KCKB UPJS

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA V KOŠICIACH

Financované Európskou úniou NextGenerationEU

PLÁN OBNOVY

MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

6. Pokročilá analýza kódu a dynamické ladenie

Definícia paradigmy ladenia

Prečo dynamická analýza? Statická analýza (čítanie disassembleru) je základným kameňom, ale v modernom prostredí často zlyháva:

- **Limity statiky:** Moderný malvér využíva polymorfizmus, metamorfizmus, silnú obfuskáciu riadenia toku (Control Flow Flattening) a šifrované payloady, ktoré sa dešifrujú až v pamäti. Statický pohľad na disk neodhaľuje skutočný zámer programu.
- **Cieľ ladenia:** Interogácia procesu v reálnom čase (Runtime Analysis). Ide o pozorovanie interakcie malvéru s operačným systémom, súborovým systémom a sieťou v momente, keď sa "cíti bezpečne" a odhalí svoj kód.

Ladenie vs. Reverzné inžinierstvo: Je kritické pochopiť rozdiel v mentálnom nastavení:

- **Vývojár:** Používa debugger na validáciu vlastných predpokladov a hľadanie chýb (bugy), ktoré spôsobujú pád alebo nesprávne správanie. Pozná zdrojový kód.
- **Analytik:** Pracuje s "čiernou skrinkou". Hľadá skrytú logiku, ktorú sa autor snažil zatajiť. Cieľom nie je opraviť program, ale dešifrovať payloady, extrahovať konfiguráciu (C2 servery) a identifikovať IoC (Indicators of Compromise) pre obranu siete.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Architektúra ELF – Dualita pohľadov

Dva paralelné pohľady na obsah súboru: Formát ELF je navrhnutý tak, aby slúžil dvom rôznym fázam životného cyklu programu:

1. **Pohľad linkovania (Linking View) – Statický:**
   - Reprezentovaný **Tabuľkou hlavičiek sekcií (Section Header Table - SHT)**.
   - Obsahuje logické celky kódu a dát: `.text` (kód), `.data` (inicializované dáta), `.bss` (nulované dáta), `.symtab` (symboly), `.strtab` (reťazce).
   - Tento pohľad je určený pre kompilátor a linker (`ld`), aby vedeli spojiť objektové súbory do výslednej binárky.
2. **Pohľad vykonávania (Execution View) – Dynamický:**
   - Reprezentovaný **Tabuľkou hlavičiek programu (Program Header Table - PHT)**.
   - Definuje **Segmenty**: Súvislé bloky dát, ktoré OS loader mapuje priamo do virtuálnej pamäte procesu. Jeden segment môže obsahovať viacero sekcií.
   - Toto je jediná časť, ktorá zaujíma loader jadra OS pri spustení (`execve`).

Kľúčový poznatok pre analytika: Pre úspešné spustenie programu je nutná iba PHT. SHT je pre beh programu redundantná.

- **Technika "Section Stripping":** Útočníci často úplne odstránia alebo poškodia SHT.
- **Dôsledok:** Nástroje závislé na sekciách (klasické disassemblery, `objdump`) môžu zlyhať alebo zobraziť nezmysly, zatiaľ čo program sa bez problémov spustí, pretože jadro číta len segmenty.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Detailná analýza štruktúr ELF

ELF Hlavička (Ehdr):

- **Magické bajty:** `0x7F 'E' 'L' 'F'`. Prvá kontrola, ktorú robí jadro.
- **Entry Point (`e_entry`):** Virtuálna adresa prvej inštrukcie, ktorá sa vykoná po inicializácii prostredia. U legitímneho softvéru mieri do `_start` (glibc), u malvéru často priamo na začiatok rozbaľovacieho stubu alebo dekódovacej rutiny.

Programové hlavičky (Phdr) - Segmenty: Loader číta tieto hlavičky, aby vedel, ako pripraviť pamäť:

- **PT_LOAD:** Segmenty, ktoré sa kopírujú zo súboru do RAM.
  - *Kritický indikátor malvéru:* Segment s kombináciou príznakov `PF_W` (zápis) + `PF_X` (vykonávanie) = **RWX**.
  - Moderné systémy vynucujú politiku **W^X (Write XOR Execute)** – pamäť môže byť buď zapisovateľná, alebo vykonateľná, nikdy nie oboje naraz. Výskyt RWX segmentu takmer vždy indikuje prítomnosť packera (potrebuje zapísať kód a potom ho spustiť), JIT kompilátora alebo shellcode injektoru.
- **PT_DYNAMIC:** Obsahuje informácie pre dynamický linker (zoznam potrebných knižníc `.so`, relokácie). Čisto staticky linkovaný malvér (časté u GoLang alebo Rust binárok s cieľom portability) tieto sekcie (PT_DYNAMIC a PT_INTERP) vôbec neobsahuje.
- **PT_INTERP:** Cesta k interpretru (zvyčajne `/lib64/ld-linux-x86-64.so.2`). Zmenou tejto cesty môže malvér načítať vlastný škodlivý linker.

Financované Európskou úniou
NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Správa pamäte procesu v Linuxe

Pochopenie rozloženia virtuálneho adresného priestoru (VAS) je nutné pre orientáciu v debuggeri.

Štandardný Memory Layout:

1. **Text Segment (.text):** `RX` (Read-Only/Exec). Obsahuje inštrukcie procesora. Pokus o zápis sem vyvolá výnimku `SIGSEGV` (Segmentation Fault), čo je bezpečnostná funkcia chrániaca integritu kódu.
2. **Data (.data) & BSS:** `RW` (Read-Write). Globálne a statické premenné. `.bss` fyzicky v súbore nezaberá miesto, OS ho pri štarte alokuje a vynuluje.
3. **Heap (Halda):** `RW`. Dynamická pamäť spravovaná cez `malloc/free` (brk/sbrk). Rastie smerom nahor (k vyšším adresám).
   - *Relevancia:* Tu malvér často ukladá stiahnuté payloady, dešifrované konfigurácie alebo ukradnuté dáta pred exfiltráciou.
4. **Memory Mapping (mmap):** Priestor pre zdieľané knižnice (`libc.so`), súbory mapované do pamäte a anonymné mapovania. Tu sa často skrýva *fileless* malvér využívajúci volania ako `memfd_create`.
5. **Stack (Zásobník):** `RW`. Rastie smerom nadol (k nižším adresám). Obsahuje lokálne premenné, návratové adresy funkcií a parametre.
   - **Red Zone:** Špecifikum x86_64 System V ABI. Ide o 128 bajtov pod aktuálnym `rsp`, ktoré môžu funkcie využiť pre dočasné dáta bez nutnosti posúvať stack pointer (optimalizácia).

ASLR (Address Space Layout Randomization): Bezpečnostná funkcia jadra, ktorá pri každom spustení náhodne posúva bázy (začiatky) týchto oblastí (okrem `.text` u non-PIE binárok), čím sťažuje exploitáciu a “hardkódovanie” adries pri analýze. U moderných binárok kompilovaných ako PIE (Position Independent Executable) je ASLR aplikované aj na samotný kód (segment .text), čo znamená, že úplne všetky adresy v debuggeri sa pri každom spustení zmenia. V GDB je preto často nutné ASLR dočasne vypnúť.

Financované Európskou úniou
NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Ladiace rozhranie ptrace

Systémové volanie `ptrace()` (Process Trace) Toto je "švajčiarsky nožík" pre manipuláciu procesov v Linuxe. Všetky bežné debuggery (GDB, LLDB) aj trasovacie nástroje (`strace`, `ltrace`) sú len nadstavbou nad týmto jedným volaním.

Mechanizmus fungovania: Umožňuje procesu (tracer) prevziať úplnú kontrolu nad iným procesom (tracee). Keď je proces sledovaný, každý signál (okrem SIGKILL) spôsobí jeho zastavenie a notifikáciu tracera.

Kľúčové operácie (`man ptrace`):

- **PTRACE_TRACEME:** Volá ho *dieťa* (tracee) po `fork()` ale pred `exec()`. Tým hovorí jadru: "Chcem byť sledovaný mojím rodičom". Toto je kritický bod pre anti-debugging techniky.
- **PTRACE_ATTACH:** Umožňuje pripojiť sa k už bežiacemu procesu (zaslanie `SIGSTOP`). Vyžaduje oprávnenia (CAP_SYS_PTRACE alebo rovnaké UID).
- **PTRACE_PEEKTEXT / POKETEXT:** Čítanie a zápis strojového kódu alebo dát priamo v adresnom priestore sledovaného procesu. Toto sa deje pri vkladaní breakpointov.
- **PTRACE_GETREGS / SETREGS:** Čítanie a prepisovanie registrov CPU (`rip`, `rsp`, `rax`...). Umožňuje debuggeru zmeniť tok programu (napr. preskočiť podmienku).

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

GDB – Typy Breakpointov a ich detekcia

Analytik musí vedieť, aký typ breakpointu použiť, aby nebol odhalený.

1. Softvérové Breakpointy (SW BP):

- **Implementácia:** Debugger prepíše prvý bajt inštrukcie na cieľovej adrese inštrukciou `INT 3` (opcode `0xCC`). Pôvodný bajt si uloží.
- **Vykonanie:** Keď CPU narazí na `0xCC`, vyvolá prerušenie (interrupt) a pošle procesu signál `SIGTRAP`. Jadro tento signál zachytí a pozastaví proces, čím odovzdá kontrolu debuggeru.
- **Riziko detekcie:** Malvér môže čítať vlastnú pamäť (Code Integrity Check) a hľadať bajty `0xCC`. Ak ich nájde, vie, že je modifikovaný/ladený a ukončí sa alebo zmení správanie.

2. Hardvérové Breakpointy (HW BP):

- **Implementácia:** Využitie špeciálnych debug registrov procesora x86 (DR0 až DR3). Do registra sa zapíše adresa, ktorú má CPU sledovať.
- **Vykonanie:** Komparátor v CPU pri každom prístupe do pamäte kontroluje zhodu s DRx registrami. Ak nastane zhoda, vyvolá sa výnimka `INT 1`. - DB (Debug Exception)
- **Výhoda:** Kód v pamäti sa vôbec nemení. Kontrola integrity kódu (CRC/Hash) nič nezistí.
- **Limit:** Procesor má obmedzený počet debug registrov.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

GDB – Automatizácia a Forenzná analýza pamäte

GDB nie je len interaktívny nástroj, ale plne skriptovateľný framework.

Skriptovanie v Pythone: Manuálne krokovanie dešifrovacej slučky (XOR loop), ktorá beží tisíckrát, je nemožné. GDB Python API umožňuje:

- Automatizovať extrakciu dát po každej iterácii.
- Obchádzať anti-debug kontroly bez manuálneho zásahu.
- **Príklad bypassu:** Napísanie hooku, ktorý zachytí volanie syscallu `ptrace`. Keď malvér zavolá `ptrace` pre kontrolu debuggera, skript automaticky podvrhne návratovú hodnotu v registri `rax` na 0 (úspech), čím malvér oklame.

Analýza Core Dumpov (`gcore`):

- **Koncept:** Vytvorenie kompletného obrazu pamäte (snapshot) bežiaceho procesu v danom čase príkazom `gcore <PID>`.
- **Výhoda pre analytika:** Namiesto riskantnej živej analýzy, kde malvér môže detegovať debugger a zničiť dáta, môžeme proces nechať "rozbaliť sa" do pamäte, potom ho zmraziť, spraviť dump a ten analyzovať staticky.
- **Obsah dumpu:** Obsahuje dešifrovaný kód, rozbalené moduly, otvorené sieťové spojenia a kľúče v čitateľnej podobe na Heap-e.
- Ideálne pre malvér, ktorý agresívne zhadzuje debugger pri dlhšom pripojení (timeouty).

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Anti-Debugging Techniky (1/2): Detekcia prostredia

1. Self-Debugging (Ptrace Check):

- **Princíp:** Linux kernel dovoľuje, aby bol proces sledovaný **maximálne jedným** tracerom súčasne.
- **Implementácia:** Malvér sám na seba zavolá `ptrace(PTRACE_TRACEME, 0, 0, 0)`.
  - Ak volanie uspeje (vráti 0), malvér vie, že je "čistý" a pokračuje.
  - Ak volanie zlyhá (vráti -1, EPERM), znamená to, že už je pripojený iný tracer (náš GDB/Strace). Malvér sa okamžite ukončí.
- **Eliminácia:** Použitie `LD_PRELOAD` knižnice, ktorá prepíše funkciu `ptrace` v user-space tak, aby vždy vracala 0, alebo použitie príkazu `catch syscall ptrace` v GDB a manipulácia návratovej hodnoty.

2. Inšpekcia `/proc/self/status`:

- **Princíp:** Súborový systém `/proc` je rozhranie k jadru. Súbor `status` obsahuje detailné informácie o procese, vrátane poľa `TracerPid`.
- **Detekcia:** Malvér otvorí tento súbor a parsuje riadok `TracerPid`. Hodnota `0` znamená, že proces nie je ladený. Akákoľvek iná hodnota je PID debuggera.
- **Eliminácia:** Toto je ťažšie obísť v user-space. Vyžaduje hookovanie funkcií `open` a `read` (napr. cez Fridu), ktoré pri pokuse o čítanie tohto konkrétneho súboru vrátia falošný obsah s `TracerPid: 0`.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Anti-Debugging Techniky (2/2): Čas a Integrita

3. Časové útoky (Timing Attacks):

- **Princíp:** Ladenie drasticky spomaľuje beh programu. Človek (analytik) potrebuje sekundy na stlačenie klávesy "Step", zatiaľ čo CPU vykoná milióny inštrukcií za milisekundu.
- **Implementácia:** Malvér zmeria čas pred a po vykonaní bloku kódu pomocou inštrukcie `RDTSC` (Read Time-Stamp Counter), ktorá vracia počet cyklov CPU od reštartu. Pokročilý malvér pred volaním RDTSC často volá inštrukciu CPUID, aby vynútil serializáciu inštrukcií (zabránil out-of-order execution) a získal tak exaktne presný čas dešifrovacej slučky.
  - `t1 = RDTSC; vykonaj_kod(); t2 = RDTSC; delta = t2 - t1;`
  - Ak `delta > Prahová hodnota` (napr. 0x10000 cyklov), je prítomný debugger alebo virtualizácia.
- **Eliminácia:** Veľmi náročné v user-mode. Efektívne riešenie vyžaduje hypervízor, ktorý zachytí inštrukciu `RDTSC` a podvrhne jej výsledok (spomalí virtuálny čas), alebo binárne patchovanie inštrukcie na `NOP`.

4. Kontrola integrity (Checksumming):

- **Princíp:** Malvér si vypočíta CRC32 alebo Hash vlastného `.text` segmentu za behu.
- **Detekcia:** Ak analytik vložil Softvérový Breakpoint (`0xCC`), hash sa zmení a nezodpovedá uloženej hodnote.
- **Dôsledok:** Malvér sa môže "ticho" poškodiť (zmení dešifrovací kľúč na základe nesprávneho hashu), takže dešifrovaný payload bude nepoužiteľný, čím zmätie analytika.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Dynamická Binárna Inštrumentácia

Čo je Frida? Revolučný nástroj v dynamickej analýze. Na rozdiel od GDB, ktoré proces "zastavuje", Frida do procesu **injektuje** JavaScript engine a umožňuje prepisovať logiku aplikácie za plného behu.

Architektúra:

1. **Client:** Python skript alebo CLI na počítači analytika.
2. **Server:** Binárka bežiaca na cieľovom systéme (napr. Android, Linux Server), ktorá komunikuje s OS.
3. **Agent:** Knižnica `.so`, ktorá je injektovaná do cieľového procesu pomocou `ptrace`, vytvorí si vlastné vlákno a naštartuje JS engine.

Výhody oproti GDB:

- **Non-intrusive:** Nezastavuje beh procesu pri každej inštrukcii, čo je kľúčové pre analýzu sieťových timeoutov a real-time systémov.
- **High-level API:** Prístup k pamäti a funkciám pomocou JavaScriptu (`Memory.readUtf8()`, `Interceptor.attach()`).
- **Univerzálnosť:** Rovnaké skripty často fungujú na Linuxe, Androide aj iOS.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Frida – Praktické využitie

Väčšina malvéru dnes komunikuje cez HTTPS (TLS). Analýza paketov vo Wiresharku zobrazí len šifrovaný šum.

Problém: Potrebujeme vidieť obsah komunikácie (príkazy z C2, exfiltrované heslá) v čitateľnej podobe (plaintext). Nemáme privátny kľúč servera útočníka.

Riešenie s Fridou (Man-in-the-Process): Šifrovanie sa deje až v user-space knižniciach (OpenSSL, GnuTLS), nie v jadre ani na sieťovej karte.

1. **Cieľ:** Funkcia `SSL_write` v knižnici `libssl.so`. Táto funkcia berie ako vstup *nešifrované dáta* a buffer, kam zapíše výsledok.
2. **Akcia:** Použijeme Fridu na "zavesenie sa" (hook) na začiatok tejto funkcie.
3. **Implementácia:**

```
Interceptor.attach(Module.findExportByName("libssl.so", "SSL_write"), {
    onEnter: function(args) {
        // args[1] je pointer na buffer s dátami
        console.log("Odosielané dáta (Plaintext):");
        console.log(Memory.readByteArray(args[1], parseInt(args[2])));
    }
});
```

Výsledok: Vidíme presný obsah požiadaviek ešte predtým, než sú zašifrované a odoslané na sieť. Obchádzame tak potrebu riešiť certifikáty a MITM proxy.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Statická Emulácia: Radare2 a ESIL

Čo robiť, ak je vzorka príliš nebezpečná na spustenie alebo cieli na architektúru, ktorú nemáme (napr. IoT MIPS malvér na x86 PC)?

Radare2 (r2): Komplexný open-source framework pre reverzné inžinierstvo.

ESIL (Evaluable Strings Intermediate Language):

- Radare2 nevykonáva natívny kód priamo. Prekladá inštrukcie (x86, ARM, MIPS...) do spoločného medzijazyka ESIL (založený na RPN - Reverse Polish Notation).
- **Výhoda:** Umožňuje **emuláciu kódu** vo virtuálnom CPU, ktorý je súčasťou Radare2. Nemusíme mať fyzický procesor danej architektúry.

Využitie – Dešifrovanie reťazcov bez spustenia: Malvér často dešifruje svoje konfiguračné reťazce (IP adresy) až pri štarte.

1. V r2 inicializujeme emuláciu: `aei` (analyze emulation init).
2. Nastavíme registre a pamäť stacku: `aeim`.
3. Necháme emulátor vykonať dešifrovaciu funkciu: `aecu <adresa_konca>`.
4. Prečítame výsledok z emulovanej pamäte.

- **Bezpečnosť:** Všetko sa deje v pamäti nástroja r2, žiadny škodlivý kód sa reálne nevykonáva na OS, nehrozí infekcia ani sieťová komunikácia.

Financované Európskou úniou
NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Techniky balenia (Packing) a Obfuskácie

Cieľ packerov: Sťažiť statickú analýzu (aby reťazce neboli viditeľné cez `strings`) a obísť antivírové signatúry (zmena hashu súboru pri zachovaní funkčnosti).

Anatómia zabaleného súboru: Namiesto pôvodného kódu súbor obsahuje len:

1. **Stub (Zavádzač):** Malý kúsok kódu, ktorý je spustiteľný.
2. **Payload:** Pôvodný program, ktorý je skomprimovaný alebo zašifrovaný a uložený ako dáta.

Indikátory zabalenia (Heuristika):

- **Vysoká entropia:** Shannonova entropia sekcií blízka hodnote 8.0 indikuje náhodné alebo šifrované dáta. Bežný kód má entropiu okolo 5-6.
- **Neštandardné názvy sekcií:** Napr. `.UPX0`, `.protect`, alebo nezmyselné reťazce.
- **Redukovaná tabuľka importov:** Zabalený program má zredukovanú sekciu PLT/GOT (Procedure Linkage Table / Global Offset Table). Často obsahuje len odkazy na dynamický linker (napr. funkcie dlopen, dlsym v Linuxe, alebo LoadLibrary/GetProcAddress vo Windows), pomocou ktorých si zvyšok knižníc malvér načíta do pamäte sám.
- **RWX Segmenty:** Ako bolo spomenuté, packer potrebuje miesto, kam rozbalí kód a odkiaľ ho následne spustí.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Postup manuálneho rozbaľovania

Univerzálna stratégia pre neznáme packery, kde nefunguje `upx -d`.

Metodika "Find OEP & Dump":

1. **Nájdenie OEP (Original Entry Point):**
   - Cieľom je nájsť moment, kedy Stub končí svoju prácu a odovzdáva riadenie pôvodnému programu.
   - Hľadáme "Tail Jump" – zvyčajne inštrukcia skoku (`JMP` alebo `CALL`), ktorá mieri na vzdialenú adresu alebo do inej pamäťovej sekcie.
   - V grafovom zobrazení disassembleru je to často jediný prechod z jednej komplexnej zhluky blokov do inej, čistej oblasti.
2. **Dynamické sledovanie:**
   - V GDB nastavíme hardvérový breakpoint na OEP: `hbreak *OEP`.
   - Spustíme program (`run`). Počkáme, kým sa dekompresia dokončí a debugger zastaví na OEP.
   - V tomto bode je payload v pamäti plne rozbalený a čitateľný.
3. **Memory Dump & Rekonštrukcia:**
   - Použijeme `gcore` alebo plugin pre dumpovanie procesu.
   - Získaný dump nie je okamžite spustiteľný. Musíme opraviť ELF/PE hlavičku, Entry Point a zrekonštruovať prepojenia na knižnice (oprava importov – IAT fix pre Windows, oprava GOT/PLT pre Linux), aby OS vedel binárku znova načítať.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Fileless Malvér & techniky "Living off the Land"

Moderný útočník sa snaží nezanechať stopy na pevnom disku (HDD/SSD), aby obišiel tradičnú forenznú analýzu.

Technika `memfd_create`:

- Zneužitie legitímneho systémového volania `memfd_create()`, ktoré bolo pôvodne určené pre optimalizáciu zdieľanej pamäte (napr. pre Wayland grafický server).
- Vytvorí **anonymný súbor** (file descriptor), ktorý existuje **iba v RAM**.
- Tento súbor sa správa ako bežný súbor, ale nie je pripojený k žiadnemu adresáru na disku. Príkaz `ls` ho nevidí.

Mechanizmus infekcie:

1. Malvér (dropper) alokuje pamäť cez `memfd_create`.
2. Stiahne zo siete ELF payload a zapíše ho do tohto deskriptora.
3. Spustí ho priamo z pamäte pomocou volania `fexecve()`, ktoré akceptuje file descriptor namiesto cesty k súboru.

Analýza a Detekcia:

- Tradičná disková forenzná analýza (image disku) tento malvér nenájde, pretože po vypnutí počítača dáta zmiznú z RAM.
- **Live Forensics:** Artefakty sú viditeľné v `/proc/<PID>/fd/`. Symbolický odkaz bude ukazovať na `/memfd:nazov (deleted)`.
- Analytik musí extrahovať obsah priamo čítaním tohto symlinku (`cp /proc/pid/fd/3 /tmp/recovered_malware`).

Financované Európskou úniou
NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

Záver

Zhrnutie pre prax:

1. **Poznajte svoje nástroje:** GDB + Python je nevyhnutnosť pre bypass ochrán. Frida je kráľom inštrumentácie pre moderné aplikácie. Radare2/Cutter poskytuje silnú statickú analýzu zadarmo.
2. **Vrstvite metódy:** Žiadna technika nie je 100%-ná. Kombinácia statickej analýzy (pre prehľad) a dynamickej analýzy (pre detaily) prináša najlepšie výsledky.
3. **Adaptabilita:** Kybernetická bezpečnosť je ako preteky v zbrojení. Útočníci neustále inovujú (eBPF rootkity, custom packery), analytik musí ovládať nízkoúrovňové princípy OS (Kernel internals), aby pochopil nové hrozby.

Etické a právne upozornenie: Všetky prezentované techniky (reverzné inžinierstvo, obchádzanie ochrán) musia byť vykonávané výhradne v izolovanom laboratórnom prostredí (Sandbox/VM) a len na vzorkách, na ktoré máte oprávnenie, alebo na verejne dostupných crackmes na edukačné účely.

Odporúčanie: Začnite s platformami ako *Crackmes.one* alebo *Root-Me*, kde si môžete tieto techniky natrénovať legálne.

Financované Európskou úniou NextGenerationEU
PLÁN OBNOVY
MINISTERSTVO INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA A INFORMATIZÁCIE SLOVENSKEJ REPUBLIKY

KCKB UPJS

UNIVERZITA
PAVLA JOZEFA ŠAFÁRIKA
V KOŠICIACH

Financované
Európskou úniou
NextGenerationEU

PLÁN OBNOVY

MINISTERSTVO
INVESTÍCIÍ, REGIONÁLNEHO ROZVOJA
A INFORMATIZÁCIE
SLOVENSKEJ REPUBLIKY

Ďakujem za pozornosť.