Windows Procesi

V magičnem svetu Windows-a so procesi ključnega pomena. Ampak kaj sploh so?

V Windows so procesi predstavljeni z strukturo _EPROCESS. Je skoraj nedokumentirana iz strani Microsofta. Njene dele lahko dobimo z raznimi rutinami, medtem ko se ona sama lahko spreminja med verzijam Windowsa.

Sama struktura izgleda nekako tako (veliko stvari izbrisanih zaradi preglednosti. Cela verzija: https://www.vergiliusproject.com/)

```
nt!_EPROCESS
   ...
   +0x464 Flags            : Uint4B
  ...
   +0x468 CreateTime       : _LARGE_INTEGER
   +0x470 ProcessQuotaUsage : [2] Uint8B
   +0x480 ProcessQuotaPeak : [2] Uint8B
  ...
   +0x4b8 Token            : _EX_FAST_REF
  ...
   +0x548 OwnerProcessId   : Uint8B
   +0x550 Peb              : Ptr64 _PEB
   +0x558 Session          : Ptr64 _MM_SESSION_SPACE
   +0x560 Spare1           : Ptr64 Void
   +0x568 QuotaBlock       : Ptr64 _EPROCESS_QUOTA_BLOCK
   +0x570 ObjectTable      : Ptr64 _HANDLE_TABLE
  ...
   +0x7d4 ExitStatus       : 0n259
  ...
```

Seveda je še ogromno drugih polj.
Meni zanimive so:

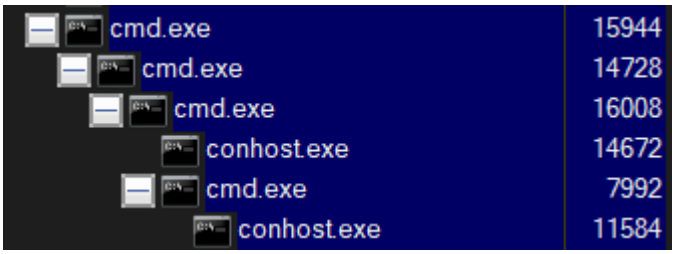
- Peb (Process Environment Block): ki je kazalec na strukturo PEB, ki je neke vrste nadomestilo za EPROCESS v user mode (EPROCESS se lahko uporablja le v jedru).
- OwnerProcessId: pove kdo je starš procesa
- Token: kaže na strukturo _EX_FAST_REF, ki vsebuje varnostne informacije

Windows se od Linuxa razlikuje o hijerarhiji procesov. Linux ima strogo hiearhijo, kar pomeni, da otrok ne more obstajat brez starša (...razen systemmd, ki je na vrhu 🙄). Če izgubi starša postane otrok njegovega starša/nekega prednika.

Na Windows lahko ubijemo starša in bo otrok obstajal brez njega.

Poskus: Izdelava sirote

1. Uporabimo process explorer. Opazimo, da je 16008 starš 7992.



2. Tu ustavimo proces 16008. Proces 7992 izpade iz drevesa procesov.

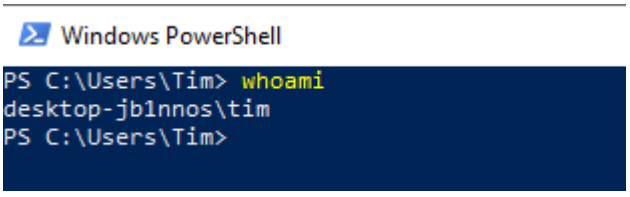
(Ime procesa, PID starša, PID otroka: "wmic process get ProcessId,ParentProcessId,Name")

3. Proces 7992 ostane. Njegov starš je še zmeraj 16008, čeprav ne obstaja več.

Naslednje bom govoril o sami varnosti procesov. Windows ti pusti VSE, če ga dosti močno šlataš. Kot je že prej bilo omenjeno ima vsak proces v svoji EPROCESS strukturi kazalec na nek token. Ta token je struktura _EX_FAST_REF. Ta je dokaj preprosta. Vsebuje kazalec do strukture _TOKEN (skoraj kazalec, zadnji 4 biti so za okras). _TOKEN struktura vsebuje veliko pomembnih stvari o katerih ne bom govoril, bom pa predstavil z primerom.

Poskus: Kaj se zgodi, če zamenjamo token nekemu procesu?

Za ta eksperiment uporabimo WinDbg (hotel sem narediti z pomočjo driverja, a sem dobil prevečkrat BSOD). Sam debugger bomo pritrdili na samo jedro sistema. Nato zaženemo powershell.



Nato najdemo kje v spominu se nahaja _EPROCESS.

```
lkd> !process 1234 0
Searching for Process with Cid == 1234
PROCESS ffff8b0f42ab0080
    SessionId: 1  Cid: 1234    Peb: 3a63873000  ParentCid: 206c
    DirBase: 19d1f0002  ObjectTable: ffff9b007c9e3280  HandleCount: 684.
    Image: powershell.exe
```

Če gremo na to lokacijo dobimo nekaj takega:

```
lkd> dt _eprocess ffff8b0f42ab0080
nt!_EPROCESS
   ...
```

```
    +0x4b8 Token              : _EX_FAST_REF
    ...
```

Našli smo offset tokna. Ta offset je pomemben, ker se lahko spremeni med različicam Windowsa. Na tem naslovu se nahaja _EX_FAST_REF objekt. Tu lahko tudi vidimo kje se skrivajo tisti 4 biti, ki jih lahko odstranimo (Glej offset).

```
lkd> dx -id 0,0,ffff8b0f3e718340 -r1 (*((ntkrnlmp!_EX_FAST_REF *)0xffff8b0f42ab0538))
(*((ntkrnlmp!_EX_FAST_REF *)0xffff8b0f42ab0538))                 [Type: _EX_FAST_REF]
    [+0x000] Object           : 0xffff9b00811b1603 [Type: void *]
    [+0x000 ( 3: 0)] RefCnt           : 0x3 [Type: unsigned __int64]
    [+0x000] Value            : 0xffff9b00811b1603 [Type: unsigned __int64]
```

Ne zanima nas kako izgleda token. Vse kar nas zanima je kater naslov je treba spremenit, da bo kazal na nek drug token.

Sedaj pa rabimo izbrat nek token, ki ima veliko privilegijev. Vsak sistem je drugačen, ampak vsak sistem ima na PID 4 proces, ki se imenuje 'System'. Kot zanimivost vsi PID na Windows so deljivi z 4 (zdi se mi, da zaradi recikliranja kode).

Tu lahko lokacijo tokna pogledamo direktno.

```
lkd> !process 4 1
Searching for Process with Cid == 4
PROCESS ffff8b0f35ab3080
    SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 001aa002  ObjectTable: ffff9b0072411040  HandleCount: 3522.
    Image: System
    VadRoot ffff8b0f37371f60 Vads 35 Clone 0 Private 27. Modified 163196.
Locked 64.
    DeviceMap ffff9b007245cba0
    Token                             ffff9b0072426810
    ElapsedTime                       00:29:01.152
    UserTime                          00:00:00.000
    KernelTime                        00:00:05.671
    QuotaPoolUsage[PagedPool]         0
    QuotaPoolUsage[NonPagedPool]      272
    Working Set Sizes (now,min,max)  (1260, 50, 450) (5040KB, 200KB, 1800KB)
    PeakWorkingSetSize                3723
    VirtualSize                       9 Mb
    PeakVirtualSize                   29 Mb
    PageFaultCount                    26034
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      52
```

Naslov lahko preprosto prekopiramo.

```
    PeakVirtualSize                   29 Mb
    PageFaultCount                    26034
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      52

lkd> eq ffff8b0f42ab0080+0x4b8 ffff9b0072426810

lkd>
```

```
Windows PowerShell
PS C:\Users\Tim> whoami
desktop-jb1nnos\tim
PS C:\Users\Tim> whoami
nt authority\system
PS C:\Users\Tim>
```

Powershell ima sedaj enake pravice kot jedro OS (o tem sanjam ponoči).

Kot zanimivost 'nt' v 'nt authority' prihaja iz Windows NT. Windows NT je bil prvi Windows 32bit OS. Vse verzije Windowsa so še zmeraj narejene na isti tehnologiji (tudi za telefon, xbox,...).

Govoril bom še malo o nitih. V enem procesu je lahko več niti. Za razliko procesov v Windows niti nimajo svojega virtualnega prostora, ampak si ga delijo z drugimi niti v procesu. Vsak proces ima vsaj eno nit (razen v posebnih primerih, ko pride do napake v jedru).

Niti delujejo na principu 'concurrency' in ne 'paralelizma'. To pomeni, da se lahko na enkrat izvaja le ena nit, ampak se te niti hitro menjajo. Za to skrbi 'scheduler'. Ker je to zahtevno Windows uporablja (popravek: je uporabljal) dva mehanizma: vlakna in UMS.

Vlakna (fiber) niso del jedra (oz. jedro ne upravlja z njimi). Ustvarja se jih z funkcijo 'ConvertThreadToFiber' in se rabijo preklapljati ročno z 'SwitchToFiber'. So del zgodovine in se redko uporabljajo. Edina uporaba za njih je portiranje aplikacij v Windows, ki se same ukvarjajo z svojimi niti.

UMS (User-mode scheduling) niti so vidne v jedru. Jedro jih naredi, menja kontekst in jih ustavi. Aplikacija sama pa jih menja ('schedula'). Z Windows 11 je Microsoft ukinil podporo za UMS niti.