

Analysis of Function Call Mechanisms: PLT/GOT vs Direct Syscall Implementation

Jiwon Hwang *Cybersecurity*
University of Maryland, College Park
jhwang97@umd.edu

Abstract—This report presents a comprehensive analysis of the Procedure Linkage Table (PLT) and Global Offset Table (GOT) mechanisms in dynamically linked C programs compared to direct system call implementations. Through systematic examination of static versus dynamic symbol resolution, this study demonstrates the lazy binding process, captures GOT update mechanics during runtime, and contrasts these with direct syscall approaches that bypass PLT/GOT entirely. The analysis provides insights into modern linking mechanisms, their security implications, and performance characteristics essential for cybersecurity professionals and system developers.

I. INTRODUCTION

The PLT and GOT are essential components in modern dynamically linked programs that enable efficient symbol resolution and code reuse. Understanding these mechanisms is crucial for cybersecurity analysis, reverse engineering, and system optimization. This analysis aims to understand:

- Fundamental differences between static and external symbol handling
- The complete lazy binding resolution process
- Runtime GOT update mechanics and timing
- Comparative analysis with direct syscall implementations
- Security implications of different linking approaches

The study employs a controlled experimental approach using custom C programs and assembly code to demonstrate each mechanism in isolation, providing clear visibility into the underlying processes.

II. METHODOLOGY

A. Environment Setup

Component	Specification
Operating System	Ubuntu 24.04.1 LTS
Architecture	x86_64 (64-bit)
Compiler	GCC v13.2.0
Debugger	GDB v15.0.50
Extension	pwndbg v2024.08.29
Analysis Tools	readelf, objdump, nm

Table I

DEVELOPMENT ENVIRONMENT CONFIGURATION

B. Disable Address Space Layout Randomization (ASLR)

To ensure consistent memory addresses across multiple runs for analysis purposes:

Listing 1. ASLR Disabling and Verification

```
1 sudo sysctl -w kernel.randomize_va_space=0
2 # Verify consistent library loading
3 ldd ./plt_got_test
4 ldd ./plt_got_test
5 ldd ./plt_got_test
```

C. Experimental Design

The experiment systematically demonstrates four distinct symbol resolution scenarios through carefully designed test cases:

- 1) **Static data access** - Direct memory addressing without indirection
- 2) **Static function calls** - Direct function invocation within the same compilation unit
- 3) **External data access** - GOT-mediated data access from external modules
- 4) **External function calls** - PLT/GOT lazy binding for external functions

D. Program Structure

1) *Primary Program Design (Appendix A)*: The main program file was structured to systematically test different memory access patterns and function call mechanisms with clear separation between each test case.

Component	Implementation Purpose
Static Data	Test direct RIP-relative memory addressing
Static Function	Analyze direct function calls without PLT
External References	Setup PLT/GOT resolution mechanisms
Main Function	Execute controlled test sequence

Table II

PRIMARY PROGRAM COMPONENTS AND THEIR TESTING PURPOSE

The test sequence in the main function follows a deliberate order to isolate each mechanism:

- Access static data using direct addressing
- Access external data through GOT indirection
- Execute static function call with direct addressing
- Trigger PLT/GOT resolution with external function call

2) *Support Module Design (Appendix B)*: The secondary file provides external symbols necessary for testing PLT/GOT mechanisms, implemented as separate compilation unit to force dynamic resolution.

Component	Testing Purpose
Global Variable	Enable GOT-mediated data access testing
Global Function	Trigger PLT/GOT lazy binding process

Table III

SUPPORT MODULE COMPONENTS FOR DYNAMIC TESTING

E. Compilation Process

The compilation strategy was designed to enable detailed PLT/GOT analysis while maintaining predictable memory layout:

Stage	Purpose
Separate Compilation	Generate independent object files for dynamic linking
Non-PIE Linking	Create executable with fixed addresses for analysis
Debug Information	Enable runtime debugging and step-through analysis

Table IV

COMPILATION STRATEGY FOR ANALYSIS

Listing 2. Compilation Commands

```
1 # Compile with debugging symbols and
  # disable PIE for fixed addresses
2 gcc -g -fno-pie -no-pie -c -o file1.o
  file1.c
3 gcc -g -fno-pie -no-pie -c -o file2.o
  file2.c
4 gcc -g -fno-pie -no-pie -o plt_got_test
  file1.o file2.o
```

F. Analysis Methodology

The analysis follows a systematic approach combining static analysis and dynamic debugging:

1) *Static Analysis Phase*: First, examine the binary structure and symbol information:

Listing 3. Static Analysis Commands

```
1 # Examine section headers and memory
  # layout
2 readelf -S plt_got_test
3
4 # View symbol table and binding
  # information
5 nm plt_got_test
6
7 # Analyze relocation entries
8 readelf -r plt_got_test
9
10 # Disassemble relevant sections
11 objdump -d plt_got_test
```

2) *Dynamic Analysis Phase*:

1) Case 1: Static Symbol Analysis

Analyze direct memory access and function call patterns:

Analysis Type	Target Information
Static Data	RIP-relative addressing calculation
Static Function	Direct call instruction without PLT

Table V

STATIC SYMBOL ANALYSIS TARGETS

Listing 4. Static Analysis Commands

```
1 gdb ./plt_got_test
2 pwndbg> b main
3 pwndbg> run
4 pwndbg> disass main
5 # Examine RIP-relative addressing for
  # static data
6 # Analyze direct call instructions
  # for static functions
```

2) Case 2: PLT/GOT Resolution Analysis

Monitor the complete resolution process for external symbols:

External Data Access Analysis:

Observation Point	Data Collected
Initial GOT State	Pre-resolution GOT entries
PLT Execution	Resolution chain execution
Memory Updates	GOT entry modifications
Final State	Post-resolution addresses

Table VI
PLT/GOT ANALYSIS COLLECTION POINTS

Listing 5. External Data Analysis

```

1 # Check data relocations
2 readelf -r plt_got_test | grep
  shared_value
3 # Examine GOT entry for data
4 pwndbg> x/gx &shared_value
5 # Analyze access pattern in main
6 pwndbg> disass main

```

External Function Call Analysis:

Listing 6. Function Resolution Tracking

```

1 # Set strategic breakpoints
2 pwndbg> b main
3 pwndbg> b *0x401060 #
  shared_function@plt entry
4 pwndbg> run
5
6 ## At main - examine initial state
7 # Check PLT entry structure
8 pwndbg> x/3i 0x401060
9 # Examine initial GOT entry (should
  point to resolver)
10 pwndbg> x/gx 0x404008
11
12 ## Continue to PLT entry - observe
  resolution
13 pwndbg> c
14 pwndbg> x/3i $rip # Current PLT
  instruction
15 pwndbg> si # Step through resolution
  process
16
17 ## After resolution - verify GOT
  update
18 pwndbg> x/gx 0x404008 # Should now
  contain actual function address

```

3) Case 3: Direct Syscall Implementation (Appendix D)

Analyze direct system call mechanism bypassing PLT/GOT:

Listing 7. Direct Syscall Analysis

```

1 gcc -m32 -nostdlib -o syscall_test
  execveShell.S
2 gdb ./syscall_test
3 pwndbg> disass _start
4 # Examine register setup for syscall
5 # Analyze stack layout for arguments

```

III. RESULTS

A. Binary Structure Analysis

The compiled binary exhibits the expected section layout for PLT/GOT functionality:

Section	Type	Address	Flags
.plt	PROGBITS	0x401020	AX (Allocated, Executable)
.got	PROGBITS	0x403fd8	WA (Writable, Allocated)
.got.plt	PROGBITS	0x403fe8	WA (Writable, Allocated)
.text	PROGBITS	0x401040	AX (Allocated, Executable)

Table VII
CRITICAL BINARY SECTIONS FOR PLT/GOT ANALYSIS

Section permissions are crucial for security and functionality:

- **PLT (AX):** Executable for resolution code, not writable for security
- **GOT (WA):** Writable to enable runtime address updates
- **Text (AX):** Contains main program code and static functions

B. Symbol Resolution Comparison

Symbol Type	Access Method	Resolution Time	Indirection
Static Data	RIP-relative	Link time	None
Static Function	Direct call	Link time	None
External Data	GOT-mediated	Load time	Single
External Function	PLT/GOT	First call (lazy)	Double

Table VIII
COMPREHENSIVE SYMBOL RESOLUTION ANALYSIS

C. Memory Access Patterns

Analysis of the disassembled main function reveals distinct access patterns:

Access Type	Instruction Pattern	Address Calculation
Static Data	mov 0x2e9a(%rip), %eax	RIP + offset
External Data	mov 0x2f8e(%rip), %rax mov (%rax), %eax	RIP + GOT offset Dereference GOT entry
Static Function	call 0x401156	Direct address
External Function	call 0x401060 <shared_function@plt>	PLT entry

Table IX
MEMORY ACCESS PATTERN ANALYSIS

D. PLT/GOT Resolution Process

The complete resolution process demonstrates the lazy binding mechanism:

Stack Position	Content	Purpose
ESP+16	0x0	String terminator
ESP+12	0x68732f2f	"//sh" string
ESP+8	0x6e69622f	"/bin" string
ESP+4	0x0	argv terminator
ESP	ptr to command	argv[0] pointer

Table X
DIRECT SYSCALL STACK LAYOUT

E. Direct Syscall Implementation

The direct syscall approach completely bypasses PLT/GOT mechanisms:

Component	Implementation	Purpose
Stack Setup	String construction in-place	Command preparation
Register Loading	Direct syscall number	System call identification
Syscall Invocation	int 0x80	Direct kernel interface

Table XI
DIRECT SYSCALL IMPLEMENTATION COMPONENTS

Stack layout for syscall execution:

Stack Position	Content	Purpose
ESP+16	0x0	String terminator
ESP+12	0x68732f2f	"//sh" string
ESP+8	0x6e69622f	"/bin" string
ESP+4	0x0	argv terminator
ESP	ptr to command	argv[0] pointer

Table XII
DIRECT SYSCALL STACK LAYOUT

IV. DISCUSSION

A. Static vs Dynamic Resolution Mechanisms

The analysis reveals fundamental differences in symbol resolution approaches:

Static Symbol Resolution:

- **Efficiency:** Direct addressing with single instruction execution
- **Predictability:** Fixed addresses determined at link time
- **Security:** Addresses visible in static analysis
- **Example:** `mov 0x2e9a(%rip), %eax` for static data access

Dynamic Symbol Resolution via PLT/GOT:

- **Flexibility:** Runtime address resolution enables shared libraries
- **Lazy Loading:** Functions resolved only when first called
- **Memory Efficiency:** Shared library code reused across processes
- **Overhead:** Additional indirection and resolution complexity

B. GOT Update Mechanism Analysis

The GOT update process follows a precise sequence ensuring thread safety and consistency:

- 1) **Initial Setup:** GOT entry contains address of PLT resolver stub
- 2) **First Access:** PLT entry redirects to resolver via GOT
- 3) **Resolution:** Dynamic linker locates actual function address
- 4) **Update:** GOT entry atomically updated with resolved address
- 5) **Subsequent Calls:** Direct jump through updated GOT entry

This mechanism provides several advantages:

- **Performance:** First call overhead, subsequent calls efficient
- **Memory Conservation:** Unused functions never resolved
- **Security:** Write-protected after resolution in some implementations

C. Security Implications

Each approach presents distinct security characteristics:

PLT/GOT Security Considerations:

- **Vulnerability:** Writable GOT enables GOT overwrite attacks
- **Mitigation:** RELRO (Read-Only Relocations) protection
- **Analysis Complexity:** Dynamic resolution complicates static analysis
- **Runtime Flexibility:** Enables function interposition and hooking

Direct Syscall Security Profile:

- **Stealth:** Bypasses library-based monitoring and hooking
- **Simplicity:** Reduced attack surface through minimal dependencies
- **Detection Difficulty:** Harder to intercept for security tools

- **Functionality Limitation:** Fixed syscall numbers and interfaces

D. Performance Analysis

The timing characteristics differ significantly between approaches:

Call Type	First Call	Subsequent	Overhead
Static Function	1 instruction	1 instruction	None
PLT/GOT (cold)	50+ instructions	2 instructions	High initially
PLT/GOT (warm)	2 instructions	2 instructions	Minimal
Direct Syscall	3–5 instructions	3–5 instructions	Context switch

Table XIII
PERFORMANCE COMPARISON OF CALL MECHANISMS

Understanding these mechanisms is essential for cybersecurity professionals, system developers, and anyone working with low-level system interactions.

The ability to observe and analyze GOT updates in real-time provides crucial insights into runtime symbol resolution, enabling better understanding of system behavior, security implications, and performance characteristics. This knowledge forms the foundation for advanced topics in system security, reverse engineering, and low-level system optimization.

E. Practical Applications

Understanding these mechanisms is crucial for:

Cybersecurity Applications:

- **Reverse Engineering:** Understanding program flow and dependencies
- **Malware Analysis:** Identifying evasion techniques and system interactions
- **Exploit Development:** Leveraging GOT overwrites and ROP chains
- **Defense:** Implementing and understanding protective mechanisms

System Development:

- **Performance Optimization:** Choosing appropriate linking strategies
- **Security Hardening:** Implementing proper memory protections
- **Debugging:** Understanding resolution failures and timing issues

V. CONCLUSION

This comprehensive analysis demonstrates the complete spectrum of function call and symbol resolution mechanisms in modern systems. The PLT/GOT system provides essential flexibility for dynamic linking while introducing complexity and potential security considerations. The lazy binding process optimizes memory usage and startup time but creates resolution dependencies that must be understood for effective system analysis.

Direct syscall implementations offer an alternative approach with different trade-offs, providing stealth and simplicity at the cost of flexibility and maintainability.

```

/mnt/Share/Assignments/9th # ldd plt_got_test
linux-vdso.so.1 (0x00007ffff7fc3000)
libshared.so => ./libshared.so (0x00007ffff7fb0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7c00000)
/lib64/ld-linux-x86-64.so.2 (0x00007ffff7fc5000)

/mnt/Share/Assignments/9th #

/mnt/Share/Assignments/9th # ldd plt_got_test
linux-vdso.so.1 (0x00007ffff7fc3000)
libshared.so => ./libshared.so (0x00007ffff7fb0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7c00000)
/lib64/ld-linux-x86-64.so.2 (0x00007ffff7fc5000)

/mnt/Share/Assignments/9th # ldd plt_got_test
linux-vdso.so.1 (0x00007ffff7fc3000)
libshared.so => ./libshared.so (0x00007ffff7fb0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ffff7c00000)
/lib64/ld-linux-x86-64.so.2 (0x00007ffff7fc5000)

```

Figure 1. Fixed Memory Address Verification

```

/mnt/Share/Assignments/9th # readelf -S plt_got_test
There are 31 section headers, starting at offset 0x3e0:

Section Headers:
[Nr] Name              Type             Address           Offset
Size              EntSize          Flags Link Info  Align
[ 0] 0000000000000000 NULL              0000000000000000 00000000
[ 1] .interp              PROGBITS         0000000000000000 00000000
[ 2] .note.gnu.pr[...]    NOTE            0000000000000000 A 0 0 1
[ 3] .note.gnu.bu[...]    NOTE            0000000000000000 A 0 0 8
[ 4] .note.ABI-tag       NOTE            0000000000000000 A 0 0 4
[ 5] .gnu.hash            GNU_HASH         0000000000000000 A 6 0 8
[ 6] .dynsym              DYNSYM           0000000000000000 A 7 1 8
[ 7] .dynstr              STRTAB           0000000000000000 A 0 0 1
[ 8] .gnu.version         VERSYM           0000000000000000 A 6 0 2
[ 9] .gnu.version_r       VERNEED         0000000000000000 A 7 1 8
[10] .rela.dyn            RELA            0000000000000000 A 6 0 8
[11] .rela.plt            RELA            0000000000000000 AI 6 24 8
[12] .init                PROGBITS         0000000000000000 AX 0 0 4
[13] .plt                 PROGBITS         0000000000000000 AX 0 0 16
[14] .plt.sec             PROGBITS         0000000000000000 AX 0 0 16
[15] .text                PROGBITS         0000000000000000 AX 0 0 16
[16] .fini                PROGBITS         0000000000000000 AX 0 0 4
[17] .rodata              PROGBITS         0000000000000000 A 0 0 8
[18] .eh_frame_hdr        PROGBITS         0000000000000000 A 0 0 8
[19] .eh_frame            PROGBITS         0000000000000000 A 0 0 8
[20] .init_array           INIT_ARRAY       0000000000000000 WA 0 0 8
[21] .fini_array           FINI_ARRAY       0000000000000000 MA 0 0 8
[22] .dynamic              DYNAMIC          0000000000000000 WA 0 0 8
[23] .got                 PROGBITS         0000000000000000 MA 0 0 8
[24] .got.plt             PROGBITS         0000000000000000 MA 0 0 8
[25] .data                PROGBITS         0000000000000000 WA 0 0 8
[26] .bss                 NOBITS           0000000000000000 WA 0 0 8
[27] .comment              PROGBITS         0000000000000000 MS 0 0 1
[28] .symtab              SYMTAB           0000000000000000 29 20 8
[29] .strtab              STRTAB           0000000000000000 0 0 1
[30] .shstrtab            STRTAB           0000000000000000 0 0 1

Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
D (mbind), l (large), p (processor specific)

```

Figure 2. Binary Section Information

```

/mnt/Share/Assignments/9th # nm plt_got_test
00000000004030c r __abi_tag
0000000000404028 B __bss_start
000000000040402c b __completed.0
0000000000404010 D __data_start
0000000000404010 W __data_start
00000000004010b0 t deregister_tm_clones
0000000000401120 T _dl_relocate_static_pie
0000000000403de0 d __do_global_ctors_aux
0000000000403de8 d __do_global_dtors_aux_fini_array_entry
0000000000404018 D __dso_handle
0000000000403de8 d _DYNAMIC
0000000000404024 D __edata
0000000000404030 B __end
00000000004011c0 T _fini
0000000000401150 t frame_dummy
0000000000403dd8 d __frame_dummy_init_array_entry
0000000000402150 r __FRAME_END__
0000000000403fe8 d __GLOBAL_OFFSET_TABLE__
0000000000402054 r __GNU_EH_FRAME_HDR
0000000000401000 T _init
0000000000402000 R __IO_stdin_used
0000000000401178 T main
0000000000401156 t private_function
0000000000404028 d private_value
00000000004010e0 t register_tm_clones
0000000000404028 B shared_value
0000000000401070 T _start
0000000000404028 D __TMC_END__

```

Figure 3. Symbol Table and Binding Information

```

/mnt/Share/Assignments/9th # readelf -r plt_got_test

Relocation section '.rela.dyn' at offset 0x520 contains 3 entries:
Offset      Info          Type            Sym. Name + Addend
00000000003fd0 000100000006 R_X86_64_GLOB_DAT 0000000000000000 __libc_start_main@GLIBC_2.34 + 0
00000000003fd8 000400000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
00000000004028 000500000005 R_X86_64_COPY     0000000000404028 shared_value + 0

Relocation section '.rela.plt' at offset 0x568 contains 2 entries:
Offset      Info          Type            Sym. Name + Addend
00000000004000 000200000007 R_X86_64_JUMP_SLO 0000000000000000 printf@GLIBC_2.2.5 + 0
00000000004008 000300000007 R_X86_64_JUMP_SLO 0000000000000000 shared_function + 0

```

Figure 4. Relocation Entries Analysis

```

pwndbg> disasm main
Dump of assembler code for function main:
0000000000401170 <+0>:  mov     rbp, rdi
0000000000401172 <+2>:  mov     rbp, rbp
0000000000401174 <+4>:  mov     esi, 0
0000000000401176 <+6>:  mov     edi, 0x40203f
0000000000401178 <+8>:  mov     eax, edi
000000000040117a <+10>: mov     eax, edi
000000000040117c <+12>: mov     eax, edi
000000000040117e <+14>: mov     eax, edi
0000000000401180 <+16>: mov     eax, edi
0000000000401182 <+18>: mov     eax, edi
0000000000401184 <+20>: mov     eax, edi
0000000000401186 <+22>: mov     eax, edi
0000000000401188 <+24>: mov     eax, edi
000000000040118a <+26>: mov     eax, edi
000000000040118c <+28>: mov     eax, edi
000000000040118e <+30>: mov     eax, edi
0000000000401190 <+32>: mov     eax, edi
0000000000401192 <+34>: mov     eax, edi
0000000000401194 <+36>: mov     eax, edi
0000000000401196 <+38>: mov     eax, edi
0000000000401198 <+40>: mov     eax, edi
000000000040119a <+42>: mov     eax, edi
000000000040119c <+44>: mov     eax, edi
000000000040119e <+46>: mov     eax, edi
00000000004011a0 <+48>: mov     eax, edi
00000000004011a2 <+50>: mov     eax, edi
00000000004011a4 <+52>: mov     eax, edi
00000000004011a6 <+54>: mov     eax, edi
00000000004011a8 <+56>: mov     eax, edi
00000000004011aa <+58>: mov     eax, edi
00000000004011ac <+60>: mov     eax, edi
00000000004011ae <+62>: mov     eax, edi
00000000004011b0 <+64>: mov     eax, edi
00000000004011b2 <+66>: mov     eax, edi
00000000004011b4 <+68>: mov     eax, edi
00000000004011b6 <+70>: mov     eax, edi
00000000004011b8 <+72>: mov     eax, edi
00000000004011ba <+74>: mov     eax, edi
00000000004011bc <+76>: mov     eax, edi
00000000004011be <+78>: mov     eax, edi
00000000004011c0 <+80>: mov     eax, edi
00000000004011c2 <+82>: mov     eax, edi
00000000004011c4 <+84>: mov     eax, edi
00000000004011c6 <+86>: mov     eax, edi
00000000004011c8 <+88>: mov     eax, edi
00000000004011ca <+90>: mov     eax, edi
00000000004011cc <+92>: mov     eax, edi
00000000004011ce <+94>: mov     eax, edi
00000000004011d0 <+96>: mov     eax, edi
00000000004011d2 <+98>: mov     eax, edi
00000000004011d4 <+100>: mov     eax, edi
00000000004011d6 <+102>: mov     eax, edi
00000000004011d8 <+104>: mov     eax, edi
00000000004011da <+106>: mov     eax, edi
00000000004011dc <+108>: mov     eax, edi
00000000004011de <+110>: mov     eax, edi
00000000004011e0 <+112>: mov     eax, edi
00000000004011e2 <+114>: mov     eax, edi
00000000004011e4 <+116>: mov     eax, edi
00000000004011e6 <+118>: mov     eax, edi
00000000004011e8 <+120>: mov     eax, edi
00000000004011ea <+122>: mov     eax, edi
00000000004011ec <+124>: mov     eax, edi
00000000004011ee <+126>: mov     eax, edi
00000000004011f0 <+128>: mov     eax, edi
00000000004011f2 <+130>: mov     eax, edi
00000000004011f4 <+132>: mov     eax, edi
00000000004011f6 <+134>: mov     eax, edi
00000000004011f8 <+136>: mov     eax, edi
00000000004011fa <+138>: mov     eax, edi
00000000004011fc <+140>: mov     eax, edi
00000000004011fe <+142>: mov     eax, edi
0000000000401200 <+144>: mov     eax, edi
0000000000401202 <+146>: mov     eax, edi
0000000000401204 <+148>: mov     eax, edi
0000000000401206 <+150>: mov     eax, edi
0000000000401208 <+152>: mov     eax, edi
000000000040120a <+154>: mov     eax, edi
000000000040120c <+156>: mov     eax, edi
000000000040120e <+158>: mov     eax, edi
0000000000401210 <+160>: mov     eax, edi
0000000000401212 <+162>: mov     eax, edi
0000000000401214 <+164>: mov     eax, edi
0000000000401216 <+166>: mov     eax, edi
0000000000401218 <+168>: mov     eax, edi
000000000040121a <+170>: mov     eax, edi
000000000040121c <+172>: mov     eax, edi
000000000040121e <+174>: mov     eax, edi
0000000000401220 <+176>: mov     eax, edi
0000000000401222 <+178>: mov     eax, edi
0000000000401224 <+180>: mov     eax, edi
0000000000401226 <+182>: mov     eax, edi
0000000000401228 <+184>: mov     eax, edi
000000000040122a <+186>: mov     eax, edi
000000000040122c <+188>: mov     eax, edi
000000000040122e <+190>: mov     eax, edi
0000000000401230 <+192>: mov     eax, edi
0000000000401232 <+194>: mov     eax, edi
0000000000401234 <+196>: mov     eax, edi
0000000000401236 <+198>: mov     eax, edi
0000000000401238 <+200>: mov     eax, edi
000000000040123a <+202>: mov     eax, edi
000000000040123c <+204>: mov     eax, edi
000000000040123e <+206>: mov     eax, edi
0000000000401240 <+208>: mov     eax, edi
0000000000401242 <+210>: mov     eax, edi
0000000000401244 <+212>: mov     eax, edi
0000000000401246 <+214>: mov     eax, edi
0000000000401248 <+216>: mov     eax, edi
000000000040124a <+218>: mov     eax, edi
000000000040124c <+220>: mov     eax, edi
000000000040124e <+222>: mov     eax, edi
0000000000401250 <+224>: mov     eax, edi
0000000000401252 <+226>: mov     eax, edi
0000000000401254 <+228>: mov     eax, edi
0000000000401256 <+230>: mov     eax, edi
0000000000401258 <+232>: mov     eax, edi
000000000040125a <+234>: mov     eax, edi
000000000040125c <+236>: mov     eax, edi
000000000040125e <+238>: mov     eax, edi
0000000000401260 <+240>: mov     eax, edi
0000000000401262 <+242>: mov     eax, edi
0000000000401264 <+244>: mov     eax, edi
0000000000401266 <+246>: mov     eax, edi
0000000000401268 <+248>: mov     eax, edi
000000000040126a <+250>: mov     eax, edi
000000000040126c <+252>: mov     eax, edi
000000000040126e <+254>: mov     eax, edi
0000000000401270 <+256>: mov     eax, edi
0000000000401272 <+258>: mov     eax, edi
0000000000401274 <+260>: mov     eax, edi
0000000000401276 <+262>: mov     eax, edi
0000000000401278 <+264>: mov     eax, edi
000000000040127a <+266>: mov     eax, edi
000000000040127c <+268>: mov     eax, edi
000000000040127e <+270>: mov     eax, edi
0000000000401280 <+272>: mov     eax, edi
0000000000401282 <+274>: mov     eax, edi
0000000000401284 <+276>: mov     eax, edi
0000000000401286 <+278>: mov     eax, edi
0000000000401288 <+280>: mov     eax, edi
000000000040128a <+282>: mov     eax, edi
000000000040128c <+284>: mov     eax, edi
000000000040128e <+286>: mov     eax, edi
0000000000401290 <+288>: mov     eax, edi
0000000000401292 <+290>: mov     eax, edi
0000000000401294 <+292>: mov     eax, edi
0000000000401296 <+294>: mov     eax, edi
0000000000401298 <+296>: mov     eax, edi
000000000040129a <+298>: mov     eax, edi
000000000040129c <+300>: mov     eax, edi
000000000040129e <+302>: mov     eax, edi
00000000004012a0 <+304>: mov     eax, edi
00000000004012a2 <+306>: mov     eax, edi
00000000004012a4 <+308>: mov     eax, edi
00000000004012a6 <+310>: mov     eax, edi
00000000004012a8 <+312>: mov     eax, edi
00000000004012aa <+314>: mov     eax, edi
00000000004012ac <+316>: mov     eax, edi
00000000004012ae <+318>: mov     eax, edi
00000000004012b0 <+320>: mov     eax, edi
00000000004012b2 <+322>: mov     eax, edi
00000000004012b4 <+324>: mov     eax, edi
00000000004012b6 <+326>: mov     eax, edi
00000000004012b8 <+328>: mov     eax, edi
00000000004012ba <+330>: mov     eax, edi
00000000004012bc <+332>: mov     eax, edi
00000000004012be <+334>: mov     eax, edi
00000000004012c0 <+336>: mov     eax, edi
00000000004012c2 <+338>: mov     eax, edi
00000000004012c4 <+340>: mov     eax, edi
00000000004012c6 <+342>: mov     eax, edi
00000000004012c8 <+344>: mov     eax, edi
00000000004012ca <+346>: mov     eax, edi
00000000004012cc <+348>: mov     eax, edi
00000000004012ce <+350>: mov     eax, edi
00000000004012d0 <+352>: mov     eax, edi
00000000004012d2 <+354>: mov     eax, edi
00000000004012d4 <+356>: mov     eax, edi
00000000004012d6 <+358>: mov     eax, edi
00000000004012d8 <+360>: mov     eax, edi
00000000004012da <+362>: mov     eax, edi
00000000004012dc <+364>: mov     eax, edi
00000000004012de <+366>: mov     eax, edi
00000000004012e0 <+368>: mov     eax, edi
00000000004012e2 <+370>: mov     eax, edi
00000000004012e4 <+372>: mov     eax, edi
00000000004012e6 <+374>: mov     eax, edi
00000000004012e8 <+376>: mov     eax, edi
00000000004012ea <+378>: mov     eax, edi
00000000004012ec <+380>: mov     eax, edi
00000000004012ee <+382>: mov     eax, edi
00000000004012f0 <+384>: mov     eax, edi
00000000004012f2 <+386>: mov     eax, edi
00000000004012f4 <+388>: mov     eax, edi
00000000004012f6 <+390>: mov     eax, edi
00000000004012f8 <+392>: mov     eax, edi
00000000004012fa <+394>: mov     eax, edi
00000000004012fc <+396>: mov     eax, edi
00000000004012fe <+398>: mov     eax, edi
0000000000401300 <+400>: mov     eax, edi
0000000000401302 <+402>: mov     eax, edi
0000000000401304 <+404>: mov     eax, edi
0000000000401306 <+406>: mov     eax, edi
0000000000401308 <+408>: mov     eax, edi
000000000040130a <+410>: mov     eax, edi
000000000040130c <+412>: mov     eax, edi
000000000040130e <+414>: mov     eax, edi
0000000000401310 <+416>: mov     eax, edi
0000000000401312 <+418>: mov     eax, edi
0000000000401314 <+420>: mov     eax, edi
0000000000401316 <+422>: mov     eax, edi
0000000000401318 <+424>: mov     eax, edi
000000000040131a <+426>: mov     eax, edi
000000000040131c <+428>: mov     eax, edi
000000000040131e <+430>: mov     eax, edi
0000000000401320 <+432>: mov     eax, edi
0000000000401322 <+434>: mov     eax, edi
0000000000401324 <+436>: mov     eax, edi
0000000000401326 <+438>: mov     eax, edi
0000000000401328 <+440>: mov     eax, edi
000000000040132a <+442>: mov     eax, edi
000000000040132c <+444>: mov     eax, edi
000000000040132e <+446>: mov     eax, edi
0000000000401330 <+448>: mov     eax, edi
0000000000401332 <+450>: mov     eax, edi
0000000000401334 <+452>: mov     eax, edi
0000000000401336 <+454>: mov     eax, edi
0000000000401338 <+456>: mov     eax, edi
000000000040133a <+458>: mov     eax, edi
000000000040133c <+460>: mov     eax, edi
000000000040133e <+462>: mov     eax, edi
0000000000401340 <+464>: mov     eax, edi
0000000000401342 <+466>: mov     eax, edi
0000000000401344 <+468>: mov     eax, edi
0000000000401346 <+470>: mov     eax, edi
0000000000401348 <+472>: mov     eax, edi
000000000040134a <+474>: mov     eax, edi
000000000040134c <+476>: mov     eax, edi
000000000040134e <+478>: mov     eax, edi
0000000000401350 <+480>: mov     eax, edi
0000000000401352 <+482>: mov     eax, edi
0000000000401354 <+484>: mov     eax, edi
0000000000401356 <+486>: mov     eax, edi
0000000000401358 <+488>: mov     eax, edi
000000000040135a <+490>: mov     eax, edi
000000000040135c <+492>: mov     eax, edi
000000000040135e <+494>: mov     eax, edi
0000000000401360 <+496>: mov     eax, edi
0000000000401362 <+498>: mov     eax, edi
0000000000401364 <+500>: mov     eax, edi
0000000000401366 <+502>: mov     eax, edi
0000000000401368 <+504>: mov     eax, edi
000000000040136a <+506>: mov     eax, edi
000000000040136c <+508>: mov     eax, edi
000000000040136e <+510>: mov     eax, edi
0000000000401370 <+512>: mov     eax, edi
0000000000401372 <+514>: mov     eax, edi
0000000000401374 <+516>: mov     eax, edi
0000000000401376 <+518>: mov     eax, edi
0000000000401378 <+520>: mov     eax, edi
000000000040137a <+522>: mov     eax, edi
000000000040137c <+524>: mov     eax, edi
000000000040137e <+526>: mov     eax, edi
0000000000401380 <+528>: mov     eax, edi
0000000000401382 <+530>: mov     eax, edi
0000000000401384 <+532>: mov     eax, edi
0000000000401386 <+534>: mov     eax, edi
0000000000401388 <+536>: mov     eax, edi
000000000040138a <+538>: mov     eax, edi
000000000040138c <+540>: mov     eax, edi
000000000040138e <+542>: mov     eax, edi
0000000000401390 <+544>: mov     eax, edi
0000000000401392 <+546>: mov     eax, edi
0000000000401394 <+548>: mov     eax, edi
0000000000401396 <+550>: mov     eax, edi
0000000000401398 <+552>: mov     eax, edi
000000000040139a <+554>: mov     eax, edi
000000000040139c <+556>: mov     eax, edi
000000000040139e <+558>: mov     eax, edi
00000000004013a0 <+560>: mov     eax, edi
00000000004013a2 <+562>: mov     eax, edi
00000000004013a4 <+564>: mov     eax, edi
00000000004013a6 <+566>: mov     eax, edi
00000000004013a8 <+568>: mov     eax, edi
00000000004013aa <+570>: mov     eax, edi
00000000004013ac <+572>: mov     eax, edi
00000000004013ae <+574>: mov     eax, edi
00000000004013b0 <+576>: mov     eax, edi
00000000004013b2 <+578>: mov     eax, edi
00000000004013b4 <+580>: mov     eax, edi
00000000004013b6 <+582>: mov     eax, edi
00000000004013b8 <+584>: mov     eax, edi
00000000004013ba <+586>: mov     eax, edi
00000000004013bc <+588>: mov     eax, edi
00000000004013be <+590>: mov     eax, edi
00000000004013c0 <+592>: mov     eax, edi
00000000004013c2 <+594>: mov     eax, edi
00000000004013c4 <+596>: mov     eax, edi
00000000004013c6 <+598>: mov     eax, edi
00000000004013c8 <+600>: mov     eax, edi
00000000004013ca <+602>: mov     eax, edi
00000000004013cc <+604>: mov     eax, edi
00000000004013ce <+606>: mov     eax, edi
00000000004013d0 <+608>: mov     eax, edi
00000000004013d2 <+610>: mov     eax, edi
00000000004013d4 <+612>: mov     eax, edi
00000000004
```

```

1 // file1.c
2 #include <stdio.h>
3
4 // External variable and function declarations
5 extern int shared_value;
6 extern void shared_function(void);
7
8 // Static variable – only visible in this file
9 static int private_value = 100;
10
11 // Static function – only visible in this file
12 static void private_function(void) {
13     printf("Inside private function, value: %d\n", private_value);
14 }
15
16 int main() {
17     printf("Private value: %d\n", private_value);
18     printf("Shared value: %d\n", shared_value);
19     private_function();
20     shared_function();
21     return 0;
22 }

```

B. Appendix B: External Symbol Provider (file2.c)

```

1 // file2.c – This will be compiled into a
2 // shared library
3 #include <stdio.h>
4
5 // Shared variable and function definitions
6 int shared_value = 42;
7
8 void shared_function(void) {
9     printf("Inside shared function\n");
10 }

```

C. Appendix C: Alternative Main Implementation (main.c)

```

1 #include <stdio.h>
2
3 static int stt_data = 1;
4 int ext_data;
5
6 static int Stt_func(){
7     printf("static function\n");
8 }
9 int Ext_func(){
10    printf("extern function\n");
11 }
12
13 int main()
14 {
15     Stt_func();
16     Ext_func();
17     printf("static variable stt_data: %d\n", stt_data);
18     printf("extern variable ext_data: %d\n", ext_data);
19 }

```

D. Appendix D: Direct Syscall Implementation (execveShell.S)

```

1 // execveShellstorm.S
2 .global _start
3 _start:
4     xor     %eax,%eax      # Zero EAX register
5     push    %eax          # Push NULL
6     terminator
7     push    $0x68732f2f    # Push "//sh"
8     push    $0x6e69622f    # Push "/bin"
9     mov     %esp,%ebx      # First arg: filename
10    pointer
11    push    %eax           # Push NULL
12    push    %ebx           # Push argv[0]
13    mov     %esp,%ecx      # Second arg: argv
14    pointer
15    mov     $0xb,%al       # syscall number for
16    execve
17    int     $0x80          # Make syscall

```

REFERENCES

- [1] Alexandre Cheron, "Exploit 101 - Format Strings," Available: <https://axcheron.github.io/exploit-101-format-strings/>
- [2] Ulrich Drepper, "How To Write Shared Libraries," Red Hat Inc., 2011.
- [3] "System V Application Binary Interface AMD64 Architecture Processor Supplement," Available: https://refspecs.linuxbase.org/elf/x86_64-abi-0.99.pdf
- [4] Ian Lance Taylor, "ELF Symbol Versioning," Available: <https://www.akkadia.org/drepper/symbol-versioning>