ECE455: Week 5 notes

Jonathan Lam

10/06/21

Contents

1 Exploiting format string vulnerabilities **1**
1.1 Overview 1

2 Hacking blind **3**

3 Smashing the stack for fun and profit **4**

4 Questions **5**

5 Review **5**

1 Exploiting format string vulnerabilities

scut/team teso (2001)

1.1 Overview

	Buffer Overflow	Format String
public since	mid 1980's	1999
danger realized	1990's	2000
# exploits	few thousand	few dozen
considered as	security threat	programming bug
techniques	evolved and advanced	basic techniques
visibility	sometimes very difficult to spot	easy to find

- Impacted software: ftpd, rpc.statd, telnetd, php3, screen, chpass, fstat
- Easy to discover, even in binary format (look at "argument deficiencies")

- If an attacker is able to provide the format string to an ANSI C format function in part or in whole, a format string vulnerability is present
- Don't print user-supplied string as format string; use `%s`
- Relevant functions:
  - `*printf` family
  - `setproctitle` – set `argv[]`
  - `syslog`
  - `err*`, `verr*`, `warn*`, `vwarn*`
- Only the calling function knows how many parameters it pushes on the stack (callee doesn't know)
- **Channeling problem**: two different types of information differentiated by a special character, user-supplied strings
- Easy to crash a program with custom format strings
- If we can see the reply of the format function we can extract information about the program
  - View the stack by printing it out in hex
  - The format function modifies the stack pointer, we can use that to get the stack address
  - Sample program:

```
/* gcc -o printf -m32 -fno-stack-protector -g printf.c */
#include <stdio.h>

int main() {
  char buf[] = "AAA3BAA1C___D___%d.%8x.%8x.|%c%c%c%c|....     \n";

  printf(buf, 123456789);

  return 0;
}
```

  - As you can see, might need some padding to line up to a four-byte boundary

  - We can arbitrarily change the address in the string and dump the contents at that address using `%s`: can coredump a program

- Like in buffer overflow, we want to modify the IP: can do this by changing the format string in a `sprintf` to overflow the buffer (and perhaps overwrite the IP)

- Even with fixed-length strings, we can write a small number (using `%n`) to a determined location

  - Have some degree over number with a length format specifier (but only small numbers); can do this for each byte

2 Hacking blind

Bittau et al., 2013

- Standard **return-oriented programming** (ROP) requires knowledge of where addresses are in the binary

- **Blind ROP** (BROP) is proposed in this paper and requires:

  - A service that restarts after a crash
  - A stack vulnerability

- BROP leaks a single bit of information based on whether a process crashed or not when given a particular input string, and gets enough information to write its own binary to the network

- Works against ASLR, NX, and stack canaries

- ROP is usually necessary on modern systems due to NX preventing code injection

- Two new techniques:

  - Generalized stack reading: technique known for leaking canaries also leak saved return addresses to defeat ASLR even when PIE is used
  - Blind ROP: remotely locates ROP **gadgets**:
    * Gadget: Carefully chosen machine instruction sequences that are already present in the machine's memory, usually ending in a return instruction

3 Smashing the stack for fun and profit

Aleph One (1996)

- Static arrays/buffers at data segment at load-time
- Dynamic variables on the stack at run-time
- We are mostly concerned with stack-based (dynamic) buffer overflows
- Text is at low memory; stack at high memory; data in between
- ESP refers to top of stack (lowest memory address); EBP refers to a fixed point in the stack (beginning/base of current stack frame)
- Function prologue: (save EIP – part of call instruction); save EBP, write new EBP
- Memory is only addressible in multiples of the word size, so each variable is aligned to a word boundary
- View of memory:

```
bottom of                                        top of
memory                                           memory
         buffer2  buffer1  sfp  ret  a  b  c
<------ [       ][       ][   ][   ][ ][ ][ ]
top of                                       bottom of
stack                                            stack
```

- `JMP~/~CALL` combination to get the address of a string in an arbitrary piece of memory
- The stack will start at the same address for every program (in the days before ASLR/PIE?)
- Program to print stack pointer:

```
// sp.c
unsigned long get_sp(void) {
  __asm__("movl %esp,%eax");
}
void main() {
  printf("0x%x\n", get_sp());
}
```

- "Use the source d00d"

4 Questions

- ASLR vs. PIE?

5 Review

- STRIDE: spoofing (authenticity), tampering (integrity), repudiability (non-repudiability), information disclosure (confidentiality), denial of service (availability), elevation of privilege (authorization)
  - These threats can basically cause all or any of these, depending on the program; at the most basic level crashing the program (DoS)
- Threat model: vulnerability, threats, malicious actors, risk
- Smashing the stack for fun and profit: buffer overflows: overflowing a fixed-size dynamic buffer and injecting shellcode (difficult nowadays with ASLR and NX)
- Return to libc (ROP): buffer overflows with NX bit: cannot inject shellcode but instead use gadgets from libc
- Exploiting format string vulnerabilities: sometimes works like a buffer overflow, but doesn't even need an overflow; can "walk the stack" (`%u`) and either dump it (`%s`) or write to it (`%n`) at arbitrary locations
- Hacking blind (BROP): more advanced method that defeats ASLR, NX, and stack canaries; requires a program with stack vulnerabilities and that automatically restarts; doesn't require having the binary at hand (as you usually need with ROP); instead extracts information slowly until you can send the entire binary over the network to perform regular ROP