MI-PB-13

Analýza toku kódu. Prolog, epilog a tělo funkce. Vstupní bod do programu.

Tok kódu

Základní blok: maximální posloupnost po sobě jdoucích instrukcí, do které lze vstoupit jedině první instrukcí posloupnosti a opustit ji jedině poslední instrukcí.

Tvorba grafu toku kódu:

- Rozdělit funkci na základní boky
- Vrcholy: základní bloky
- Orientované hrany: tok kódu z jednoho bloku do druhého

Příkaz if:

```
    aritmerická operace počítající podmínku
    jxx přeskočit
    výraz
přeskočit:
```

```
    aritmerická operace počítající operand podmínky
    cmp
    jxx přeskočit
    výraz
přeskočit:
```

Smyčka while:

```
zacatek:
    test ecx,ecx
    je za_smyčkou
    tělo
    jmp zacatek
za_smyčkou
```

Smyčka for:

```
zacatek:
    test ecx,ecx
    jle za_smyčkou
    tělo
    inc eax
    cmp eax, ecx
    jl zacatek
za_smyčkou
```

Switch:

- Použití `sub` / `dec` : argument switche se postupně snižuje a testuje na 0
- Použití `cmp` : argument switche se porovnává s hodnotami
- Skoková tabulka: tabulka s adresami začátků `case` , skok na adresu z ní

Reverzování funkcí: Prolog, tělo, epilog

Každá funkce má:

- **Prolog** (volitelně): několik prvních instrukcí, které tvoří rámec zásobníku, alokují prostor pro lokální proměnné, zajístí zarovnání zásobníku a uloží všechny registry, které je nutno zachovat. Mohou uložit i kanárka nebo strukturu pro obsluhu výjimek
  - Natavení `ebp` na hodnotu `esp` : pomocí `ebp` lze odkazovat na zásobník (`ebp+offset` ≥ 8 odkazuje na **argumenty**, `ebp-offset` odkazuje na lokální proměnné)
  - Analytikovi řekne:
    - Kolik bytů mají lokální proměnné
    - Zarovnání zásobníku
    - Využití rámce zásobníku
    - Využití kanárků
    - Využití SEH
- **Tělo**
- **Epilog** (volitelně): poslední instrukce funkce těsně před instrukcí `ret` , ruší rámec zásobníku, obnoví registry, ověří kanárka, volitelně vyčistí parametry ze zásobníku
  - Analytikovi řekne:
    - Kde funkce končí
    - Testy kanárků
    - Obnovení `EXCEPTION_REGISTRATION`
    - Volací konvence (`ret XXX` vs `ret`)

Analýza rámce zásobníku:

- Rozdělení prostoru vytvořeného instrukcí `sub esp, __LOCAL_SIZE`
- Analýza instrukcí pracujících se zásobníkem (`mov, push, lea, ...`)
  - V každé instrukci zaměření na: offset (pozice v zásobníku), velikostní modifikátor (byte, word, dword...)

Kanárky:

- Prevence stack smashing (úmyslné přepsání zásobníku)
- Náhodné slovo vložené na zásobník
- Před ukončením funkce zkontrolován, pokud nastane neshoda, program okamžitě ukončen

Analýza znamének typů:

- Podle aritmetických instrukcí a jim oodpovídajících instrukcí `cmp/test` , `jxx` :
  - Pokud skok založen na příznaku `CF` , jde o bezznaménkový typ
  - Pokud na `SF` / `OF` , jde o znaménkový

Analýza API volání:

- Volání externích (zdokumentovaných) modulů -- lze odvodit typy, délky a význam parametrů

Hlavní vstupní bod

Adresa vstupního bodu: `AddressOfEntryPoint` v Optional Header PE souboru

Vstupní bod poskytnut runtime knihovnou jazyka (`mainCRTStartup`): *(CRT = C Run-Time)*

- Volání **inicializátorů**
  - Inicializace globálních proměnných, standardních typů, globálních konstant
  - Ruční inicializátory: `#pragma section(".CRT$XIB")` - umístění ukazatele na funkci do sekce inicializátorů
  - Funkce `_initterm_e` : Procházení pole s ukazateli na inicializační funkce, volání funkcí
- **Terminátory:** Funkce volané po ukončení programu
  - Ukazatele na funkce v poli, na které ukazují `__onexitbegin` a `__onexitend` . Funkce se volají po zavolání C funkce `exit`

Nalezení `main` : často je u konce vstupní funkce, pozná se podle předávaných argumentů

Další aspekty

Kódování ukazatelů: zabrání útočníkovi využití (přepisu) ukazatelů na zásobníku/haldě
API `EncodePointer` / `DecodePointer`

Podpora hot-patching: Instrukce `nop` před začátkem funkce -- snadná úprava bez nutnosti restartovat aplikaci
Ze začátku funkce se skočí na začátek nopů, kam je zapsán skok na upravenou funkci

Skok na instrukci `jmp` : umožňuje za běhu nahradit libovolnou funkci

Import Address Table:

- Jména modulů a symbolů (nebo jejich pořadová čísla) jsou v **importním adresáři** PE souboru
- Používání nepřímých skoků -- IAT obsahuje adresy importovaných funkcí
- Změna v IAT -- přesměrování všech volání daného API
  - IAT standardně read-only, ale zápis lze povolit (např. pomocí `VirtualProtect`)
  - Lze se dostat i za hranice procesu