Shellcoding

Peter Švec
peter_svec@stuba.sk

Prečo **shellcode**?

- Postupnosť bajtov, reprezentujúca strojový kód vhodná na injektovanie do binárneho súboru (<=> XSS pri web hackingu)
- Von Neumann architektúra -> dáta = kód
- V minulosti bolo hlavným cieľom spustiť shell

Ukážka

Cieľom je spustiť shell. V Cčku: ***execve("/bin/sh", NULL, NULL);***

execve -> číslo systémového volania 59 (**RAX**)

const char *filename -> cesta k programu (**RDI**)

const char *const argv[] -> argumenty programu (**RSI**)

const char *const envp[] -> premenné prostredia (**RDX**)

```
mov rax, 59                 # číslo systémového volania
lea rdi, [rip+binsh]        # pointer na reťazec /bin/sh
mov rsi, 0                  # argv = NULL
mov rdx, 0                  # envp = NULL
syscall                     # vyvolanie systémového volania
binsh:                      # návestie na označenie, kde sa
.string "/bin/sh"           # nachádza reťazec (nezaberá miesto)
```

Dáta v shellcode

Ako vložiť reťazec “/bin/sh”?

- Možnosť 1:
  - .string “/bin/sh” -> “/bin/sh\0”
- Možnosť 2:
  - .ascii “/bin/sh” -> “/bin/sh”
- Možnosť 3:
  - **mov** rbx, 0x0068732f6e69622f # 0x2f = ‘/’, 0x62 = ‘b’, 0x69 = ‘i’, ...
  - **push** rbx
  - **mov** rdi, rsp

Ako vytvoriť shellcode?

Zdrojový kód (shellcode.s):

```
.global _start
_start:
.intel_syntax noprefix
    kód...
```

Kompilácia:

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

Extrakcia bajtov:

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

Rýchle disassemblovanie:

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

Ako ladiť shellcode?

- Najrýchlejšia high-level kontrola:
  - strace ./shellcode-elf
- Prostredníctvom GDB:
  - gdb ./shellcode-elf
  - si (**S**tep **I**nstruction) -> ďalšia inštrukcia (+vnáranie sa do fukcií CALL)
  - ni (**N**ext **I**nstruction) -> ďalšia inštrukcia (CALL sa preskakuje)
  - Prehliadanie registrov/pamäte
    - x/gx $rsp, x/8b $rax, x/20i $rip
- Breakpointy:
  - Manuálne int3 inštrukcia (0xCC)
  - Návestia v kóde
  - break *adresa (break *0x400000)

Obmedzenia

- Vstupy do aplikácie môžu mať rôzne obmedzenia: \0 byte (pri strcpy), dĺžka, transformácia vstupu, šifrovanie, prekódovanie,...
- Exploit/shellcode musíme vedieť prispôsobiť

mov rax, 0	48 c7 c0 00 00 00 00
mov rax, 5	48 c7 c0 05 00 00 00
mov rax, 10	48 c7 c0 0a 00 00 00
mov rbx, 0x67616c662f	48 bb 2f 66 6c 61 67 00 00 00

xor rax, rax	48 31 c0
xor rax, rax; **mov** al, 5	48 31 c0 b0 05
mov rax, 9; **inc** rax	48 c7 c0 09 00 00 00 48 ff c0
mov ebx, 0x67616c66; **shl** rbx, 8; **mov** bl, 0x2f	bb 66 6c 61 67 48 c1 e3 08 b3 2f

Shellcoding v súčasnosti

- Koncept oprávnení na stránky vo virtuálnom adresnom priestore (**R**ead **W**rite e**X**ecute)
- Moderné systémy sú už chránené (NX, DEP) -> zásobník, halda nie nemajú **X** oprávnenie
- Systémové volanie mprotect vie nastaviť **X** na stránku, ako však spustiť kód keď nevieme spustiť kód (ROP -> blok 4)
- JIT kompilátory (JavaScript v prehliadačoch)
  - var evil_code = 0x48c70c...;