Software Security 1

Exercise Session

> 07.11.2024

Assignment 2

- Echo - **Format String**
- Echo 2 - **Format String + ROP + Pain**
- Coal Mine (`coalmine`) - **Stack Canary + Return 2 libc (ROP)**
- Dropped - **ROP**
- Nuggets - **ROP**
- Peeky Blinders (`peeky-blinders`) - **Shellcoding**
- Over 9000 (`over9000`) - **Integer Overflow + Return 2 Win**

Echo (1/2)

- During `make`, the compiler will warn about `printf` being misused - **Format String**.
- The regex does not check for positional specifiers (e.g. `%42$n`).
- It's possible to leak pointers to the libc (`%35$p`) and the binary (`%37$p`).
- `checksec` will tell `Partial RELRO` .This means we can overwrite `GOT` entries and replace `printf` with `system`.
- Use `%hhn` (byte) and `%hn` (short) writes. Sort the writes by value.

Echo (2/2)

- You can also find your format string on the stack, e.g. via `%14$lx`. This makes it a lot easier to write to arbitrary addresses with `%n`.
- Otherwise, you have to find a pointer chain on the stack (use the first pointer to partially overwrite the second pointer, then use the second pointer for the write).
- You could use `pwnlib.fmtstr` to help you in this task.

Echo 2

- During `make`, the compiler will warn about `printf` being misused - **Format String**.
- Make it long enough that `fmt_len` is large, and we don't have to reallocate later when we've rewritten the `fmt` pointer.
- We can't put addresses in the format string since we can't reach it via `%...$n`.
- Use a pointer chain to overwrite the `fmt` pointer to point to the stack.
- We use `(stack + 0x58)` —▸ `(stack + 0x148)` —▸ ..., i.e. `%17` —▸ `%47` —▸ ...
- It's possible to leak pointers to the binary `(%15$p)` stack `(%13$p)` and libc `(%17$p)`.

Coal Mine (`coalmine`)

- We need to find a way to defeat **Stack Canaries**. By the fork server nature of the program, **Brute-Force** technique is feasible.
- Use your local environment for reference on the offsets for the stack canary and libc.
- Each time the child process dies, neither the stack canary nor the libc address will change. Use this behavior to leak these values.
- As `-fstack-protector` is enabled, `__stack_chk_fail` verification will crash your program when you hit the canary. You can use this to try to guess the canary byte by byte.
- You can also use the same approach to brute-force and get libc address. You can take educated guesses by supposing that the address is probably at `0x7f ?? ?? ?? ?? ?? ?0 00` to `0x0`.

Dropped

- The bug is `read()` with a wrong buffer size.
- `checksec` outputs `No canary` and `No PIE`.
- This time, the **stack isn't executable**, and **ASLR is enabled**.
- Write an ROP chain to the stack.
- You can build your ROP chain by hand or use `pwnlib.rop`.
- There is no address leak, so you are limited to gadgets in the (non-PIE) binary (you can use `pwndbg`'s `leakfind` to find an address leak).
- There are `pop rdi` and `pop rsi` gadgets in the binary.

Nuggets

- Very similar to the example given in class. But now it's linking to `libz.a`.
- You can *"bring your own `/bin/sh`"* to `.bss` section.
- There are gadgets that enable you to `write` and `read` from memory.

Peeky Blinders (`peeky-blinders`)

- The flag only changes when you spawn a new instance. So you could abuse it by trying to read the flag byte by byte, using some sort of computation to check if the result matches or not.
- The flag matches the following regular expression: `softsec\{[0-9a-zA-Z_-]{64}\}`. You don't need brute-force all ASCII. You could also use binary search.
- **Note:** If you disable coredump locally, it will be faster.

Over 9000 (`over9000`)

- Integer Overflow challenge. There is an `if` statement that verifies if the buffer you are writing is inside a defined limit.
- Later, the `fgets` inside `measure_power_level` substracts the number by one. If you input `INT_MIN` when it gets to `INT_MIN - 1`, it will result in `INT_MAX`.
- After getting to the buffer, you could do multiple exploits. Maybe the easiest way is to return to `its_over_9000()`.