broken support for Macs)
- All commands can be abbreviated to either one or two characters (“next” is the same as “n”)
- All numbers default to hex

```
[gdb-peda$ r
[----------------------------------registers-----------------------------------]
EAX: 0x1
EBX: 0xf7fc3ff4 --> 0x1a4d7c
ECX: 0xffffbb34 --> 0xffffbc75 ("/home/vagrant/Hack-Night/2015-Fall/Reverse_Engi
EDX: 0xffffbac4 --> 0xf7fc3ff4 --> 0x1a4d7c
ESI: 0x0
EDI: 0x0
EBP: 0xffffba98 --> 0x0
ESP: 0xffffba70 --> 0xffffffff
EIP: 0x80483e5 (<main+17>:      cmp    DWORD PTR [esp+0x1c],0x0)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80483d7 <main+3>:  and    esp,0xfffffff0
   0x80483da <main+6>:  sub    esp,0x20
   0x80483dd <main+9>:  mov    DWORD PTR [esp+0x1c],0x0
=> 0x80483e5 <main+17>: cmp    DWORD PTR [esp+0x1c],0x0
   0x80483ea <main+22>: je     0x80483fa <main+38>
   0x80483ec <main+24>: mov    DWORD PTR [esp],0x80484e0
   0x80483f3 <main+31>: call   0x80482f0 <puts@plt>
   0x80483f8 <main+36>: jmp    0x8048406 <main+50>
[--------------------------------------stack-------------------------------------]
0000| 0xffffba70 --> 0xffffffff
0004| 0xffffba74 --> 0xf7e51dc6 (add    ebx,0x17222e)
0008| 0xffffba78 --> 0xf7fc3ff4 --> 0x1a4d7c
0012| 0xffffba7c --> 0xf7e51e55 (<__cxa_atexit+53>:    add    esp,0x18)
0016| 0xffffba80 --> 0xf7feb280 (push   ebp)
0020| 0xffffba84 --> 0x0
0024| 0xffffba88 --> 0x8048419 (<__libc_csu_init+9>:   add    ebx,0x1bdb)
0028| 0xffffba8c --> 0x0
[------------------------------------------------------------------------------]
```

Symbols and Code

- Debugger tries to load as much helpful information as it can
- Symbols include function names, global variables and standard library names
- Source code can also be included
- Code can be seen with “list”

```
[gdb-peda$ list
1
2       #include <stdio.h>
3
4       int main()
5       {
6           int x = 0;
7
8           if (x) {
9               puts("You did it! :)");
10          } else {
[gdb-peda$
11              puts("Nope, you didn't do it :(");
12          }
13      }
gdb-peda$
```

Breakpoints

- Let you know when a piece of code is reached by the program
- GDB Syntax: bp *<address> or bp <function name>
- Delete them with “delete <breakpoint number>”

```
[gdb-peda$ b 6
 Breakpoint 2 at 0x80483dd: file crackme.c, line 6
[gdb-peda$ b *0x80483dd
 Breakpoint 3 at 0x80483dd: file crackme.c, line 6
[gdb-peda$ b main
 Breakpoint 4 at 0x80483dd: file crackme.c, line 6
 gdb-peda$
```

Stepping

- Lets you walk through a program step by step or instruction by instruction
  - next - next instruction (treats functions as single instruction)
  - step - basically next, but will go into functions
  - continue - continues code execution until breakpoint is hit

Viewing Registers

- Registers are the program's temporary variables…basically
- Reference register with a dollar sign before the register (ex. $rip)
- GDB Syntax: info registers (shows all registers)
- GDB Syntax: info registers $<register>

```
[gdb-peda$ info registers
eax            0x1      0x1
ecx            0xffffbb34       0xffffbb34
edx            0xffffbac4       0xffffbac4
ebx            0xf7fc3ff4       0xf7fc3ff4
esp            0xffffba70       0xffffba70
ebp            0xffffba98       0xffffba98
esi            0x0      0x0
edi            0x0      0x0
eip            0x80483dd        0x80483dd <main+9>
eflags         0x282    [ SF IF ]
cs             0x23     0x23
ss             0x2b     0x2b
ds             0x2b     0x2b
es             0x2b     0x2b
fs             0x0      0x0
gs             0x63     0x63
gdb-peda$
```

Print

- Prints something out in a specified format and stores it in a variable
  - print/d 0x10 —> $1 = 16
  - print/x $rip —> $2 = 0x7fff9129fd10
  - print/d 0x10 + 0x50 —> $5 = 96

Examine Memory

- Lets you look at everything that program has stored in its memory space
- GDB Syntax: x/<amt><data type> <address>
- Ex. x/40x $esp will show you the first 40 words (4 or 8 bytes) at the top of the stack

```
[gdb-peda$ x/40x $esp
0xffffba70:     0xffffffff      0xf7e51dc6      0xf7fc3ff4      0xf7e51e55
0xffffba80:     0xf7feb280      0x00000000      0x08048419      0xf7fc3ff4
0xffffba90:     0x08048410      0x00000000      0x00000000      0xf7e384d3
0xffffbaa0:     0x00000001      0xffffbb34      0xffffbb3c      0xf7fda858
0xffffbab0:     0x00000000      0xffffbb1c      0xffffbb3c      0x00000000
0xffffbac0:     0x0804821c      0xf7fc3ff4      0x00000000      0x00000000
0xffffbad0:     0x00000000      0x92f93178      0xaa855568      0x00000000
0xffffbae0:     0x00000000      0x00000000      0x00000001      0x08048320
0xffffbaf0:     0x00000000      0xf7ff06b0      0xf7e383e9      0xf7ffcff4
0xffffbb00:     0x00000001      0x08048320      0x00000000      0x08048341
gdb-peda$
```

Lets take a look

- do something cool

Cracking Programs

- Software Cracking - modification of software to remove or disable features which are considered undesirable by the person cracking the software, especially copy protection features (thanks wikipedia)

- Should never be used on paid software as this is illegal

Where to start?

- An annoying message that pops up
- Looking for functions called “check_if_registered” or something similar
- Calls to getting system time

Steps to “Fix” the Program

- Locate the code that is making the decision do the annoying thing
- Create assembly program to prevent the program from ever doing the annoying thing
- Replace existing code, with your new code
- …
- Enjoy your new software :3

Keygens

- A program that generates keys that the program accepts
- Trace the key you enter throughout the program until key verification code is found
- Reverse engineer key verifying code

Modify a program

- do something cool

Other Tools

- PEDA - Useful tools for using GDB when developing exploits
- lldb - Debugger for Macs
- windbag - Debugger for Windows
- Z3 - Math Solver