IT drošības incidenta pierādījumu materiāla iegūšana: operatīvā atmiņa

Ievads

Rezultatīvas IT drošības incidenta izmeklēšanas pamatā ir informācija par notikumiem un darbībām, kas reģistrēti laika nogrieznī pirms, pēc un incidenta brīdī.
IT drošības incidenta gadījumā vēlama šādu informācijas avotu pieejamība un apstrāde:

- centralizētu auditācijas pierakstu jeb žurnālfailu uzkrāšana;
- datorsistēmu diska kopijas (attēla) iegūšana;
- datorsistēmu operatīvās atmiņas kopijas (attēla) iegūšana;
- tīkla plūsmas auditācija;
- *Endpoint Detection and Response* (*EDR)* risinājumu integrācija, reglamentēta izmeklēšanas materiāla iegūšanas procedūra un paredzamā reakcija palīdzēs augstākminēto sasniegt efektīvāk (https://en.wikipedia.org/wiki/Endpoint_detection_and_response);
- anomāliju identificēšanas spēja visos augstākminētajos informācijas avotos.

CERT.LV uzsver, ka pieļaujamas novirzes no vēlamā scenārija, atkarībā no mērķa organizācijas IT drošības brieduma (*maturity*) pakāpes.

Operatīvā atmiņa ir viens no svarīgākajiem pierādījumu avotiem, it īpaši incidentos, kas saistīti ar ļaunatūru. Mūsdienu ļaunatūra var izvairīties no failu izmainīšanas, neatstājot nekādus pierādījumus ierīces pastāvīgajā atmiņā (cietajā diskā, atmiņas kartēs, tīkla failu sistēmās), taču jebkuram kodam, kas izpildās procesorā, ir vismaz uz brīdi jāparādās operatīvajā atmiņā.

Operatīvā atmiņa atspoguļo visu, kas darbojas *operētājsistēmā*, kā arī satur iepriekš izmantoto datu (nesen aizvērtas programmas, dzēsti faili, u.c.) atliekas, tad analīzes rezultātā var iegūt izmeklēšanai būtisku informāciju:

- izpildītos procesus
- atvērtos failus, to saturu
- tīkla komunikācijas artefaktus

- šifrēšanas atslēgas un atšifrētus datus
- paroles, lietojumu sesijas, autentifikācijas marķierus (tokens)
- atpakotu programmu kodu
- ļaunatūras kodu, kas injicēts citās programmās vai bibliotēkās
- sistēmas draiverus un bibliotēkas
- modificētas OS datu struktūras (liecina par rootkit)

Atkarībā no situācijas, var analizēt gan "dzīvas" (*live*) sistēmas, gan arī iepriekš iegūtus operatīvās atmiņas attēlus (*memory dump*).

Pirmais variants – "dzīvās" sistēmās analīze – ir populāra lielos korporatīvos tīklos, kur tādā veidā paralēli var analizēt simtiem vai tūkstošiem sistēmu un iegūt rezultātus tuvu reālajam laikam. Taču tas prasa ievērojamus ieguldījumus, tādēļ parasti šāds risinājums ir integrēts organizācijas uzraudzīšanas (*monitoringa*) risinājumā un ar datu apstrādi nodarbojas atsevišķa IT nodaļa, vai arī ar tā izveidi nodrošina ārpakalpojuma sniedzējs, kas ir pieaicināts apjomīga kiberincidenta risināšanā.

Tālāk šajā rakstā apskatīsim otro variantu – iepriekš iegūtu operatīvās atmiņas attēlu analīzi – kuras veikšanai nepieciešami divi soļi:

1. pierādījumu iegūšanu veic organizācijas/uzņēmuma IT darbinieki
2. iegūtos pierādījumus iespējams nodot tālākai analīzei CERT.LV, Valsts Policijai vai ārpakalpojuma sniedzējam, turklāt to var darīt arī paralēli un nepieciešamības gadījumā analīzi var arī atkārtot vēlāk, ja rodas jauni pierādījumi (piem. tiek atrastas šifrēšanas atslēgas)

Incidenta gadījumā pilnvērtīgāko pierādījumu kopu iespējams iegūt tikai pie nosacījumiem, ka mērķa sistēma netiek papildus ārēji ietekmēta. To nedrīkst izslēgt vai restartēt, kā arī, pirms iegūts atmiņas attēls, nav vēlama atslēgšana no tīkla, papildu pretvīrusu pārbaude vai jebkādas citas darbības ar sistēmu, kas nav tieši saistītas ar pierādījumu saglabāšanu.

Svarīgi! Kā pirmo jāiegūst atmiņas attēlu, veicot tikai tā iegūšanai nepieciešamās darbības!

Atmiņas un diska attēla iegūšana no *Windows* operētājsistēmas bez virtualizācijas:

1) Savlaicīgi sagatavo ārēju datu nesēju, vai tīkla disku, kura kapacitāte pārsniedz mērķa sistēmas diska un/vai operatīvās atmiņas apjomu. Jāpārliecinās par datu nesēja failu sistēmas piemērotību (piemēram *FAT32* failsistēma neļauj saglabāt failus lielākus par 4GB). Ātrāk darbs notiks, izmantojot *SSD* ārējos datu nesējus, īpaši svarīgi izmatot *SSD* datu nesēju ir atmiņas attēla iegūšanas procesā.
2) Uz ārējā datu nesējā jāieraksta atmiņas un diska attēla iegūšanai nepieciešamos rīkus: *Windows* gadījumā – *FTK imager*, vai *WinPMEM*

Atmiņas un diska attēla iegūšana no *Linux* oprētājsistēmas bez virtualizācijas:

1) Savlaicīgi sagatavo ārēju datu nesēju, vai tīkla disku, kura kapacitāte pārsniedz mērķa sistēmas diska un/vai operatīvās atmiņas apjomu. Jāpārliecinās par failu sistēmas piemērotību. Ātrāk darbs notiks, izmantojot SSD ārējos datu nesējus, īpaši svarīgi izmatot SSD datu nesēju ir atmiņas attēla iegūšanas procesā.

2) Uz ārējā datu nesējā jāieraksta atmiņas un diska attēla iegūšanai nepieciešamos rīkus *Linux* gadījumā - jau sakompilēts *AVML* (https://github.com/microsoft/avml) vai *LIME* atmiņas attēla iegūšanai, dd (https://en.wikipedia.org/wiki/Dd_(Unix)) rīks diska attēla iegūšanai parasti būs pieejams uz mērķa sistēmas;

Pierādījumu iegūšana bez virtualizācijas

Operatīvās atmiņas attēlu veidošana no pašas operētājsistēmas, kas tiek analizēta, jāveic tikai tad, ja to nevar panākt savādāk, t.i., ja apskatām fizisku darbstaciju, portatīvo datoru vai serveri, kas nav virtualizēts. Šī darbība ir saistīta ar vairākiem riskiem, kas jāapzinās, it īpaši, ja ir aizdomas par ļaunatūru vai aktīvu ielaušanos:

- ļaunatūra var pamanīt kriminalistikas (*forensics*) rīku, kas tiek izmantots atmiņas attēla iegūšanai (pēc nosaukuma, faila kontrolsummas, vai izmantotajiem *API*), un reaģēt uz to:
    - signalizējot operatoram par to, ka viņa darbības ir pamanītas un ir uzsākta pretreakcija
    - apslēpjot ļaunatūru atmiņā un diskā
    - dzēšot ļaunatūras pēdas sistēmā vai tīklā
    - izpildot ieplānotās destruktīvās darbības, kā, piemēram, diska šifrēšanu
    - bloķējot kriminalistikas (*forensics*) rīku vai pārveidojot tā darbības rezultātus
- ļaunatūra var inficēt pievienoto datu nesēju vai tīkla disku
- atmiņas attēla ierakstīšana failu sistēmā var pārrakstīt neseno darbību pēdas
- atmiņas attēla ierakstīšanai var nepietikt vietas failu sistēmā (ja tas tiek rakstīts lokālajā diskā)

Tādēļ, ja šādas darbības ir jāveic, vēlams tās iepriekš notestēt uz līdzīgām sistēmām un, atkārtojot potenciāli inficētajā sistēmā, darboties pēc iespējas ātrāk, bez liekas kavēšanās.

Ja pētāmā sistēma ir pieslēgta *Windows* aktīvai direktorijai (*Active Directory*), vai ir kā savādāk centralizēti pārvaldīta, atmiņas attēla veidošanu var automatizēt, izpildot komandas

un rakstot rezultātus koplietojamā diskā/failu sistēmā. Gadījumā, ja šīs darbības tiek izpildītas uz vairākām sistēmām vienlaicīgi, svarīgi pievērst uzmanību koplietojamās failu sistēmas tiesībām (*permissions*), lai inficētās sistēmas nevar kompromitēt koplietotos failus, tādā veidā bojājot pierādījumus, kas iegūti no citām sistēmām.

Būtiska problēma, kas skar atmiņas attēlu veidošanu no operētājsistēmu iekšpuses, ir tā saucamais "*memory smearing*" efekts. Tā pamatā ir fakts, ka atmiņas nolasīšana un ierakstīšana datu nesējā aizņem ievērojamu laiku – tas ir proporcionāls operatīvās atmiņas apjomam un datu nesēja ierakstīšanas ātrumam. Operētājsistēmā aktīvie procesi šajā laikā turpina darboties, tādēļ pie lielas slodzes un liela operatīvās atmiņas apjoma var gadīties, ka operatīvās atmiņas saturs būtiski izmainās attēla veidošanas laikā. Rezultātā izveidotais atmiņas attēls var saturēt pretrunīgu vai bojātu informāciju. Tas nav atkarīgs no izmantotā rīka, un, lai nodrošinātu, ka tā nenotiek ir jāizmanto kāda no šīm metodēm:

- atmiņas attēla veidošana no virtualizācijas *host* iekārtas puses (gadījumā, ja tiek izmantota virtualizācija)
- [invazīvi] "*cold boot*" metode, kad sistēma tiek momentāni atslēgta (piem., atslēdzot elektrības padevi) un tūlīt pat pārstartēta *LiveCD* režīmā, kas nolasa operatīvajā atmiņā atlikušos datus
- [invazīvi] *BSOD* izsaukšana *Windows* vidē, iepriekš norādot sistēmas iestatījumos pilna *crash dump* izveidi šajā situācijā
- [invazīvi] *kexec* mehānisma izmantošana *Linux* vidē, lai apstādinātu sistēmas darbību un nomainītu kodolu (ar iebūvētu *initramfs*), nezaudējot iepriekšējo operatīvās atmiņas saturu

Windows

Svarīgi! Kā pirmo jāiegūst atmiņas attēlu, veicot tikai tā iegūšanai nepieciešamās darbības!

Atmiņas attēla veidošanai *Windows* operētājsistēmā ir pieejami daudzi risinājumi, starp populārākajiem, kas joprojām tiek atbalstīti, ir:

- *FTK Imager*
- *Memoryze*
- *Belkasoft Ram Capturer*

Dažādiem rīkiem ir atšķirīgas priekšrocības un papildu funkcionalitāte, kas var būt noderīga īpašos apstākļos. Viens no vienkāršākajiem rīkiem, kas ir arī bezmaksas atvērtā koda risinājums un tiek aktīvi attīstīts, šobrīd ir *WinPmem*: https://github.com/Velocidex/WinPmem/releases

Izmantojot šo rīku operatīvās atmiņas izveidošanai jānorāda vien faila nosaukums, kur saglabāt izveidoto attēlu:

> *winpmem.exe E:\\location.dmp*

Pie tam, failu var rakstīt arī koplietotajā mapē, izmantojot sintaksi:

> *winpmem.exe \\server\shared-folder-name\path*

Linux

Linux vidē ilgu laiku par standarta rīku atmiņas attēlu veidošanai tika uzskatīts *LiME* (https://github.com/504ensicsLabs/LiME/), taču pēc struktūras tas ir *Linux* kodola modulis, un līdz ar to tas ir jākompilē konkrētai sistēmai (precīzāk konkrētai kodola versijai). Lai gan kompilācija nav obligāti jāveic pašā sistēmā, tomēr šis process ir salīdzinoši neērts.

Tādēļ šobrīd ieteicams atmiņas attēlus veidot, izmantojot jaunāku rīku - *AVML* (https://github.com/microsoft/avml), kas ir rakstīts *Rust* valodā un tiek izplatīts kā statiski kompilēts izpildāmais fails (*binary*), līdz ar to ir vienkāršāk izmantojams:

> *$ sudo avml linux-example.dmp*

Konteineri

Docker, LXD/LXC un citi konteineru risinājumi patiesībā ir parasti *Linux* procesi, lai gan ar specifisku *sandboxing* funkcionalitāti. To atmiņas attēlus var iegūt no uzturētāja (*host*) puses, saglabājot visu procesu *core dumps*, kuri ir palaisti konteinerī. Piemēram, noskaidrojot kāda konteinera *PID*, var atrast visus tā palaistos procesus, tad iziet tiem cauri un saglabāt *core dump* katram no šiem procesiem:

```
$ sudo pstree -p
$ sudo cat /proc/XXX/task/XXX/children
$ sudo gcore $PID1 $PID2 $PID3
$ sudo gcore $(cat /proc/XXX/task/XXX/children)
```

Parasti, procesi konteinerī nevar paslēpties no *host* sistēmas, tāpēc *root* lietotājam būtu jāredz visi procesi, kurus ir izpildījis parasts lietotājs. Taču *container escape* uzbrukumu rezultātā arī konteineru lietotājs var izmukt no konteineriem, vai pat kļūt par *root* (*privilege escalation*). Šajā gadījumā sistēmā var darboties procesi, kurus nevarēs redzēt ar parastiem rīkiem (*ps, top, htop*). Šajā gadījumā drošāk ir uzskatīt visu serveri par kompromitētu un jāiegūst pilns operatīvās atmiņas attēls, kā tas ir aprakstīts augstāk šajā rakstā.

Virtualizētās sistēmas

Ja sistēma, kas tiek analizēta, ir virtualizēta, tad visdrošākais veids, kā iegūt tās atmiņas attēlu, ir no virtualizācijas uzturētājsistēmas *(host)*. Gadījumā, ja serveris tiek mitināts mākonī vai to nodrošina ārpakalpojums, var būt nepieciešams vērsties pēc palīdzības šī mākoņa vai ārpakalpojuma atbalsta dienestā.

Operatīvās atmiņas attēla veidošanai virtualizācijas uzturētājsistēmas (*host)* līmenī ir vairāki ieguvumi:

- rezultāts būs tūlītējs un korekts (nav jāuztraucas par *smearing*, jo atmiņas saturs var mainīties fonā strādājošo procesu dēļ)
- ļaunatūra (t.sk. *rootkits*) nevar pamanīt atmiņas nolasīšanu un nevar izvairīties vai paslēpties
- atmiņas iegūšanas process nav atkarīgs no virtualizētās operētājsistēmas un programmatūras versijām
- operatīvās atmiņas attēlu var iegūt, neveicot nekādas izmaiņas strādājošā sistēmā

Turklāt, šādā veidā operatīvo atmiņu var nolasīt arī tad, ja nav tiešas pieejas virtualizētajai mašīnai, piem., nav zināma lietotāja parole (piem., serveris nopirkts vīrusu vai *botnetu C2* uzturēšanai). Turklāt virtuālās mašīnas operatīvā atmiņa var tikt nolasīta arī tad, ja diska saturs ir šifrēts un faila sistēmai tieši piekļūt nav iespējams.

Hyper-V

Hyper-V gadījumā operatīvās atmiņas attēlu ieteicams iegūt, izmantojot *WinDBG* vai *kd* (*kernel debugger*) un *LiveKD* rīku no *Sysinternals*.

Debugger instalēšanai nepieciešams lejupielādēt *Windows 10 SDK* (neskatoties uz nosaukumu, darbojas arī *Windows Server* vidē), instalēšanas solī gan var izņemt ķekšus no visām opcijām, izņemot "*Debugging Tools for Windows*": https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk/

LiveKD rīku iespējams iegūt *Sysinternals* portālā:
http://technet.microsoft.com/sysinternals/bb897415.aspx

Pēc šo rīku instalēšanas var pārbaudīt, vai *Hyper-V* atbalsts darbojas un vai ir redzama nepieciešamā virtuālā mašīna:

```
> livekd64.exe -hvl
```

Ja instalēšanas laikā *debugger* rīki netika pievienoti sistēmas *PATH*, to var izdarīt komandrindas sesijā, piem., *PowerShell* gadījumā:

```
> $env:path = $env:path + 'C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\'
```

Operatīvās atmiņas attēlu var iegūt šādi:

```
> livekd64.exe -y 'srv*c:\symbols\*https://msdl.microsoft.com/download/symbols' -
hv TestVM -p -o TestVM.dmp -vsym
```

Šajā piemērā "-y" opcija izmantota, lai norādītu standarta *Microsoft Symbols* serveri un lokālo direktoriju, kas tiks izmantota kešatmiņas veidošanai; "*-hv*" tiek izmantots, lai norādītu virtuālās mašīnas vārdu, kam jāveido atmiņas attēls; "*-p*" norāda, ka mašīna jāapstādina uz atmiņas attēla veidošanas laiku; "*-o*" norāda failu, kurā tiks saglabāts atmiņas attēls un *"-vsym*" parādīs simbolu lejupielādes progresu.

Ja *Hyper-V* serverī nav iespējams instalēt *debugging* rīkus, var arī vienkārši eksportēt visu virtuālo mašīnu (operatīvo atmiņu un disku), tā, lai analītiķis, kas veiks tālāko analīzi, varētu to importēt un veikt tālākās darbības patstāvīgi:

```
> Get-VM
> Export-VM -Name TestVM -Path TestVM_Exported -CaptureLiveState
CaptureSavedState
```

"-CaptureLiveState CaptureSavedState" norāda uz to, lai eksportētajos failos tiktu iekļauta arī operatīvā atmiņa (kas pēc noklusējuma *Hyper-V* vidē netiek darīts).

VMWware

VMWare gadījumā atmiņas attēla vietā var izmantot iebūvēto Snapshots funkcionalitāti, jo tā satur to pašu informāciju, kas nepieciešama operatīvās atmiņas analīzei. Atmiņas attēla iegūšanai ir nepieciešami faili ar paplašinājumu *.vmss* un *.mem* (abi tiek veidoti vienlaicīgi un ir nepieciešami tie abi, nevis viens no tiem).

Tālākai analīzei var nodot šos abus failus, vai arī var no tiem iegūt *"raw memory dump"*, izmantojot *Volatility 2*:

```
$ volatility -f windows-infected.vmem -O windows-infected.dmp --
profile=Win10x64_19041 raw2dump
```

VirtualBox

VirtualBox virtualizācijas gadījumā atmiņas attēlu var iegūt, izmantojot komandrindu. Virtuālo mašīnu sarakstu var iegūt ar:

```
$ VBoxManage list vms
```

Atrodot vajadzīgās VM vārdu vai identifikatoru, tās operatīvās atmiņas attēlu var iegūt ar:

```
$ VBoxManage debugvm "Test-VM" dumpvmcore --filename=TestVM.dmp
```

Libvirt / Virt-manager (KVM/QEMU)

Libvirt / virt-manager (parasti tiek izmantoti *KVM/QEMU* pārvaldībai) gadījumā operatīvās atmiņas attēlu var iegūt šādi:

```
$ virsh -c qemu:///system dump --memory-only win10 win10.dmp
```

Profilu veidošana strukturētai analīzei

Daļu no nepieciešamās informācijas par datiem, kas atradās atmiņā pierādījumu gūšanas solī, var iegūt uzreiz, vienkārši meklējot baitu ķēdes, kas atbilst zināmiem šabloniem. Lai to iegūtu, var izmantot:

- universālus rīkus kā *grep, strings (GNU* vai *Sysinternals)*
- specializētus rīkus kā *Yara*, *bulk_extractor*, *BelkaCarving*, *scalpel*, *foremost*

Taču, lai veiktu strukturētu analīzi, atgūstot informāciju par procesiem, failiem, utt., ir nepieciešama informācija par operētājsistēmas datu struktūrām un to atrašanās vietām. Tā kā šīs ir iekšējas operētājsistēmas kodola struktūras un nevis publiski interfeisi, tad bieži vien šīs struktūras netiek oficiāli dokumentētas un var tikt izmainītas pēc jebkuriem operētājsistēmas atjauninājumiem, kas ievieš izmaiņas kodolā, *draiveros* vai zema līmeņa bibliotēkās.

Konfigurācijas faili, kas apraksta konkrētās operētājsistēmas versijas datu struktūru atrašanās vietas, tiek saukti par profiliem, un to formāts ir specifisks rīkam, kas tiks izmantots strukturētai analīzei.

Gadījumā, ja tiek lietota vecāka *Windows* versija, tai visticamāk jau būs izveidoti nepieciešamie profili, un vienīgais, kas jādokumentē, ir precīza kodola versija. Taču *Linux* gadījumā profili precīzai versijai var arī neeksistēt, īpaši, ja tiek lietots kāds mazāk populārs *distributīvs* (precīzas *Linux* profilu versijas, kas pieejamas *Volatility 2*, redzamas šeit: https://github.com/volatilityfoundation/profiles/tree/master/Linux).

Gadījumā, ja tiek izmantota jaunākā operētājsistēmas versija (proti, ir instalēti visi pieejamie atjauninājumi), var gadīties, ka tai vēl nebūs izveidots profils. Šajā gadījumā analītiķis varēs replicēt šo sistēmu (uzinstalējot tādu pašu operētājsistēmu un pilnībā to atjaunojot), lai atrastu nepieciešamās datu struktūras un izveidotu profilu, ja tas ir nepieciešams. Taču, šis process var būt laikietilpīgs, kā arī var gadīties, ka līdz laikam, kad notiks atmiņas attēla analīze, pieejami kļūs jauni operētājsistēmas atjauninājumi, kas izmainīs datu struktūras, un profilu izveidot neizdosies.

Šo apsvērumu dēļ pēc atmiņas attēla izveidošanas ieteicams vienmēr pārliecināties, ka iegūtos datus būs iespējams korekti analizēt, vienā no uzskaitītajiem veidiem:

- *Windows* gadījumā: saglabājot nepieciešamo informāciju, kas nepieciešama profila veidošanai

- pārbaudot, ka dotajai operētājsistēmas versijai jau ir izveidots *Volatility 2* profils (*Windows* un *Linux* profili atrodami dažādos *Github* repozitorijos)
- *Windows* gadījumā: pārbaudot, ka iegūtais attēls ir analizējams ar Volatility 3, kam nav nepieciešami operētājsistēmas-specifiski profili
- *Linux* gadījumā: izveidojot dotajai sistēmai specifisku profilu

Virtualizācijas gadījumā kā rezerves variantu var izmantot *Snapshot* vai *Export* funkcionalitāti. Izveidojot un saglabājot sistēmas *Snapshot* (jāpārliecinās, ka tajā tiek iekļauts operatīvās atmiņas stāvoklis), nepieciešamības gadījumā pie tā varēs atgriezties, lai iegūtu vairāk informācijas, piemēram, ja neizdosies izveidot profilu vai būs nepieciešams izveidot operatīvās atmiņas attēlu atkārtoti. Savukārt, *Export* funkcionalitāti var izmantot, lai saglabātu virtuālās mašīnas rezerves kopiju un iespējams pat nosūtīt to tālākai analīzei (atsevišķu operatīvās atmiņas/diska attēlu vietā). Iegūto arhīvu varēs importēt, lai iegūtu *VM* tādā stāvoklī, kādā šī sistēma bija apskates momentā. Atšķirībā no *Snapshot*, *VM* varēs nepieciešamības gadījumā atjaunot jau uz cita servera.

Windows

Parasti *Windows* kodola izmaiņas tiek ieviestas tikai lielajos atjauninājumos, tādēļ vissvarīgākais ir pierādījumu iegūšanas laikā fiksēt *Windows build* versiju. Tālākās darbības veic tikai PĒC atmiņas attēla veiksmīgas iegūšanas.

Jaunākajās *Windows* versijās versijas informāciju var attēlot ar

```
> winver
```

bet drošāk ir apkopot pilnu informāciju par sistēmu ar

```
> systeminfo > systeminfo.txt
```

Izpildītā komanda saglabās sistēmas informāciju teksta failā *systeminfo.txt.*

End-of-life sistēmu gadījumā (tādas kā *Windows 7*) kodola atjauninājumi, kas sakrājušies, var ieviest diezgan būtiskas izmaiņas, lai arī operētājsistēmas versijas "*build"* skaitlis formāli paliek nemainīgs, tādēļ šajā gadījumā vēlams papildus *"build"* versijai noskaidrot arī tekošo *"revision",* ko var iegūt ar *PowerShell* komandu satrpniecību:

```
> [environment]::OSVersion.Version
```

Profilus, kas jau izveidoti *Windows* operētājsistēmas versijām *Volatility 2*, var redzēt šeit: https://github.com/volatilityfoundation/volatility/tree/master/volatility/plugins/overlays/windows. Savukārt, *Volatility 3* atpazīst *Windows* datu struktūras automātiski caur publisko *Microsoft Symbol* serveri.

Praksē *Windows 10* un vecāku operētājsistēmu atbalsts ir labs, taču retāk izmantoto *Windows Server* versiju, kā arī *LTSC* un *Windows Insider* versijas dažreiz netiek automātiski atpazītas, un tām jāveido profili manuāli.

Par laimi, *Windows* gadījumā galvenās struktūras var iegūt, analizējot vienu failu (kas satur *Windows* kodolu):

```
c:\windows\system32\ntoskrnl.exe
```

Tādēļ pierādījumu iegūšanas stadijā vēlams saglabāt šo failu no mērķa sistēmas, gadījumam, ja tas būs nepieciešams vēlāk (ja tiek veidota arī diska kopija, tad šis fails atsevišķi nav jāglabā, jo nepieciešamības gadījumā to varēs atgūt no failu sistēmas).

Vairāk informācijas par *Windows* profilu veidošanu:
https://www.osdfcon.org/presentations/2020/Jamie-Levy_Troubleshooting-Memory.pdf

Linux profili

GNU/Linux gadījumā profili ir galvenokārt atkarīgi no kodola versijas un kompilācijas opcijām. Publiski pieejamie profili *Volatility 2* ir redzami šeit: https://github.com/volatilityfoundation/profiles/tree/master/Linux. *Volatility 3 Linux* operētājsistēmas atbalsts šobrīd aprobežojas ar *debug* kodoliem, kas tipiski netiek izmantoti produkcijas vidē, tādēļ digitālajā kriminalistikā (*forensics*) šīs sistēmas joprojām tiek analizētas tieši ar *Volatility 2* rīku.

Linux distributīva un izmantoto kodolu versijas var noskaidrot ar komandām:

```
$ cat /etc/os-release
$ uname -a
```

Ja sistēma ir tikusi laicīgi atjaunināta un tiek izmantotas jaunākās programmatūras (un kodola) versijas no standarta repozitorijiem, tad sistēmai specifisko profilu var arī neveidot, jo analītiķis varēs to izveidot patstāvīgi, instalējot virtuālajā mašīnā tieši tādu pašu *Linux distributīva* versiju. Taču bieži vien profili tomēr ir jāveido šo iemeslu dēļ:

- servera programmatūra tika atjaunināta daļēji vai nepilnīgi
- tiek izmantots vecs, trešo pušu veidots vai patstāvīgi kompilēts *Linux* kodols
- tiek izmantots rets vai maksas *distributīvs*, ko būs grūti atkārtot analīzes stadijā

Tā kā mūsdienās vairumā gadījumu *Linux* serveri darbojas virtualizētā vidē, tad no profila veidošanas iespējams izvairīties, eksportējot virtuālo mašīnu (vai palūdzot to eksportēt sava mitināšanas pakalpojuma sniedzējam) un nododot analīzei visu eksportēto arhīvu. Šajā gadījumā analītiķis varēs importēt visu virtuālo mašīnu savā darbstacijā un izveidot profilu (ja tas ir nepieciešams) patstāvīgi. Taču eksporta/importa funkcionalitāte dažādos virtualizācijas / mākoņrisinājumos atšķiras, tādēļ jānoskaidro, kas ir iekļauts eksportētajā arhīvā un vai ar šiem datiem pietiks, lai pilnībā atkārtotu sistēmas iestatījumus.

Linux profila veidošanai sākumā jāinstalē *Linux headers* (informācija, kas nepieciešama kodola moduļu izstrādei) tekošai kodola versijai, kā arī *gcc, make* un *dwarfdump* rīki.

Konkrētās komandas šīs programmatūras instalēšanai ir atkarīgas no *Linux distributīva* un tā versijas, tāpēc komandrindas var būt nepieciešams pielabot:

```
# Debian/Ubuntu
$ sudo apt install dwarfdump build-essential pcregrep libpcre++-dev python-dev
python-pip linux-headers-$(uname -r)
# Red Hat/Centos/Fedora
$ sudo yum –enablerepo=epel install libdwarf-tools elfutils-libelf-devel kernel-devel
gcc
# OpenSUSE
$ sudo zypper install -t pattern devel_basis libdwarf-tools
```

Tālāk jāuzinstalē *Volatility 2*:

```
# opcionāli var instalēt Python moduļus iekš virtualenv,
# lai nepiesārņotu sistēmu:
#   virtualenv -p python2 venv && . venv/bin/activate
$ pip install pycrypto distorm3 openpyxl ujson
$ git clone https://github.com/volatilityfoundation/volatility
$ cd volatility/tools/linux
$ make -C /lib/modules/$(uname -r)/build CONFIG_DEBUG_INFO=y M=$PWD
modules
dwarfdump -di ./module.o > module.dwarf
$ zip Profila-nosaukums.zip module.dwarf /boot/System.map-$(uname -r)
```

Vairāk informācijas par Linux profilu veidošanu:
https://github.com/volatilityfoundation/volatility/wiki/Linux#making-the-profile