

# ENPM691 Homework 09: Exploiting Format String Vulnerability with Direct GOT Overwrite

Kalpesh Parmar  
M.Eng Cybersecurity  
University [REDACTED] Park  
[REDACTED]  
UMD Directory ID [REDACTED]  
Course and section - ENPM691 0101

**Abstract**—In this assignment, I explored a format string vulnerability in a 32-bit compiled C program and used it to redirect program execution to a custom function with elevated privileges. I explain step-by-step the exploitation process, system configuration, compilation commands, debugging steps, and payload construction. References to relevant course materials and documentation are included [1][2][3].

## I. INTRODUCTION

I implemented a format string attack to redirect a vulnerable program's execution flow to a custom target function. This target function executes a child process with root privileges to print a message. All operations were conducted in a controlled Kali-Linux-2025.2 environment using gcc 14.3.0 with multilib support and GDB 16.3 with the pwndbg extension [1].

## II. SYSTEM CONFIGURATION

- **Operating System:** Kali-Linux-2025.2
- **Compiler:** gcc 14.3.0 (Debian 14.3.0-5) with multilib
- **Debugger:** GDB 16.3 with pwndbg extension
- **Architecture:** Compiled and executed in 32-bit mode

## III. COMPILATION

I compiled the program with the following command:

```
gcc -m32 -fno-stack-protector -no-pie -Wl,  
-z,norelro -g directPrintf.c -o  
directPrintf
```

### Explanation:

- `-m32`: Compile as 32-bit
- `-g`: Include debug symbols for GDB
- `-fno-stack-protector`: Disable stack canaries to allow exploitation
- `-no-pie`: Disable position independent executable for fixed addresses
- `-Wl,-z,norelro`: Disable RELRO (read-only relocations) to allow GOT overwrite

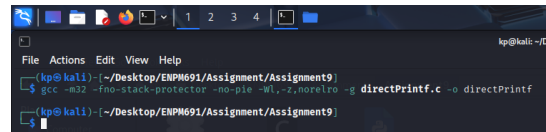


Fig. 1: Compilation output showing no errors and the program ready for exploitation.

## IV. SETTING SUID BIT

To execute the program with root privileges, I set the SUID bit:

```
sudo chown root:root directPrintf  
sudo chmod u+s directPrintf  
ls -ll
```

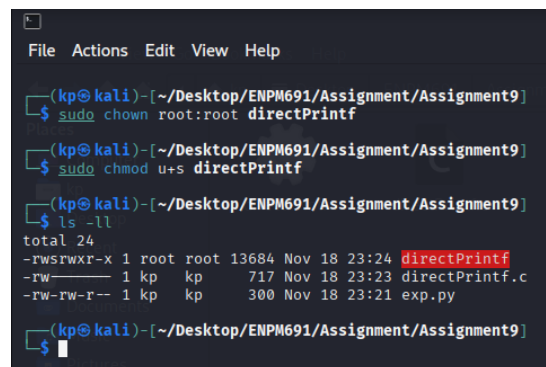


Fig. 2: SUID bit set on directPrintf for root execution.

## V. DISABLING ASLR

I disabled Address Space Layout Randomization (ASLR) to make exploitation deterministic:

```
sudo sysctl -w kernel.randomize_va_space=0  
cat /proc/sys/kernel/randomize_va_space
```



## XI. TARGET FUNCTION EXECUTION

After successfully overwriting the GOT entry of `putchar`, execution redirected to `target()`:



Fig. 13: In target function, privileges escalated and child process prepared [2].

## XII. EXPLOIT RESULTS

Finally, the payload executed the target function, printing the root-level message:

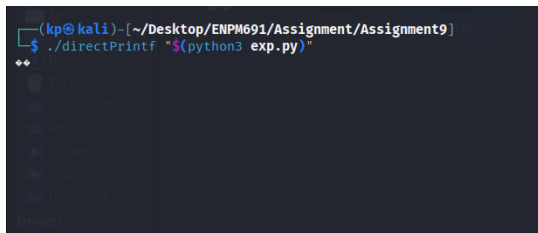


Fig. 14: Exploit output part 1 [2].

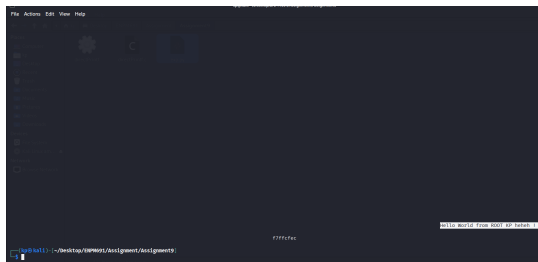


Fig. 15: Exploit output part 2 showing "Hello World from ROOT KP heheh !" [2].

## XIII. CONCLUSION

I successfully exploited a format string vulnerability to hijack program execution, redirecting it to a privileged function. Through this exercise, I reinforced my understanding of memory corruption vulnerabilities, GOT manipulation, format string payload construction, and controlled privilege escalation. This practical work demonstrated the importance of compiler protections, stack layout awareness, and precise memory inspection in real-world exploitation scenarios.

Additionally, I observed firsthand how compiler flags like `-fno-stack-protector`, `-no-pie`, and disabling ASLR contribute to exploit feasibility, and how these conditions can affect program security. This reinforces the critical

need for proper defensive programming and runtime protections in secure software development.

## XIV. WHAT I LEARNED

From this experiment, I learned:

- How format string vulnerabilities can be used to overwrite GOT entries and hijack program execution.
- The importance of stack analysis and offset determination for constructing reliable exploits.
- How system protections like SUID, ASLR, and RELRO interact with memory corruption vulnerabilities.
- Practical skills in using GDB and pwndbg to debug and validate exploit payloads.
- The real-world implications of these vulnerabilities and the importance of designing software with security in mind.

## XV. REFERENCES

- 1) ENPM691 Lecture 08 – Return Oriented Programming (ROP) Part 2.
- 2) pwndbg Documentation, <https://pwndbg.com/>
- 3) Linux man pages for gcc, chmod, chown, printf, and sysctl.