Software Security 1

Flipped Classroom

> 30.10.2024

Agenda

- Quick refresher on **Stack-based Buffer Overflow**
- Hunting down **Stack Canaries**
- **ASLR** defeating may not be as hard as pronouncing it
- Never trust user input - A primer on **Format String** vulnerabilities
- **ROP**in' like a cowboy - Ridin' those gadgets all the way to a shell!

Recap: Stack-based Buffer Overflow

Recap: Stack-based Buffer Overflow

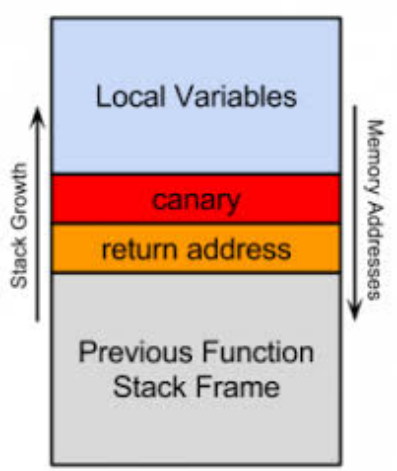
- The objective is to overwrite the return address.
- Do you remember deprecated? [Live Demo]

```c
#include <stdio.h>

int main(void)
{
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);

    char name[16];
    printf("What is your name? ");
    gets(name); // What is the problem here?

    printf("Hello %s!\n", name);
}
```

Stack Canaries

Stack Canaries

- Writes a guard value in the stack and tries to check if it is unmodified before returning [Live Demo]
- Named after **coal mine's** canary



Defeating Stack Canaries - some methods

- Leaking the Canary
- Brute-forcing [Live Demo]
- Skipping/Jumping

Defeating Stack Canaries - Brute-Forcing

```
int main() {
  char buf[16];
  while(true) {
    if(fork()) {
          wait(0);
    } else {
      read(stdin, buf, 128);
      return;
    }
  }
}
```

Adress Space Layout Randomization - ASLR

Defeating ASLR - some methods

- Leaking an address of the stack (just like in `deprecated`)
- Brute-force
- Partial Overwrite [Live Demo]

Defeating ASLR - Partial Overwrite

- Memory pages are aligned at `0x1000` boundaries
  - It can be mapped at: `0xCAFE0000` or `0x00403000` or `0x12345000`
  - But not at `0xCAFECAFE`
- If we overwrite the two least significant bytes of a pointer, we only have to brute force one nibble (= 16 values) to successfully redirect the pointer to another location on the same memory page
- This gives us limited control
- On little endian, these are the first two bytes that we overwrite! [Live Demo]

Format Strings

Format Strings

`printf(const char *format, ...);`

- printf() takes a variable number of arguments

```
printf("foobar");
printf("foo%s", bar);
printf("%d", 42);
printf("%d / %d = %f", 13, 37, 13.0/37.0);
```

- printf knows the number of arguments it should read based on the format string

Format Strings - Reading data

- Specifier characters
  - `%c` - single char
  - `%s` - null-terminated string
  - `%p` - pointer
  - `%d`, `%i` - signed 4 byte integer
  - `%o`, `%u`, `%x`, `%X` - unsigned 4 byte integer
- Length specifier
  - `hh`, `h`, `l`, `ll`, and more ...

Format Strings - Writing data

- Using `%n` data can be written
- `%n` can also use length and position specifiers.
- [Live Demo]

Return Oriented Programming

Return to libc

- x86 arguments are passed to the stack, so it would be the case that we could align the stack and modify its arguments
- For x64, can still do a similar of exploitation but ignoring the arguments

So what is actually ROP?

- Reusing fragments of instructions to make the program do what you want
- It's like shellcoding, but you only can reuse code existing in the program already

ROP Gadgets

- Useful tools: pwndbg, rop, ropper, pwntools

ROP Chaining

- The act of chaining together gadgets in order to get your intended outcome