

ENPM691 Homework 05: Injection of Shell Code

Kalpesh Bharat Parmar

M.Eng Cybersecurity, University of Maryland, College Park

kalpesh@umd.edu

UMD Directory ID – kalpesh

Course and section - ENPM691 0101

Abstract—This report documents an observational study of two classic stack-redirection techniques. The first approach demonstrates overwriting a saved return address to redirect control flow to bytes in a local stack buffer (“return-to-stack”); the second approach demonstrates an indirect function-pointer call into a local byte buffer. Using gcc and gdb [1] (with pwndbg [2]), I captured debugger evidence of (i) the location of the saved return address, (ii) the precise instruction that overwrites it (first approach) or the indirect call site (second approach), and (iii) the transfer of control to stack-resident bytes followed by process exit. Interpretation is grounded in the i386 System V ABI [4] and classic background literature [3], with notes on modern DEP/NX protections [5]. The build commands, system configuration, and explanations are included.

I. PURPOSE

The goal is to analyze two stack-hacking techniques using intentionally vulnerable examples:

- 1) **Saved-return overwrite** (“return-to-stack”): a local pointer targets the caller’s saved return slot and sets it to the address of a stack buffer, consistent with the i386 calling convention layout [3], [4].
- 2) **Function-pointer cast and call**: a local byte buffer’s address is cast to a function pointer and called indirectly [4].

II. SYSTEM CONFIGURATION

- **Operating System:** Kali-Linux-2025.2
- **Compiler:** gcc 14.3.0 (Debian 14.3.0-5) with multilib support
- **Debugger:** GDB 16.3 with pwndbg extension [1], [2]
- **Mode:** Compiled and executed in 32-bit mode (i386 ABI) [4]

III. COMPILATION COMMANDS AND EXPLANATIONS

```
gcc -m32 -g exit_shell12.c -o exit_shell12 \
-fno-stack-protector -no-pie -z execstack
```

Flags:

- **-m32:** Produces a 32-bit executable.
- **-g:** Includes debugging symbols for GDB.
- **-o:** Specifies the output binary name.
- **-fno-stack-protector:** Disables stack canaries (for lab observation).
- **-no-pie:** Fixed image base (addresses are stable in disassembly).
- **-z execstack:** Marks the stack executable (contrasts with standard DEP/NX policies) [5].

Fig. 1. Program 1 — Break at main: mixed disassembly (pwndbg [2]) shows the LEA of &shellcode and the store into the saved return slot at [EBP+4] per i386 ABI [4].

IV. METHOD (HIGH-LEVEL)

Following standard debugging practice [1], for each program I:

- 1) Compiled with the flags above.
- 2) In gdb, broke at main and captured baseline registers and mixed source/assembly (pwndbg views where helpful) [2].
- 3) For Program 1, stepped to the store that writes the saved return slot; captured “before” and “after” snapshots of [EBP+4] (per i386 ABI) [4]; stepped the epilogue to show ret transferring control into stack bytes.
- 4) For Program 2, observed the indirect call where the computed pointer equals the local buffer address; captured exit status.

V. RESULTS

A. Program 1: Saved-return Overwrite (exit_shell1)

B. Program 2: Function-Pointer Call (exit_shell12)

VI. DISCUSSION

A. Mechanics

In Program 1, a local pointer computes the address of the saved return slot relative to the current frame and stores the

(registers, stack words, disassembly) to C-level constructs. The study emphasizes conceptual understanding and the role of modern mitigations [5], while referencing foundational background and official tooling/ABI documentation [1]–[4].

Two passes resolve cross-references. If you switch to a .bib file later, add a bibtex step between passes.

BUILD & DEBUG COMMANDS (FOR REFERENCE)

```
# Program 1
gcc -m32 -g exit_shell.c -o exit_shell \
    -fno-stack-protector -no-pie -z execstack

# Program 2
gcc -m32 -g exit_shell2.c -o exit_shell2 \
    -fno-stack-protector -no-pie -z execstack

# GDB (pwndbg may autoload)
gdb -q ./exit_shell
gdb -q ./exit_shell2
```

REFERENCES

- [1] GNU Project, “Debugging with GDB,” Free Software Foundation. (GDB manual).
- [2] pwndbg Project, “pwndbg: A GDB plugin for exploit development,” documentation pages.
- [3] E. Levy (Aleph One), “Smashing the Stack for Fun and Profit,” *Phrack* 49:14, 1996.
- [4] Intel Corporation, “*System V Application Binary Interface: i386 Architecture Supplement*,” (calling conventions and stack frame layout).
- [5] General documentation on DEP/NX (Data Execution Prevention / Non-Executable memory) from OS-vendor and PaX resources.

APPENDIX A SOURCE CODE

A. exit_shell.c

```
#include <stdio.h>
#include <string.h>

int main( int argc , char *argv ){
    char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";

    int *ret;
    ret = (int *)&ret + 6;
    (*ret) = (int)shellcode;
}
```

B. exit_shell2.c

```
#include <stdio.h>
#include <string.h>

/*shellcode for sum of 2 numbers*/
int main( int argc , char *argv ){
    char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";

    int (*shell)(void) = (void *)&shellcode;
    return shell();
}
```