Lab 9: The Cryptic Stack

1 Introduction

The suspect isn't talking, Detective. We have to make them slip up.

You have proven you can pick locks (Network) and assume identities (Web). But the deepest secrets are not hidden behind passwords; they are buried in the raw memory of the machine itself.

The Baker Street Society has intercepted a piece of software used by the target organization's courier service. It is a simple program—a digital clerk that takes a name and prints it back. But our intelligence suggests this clerk is easily overwhelmed. If you speak too much, and too fast, the clerk forgets what it was doing and starts following *your* instructions instead.

Your mission is **Binary Exploitation**. You must analyze this program, find the breaking point (Buffer Overflow), and rewrite the execution flow to force it to reveal the secret **Flag** hidden within its code.

Crash the program. Control the Instruction Pointer. Seize the data.

2 The PI Code (Binary Measures)

You are no longer working with high-level languages. You are working with the raw metal.

Precision is Paramount. In binary exploitation, a single byte out of place means the difference between a successful hack and a segmentation fault. You must measure your inputs exactly.

The Tool of Choice. While raw socket programming is possible, a master craftsman uses the best tools. The Society highly recommends utilizing **pwntools**, a Python framework designed specifically for this kind of digital surgery. It handles the raw bytes so you can focus on the logic.

3 The Setup

This operation requires two stages: local analysis and remote exploitation.

3.1 Establishing the Connection

You will access the Jump Host using the SSH protocol. To reach the server, you must tunnel through the Jump Host just as you did in the previous mission:

```
ssh -L 4331:130.208.246.231:4331 -L 4332:130.208.246.231:4332 -L
    4333:130.208.246.231:4333 -L 4334:130.208.246.231:4334
                <username>@130.208.246.239 -N
```

3.2 The Artifact (Local Analysis)

First, download the vulnerable binary provided on The Artifact Directory

3.3 The Target (Remote Exploitation)

Once your exploit is ready, you must deploy it against the live server where the real flags are stored.

Once the tunnel is active, the target services will be listening on your machine at:

- **Level I:** `localhost:4332`
- **Level II:** `localhost:4333`
- **Level III:** `localhost:4334`

4 The Toolkit

To manipulate memory, you need a different set of tools.

GDB (The Magnifying Glass). The GNU Debugger allows you to pause the program mid-execution and inspect its mind. You can see exactly what is on the Stack and where the Instruction Pointer (RIP/EIP) is pointing.

Pwntools (The Skeleton Key). This Python library is your weapon. It allows you to script interactions with the binary, sending raw bytes and parsing responses.

- *Why use it?* Writing raw sockets in Python is tedious. Pwntools allows you to simply write `conn.sendline(payload)` and `conn.interactive()`.

5 The Job

Your objective is to redirect the program's execution flow across three distinct levels of difficulty.

5.1 Level I: The Basic Rewrite

The binary contains a hidden function called `reachMe()`. Your goal is to force the program to jump to this function.

Hints from Intelligence:

- **Analysis:** Use `objdump -d <binary> | grep reachMe` to find the memory address of the target function.
- **Sockets:** Remember, you are attacking a remote service, not running a local program. A socket is essentially a pipe for data. Unlike running a program in your terminal where you type input, here you must push bytes across the network. Pwntools handles this connection for you using `remote()`.

5.2 Level II: The Argument (ROP)

This level is protected. The target function requires a specific argument to unlock the flag. You cannot simply jump there; you must set up the registers first using **Return Oriented Programming (ROP)**.

Hints from Intelligence:

- You need to find "gadgets" (snippets of code ending in `ret`).
- Tools: Use `ROPGadget` or Pwntools' internal `rop.find_gadget` feature.
- **Objective:** Find a gadget to pop a value into the `%rdi` register (which holds the first argument in x64) before jumping to the function.

5.3 Level III: Write-What-Where (Bonus)

The final binary is fortified. You must manually construct a call to the system shell.

Hints from Intelligence: The binary contains specific helper functions designed to aid your exploit. Look for these gadgets:

```
void pop_rdi_ret() { __asm__("pop %rdi; ret;"); }
void pop_rsi_ret() { __asm__("pop %rsi; ret;"); }
void mov_rdi_rsi_ret() { __asm__("movq %rsi, (%rdi); ret;"); }
```

Use these tools to write a command string into memory and then call `system()`.

6 Deliverables

The Baker Street Society requires concrete proof of your success. We do not need a written report, but we do require the data you seized and the tools you used to seize it.

6.1 The Transmission (Flags)

You must submit the captured flags in a specific format to verify your breach. Concatenate your findings into a single string separated by semicolons:

`<Flag_Level_1>;<Flag_Level_2>;<Flag_Level_3>`

Note: If you did not complete the bonus level, simply omit the final flag and semicolon.

6.2 The Archive (Exploit Scripts)

The Society prefers to see the methodology behind the madness. Please attach your code exploit scripts (e.g., `exploit_1.xx`, `exploit_2.xx`) as file uploads alongside your submission. We want to see how you constructed your skeleton keys.

The memory is shattered. The truth is revealed. Welcome to the inner circle, Detective.