ENPM691 Homework 05: Injection of Shell Code

Kalpesh Bharat Parmar
M.Eng Cybersecurity, University of Maryland, College Park
[redacted]
[redacted]D Directory ID [redacted]
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

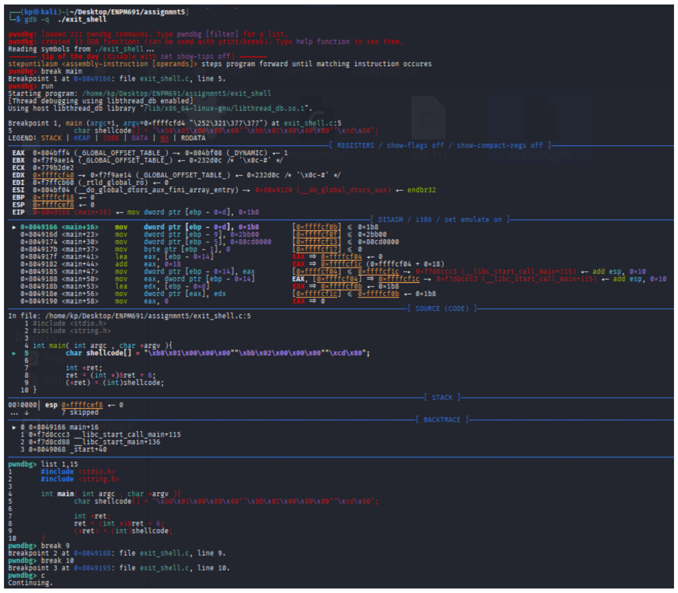
- **Operating System**: Kali-Linux-2025.2
- **Compiler**: gcc 14.3.0 (Debian 14.3.0-5) with multilib support
- **Debugger**: GDB 16.3 with pwndbg extension [1], [2]
- **Mode**: Compiled and executed in 32-bit mode (i386 ABI) [4]

III. Compilation Commands and Explanations

```
gcc -m32 -g exit_shell2.c -o exit_shell2 \
   -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].



```
┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ gdb -q  ./exit_shell

pwndbg: loaded 211 pwndbg commands. Type pwndbg [filter] for a list.
pwndbg: created 13 GDB functions (can be used with print/break). Type help function to see them.
Reading symbols from ./exit_shell ...
────── tip of the day (disable with set show-tips off) ──────
stepuntilasm <assembly-instruction [operands]> steps program forward until matching instruction occures
pwndbg> break main
Breakpoint 1 at 0×8049166: file exit_shell.c, line 5.
pwndbg> run
Starting program: /home/kp/Desktop/ENPM691/assignmnt5/exit_shell
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0×ffffcfd4 "\252\321\377\377") at exit_shell.c:5
5               char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
─────────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────────
 EAX  0×804bff4 (_GLOBAL_OFFSET_TABLE_) —▸ 0×804bf08 (_DYNAMIC) ◂— 1
 EBX  0×f7f9ae14 (_GLOBAL_OFFSET_TABLE_) ◂— 0×232d0c /* '\x0c-#' */
 ECX  0×779b2de2
 EDX  0×ffffcf40 —▸ 0×f7f9ae14 (_GLOBAL_OFFSET_TABLE_) ◂— 0×232d0c /* '\x0c-#' */
 EDI  0×f7ffcb60 (_rtld_global_ro) ◂— 0
 ESI  0×804bf04 (__do_global_dtors_aux_fini_array_entry) —▸ 0×8049120 (__do_global_dtors_aux) ◂— endbr32
 EBP  0×ffffcf18 ◂— 0
 ESP  0×ffffcef8 ◂— 0
 EIP  0×8049166 (main+16) ◂— mov dword ptr [ebp - 0×d], 0×1b8
───────────────────────────────────────────────────[ DISASM / i386 / set emulate on ]──────────────────────
 ▶ 0×8049166 <main+16>    mov    dword ptr [ebp - 0×d], 0×1b8         [0×ffffcf0b] ≤ 0×1b8
   0×804916d <main+23>    mov    dword ptr [ebp - 9], 0×2bb00         [0×ffffcf0f] ≤ 0×2bb00
   0×8049174 <main+30>    mov    dword ptr [ebp - 5], 0×80cd0000      [0×ffffcf13] ≤ 0×80cd0000
   0×804917b <main+37>    mov    byte ptr [ebp - 1], 0                [0×ffffcf17] ≤ 0
   0×804917f <main+41>    lea    eax, [ebp - 0×14]                    EAX ⇒ 0×ffffcf04 ◂— 0
   0×8049182 <main+44>    add    eax, 0×18                            EAX ⇒ 0×ffffcf1c (0×ffffcf04 + 0×18)
   0×8049185 <main+47>    mov    dword ptr [ebp - 0×14], eax          [0×ffffcf04] ≤ 0×ffffcf1c —▸ 0×f7d8ccc3 (__libc_start_call_main+115) ◂— add esp, 0×10
   0×8049188 <main+50>    mov    eax, dword ptr [ebp - 0×14]          EAX, [0×ffffcf04] ⇒ 0×ffffcf1c —▸ 0×f7d8ccc3 (__libc_start_call_main+115) ◂— add esp, 0×10
   0×804918b <main+53>    lea    edx, [ebp - 0×d]                     EDX ⇒ 0×ffffcf0b ◂— 0×1b8
   0×804918e <main+56>    mov    dword ptr [eax], edx                 [0×ffffcf1c] ≤ 0×ffffcf0b ◂— 0×1b8
   0×8049190 <main+58>    mov    eax, 0                               EAX ⇒ 0
─────────────────────────────────────────────────────────[ SOURCE (CODE) ]──────────────────────
In file: /home/kp/Desktop/ENPM691/assignmnt5/exit_shell.c:5
    1 #include <stdio.h>
    2 #include <string.h>
    3
    4 int main( int argc , char *argv ){
 ►  5         char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
    6
    7         int *ret;
    8         ret = (int *)&ret + 6;
    9         (*ret) = (int)shellcode;
   10 }
──────────────────────────────────────────────────────────────[ STACK ]──────────────────────
00:0000│ esp 0×ffffcef8 ◂— 0
... ↓        7 skipped
────────────────────────────────────────────────────────────[ BACKTRACE ]──────────────────────
 ► 0 0×8049166 main+16
   1 0×f7d8ccc3 __libc_start_call_main+115
   2 0×f7d8cd88 __libc_start_main+136
   3 0×8049068 _start+40
─────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> list 1,15
1       #include <stdio.h>
2       #include <string.h>
3
4       int main( int argc , char *argv ){
5               char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
6
7               int *ret;
8               ret = (int *)&ret + 6;
9               (*ret) = (int)shellcode;
10      }
pwndbg> break 9
Breakpoint 2 at 0×8049188: file exit_shell.c, line 9.
pwndbg> break 10
Breakpoint 3 at 0×8049195: file exit_shell.c, line 10.
pwndbg> c
Continuing.
```

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

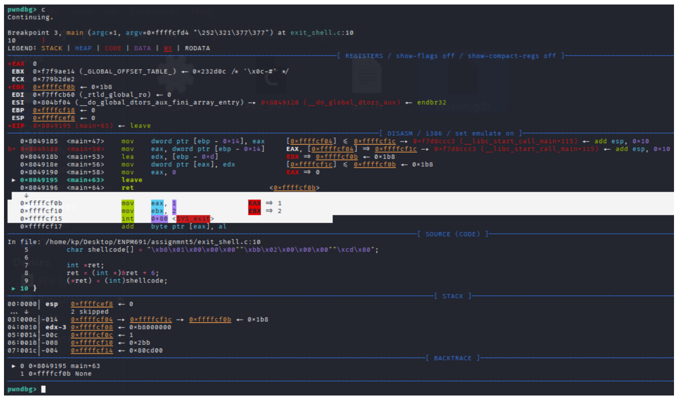
A. *Program 1: Saved-return Overwrite* (`exit_shell`)

B. *Program 2: Function-Pointer Call* (`exit_shell2`)

VI. Discussion

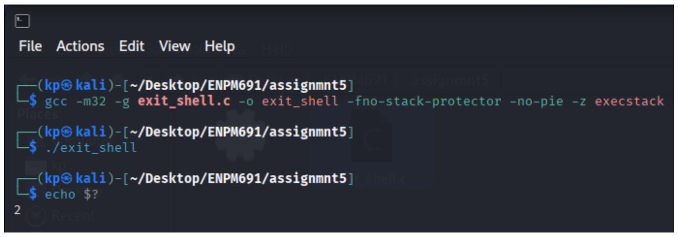
A. *Mechanics*

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



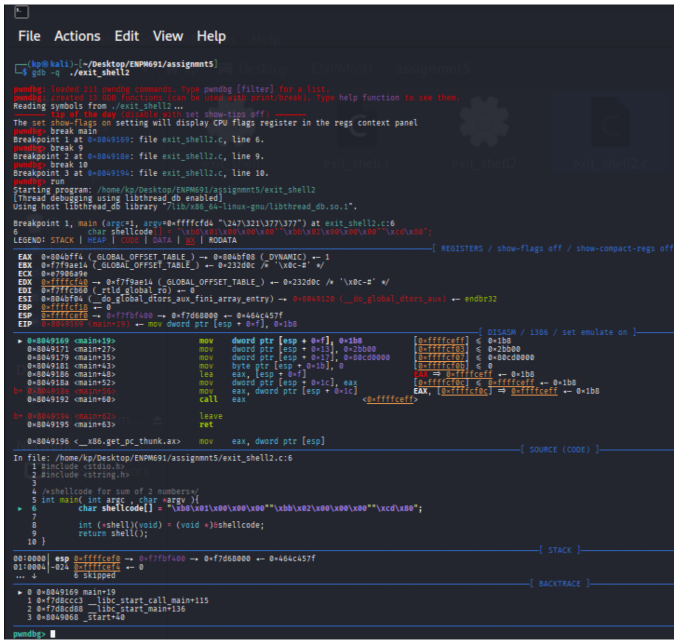
```
pwndbg> c
Continuing.

Breakpoint 3, main (argc=1, argv=0xffffcfd4 "\252\321\377\377") at exit_shell.c:10
10      }
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
──────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off ]──────
*EAX  0
 EBX  0xf7f9ae14 (_GLOBAL_OFFSET_TABLE_) ← 0x232d0c /* '\x0c-#' */
 ECX  0x779b2de2
*EDX  0xffffcf0b ← 0x1b8
 EDI  0xf7ffcb60 (_rtld_global_ro) ← 0
 ESI  0x804bf04 (__do_global_dtors_aux_fini_array_entry) → 0x8049120 (__do_global_dtors_aux) ← endbr32
 EBP  0xffffcf18 ← 0
 ESP  0xffffcef8 ← 0
*EIP  0x8049195 (main+63) ← leave
──────────────────────────────────────────[ DISASM / i386 / set emulate on ]──────────────
   0x8049185  <main+47>    mov    dword ptr [ebp - 0x14], eax     [0xffffcf04] ≤ 0xffffcf1c → 0xf7d8ccc3 (__libc_start_call_main+115) ← add esp, 0x10
b+ 0x8049188  <main+50>    mov    eax, dword ptr [ebp - 0x14]     EAX, [0xffffcf04] ⇒ 0xffffcf1c → 0xf7d8ccc3 (__libc_start_call_main+115) ← add esp, 0x10
   0x804918b  <main+53>    lea    edx, [ebp - 0xd]                EDX ⇒ 0xffffcf0b ← 0x1b8
   0x804918e  <main+56>    mov    dword ptr [eax], edx            [0xffffcf1c] ≤ 0xffffcf0b ← 0x1b8
   0x8049190  <main+58>    mov    eax, 0                          EAX ⇒ 0
 ► 0x8049195  <main+63>    leave
   0x8049196  <main+64>    ret                              <0xffffcf0b>
    ↓
   0xffffcf0b              mov    eax, 1                 EAX ⇒ 1
   0xffffcf10              mov    ebx, 2                 EBX ⇒ 2
   0xffffcf15              int    0x80 <SYS_exit>
   0xffffcf17              add    byte ptr [eax], al
─────────────────────────────────────────────────[ SOURCE (CODE) ]───────────────────────
In file: /home/kp/Desktop/ENPM691/assignmnt5/exit_shell.c:10
    5          char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
    6
    7          int *ret;
    8          ret = (int *)&ret + 6;
    9          (*ret) = (int)shellcode;
 ► 10 }
──────────────────────────────────────────────────────[ STACK ]───────────────────────────
00:0000│ esp   0xffffcef8 ← 0
... ↓          2 skipped
03:000c│-014   0xffffcf04 → 0xffffcf1c → 0xffffcf0b ← 0x1b8
04:0010│ edx-3 0xffffcf08 ← 0xb8000000
05:0014│-00c   0xffffcf0c ← 1
06:0018│-008   0xffffcf10 ← 0x2bb
07:001c│-004   0xffffcf14 ← 0x80cd00
────────────────────────────────────────────────[ BACKTRACE ]────────────────────────────
 ► 0 0x8049195 main+63
   1 0xffffcf0b None
──────────────────────────────────────────────────────────────────────────────────────────
pwndbg> 
```

Fig. 2. Program 1 — Epilogue and transfer: `ret` reads the overwritten saved return address and jumps into stack bytes; process then exits with code 2. This aligns with classic return-to-stack descriptions [3].



```
┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ gcc -m32 -g exit_shell.c -o exit_shell -fno-stack-protector -no-pie -z execstack

┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ ./exit_shell

┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ echo $?
2
```

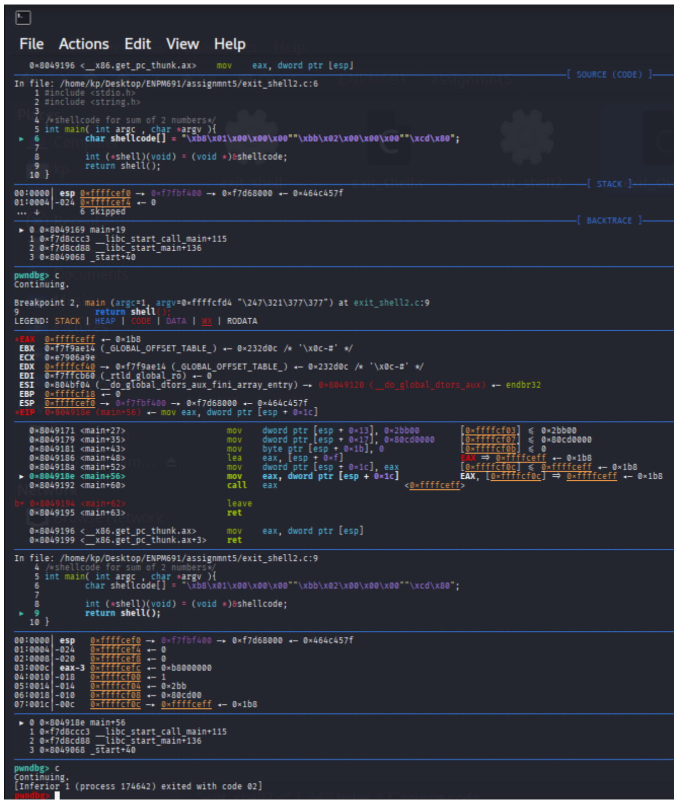
Fig. 3. Program 1 — Terminal run of `exit_shell`.



```
┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ gdb -q  ./exit_shell2

pwndbg: loaded 211 pwndbg commands. Type pwndbg [filter] for a list.
pwndbg: created 13 GDB functions (can be used with print/break). Type help function to see them.
Reading symbols from ./exit_shell2...
────── tip of the day (disable with set show-tips off) ──────
The set show-flags on setting will display CPU flags register in the regs context panel
pwndbg> break main
Breakpoint 1 at 0x8049169: file exit_shell2.c, line 6.
pwndbg> break 9
Breakpoint 2 at 0x804918e: file exit_shell2.c, line 9.
pwndbg> break 10
Breakpoint 3 at 0x8049194: file exit_shell2.c, line 10.
pwndbg> run
Starting program: /home/kp/Desktop/ENPM691/assignmnt5/exit_shell2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main (argc=1, argv=0xffffcfd4 "\247\321\377\377") at exit_shell2.c:6
6               char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
──────────────────────────────────────────[ REGISTERS / show-flags off / show-compact-regs off
 EAX  0x804bff4 (_GLOBAL_OFFSET_TABLE_) → 0x804bf08 (_DYNAMIC) ← 1
 EBX  0xf7f9ae14 (_GLOBAL_OFFSET_TABLE_) ← 0x232d0c /* '\x0c-#' */
 ECX  0xe7906a9e
 EDX  0xffffcf40 → 0xf7f9ae14 (_GLOBAL_OFFSET_TABLE_) ← 0x232d0c /* '\x0c-#' */
 EDI  0xf7ffcb60 (_rtld_global_ro) ← 0
 ESI  0x804bf04 (__do_global_dtors_aux_fini_array_entry) → 0x8049120 (__do_global_dtors_aux) ← endbr32
 EBP  0xffffcf18 ← 0
 ESP  0xffffcef0 → 0xf7fbf400 → 0xf7d68000 ← 0x464c457f
 EIP  0x8049169 (main+19) ← mov dword ptr [esp + 0xf], 0x1b8
───────────────────────────────────────────────────────[ DISASM / i386 / set emulate on ]─────
 ► 0x8049169 <main+19>              mov    dword ptr [esp + 0xf], 0x1b8         [0xffffceff] ≤ 0x1b8
   0x8049171 <main+27>              mov    dword ptr [esp + 0x13], 0x2bb00      [0xffffcf03] ≤ 0x2bb00
   0x8049179 <main+35>              mov    dword ptr [esp + 0x17], 0x80cd0000   [0xffffcf07] ≤ 0x80cd0000
   0x8049181 <main+43>              mov    byte ptr [esp + 0x1b], 0             [0xffffcf0b] ≤ 0
   0x8049186 <main+48>              lea    eax, [esp + 0xf]                     EAX ⇒ 0xffffceff ← 0x1b8
   0x804918a <main+52>              mov    dword ptr [esp + 0x1c], eax          [0xffffcf0c] ≤ 0xffffceff ← 0x1b8
b+ 0x804918e <main+56>              mov    eax, dword ptr [esp + 0x1c]          EAX, [0xffffcf0c] ⇒ 0xffffceff ← 0x1b8
   0x8049192 <main+60>              call   eax                          <0xffffceff>

b+ 0x8049194 <main+62>              leave
   0x8049195 <main+63>              ret

   0x8049196 <__x86.get_pc_thunk.ax>    mov    eax, dword ptr [esp]
──────────────────────────────────────────────────────────────[ SOURCE (CODE) ]──────────────
In file: /home/kp/Desktop/ENPM691/assignmnt5/exit_shell2.c:6
    1 #include <stdio.h>
    2 #include <string.h>
    3
    4 /*shellcode for sum of 2 numbers*/
    5 int main( int argc , char *argv ){
 ►  6         char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
    7
    8         int (*shell)(void) = (void *)&shellcode;
    9         return shell();
   10 }
────────────────────────────────────────────────────────────────────[ STACK ]────────────────
00:0000│ esp 0xffffcef0 → 0xf7fbf400 → 0xf7d68000 ← 0x464c457f
01:0004│-024 0xffffcef4 ← 0
... ↓        6 skipped
─────────────────────────────────────────────────────────────────[ BACKTRACE ]───────────────
 ► 0 0x8049169 main+19
   1 0xf7d8ccc3 __libc_start_call_main+115
   2 0xf7d8cd88 __libc_start_main+136
   3 0x8049068 _start+40
─────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> 
```

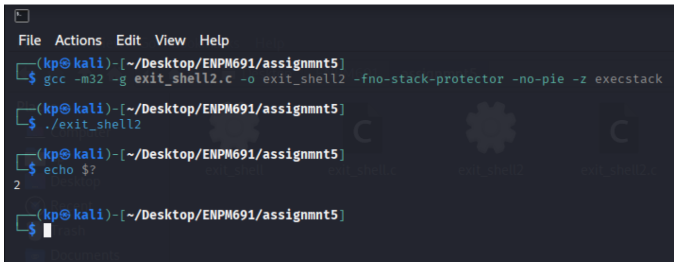
Fig. 4. Program 2 — Breakpoints and mixed disassembly: `EAX` set to `&shellcode` and indirect `call eax`. Built with `-z execstack` to contrast with DEP/NX defaults [5].

address of a local byte buffer there; on epilogue, `ret` transfers control into that buffer. This is consistent with the i386 stack layout and return semantics [3], [4]. In Program 2, a function pointer is explicitly set to the address of the local buffer and invoked via indirect call; this highlights the role of calling conventions and executable memory permissions [4], [5].



```
   0x8049196 <__x86.get_pc_thunk.ax>    mov    eax, dword ptr [esp]
──────────────────────────────────────────────────────────────[ SOURCE (CODE) ]──────
In file: /home/kp/Desktop/ENPM691/assignmnt5/exit_shell2.c:6
    1 #include <stdio.h>
    2 #include <string.h>
    3
    4 /*shellcode for sum of 2 numbers*/
    5 int main( int argc , char *argv ){
 ►  6         char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
    7
    8         int (*shell)(void) = (void *)&shellcode;
    9         return shell();
   10 }
────────────────────────────────────────────────────────────────────[ STACK ]────────
00:0000│ esp 0xffffcef0 → 0xf7fbf400 → 0xf7d68000 ← 0x464c457f
01:0004│-024 0xffffcef4 ← 0
... ↓        6 skipped
─────────────────────────────────────────────────────────────────[ BACKTRACE ]───────
 ► 0 0x8049169 main+19
   1 0xf7d8ccc3 __libc_start_call_main+115
   2 0xf7d8cd88 __libc_start_main+136
   3 0x8049068 _start+40
─────────────────────────────────────────────────────────────────────────────────────
pwndbg> c
Continuing.

Breakpoint 2, main (argc=1, argv=0xffffcfd4 "\247\321\377\377") at exit_shell2.c:9
9               return shell();
LEGEND: STACK | HEAP | CODE | DATA | WX | RODATA
─────────────────────────────────────────────────────────────────────────────────────
*EAX  0xffffceff ← 0x1b8
 EBX  0xf7f9ae14 (_GLOBAL_OFFSET_TABLE_) ← 0x232d0c /* '\x0c-#' */
 ECX  0xe7906a9e
 EDX  0xffffcf40 → 0xf7f9ae14 (_GLOBAL_OFFSET_TABLE_) ← 0x232d0c /* '\x0c-#' */
 EDI  0xf7ffcb60 (_rtld_global_ro) ← 0
 ESI  0x804bf04 (__do_global_dtors_aux_fini_array_entry) → 0x8049120 (__do_global_dtors_aux) ← endbr32
 EBP  0xffffcf18 ← 0
 ESP  0xffffcef0 → 0xf7fbf400 → 0xf7d68000 ← 0x464c457f
*EIP  0x804918e (main+56) ← mov eax, dword ptr [esp + 0x1c]
─────────────────────────────────────────────────────────────────────────────────────
   0x8049171 <main+27>              mov    dword ptr [esp + 0x13], 0x2bb00      [0xffffcf03] ≤ 0x2bb00
   0x8049179 <main+35>              mov    dword ptr [esp + 0x17], 0x80cd0000   [0xffffcf07] ≤ 0x80cd0000
   0x8049181 <main+43>              mov    byte ptr [esp + 0x1b], 0             [0xffffcf0b] ≤ 0
   0x8049186 <main+48>              lea    eax, [esp + 0xf]                     EAX ⇒ 0xffffceff ← 0x1b8
   0x804918a <main+52>              mov    dword ptr [esp + 0x1c], eax          [0xffffcf0c] ≤ 0xffffceff ← 0x1b8
 ► 0x804918e <main+56>              mov    eax, dword ptr [esp + 0x1c]          EAX, [0xffffcf0c] ⇒ 0xffffceff ← 0x1b8
   0x8049192 <main+60>              call   eax                          <0xffffceff>

b+ 0x8049194 <main+62>              leave
   0x8049195 <main+63>              ret

   0x8049196 <__x86.get_pc_thunk.ax>    mov    eax, dword ptr [esp]
   0x8049199 <__x86.get_pc_thunk.ax+3>  ret
─────────────────────────────────────────────────────────────────────────────────────
In file: /home/kp/Desktop/ENPM691/assignmnt5/exit_shell2.c:9
    4 /*shellcode for sum of 2 numbers*/
    5 int main( int argc , char *argv ){
    6         char shellcode[] = "\xb8\x01\x00\x00\x00""\xbb\x02\x00\x00\x00""\xcd\x80";
    7
    8         int (*shell)(void) = (void *)&shellcode;
 ►  9         return shell();
   10 }
─────────────────────────────────────────────────────────────────────────────────────
00:0000│ esp   0xffffcef0 → 0xf7fbf400 → 0xf7d68000 ← 0x464c457f
01:0004│-024   0xffffcef4 ← 0
02:0008│-020   0xffffcef8 ← 0
03:000c│ eax-3 0xffffcefc ← 0xb8000000
04:0010│-018   0xffffcf00 ← 1
05:0014│-014   0xffffcf04 ← 0x2bb
06:0018│-010   0xffffcf08 ← 0x80cd00
07:001c│-00c   0xffffcf0c → 0xffffceff ← 0x1b8
─────────────────────────────────────────────────────────────────────────────────────
 ► 0 0x804918e main+56
   1 0xf7d8ccc3 __libc_start_call_main+115
   2 0xf7d8cd88 __libc_start_main+136
   3 0x8049068 _start+40
─────────────────────────────────────────────────────────────────────────────────────
pwndbg> c
Continuing.
[Inferior 1 (process 174642) exited with code 02]
pwndbg> 
```

Fig. 5. Program 2 — At the indirect call site; program exits with status 2. Stepping and register/stack inspection followed standard GDB workflow [1].



```
┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ gcc -m32 -g exit_shell2.c -o exit_shell2 -fno-stack-protector -no-pie -z execstack

┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ ./exit_shell2

┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ echo $?
2

┌──(kp㉿kali)-[~/Desktop/ENPM691/assignmnt5]
└─$ 
```

Fig. 6. Program 2 — Terminal run of `exit_shell2` and exit status 2.

B. *Effect of Protections*

Modern systems deploy mitigations like non-executable stacks (DEP/NX) [5], stack canaries, ASLR, and PIE. These impede direct execution of stack bytes and make addresses unpredictable. For pedagogical reasons, Program 2 was compiled with `-z execstack`, `-no-pie`, and `-fno-stack-protector` to create a controlled setting for observing call/return mechanics (debugged with GDB/pwndbg [1], [2]). Such flags should be used only in sanctioned lab environments.

C. *Reliability Considerations*

Hard-coding a frame offset (e.g., "+6" words) is ABI/compiler dependent and brittle. Mixed source/assembly in gdb provides ground truth for the specific build [1], [4].

VII. Conclusion

The captured evidence demonstrates two control transfers to stack-resident bytes and ties observable machine state (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();
}
```