Hack Night

@ Boston University 2016

Week 2

Review of last week

- Stay if you want to learn and will put the time in
- Going to have a lot of fun
- Resources to learn from linked in the previous slides
  - https://github.com/buhacknight/main/blob/master/slides/Hack%20Night%201.pdf
- Set up both Linux and Windows environments (VM is best)
- Life of a binary file, and binary formats
- Static reverse engineering to crack applications
- $> Live @ `0night` solve of previous challenge, cmd1 on pwnable.kr

Overview

1. Command line and scripting review
2. Dynamic analysis tools, reverse engineering, and more
    a. Introduction to CTFs and live solve
3. Intro to the stack

Keywords: Dynamic analysis, reverse engineering, CTFs, the stack, command line, scripting

What is the shell?

- The shell takes your commands and gives them to the operating system
- Commonly used to start programs
- Common shells:
  - Bash/Dash
  - Zsh
  - Csh
  - Windows Shell
- Simple pseudocode implementation of a “shell”:

```
while True:
        cmd = receive_input();
        os.fork();
        os.exec(cmd);
```

Let's review the *nix shell and some tools

```
$> ssh bandit3@bandit.labs.overthewire.org
~$> cd inhere/
~/inhere$> ls -la
total 12
drwxr-xr-x 2 root    root    4096 Nov 14  2014 .
drwxr-xr-x 3 root    root    4096 Nov 14  2014 ..
-rw-r----- 1 bandit4 bandit3   33 Nov 14  2014 .hidden
~/inhere$> cat .hidden
pIwrPrtPN36QITSp3EQaw936yaFoFgAB
```

```
$> ssh bandit6@bandit.labs.overthewire.org
~$> cd /
/$> find . -size 33c -user bandit7 -group
bandit6 2>/dev/null
./var/lib/dpkg/info/bandit7.password
/$> cat ./var/lib/dpkg/info/bandit7.password
HKBPTKQnIay4Fw76bEy8PVxKEDQRKTzs
```

```
$> ssh bandit6@bandit.labs.overthewire.org
~$> cat data.txt | strings | grep ======
l========== the6
========== password
========== ism
========== truKLdjsbJ5g7yyJ2X2R0o3a5HQJFuLk
```

*nix shell review (2)

```
$> ssh bandit16@bandit.labs.overthewire.org
~$> nmap localhost -p 31000-32000

Starting Nmap 6.40 ( http://nmap.org ) at 2016-01-24 22:41 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.00046s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
31046/tcp open  unknown
31518/tcp open  unknown
31691/tcp open  unknown
31790/tcp open  unknown
31960/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
```

```
$> ssh bandit24@bandit.labs.overthewire.org
```

A daemon is listening on port 30002 and will give you the password for bandit25 if given the password for bandit24 and a secret numeric 4-digit pincode. There is no way to retrieve the pincode except by going through all of the 10000 combinations, called brute-forcing.

- Port 30002
- Bruteforce 4-digit pincode

```
$> Live @ 0night
```

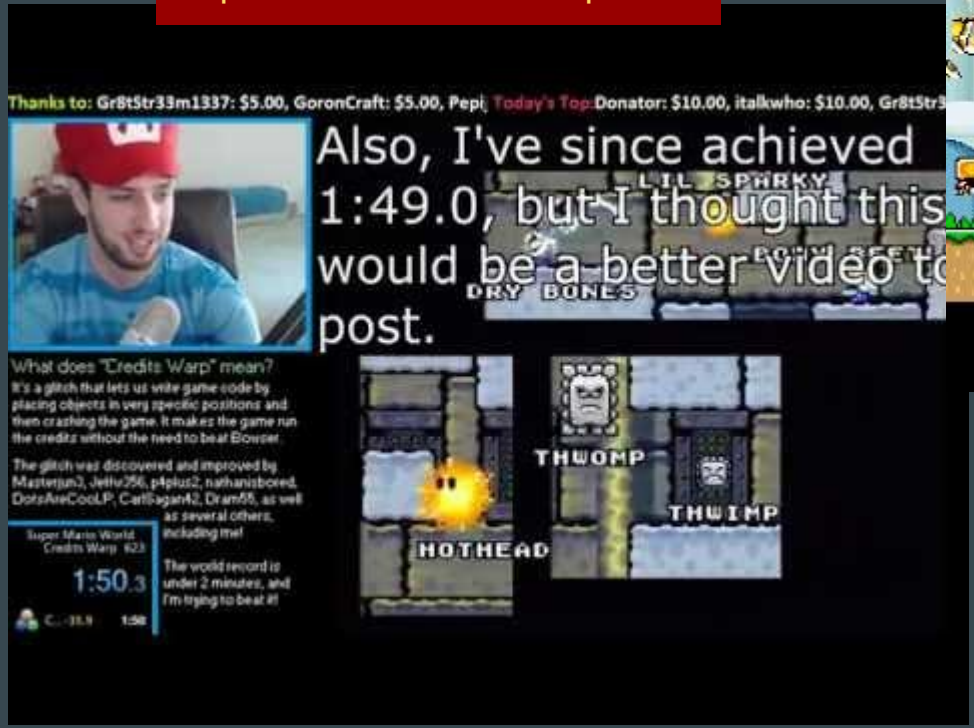
Basic *nix command line (CLI) tools

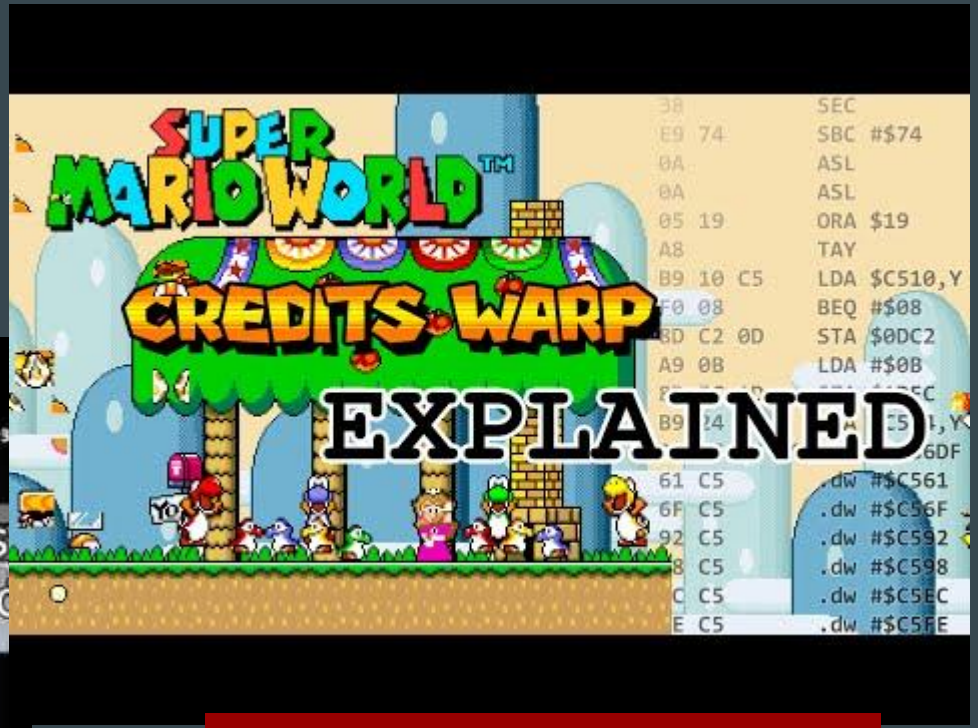
- Previous slides: `ssh`, `cd`, `cat`, `mv`, `cp`, `nc`, `grep`, `strings`, `find`, `ls`, `python`
- Some more:
  - `readelf` - Dump the contents of an ELF file
  - `objdump` - Simple disassemble an executable file
  - `printf` - Print any characters, even non-printable ones (\x00)
  - `xxd` - Make a hexdump or do the reverse
  - `hexdump` - Self explanatory
  - `file` - Guess what a file is
  - `tar` - Create/extract .tar.gz, .tar.bz2, .tar, etc. archives
  - `zip/unzip` - Create/extract .zip archives
  - `wget` - Get a file off the web
  - `scp` - Copy files from one computer to another
  - `git` - Version control your code or pull other code online
  - `curl` - Emulate a web browser
- https://www.win.tue.nl/~aeb/linux/hh/hh-3.html#ss3.1

Overview

1. Command line and scripting review
2. Dynamic analysis tools, reverse engineering, and more
   a. Introduction to CTFs and live solve
3. Intro to the stack

Super Mario Credits Warp

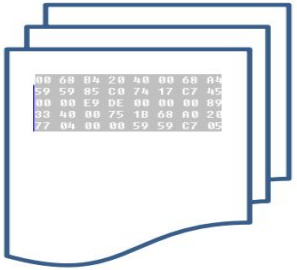




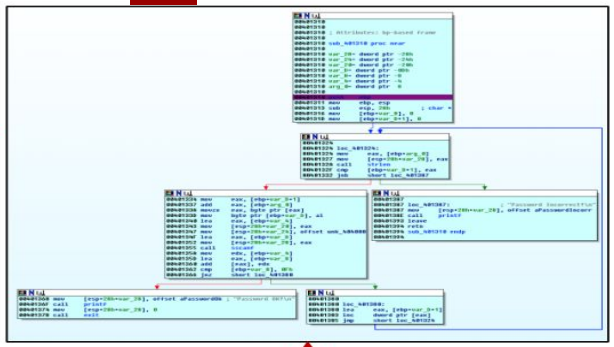
Super Mario Credits Warp Explained

RE Domain

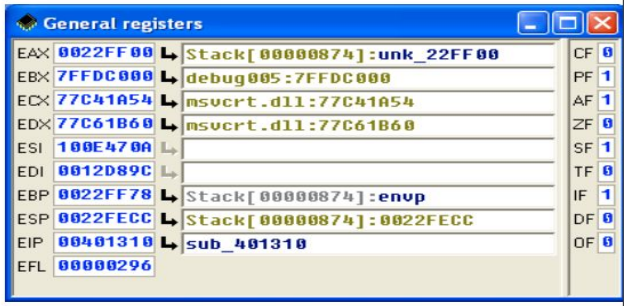
1 Libraries



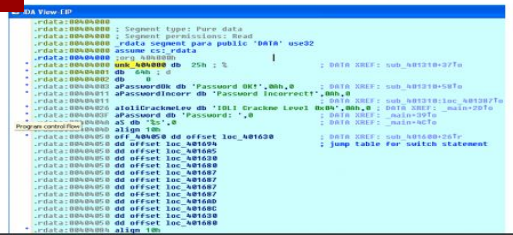
2 Code



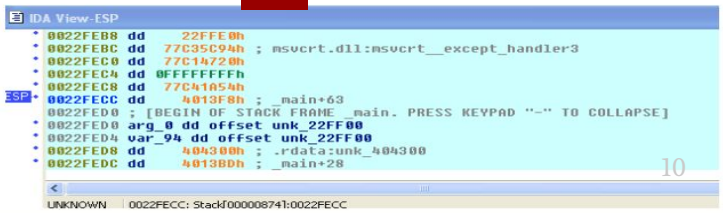
3 Registers



5 Other Memory



4 Stack



How to enter the RE domain

- gdb
  - The classic meat and potatoes
- IDA
  - The most popular and powerful disassembler out there
  - Decompiler support
- objdump
  - As simple as it gets, linear disassembly
  - Static analysis only
- radare2
  - Open source tool on the command line
  - In house support from developer, crowell
- hopper
  - More user friendly than IDA
  - Graphical
  - Designed for Mac

Windows specific:

- windbg
  - Window's version of gdb
- ollydbg

Pimp and master your GDB (½)

- gdb is a nice tool, but not very user friendly and requires A LOT of typing
  - Luckily it's expandable
- PEDA - Python Exploit Development Assistance for GDB
  - https://github.com/longld/peda
- Alternative reskins:
  - http://bit.ly/1ntm1rg

No PEDA...

```
(gdb) b main
Breakpoint 1 at 0x400602
(gdb) r
Starting program: /home/eugenek/code/eugenek-private/notes/hacknight-material/demo

Breakpoint 1, 0x0000000000400602 in main ()
(gdb)
```

Yes PEDA...

```
gdb-peda$ b main
Breakpoint 2 at 0x400602
gdb-peda$ r
Starting program: /home/eugenek/code/eugenek-private/notes/hacknight-material/demo
[----------------------------------registers-----------------------------------]
RAX: 0x4005fe (<main>:  push   rbp)
RBX: 0x0
RCX: 0x0
RDX: 0x7fffffffdce8 --> 0x7fffffffe0ba ("XDG_VTNR=7")
RSI: 0x7fffffffdcd8 --> 0x7fffffffe079 ("/home/eugenek/code/eugenek-private/notes/hacknight-material/demo")
RDI: 0x1
RBP: 0x7fffffffdbf0 --> 0x0
RSP: 0x7fffffffdbf0 --> 0x0
RIP: 0x400602 (<main+4>:        sub    rsp,0x30)
R8 : 0x7ffff7dd4e80 --> 0x0
R9 : 0x7ffff7dea560 (<_dl_fini>:        push   rbp)
R10: 0x7fffffffda80 --> 0x0
R11: 0x7ffff7a36dd0 (<__libc_start_main>:       push   r14)
R12: 0x4004d0 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffdcd0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x4005fd <myprint+64>:
   0x4005fe <main>:    push   rbp
   0x4005ff <main+1>:   mov    rbp,rsp
=> 0x400602 <main+4>:   sub    rsp,0x30
   0x400606 <main+8>:   mov    DWORD PTR [rbp-0x14],edi
   0x400609 <main+11>:  mov    QWORD PTR [rbp-0x20],rsi
   0x40060d <main+15>:  mov    QWORD PTR [rbp-0x28],rdx
   0x400611 <main+19>:  mov    DWORD PTR [rbp-0x10],0xdeadbeef
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffdbf0 --> 0x0
0008| 0x7fffffffdbf8 --> 0x7ffff7a36ec5 (<__libc_start_main+245>:       mov    edi,eax)
0016| 0x7fffffffdc00 --> 0x0
0024| 0x7fffffffdc08 --> 0x7fffffffdcd8 --> 0x7fffffffe079 ("/home/eugenek/code/eugenek-private/notes/hacknight-material/demo")
0032| 0x7fffffffdc10 --> 0x100000000
0040| 0x7fffffffdc18 --> 0x4005fe (<main>:      push   rbp)
0048| 0x7fffffffdc20 --> 0x0
0056| 0x7fffffffdc28 --> 0x9e9023172269900c
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 2, 0x0000000000400602 in main ()
gdb-peda$
```

Pimp and master your GDB (2/2)

- Using GDB w/ PEDA
  - `b <func_name>` – class breakpoint
    - b *0x123123 - break at address 0x123123
  - `pdisas` – pretty disassemble
  - `vmmap` – list mapped memory
  - `telescope 200` – view 200 words of data at the stack pointer
    - `telescope 0x123123` – view the data at 0x123123
  - `context all` – print registers, the stack, and the code
  - `procinfo` – print info from the procfs
  - `find 0xdeadbeef` – find the data 0xdeadbeef in any mapped mem region
  - `checksec` – list the security bits set on the binary
  - `readelf` – list ELF information
  - `pattern create 512` – generate a cyclic pattern of size 512

Master IDA

- IDA is immensely powerful for static and dynamic RE
  - space bar - Toggle text/graphic view
  - n - Rename variable
  - ; - Comment
  - x - go to cross reference
  - ctrl+x - list all cross references
  - tab - jump to pseudocode
  - p - define a procedure (function)
  - d - define as data
- The IDA Pro Book (Chris Eagle)

```
[ IDA Pro shortcuts ]
Navigation:
Enter     :   Jump to operand    |   ESC       :   Jump to previous position
G         :   Go to address      |   Ctrl+L    :   Jump by name
CTRL+P    :   Jump to function   |   X         :   xref
CTRL+E    :   Jump to entry point

Search
Alt+C     :   Next code          |   Ctrl+D    :   Next data
Alt+I     :   Immediate value    |   Crtl+I    :   Next immediate value
Alt+T     :   Text               |   Ctrl+T    :   Next text
Alt+B     :   Sequence of bytes  |   Ctrl+B    :   Next sequence of bytes

Graphing
F12       :   Flow chart         |   Ctrl+F12  :   Function calls

Subviews
Shift+F4  :   Name               |   Shift+F3  :   Functions
Shift+F12 :   Strings            |   Shift+F7  :   Segments

Debugger
F9        :   Start              |   Ctrl+F2   :   Stop process
F7        :   Step into          |   F8        :   Step over
Ctrl+F7   :   Run until return   |   Ctrl+Alt+B:   List breakpoints

Other
C         :   Code               |   D         :   Data
U         :   Undefine           |   N         :   Rename
Shift+;   :   Enter comment      |   ;         :   Enter repeatable comment
P         :   Create function    |   Alt+P     :   Edit function
E         :   Set function end   |   Y         :   Declare function type
M         :   Member enumeration |   Shift+F2  :   Run script
```

IDA Features

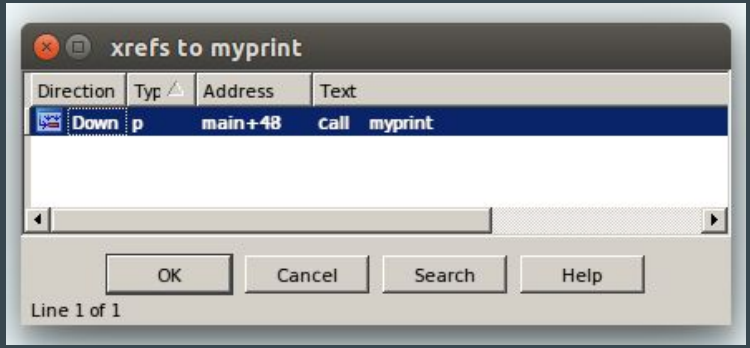
Graphical branch view

```
; Attributes: bp-based frame

public myprint
myprint proc near

var_C= dword ptr -0Ch
arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
sub     esp, 28h
mov     [ebp+var_C], 0
jmp     short loc_80484A6
```

```
loc_80484A6:
mov     eax, [ebp+var_C]
cmp     eax, [ebp+arg_4]
jl      short loc_804848C
```

```
loc_804848C:
mov     edx, [ebp+var_C]
mov     eax, [ebp+arg_0]
add     eax, edx
movzx   eax, byte ptr [eax]
movsx   eax, al
mov     [esp], eax      ; c
call    _putchar
add     [ebp+var_C], 1
```

```
nop
leave
retn
myprint endp
```



List xrefs to functions
Also known as “what the hell calls this”

IDA Features 2

Decompiler

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 20h
mov     dword ptr [esp+14h], 0DEADBEEFh
mov     dword ptr [esp+18h], 0C001C0DEh
mov     dword ptr [esp+1Ch], offset aHelloWorld ; "Hello world!"
mov     eax, [esp+1Ch]
mov     [esp], eax      ; s
call    _puts
mov     eax, [ebp+argv]
add     eax, 4
mov     eax, [eax]
mov     [esp], eax      ; nptr
call    _atoi
mov     [esp+4], eax
mov     eax, [esp+1Ch]
mov     [esp], eax
call    myprint
mov     eax, 0
leave
retn
main endp
```

Original

```
int main(int argc, char*  argv[], char* envp[]) {
        int first_int = 0xdeadbeef;
        int second_int = 0xc001c0de;

        char * first_str = "Hello world!";

        printf("%s\n", first_str);

        myprint(first_str, atoi(argv[1]));

        return 0;
}
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // ST04_4@1

  puts("Hello world!");
  v3 = atoi(argv[1]);
  myprint((int)"Hello world!", v3);
  return 0;
}
```

Wow!

radare2 - commandline disassembler, debugger, etc.

- free, open source, powerful, but steep learning curve
- https://github.com/radare/radare2
- http://radare.org/r/docs.html
- http://radare.org/r/talks.html
- #radare on irc.freenode.net

```
[0x00003c9c 255 /usr/bin/r2]> pd $r @ sym..L94+4869 # 0x3c9c
            0x00003c9c    e970effffff   jmp 0x100002c11 ; (fcn.00002390) ;[1]
            0x00003ca1    8bbba4010000  mov edi, [ebx+0x1a4]
            0x00003ca7    8b74247c      mov esi, [esp+0x7c]
            0x00003cab    8b8424940000. mov eax, [esp+0x94]
            0x00003cb2    c7442404000.  mov dword [esp+0x4], 0x0
            0x00003cba    890424        mov [esp], eax
            0x00003cbd    e81ee2ffff    call 0x100001ee0 ; (sym.imp.r_core_prompt) ;[2]
               sym.imp.r_core_prompt()
            0x00003cc2    85c0          test eax, eax
            0x00003cc4    0f8eaa000000  jle 0x3d74 ;[3]
            0x00003cca    85f6          test esi, esi
            0x00003ccc    7408          jz 0x3cd6 ;[4]
            0x00003cce    893424        mov [esp], esi
            0x00003cd1    e84ae4ffff    call 0x100002120 ; (sym.imp.r_th_lock_enter) ;[5]
               sym.imp.r_th_lock_enter()
            0x00003cd6    8b8424940000. mov edx, [esp+0x94]
            0x00003cdd    891424        mov [esp], edx
            0x00003ce0    e80be4ffff    call 0x1000020f0 ; (sym.imp.r_core_prompt_exec) ;[6]
               sym.imp.r_core_prompt_exec()
            0x00003ce5    8984249c000.  mov [esp+0x9c], eax
            0x00003cec    83c001        add eax, 0x1
            0x00003cef    0f8424010000  jz 0x3e19 ;[7]
            0x00003cf5    85f6          test esi, esi
            0x00003cf7    7408          jz 0x3d01 ;[8]
            0x00003cf9    893424        mov [esp], esi
            0x00003cfc    e87fe2ffff    call 0x100001f80 ; (sym.imp.r_th_lock_leave) ;[9]
               sym.imp.r_th_lock_leave()
            0x00003d01    83bc2498000.  cmp dword [esp+0x98], 0x0
            0x00003d09    745b          jz 0x3d66 ;[?]
            0x00003d0b    8b8424980000. mov eax, [esp+0x98]
            0x00003d12    890424        mov [esp], eax
            0x00003d15    e806e5ffff    call 0x100002220 ; (sym.imp.r_th_wait_async) ;[?]
               sym.imp.r_th_wait_async()
            0x00003d1a    85c0          test eax, eax
            0x00003d1c    7548          jnz 0x3d66 ;[?]
            0x00003d1e    8b07          mov eax, [edi]
            0x00003d20    c7442408120.  mov dword [esp+0x8], 0x12
```

Using GDB and IDA

NYU's CSAW CTF 2013 | Reversing 400: Keygenme https://github.com/crowell/keygenme

$> live @ 0night ->
https://www.eugenekolo.com/blog/a-walk-through-the-binary-with-ida/

Reversing : keygenme
someone has leaked a binary from an activation server.
can you crack the keygen algorithm for me?

using the ELF provided, reverse the keygeneration algorithm.
The server listening at raxcity.com on port 2000 will ask you for

the passwords of various usernames. If you can provide 10 passwords, you might get a nice flag :-)

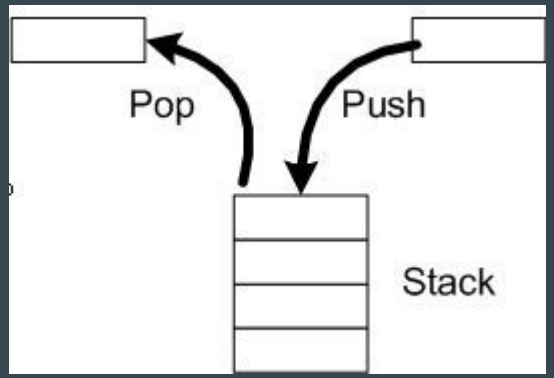
- ELF binary
- Reverse engineering
- Remote server
- Must get 10 correct in a row

Overview

1. Command line and scripting review
2. Dynamic analysis tools, reverse engineering, and more
   a. Introduction to CTFs and live solve
3. Intro to the stack

The stack

- An abstract data structure that is similar to a real world “stack” of magazines
  - The last object in, is the first object out, LIFO
  - Two operations to alter the stack, push and pop
    - push something onto the top of the stack
    - pop the item at the top of the stack off
  - Computer scientists love them
- LIFO property allows simple and efficient way to backtrack to previous function caller
- Used to pass local variables to functions
- Used to keep track of scope



Stack from 100 feet away

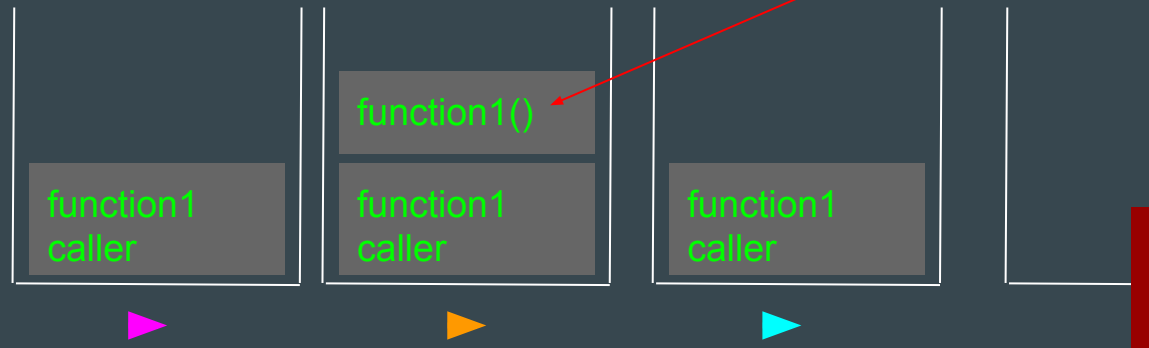
```
function1() {
        function2();
    do something after calling function2….
}
```

```
function2() {
    do something…
}
```

CPU: “I’m done. Where do I go now?”

Instruction Pointer (EIP)

Sequence is:
pink -> orange -> cyan -> red



CPU: “Okay we’re back in the function1 caller. Whoever that may be.”

More complete stack view

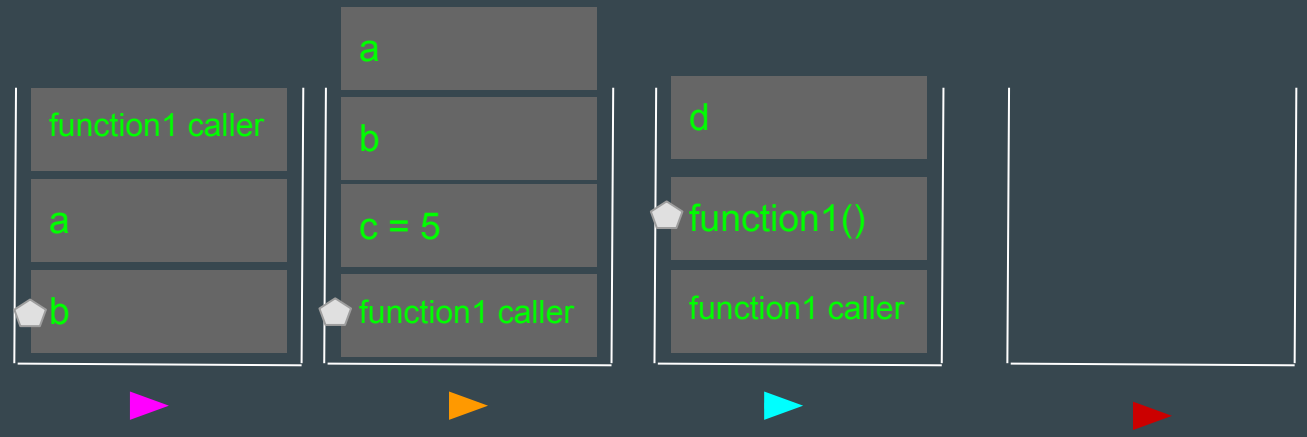
```
function1(a, b) {
        c = 5;
        d = function2(a, b, c);
        return d;
}
```

```
function2(a, b, c) {
        d = a + b + c;
        return d;
}
```

CPU: “I’m done. Good thing I saved where to go in the base pointer.”

- Base Pointer (EBP)
- Instruction Pointer (EIP)

Sequence is:
pink -> orange -> cyan -> red



EAX: d
(Return register)

Absolutely complete stack

$> Live @ 0night

(if time allows...)

Resources

- Formally learn reverse engineering:
  - http://beginners.re/Reverse_Engineering_for_Beginners-en-lite.pdf ~100 pages
    - Great, quick, touch on everything
  - Practical Reverse Engineering ~ 350 pages
    - The best book on reversing out there, touch on everything in depth
- Quickly learn x86:
  - https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture
- Cheat sheets:
  - http://disi.unitn.it/~bianchi/content/TeachingMaterial/Arch2013/X86_Win32_Reverse_Engineering_Cheat_Sheet.pdf
  - http://r00ted.com/cheat%20sheet%20reverse%20v5.png

Review/Practice

- C code review challenge: ssh lotto@pwnable.kr -p2222 (pw:guest)
- Dynamic analysis challenge: http://pwnable.kr/bin/flag
  - Hint: It's obfuscated and packed

Some wargame videos at the end of this powerpoint.

Stay tuned next week for...

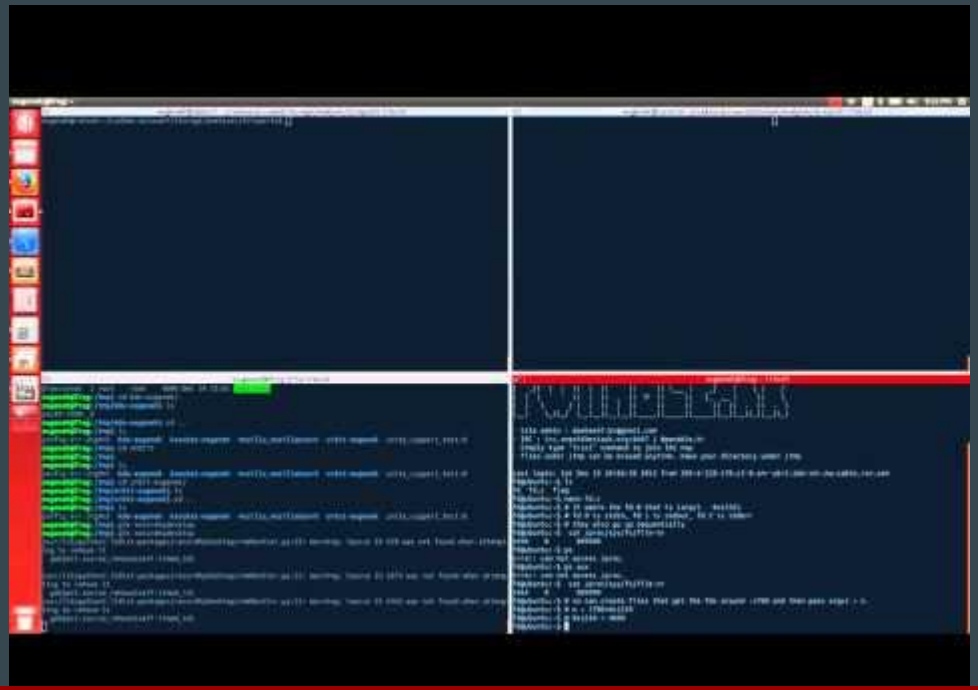
- Smashing the stack (oh my!)
  - Redirecting program flow
  - Shellcode
  - http://insecure.org/stf/smashstack.html
- Memory corruption
- Exploiting with libc (format strings, ret2libc, GOT/PLT, ...)

Frank Wang from Cybersecurity Factory, a summer program for security startups at MIT will be presenting!

FREE PIZZA NEXT WEEK!

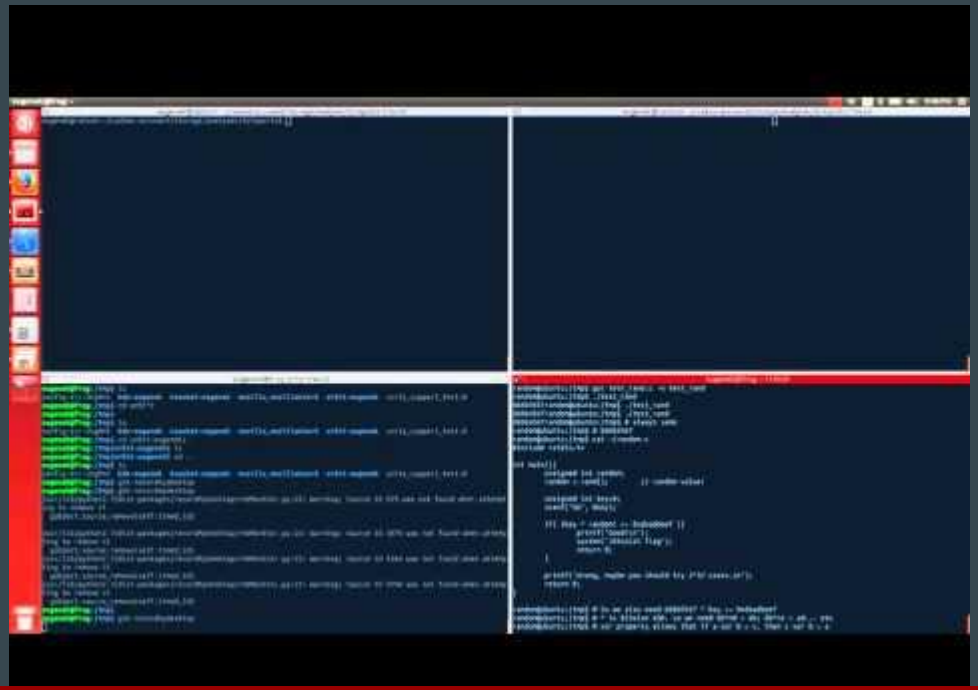
__libc_fini

Let's wargame



fd (pwnable.kr) - https://www.youtube.com/watch?v=gEpUhXLcnMM

Let’s wargame



random (pwnable.kr) - https://www.youtube.com/watch?v=9kNd3qxbBjw