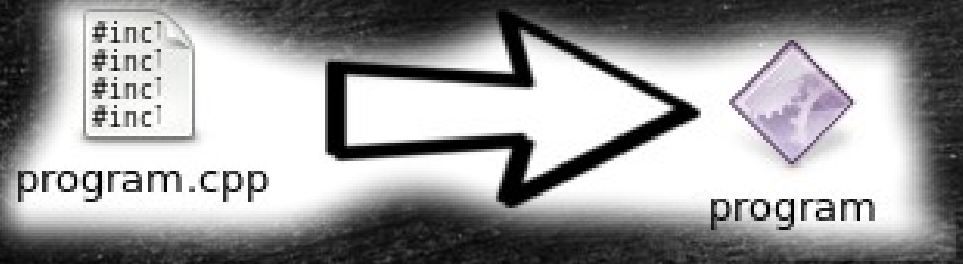
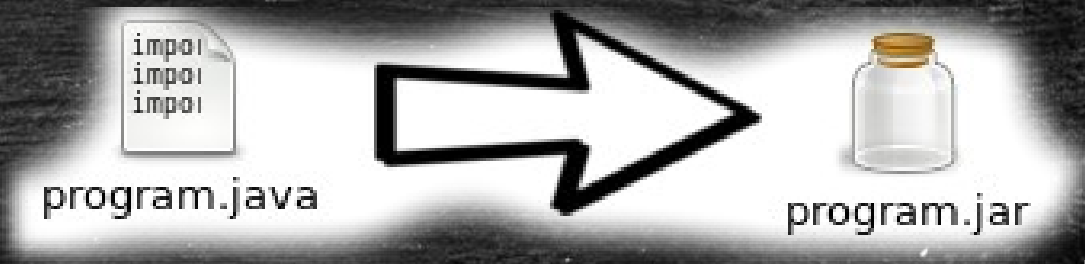
Kódvisszafejtés és exploitálás

Smashing the stack

Reverse engineering - what

- Forráskód -> Futtatható program
- Cél: Az eredeti kód vagy a számunkra érdekes kódrészek visszanyerése/megértése/módosítása





Reverse engineering - why

- Kíváncsiak vagyunk, hogyan működik egy adott program
- Módosítani szeretnénk a programban valamit
  - Funkció hozzáadás
  - DRM / Trial version megkerülés
  - Cheatelés játékokban (pl. módosított kliens)
- Sebezhetőséget keresünk, amit kihasználhatunk

Reverse engineering - how

- Decompilers: Eredeti forráskódot (próbálja) visszaállítani. pl. jd, REC studio, Hex-rays
- Disassembler: Gépi kódot átalakítja ember által olvasható(bb) assembly kóddá. pl. objdump
- Debugger: A lefordított programon utasításonként végiglépeget, számon tartja közben a memória, regiszterek, flagek, stb. állapotát. pl. gdb, edb, ollydbg

Assembly

- Standalone programok: Gépi kódot tartalmazó binárisok -> Instrukciók sorozata.
- Különböző architektúrákon (pl. 8085, x86, arm) másfajta instrukciók vannak
- Memória (stack, heap, globális) + regiszterek (pl. eax, ebx, esp, eip)

Assembly – Függvény felépítése

C kód:

```
void fuggveny()
{
  int a, int b, int c;
  ...
  return;
}
```

```
fuggveny();
```

Assembly kód:

```
0804975a<fuggveny>:
 push ebp
 mov ebp, esp
 sub esp, 12
 ...
 mov esp, ebp
 pop ebp
 ret
```

```
call 804975a <fuggveny>
```

Assembly – Függvény felépítése

call 804975a

- == **push eip**
  jmp 804975a

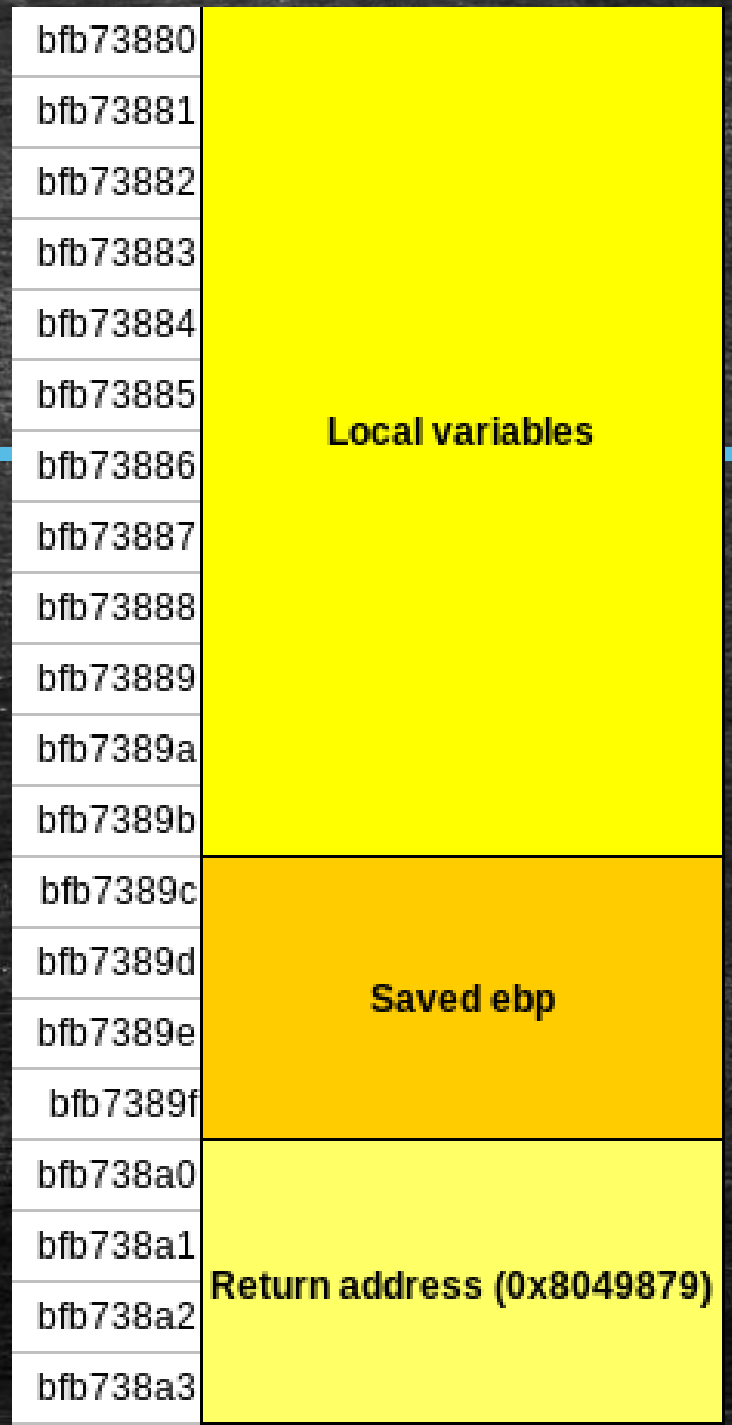
ret

- == **pop eip**

Assembly

```
0804975a<fuggveny>:
 push ebp
 mov ebp, esp
 sub esp, 12
 ...
 mov esp, ebp
 pop ebp
 ret

8049874:
call 804975a <fuggveny>
```



Demo

- VM-ben futó szervert akarunk pwnolni
- Cél: szerveren lévő titkos fájl tartalmát kiiratni (csak root olvashatja)

Források bináris pwnolásra:

- exploit-exercises.com
- owasp.org
- Shellcode database: shell-storm.org
- Korszerűbb védekezések, támadások: DEP, ASLR, ROP
- Egyéb sérülékenységek: Format string, heap overflow, use-after-free