Curs

→ Curs 1 - 24.02.2023

↳ detalii administrative

↳ materiale pe Teams

→ N = 60% lab + 40% proiect final

taskuri de reverse engineering complexe

peste fișiere binare primite

↳ pt. trecere >5 la ambele părți + fără plagiat

→ laboratoarele sunt obligatorii 7/9 (max. 2 absențe)

⇒ ca setup:

↳ mașini virtuale

→ python, C

→ pt. proiectul 0

↳ cerințe pe site

→ pt. a trimite cheia → form la sf. săptămânii pe Teams

→ cerințe + arhive pe site

→ pași care trebuie urmați:

↳ scrie cod pt. algoritm

→ testează cu datele date ca test

→ scoate cheia și verifică de. JSON-ul este corect

→ # Curs 2 — 03.03.2023

↳ Tehnici de analiză a fișierelor binare

↳ black box → nu se poate "desface", poți doar obs. interacțiunile cu SO, librării, alte binare etc.

↗ need to create this one for dissertation

→ white box → analiză pe assembly + cu ce interacționează

→ grey box → white box + black box

→ proces de compilare

$$\begin{tikzcd} \text{cod sursă} \arrow[r, "\text{transf.}"'] & \text{cod sursă assembly} \arrow[r, "\text{asamblor}"'] & \text{fișier obiect} \arrow[r, "\text{link}"'] & \text{binary file} \arrow[r, "\text{load}"'] & \text{Proces (binary file în execuție)} \\ & & \text{librării ext} \arrow[ur] & & \end{tikzcd}$$

→ objdump -d fișier_compilat_cu_cod_mașină conține

↳ adr. la care se află codul

→ codul mașină

→ codul assembly

→ un output al com. objdump nu se poate edita cu un edit. obișnuit (nu sunt toate chs. reprezentabile)

Task: editează fișierul binar simplu (cu printf) pt. a afișa altceva

→ fișiere binare

↳ conțin cod mașină, nu assembly

→ codul mașină este înconjurat de headers + alte info pt. a forma fișiere binare:

↳ headers → ELF → linux
→ PE → windows
→ WASM → web (nativ) DOOM 3 a fost portat pe WASM = web assembly

// cum știe procesorul să se oprească într-un fișier
// binar? Teoria de lg. variabilă a lui Shannon
// codare cu lg. var. → pt. a scurta codările secvențelor
// în fct. de frecvența apelării + fiecare are decodare
// unică (codarea scurtă → în fct. de manualul arhitecturii pt. că
// aceste valori sunt înțelese de hardware)

→ cod assembly

↳ cod mai rapid

→ de la cod mașină $\xrightarrow{\text{imposibil}}$ cod

→ ACC: registers

↳ SSE / SSE2 → SIMD ? → to search SSE, SSE2, SIMD
single instruction, multiple data

→ GPR, RIP, RSP, RBP

$\underset{\text{Stack pointer}}{\uparrow}$ RSP, $\underset{\text{Base pointer}}{\uparrow}$ RBP

→ stack = memoria unui program

↳ are dim. fixă, heap-ul are memorie dinamică

→ ACC: arithmetic and log

```
MOV RAX, 0      ; rax = 0
XOR RAX, RAX    ; rax = rax ^ rax
```

↳ rax → 64b. => mov rax funcționează corect de-primește val. de 64b => codificarea ei este mult mai lungă decât xor rax, rax doar transformă 0 în 64b (ceea ce este inutil), iar xor rax, rax nu face această codificare