Mérési útmutató a

“Szoftver implementációs biztonsági rések vizsgálata”

című méréshez

2016. február

A mérést kidolgozta:

Pék Gábor

BME, CrySys Adat- és Rendszerbiztonsági Laboratórium

Elméleti összefoglaló

A méréshez kapcsolódó elméleti összefoglaló a **Memory Corruption** diákban (előadás és gyakorlat) található meg a **Computer Security** tárgy honlapján. További hasznos PEDA + GDB parancsok érhetőek el itt.

A diák letöltéséhez szükséges jelszó: ts72ha74w64hd72jq91jaq82j

A mérés célja, hogy a hallgató megismerkedjen a szoftver implementációs hibák néhány fontosabb kategóriájával. A hibák felismerésén felül a mérés során a hallgatók néhány fontosabb segédeszköz használatát sajátítják el, melyek segítségével implementációs hibák kihasználására is lehetőségük nyílik.

Néhány tool fontosabb parancsai:

GDB+PEDA

- **aslr:** beállítja, illetve megmutatja az ASLR állapotát a GDB-n belül
- **disas:** Egy adott függvény, cím disassembly-je
- **pdisass:** formatált kimenete a GDB disas parancsnak
- **x/[num]gx <address>:** num*8 byte hosszan kiírja egy memóriaterület tartalmát
- **telescope [memory] <num>:** Stack/memória kilistázása num * 8 byte hosszban.
- **b *<address>:** Breakpoint elhelyezése egy megadott címre
- **info files:** Kiírja, hogy a GDB milyen file-okat és szimbólumokat használ éppen. Többek között az éppen debuggolt fájl szegmenseit is megtudhatjuk így.

PWNTOOLS

- **from pwn import ***: Pwntools modulok betöltése python-ba
- **p64(address):** 64-bites little-endian-ba alakít egy adott címet

Mérési eszközök

A mérés elvégzéséhez az *avatao* online feladatmegoldó platformot kell használni, melyre a hallgatókat a mérést megelőzően meghívjuk.

A mérés során használt legfontosabb segédprogramok a következők:

- GDB (Gnu Debugger)
- GDB PEDA plugin
- checksec
- ROPgadget
- objdump
- pwntools

Feladatok

1. Buffer overflow 101

Indítsuk el a *BoF 101* feladatot az *avataon* keresztül és jelentkezzünk be SSH segítségével a containerbe a megadott felhasználónév-jelszó párossal, majd válaszolja meg az alábbi kérdéseket.

1.1. Milyen sérülékenységet tartalmaz a main.c fájl és miért?

1.2. Indítsa el a programot, majd adjon meg olyan felhasználói inputot, hogy a program Segmentation Fault-tal lépjen ki. Adja meg a futtatott parancsot az alábbi mezőben.

1.3. Futassa le a *checksec* alkalmazást a *main* binárisra és értelmezze a program kimenetét. Mit jelentenek az egyes sorok?

1.4. Töltse be a programot gdb-be, majd helyezzen el egy breakpointot a *vuln* függvény *gets* függvényénél. A *disas* parancs segíthet. Vizsgálja meg, hogy a breakpoint fel lett-e véve.

1.5. [SZORGALMI] Indítsa újra a programot gdb-vel, majd vizsgálja meg, hogy változott-e az gets címe. Mit állapít meg és mi ennek az oka? Tud-e ebből bármilyen következtetést levonni az ASLR –re vonatkozólag?

1.6. Mekkora a stack frame mérettel inicializálódik a vuln függvény?

1.7. Milyen függvényhívási konvenciót használunk és miért? Hol helyezkednek el a függvényhívási paraméterek?

1.8. Hol helyezkedik el a *buf* változó a stacken? Adja meg a gdb parancsot, mellyel ezt meghatározta.

1.9. Győződjön meg, hogy az gets lefutás után valóban megtörtént-e a string beolvasása a buf bufferbe.

1.10. Hol tárolódik a vuln függvényen belül a main függvény visszatérési címe? A megoldáshoz segítséget nyújt a *x/xg* gdb vagy a *disas* parancs. Mit jelent a g kapcsoló a parancsban?

1.11. Tesztelje, hogy egy bemeneti string mely bájtjaira kapunk Segmentation Fault-ot. Mi a visszatérési cím értéke közvetlen a *leave ret* lefutása előtt? Használja ehhez a következő patternt:

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A

1.12. Az előbbi pattern alapján hány bájt szükséges ahhoz, hogy a visszatérési címet felülírjuk? Segítségként használhatja az *x/s* parancsot.

1.13. Alakítsa át a korábban készített patternt shellcode-dá annak első 16 bájtjának 'NOP sled'-del történő felülírásával. A shellcode ugorjon rá erre a NOP sledre. Figyeljen a cím little endiánban történő megadására. Mentse ki a shellcode-ot egy fájlba (shellcode.bin), majd ellenőrizze a kapott fájlt *hexdump –C* parancs segítségével. A shellcode generálásához érdemes python-t használni.

1.14. Injektálja a shellcode-ját gdb –n belül az *r < shellcode.bin* parancs segítségével. Mit tapasztal? Vizsgálja meg, hogy minden a helyére került-e a másolás után.

1.15. Futassa le az *execstack –s main* parancsot, majd menjen vissza gdb-be és futassa újra az előbbi exploit kezdeményezést. Milyen változást fedez fel? Vizsgálja meg az eredményt *checksec* segítségével is.

1.16. [SZORGALMI] Injektálja az alábbi shellcode-ot, hogy meggyőződjön valóban tudunk-e shell-t futtatni. Részletesebb leírása a shellcode-nak itt található. A feladat megoldását egyszerűsíti a pwntools (from pwn import *) p64 függvénye.

'\x48\x31\xff\x57\x57\x5e\x5a\x48\xbf\x2f\x2f\x62\x69\x6e\x2f\x73\x68\x48\xc1\xef\x08\x57\x54\x5f\x6a\x3b\x58\x0f\x05'

Alternatívaként, használhatja a pwntools belső shellcode generátorát is.

```
from pwn import *
context.update(arch='amd64', os='linux')
shellcode = asm(shellcraft.amd64.linux.sh())
```

1.17. Derítse ki, hogy milyen libc verzió van a környezeten belül a gdb *vmmap* parancs segítségével. Mire való a vmmap és mi a különbség a kimenetében, ha az adott program fut vagy sem.

1.18. A ROPgadget toollal keressen a libc.so-ban 'pop rdi ; ret' és 'pop rsi ; ret' gadget-eket. A tool kimenetét érdemes először egy fájlba kimenteni és utána a fájlban keresni (gyorsabb).

1.19. Számítsa ki, illetve keresse meg, hogy a memóriában a megfelelő helyeken valóban a kívánt gadget-ek vannak-e. Használja ehhez a *gdb x/i* vagy *pdisass* parancsát. Fontos, hogy a libc fájlban offsetek vannak és nem tényleges virtuális címek!

1.20. Keressen a memóriában '/bin/sh' stringet

1.21. Írja meg a ROPchain-t, mely egy system('/bin/sh') parancsot hív meg. Gondolja végig, hogy a ROPchain-nek hogyan kell elhelyezkednie a memóriában miután a vuln függvény visszatérési címét felülírta. Az egyszerűség kedvéért talán most is érdemes a *pwntools*-t használni. Futassa kódját GDB-n belül.

1.22. [SZORGALMI] Ismételje meg az előző pontot, de most execve-t használjon a system helyett. Mi a különbség?

2. Címek vizsgálata

Indítsa el *avatao-n* a Sample 4 challenge-et és válaszolja meg az alábbi kérdést.

2.1. Futtassa a *main* binárist többször és vizsgálja meg a kimenetet. Mit állapít meg belőle?

További információk

Jegyzőkönyv

A jegyzőkönyvet a mérés után egy héten belül el kell küldeni a mérésvezetőnek pdf formátumban. A jegyzőkönyvnek az alábbiakat kell tartalmaznia:

- Hallgató(k) neve és Neptun kódja
- Mérés neve
- Mérés időpontja
- Feladatok megoldása

A megoldások leírásánál törekedni kell a tömör, de érthető válaszra. A leírásból a megoldásnak reprodukálhatónak kell lennie (hosszabb kód mellékelhető a pdf-hez, nem feltétlenül kell belerakni)!