Technologic Papers

Memory Analysis על קצה המזלג

מאת יניב מרקס ואפיק קסטיאל (cp77fk4r)

הקדמה

במאמרים הקודמים שפורסמו באתר, נותחו שיטות שונות של החדרת malwares למערכת ההפעלה והסתרתם מתוכנות האבטחה הקיימות (user/kernel mode rootkit). מאחר ומערכי ההגנה הסטנדרטיים מתבססים רבות על חתימות ומאחר וההתקפות מבוצעות תוך שימוש בשיטות חדשות וקשות לגילוי, ישנו פער ההולך וגדל של אותם מערכי הגנה.

אחת השיטות הקיימות היום בזיהוי תוכנה זדונית הינה ניתוח זיכרון חי (live memory analysis) של המחשב אותו אנו מעוניינים לבדוק. אנו נרצה לבצע בדיקה זו, למשל, כתוצאה מחשד שעולה לגבי מחשב מסוים או כחלק מתהליך בדיקה שגרתי של מחשבים ברשת ארגונית.

ישנם בשוק כלים מסחריים ותוכנות open-source המאפשרים לבצע בדיקה זו אך אנו במאמר זה נשתמש בתוכנת ה-volatility - open source (code.google.com/p/volatility/) על מנת להסביר ולהדגים מה ניתן לנתח וללמוד מהנתונים שנשלפו מהזיכרון.

שלב א' - יצירת Memory Dump

על מנת שניתן יהיה לנתח אתהזיכרון, מומלץ לבצע שמירה של הזיכרון על ממת שניתן יהיה לבצע את הניתוח במחשב אחר ללא חשש שהתוכנה הזדונית תופעל גם על המחשב הבודק. לצורך כך, קיימות מגוון תוכנות המאפשרות לנו לבצע זאת, כדוגמת mdd_1.3.exe, ניתן להוריד אותה מהקישור הבא:

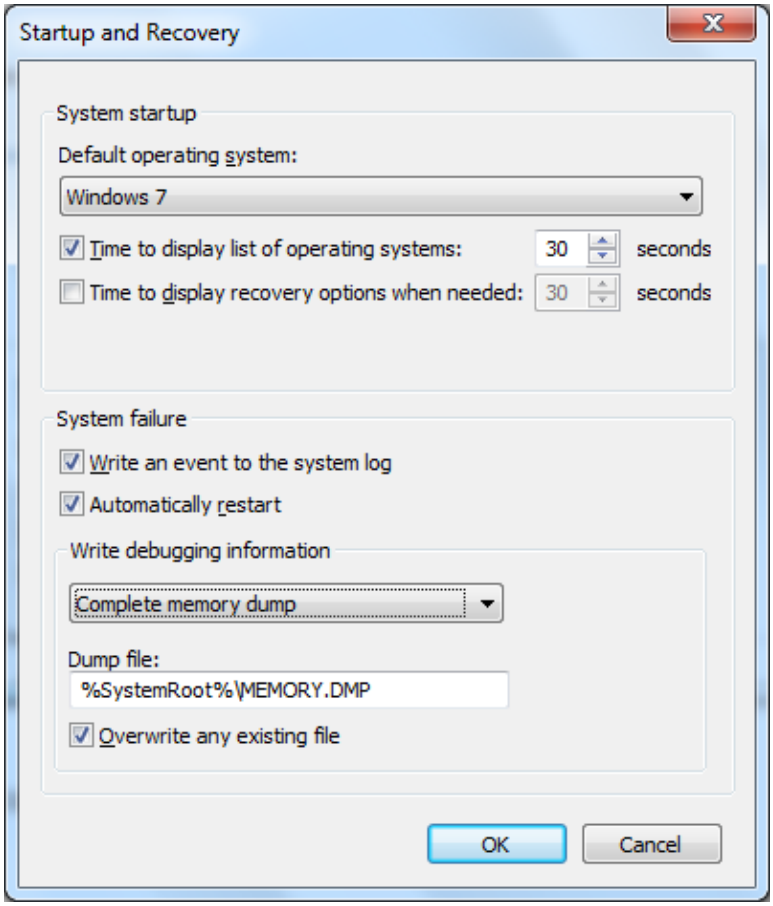
http://sourceforge.net/projects/mdd/files/mdd/mdd-1.3/mdd_1.3.exe/download

או להשתמש בפיצ'רים של מערכת ההפעלה.

על מנת להשיג "Full Memory Dump" של מערכת ההפעלה ברגע נתון, יש ראשית כל לאפשר את האופציה של יצירת Full Memory Dump בעת קריסת המערכת. נבצע זאת בעזרת השלבים הבאים:

- יש להכנס ל-Control Panel ושם להכנס ל-System.
- יש לבחור ב-Advanced ושם ל-Settings ואז ל-Startup and Recovery.

בשלב זה, נגיע לחלון הבא:



תלת "Write Debugging Information" יש לבחור ב-"Complete memory dump".

לאחר מכן נרצה את האפשרות של ליזום קריסה של המערכת על ידי טריגר מהמקלדת - בכל זאת שנרצה, נאפשר זאת בעזרת הצעדים הבאים:

- אם מדובר במקלדת המחוברת באמצעות PS/2, יש להכנס ל-Registry, תחת המפתח הבא:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\i8042prt\Parameters
```

ולהוסיף שם מפתח מסוג REG_DWORD, בשם CrashOnCtrlScroll עם הערך: 0x01.

- במידה ואנו משתמשים במקלדת המחוברת באמצעות USB, יש לבצע את אותו הדבר תחת הערך הבא:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\kbdhid\Parameters
```

לאחר שנבצע Reset למערכת ההפעלה, נוכל ליזום את קריסת המערכת באופן יזום ע"י לחיצת המקשת Right Ctrl ובאותו הזמן לחיצה על המקש SCROLL-LOCK פעמיים. הפעלת הטריגר הנ"ל תשלח למערכת קריאת KeBugCheck עם הערך 0xE2 ("MANUALLY_INITIATED_CRASH") שתגרום לקריסת המערכת ויצירת קובץ ה-Dump.

במידה ותרצו, קיימת אפשרות לשנות את הכפתורים האחראים על יצירת הטריגר, על מנת לעשות זאת, יש להכנס ב-Registry למפתח הבא:

- לבעלי מקלדות PS/2:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\crashdump
```

- ולבעלי מקלדות USB:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\kbdhid\crashdump
```

וליצור מפתח מסוג REG_DWORD בשם Dump1Keys עם אחד מהערכים הבאים:

Value	First key used in the keyboard shortcut sequence
0x01	Rightmost SHIFT key
0x02	Rightmost CTRL key
0x04	Rightmost ALT key
0x10	Leftmost SHIFT key
0x20	Leftmost CTRL key
0x40	Leftmost ALT key

[במקור: http://msdn.microsoft.com/en-us/library/ff545499.aspx]

שינוי לאחד מהערכים הנ"ל יחליף את כפתור ב-Ctrl, אך ישאיר את כפתור ה-SCROLL-LOCK, ניתן להחליף גם אותו, אך על זה תוכלו לקרוא בקישור הבא:

http://msdn.microsoft.com/en-us/library/ff545499.aspx

מעכשיו, נוכל לגרום לקריסה של המערכת בכל רגע שנרצה, לאחר עליית המערכת מחדש, יופיע קובץ Full Dump במיקום:

```
%systemRoot%\MEMORY.DMP
```

לפעמים נרצה לחקור תוכנה מסוימת החשודה כתוכנה זדונית ע"ג VM (VirtualMachine) ואז נרצה לחקור את הזיכרון של המכונה הוירטואלית.

ישנן כל מיני דרכים ליצור memory image גם עבור VM, השיטה שהוצגה בשורות האחרונות תעבוד גם במקרה של VM. ניתן לקרוא על כך גם באתר הבא:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1001624

בנוסף, במאמר זה נשתמש בכלי Volatility שתומך גם בפורטמים vmss (קובץ ה-"suspended state" של המכונה הוירטואלית) ו-vmx (קובץ הזיכרון של המכונה הוירטואלית), כך שלקיחת Snapshot יכולה להספיק בהחלט.

שלב ב' - ניתוח המידע

לצורך משימה זו קיימות מספר תוכנות, אך במאמר זה נשתמש בתוכנה volatility. לפני שניגש לאופן השימוש בה, נספר עליה מעט.

Volatility הינה אחת התוכנות החופשיות החזקות כיום המאפשרות לנו לבצע Memory analysis באופן פשוט יחסית. תוכנה זו נכתבה ב-Python ולכן נדרש גם להתקין Python interpreter על המחשב, או לחילופין ניתן להוריד מהאתר גם גרסת standalone המאפשרת לנו לנתח את הזיכרון ללא צורך בהתקנת Python interpreter בכלל.

העובדה כי התוכנה נכתבה ב-Python מקנה לה גמישות רבה ונתון זה מתבטא בכך כי חוקרי זכרון רבים (וביחוד חוקרי וירוסים, סוסים טרויאנים, תולעים ושאר נוזקות) כותבים "פלאגינים" לתוכנה. ומאותו הצד - החברה משתפת פעולה עם הקהילה ומוסיפה פלאגינים אלו ל-"Main Core" של המערכת.

התוכנה נכתבה ע"י החברה Volatile Systems LLC, ומופצת תחת הרישיון GNU. נכון לכתיבת שורות אלו, התוכנה תומכת במערכת הפעלה הבאות:

- 32-bit Windows XP Service Pack 2 and 3
- 32-bit Windows 2003 Server Service Pack 0, 1, 2
- 32-bit Windows Vista Service Pack 0, 1, 2
- 32-bit Windows 2008 Server Service Pack 1, 2
- 32-bit Windows 7 Service Pack 0, 1
- 64-bit Windows XP Service Pack 1 and 2
- 64-bit Windows 2003 Server Service Pack 1 and 2
- 64-bit Windows Vista Service Pack 0, 1, 2
- 64-bit Windows 2008 Server Service Pack 1 and 2
- 64-bit Windows 2008 R2 Server Service Pack 0 and 1
- 64-bit Windows 7 Service Pack 0 and 1
- 32-bit Linux kernels 2.6.11 to 3.5
- 64-bit Linux kernels 2.6.11 to 3.5

החברה מקיימת סדנאות וקורסים רבים בנושא, הן באופן וירטואלי והן בכנסים (כגון DefCon / BlackHat וכו'). קיימים לא מעט בלוגים עם מידע רב אודות התוכנה, אך הבלוג הרשמי של החברה הינו:

http://volatility.tumblr.com

הגרסא האחרונה של Volatility הינה 2.2, היא יצאה באוקטובר 2012, וניתן להוריד אותה בחינם מהקישור הבא:

https://code.google.com/p/volatility/wiki/Release22

התמיכה בקרנל של לינוקס (מ-2.6.11 עד 3.5) קיימת רק הגרסא האחרונה.

הפלאגינים העיקריים של Volatility מתחלקים לנושאים הבאים:

- **Image Identification** - פלאגינים שתפקידם לאתר את גרסאת המערכת ממנה נלקחה דגימת הזכרון.

- **Process and DLLs** - פלאגינים שתפקידם לבצעת את העבודה העיקרית עבור ניתוח תהליכי מערכת וקבצי DLL שנטענו לזיכרון. בין הפלאגינים ניתן למצוא פלאגינים שתפקידם להחזיר את רשימת התהליכים הרצים בזכרון ומידע אודותם, פלאגינים שתפקידם להחזיר את רשימת קבצי ה-DLL שנטענו לתהליך מסויים, פלאגינים שתפקידם לחלץ מידע ממבני נתונים הקיימים בתהליכים ספציפיים במערכת, פלאגינים המחזירים את משתני הסביבה של כל תהליך ותהליך ועוד.

- **Process Memory** - פלאגינים שתפקידם לבצע מיפוי והקלה של העבודה מול זיכריון של תהליך / תהליכים ספציפיים.

- **Kernel Memory and Objects** - משפחה של פלאגינים שתפקידם להקל בעת עבודה מול זיכרון של רכיבים ואובייקטים ברמת ה-Kernel של מערכת ההפעלה.

- **Win32k / GUI Memory** - מדובר במשפחה חדשה של פלאגינים, תפקידה להקל בעת העבודה מול רכיבים הקשורים בעיקר לשולחן העבודה ולממשקי משתמש, ניתן למצוא בה פלאגינים אשר תפקידם לאתר את רשימת ה-Session-ים המחוברים כעת למערכת, רשימת ה-Handles הפתוחים, החזרת ה-Z-Order של החלונות בשולחן העבודה, החזרת רשימת החלונות הפתוחים וכו'.

- **Networking** - קבוצה של פלאגינים שתפקידם לעזור בעת ניתוח אירועי מערכת ההפעלה הקשורים לרשת התקשורת, ניתן למצוא בה פלאגינים המחזירים את ה-Socket-ים הפתוחים, רשימת ה-Connection-ים הפתוחים ומצבם, רשימת ה-TCPObjects ועוד.

- **Registry** - משפחה של פלאגינים שתפקידם להקל בעת העבודה מול רכיב ה-Registry של מערכת ההפעלה, ניתן למצוא פלאגינים המבצעים חיפוש אחר מחרוזות ב-Registry, פלאגינים לשליפת מפתחות שלמים או Hive-ים שלמים, פלאגינים לפענוח מבני נתונים הקיימים ב-Registry ועוד.

- **File Formats -** משפחה של פלאגינים שתפקידה לעזור בעת העבודה עם קובץ הזיכרון, ניתן למצוא בה פלאגינים כדוגמאת **crashinfo** - פלאגינים המחזיר מידע אודות קובץ ה-Dump. **Hibinfo** - פלאגין המחלץ מידע מקובץ ה-hibernation של מערכת ההפעלה ועוד.

- **Malware -** משפחה של פלאגינים לטובת מחקר וירוסים וקבצים זדוניים במערכת, ניתן למצוא בה פלאגינים המחפשים Hook-ים על תהליכים, פלאגינים המחפשים קוד או קבצי DLL המוזרקים לתוך תהליכים, פלאגינים המאתרים תהליכים "בלתי-נראים" ועוד.

- **Miscellaneous -** שונות, פלאגינים "יעודיים" הקשורים לנושאים שונים.

כאמור, הגרסא האחרונה תומכת גם בביצוע ניתוח זכרון של מערכות ההפעלה המבוססות על הקרנל של Linux (מ-2.6.11 עד 3.5), רב הפלאגינים דומים לפלאגינים הקיימים עבור מערכות ההפעלה מבוססות Windows, והם מחולקים לקטגוריות הבאות:

- **Processes**
- **Process Memory**
- **Networking**
- **Malware/Rootkits**
- **System Information**

לא נרחיב עליהם במאמר זה.

את רשימת הפלאגינים המלאה בה תומכת Volatility ופירוט אודותם, ניתן למצוא בקישור הבא:
https://code.google.com/p/volatility/wiki/Release22

באופן כללי, השיטה להפעלת volatility הינה (כאשר עובדים עם גרסת ה-standalone):

```
Volatility.exe -f mem.dump command
```

ועבור גרסת ה-Python:

```
python vol.py -f mem.dump command
```

- Mem.dump - הינו קובץ שנוצר בשלב א'.
- Command - הנה הבדיקה אותה אנו מעוניינים לבצע.

לדוגמא, סוגי הבדיקות / פלאגינים הקיימים במערכת (ישנן לא מעט פקודות שהתוכנה מאפשרת אך אנו נסקור רק חלק מהן).

דוגמאות

Imageinfo - חילוץ מידע אודות קובץ הזכרון שברשותנו.

כאשר אנו עובדים עם זכרון של מערכת ההפעלה - עלינו לדעת מול איזה מערכת הפעלה אנו עובדים, במידה ואנחנו לקחנו את ה-Dump אין שום סיבה שלא נדע את גרסאת מערכת ההפעלה. אך לא תמיד אנו לקחנו אותו, ולא פעם נוצר מצב שיש ברשותנו קובץ Dump שאין לנו מושג אודותיו.

Volatility מציעה לנו פתרון המאפשר לנו לדעת מאיזו מערכת הפעלה נלקח ה-Dump ע"י ניתוח וחיפוש אחר מידע הקיים ב-Header של הקובץ וחיפוש אחר מבנים מסויימים הקיימים בקובץ Dump הנוצר במערכות הפעלה ספציפיות וע"י כך לאתר את גרסאתה.

הפקודה:

```
python vol.py -f mem.dump Imageinfo
```

איתור גרסאת מערכת ההפעלה חשובה בעת עבודה עם Volatility, מפני שיש אפשרות להפעיל פקודה מסויימת על קובץ הזכרון תחת פרופיל של מערכת הפעלה ספציפית ולקבל את הפלט באופן מדוייק יותר.

פלט לדוגמא:

```
python vol.py -f mem.dump imageinfo
Volatile Systems Volatility Framework 2.0
Determining profile based on KDBG search...
             Suggested Profile : Win7SP1x86, Win7SP0x86
                    AS Layer1 : JKIA32PagedMemory (Kernel AS)
                    AS Layer2 : FileAddressSpace (/Users/M/Desktop/win7.dmp)
                     PAE type : No PAE
                          DTB : 0x185000
                         KDBG : 0x8296cbe8
                         KPCR : 0x8296dc00
            KUSER_SHARED_DATA : 0xffdf0000
          Image date and time : 2010-07-06 22:40:28
     Image local date and time : 2010-07-06 22:40:28
          Number of Processors : 2
                    Image Type :
```

[במקור: https://code.google.com/p/volatility/wiki/CommandReference#imageinfo]

כחלק מהרצת הפקודה, חוזר הפלט "Suggested Profile", הפלט הנ"ל ממליץ לנו תחת איזה פרופיל יש להריץ את ניתוח הזכרון מערכת. כעת, לכל פקודת Volatility שנריץ את הקובץ הנ"ל נוסיף את המתג:

```
--profile= Win7SP1x86
```

או:

```
--profile= Win7SP0x86
```

על מנת להורות Volatility בעת ניתוח הזיכרון תחת איזה מערכת הפעלה נלקח ה-Dump ובכך להקל עליה.

לאחר שהבנו תחת איזו מערכת הפעלה נלקח ה-Dump, נוכל להתחיל לעבוד עליו. נוכל להריץ לדוגמא:

Connscan - בדיקת רשימת חיבורי רשת (מקביל ל-Netstat)

הפקודה:

```
python vol.py -f mem.dump --profile=Win7SP0x86 connscan
```

תוצאה לדוגמא:

```
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x02214988 172.16.176.143:1054       193.104.41.75:80          856
0x06015ab0 0.0.0.0:1056              193.104.41.75:80          856
```

[מקור: http://behindthefirewalls.blogspot.co.il/2013/07/zeus-trojan-memory-forensics-with.html]

על ידי הפעלת הפקודה הבאה, ניתן לראות אילו תהליכים יצרו תקשורת IP ומהן כתובות היעד, המקור ומספרי הפורט. בדוגמא הנ"ל, ניתן לראות כי תהליך שמספרו 856 ניסה לפנות לכתובת IP - 193.104.41.75 בפורט 80 (HTTP), ניתן גם לראות את מיקום התהליך בזיכרון.

Pstree - רשימת התהליכים (מקביל ל-Tasklist).

הפקודה:

```
python vol.py --profile=Win7SP0x86 -f mem.dump pstree
```

תחזיר תוצאה כגון:

```
Volatile Systems Volatility Framework 2.0
Name                                 Pid    PPid   Thds   Hnds   Time
 0x84E6E3D8:wininit.exe              384    340      3     73 2010-07-06 22:28:53
. 0x8D4CC030:services.exe            492    384     12    216 2010-07-06 22:28:54
.. 0x84E19030:svchost.exe           1920    492      8    115 2010-07-06 22:33:17
.. 0x8D4E5BB0:schtasks.exe          2512    492      2     60 2010-07-06 22:39:09
.. 0x8D7E9030:wsqmcons.exe          2576    492      1      3 2010-07-06 22:39:11
.. 0x8D5B18A8:dllhost.exe           1944    492     16    187 2010-07-06 22:31:21
.. 0x8D7EE030:taskhost.exe          1156    492     10    155 2010-07-06 22:37:54
.. 0x84D79D40:msdtc.exe              284    492     15    152 2010-07-06 22:31:24
.. 0x8D6781D8:svchost.exe           1056    492     16    589 2010-07-06 22:29:31
.. 0x8D777D40:taskhost.exe          2520    492     11    224 2010-07-06 22:39:10
.. 0x8D759470:sdclt.exe             2504    492      1      4 2010-07-06 22:39:09
.. 0x8D5574D8:rundll32.exe          2484    492      1      5 2010-07-06 22:39:08
.. 0x84D82C08:SearchIndexer.        1464    492     18    624 2010-07-06 22:33:20
... 0x8D759760:SearchFilterHo       1724   1464      6     82 2010-07-06 22:37:36
... 0x8D55E678:SearchProtocol       2680   1464      8    231 2010-07-06 22:39:27
.. 0x8D5CC030:svchost.exe           1140    492     17    375 2010-07-06 22:29:51
...
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference#pstree]

על ידי הפעלת הפקודה הבאה, ניתן לראות רשימת תהליכים הרצים במערכת, כולל את מספרי התהליכים שייצרו אותם (PPid), ניתן לראות כי למשל SearchIndexer נוצר על ידי תהליך מספר 492, בחיפוש נוסף בטבלה נוכל לראות כי התהליך 492 הוא services.exe.

Printkey - הצגת ערכי מפתחות ב-Registry:

הפקודה:

```
python vol.py -f mem.dump dump --profile=Win7SP0x86 printkey -K
"Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
```

תחזיר פלט כגון:

```
Volatile Systems Volatility Framework 2.0
Legend: (S) = Stable (V) = Volatile

----------------------------
Registry: \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
Key name: Winlogon (S)
Last updated: 2008-11-26 07:38:23

Subkeys:

Values:
REG_SZ ParseAutoexec : (S) 1
REG_SZ ExcludeProfileDirs : (S) Local Settings;Temporary Internet Files;History;Temp
REG_DWORD BuildNumber : (S) 2600
----------------------------
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\default
Key name: Winlogon (S)
Last updated: 2008-11-26 07:39:40

Subkeys:

Values:
REG_SZ ParseAutoexec : (S) 1
REG_SZ ExcludeProfileDirs : (S) Local Settings;Temporary Internet Files;History;Temp
REG_DWORD BuildNumber : (S) 2600
...
```

[מקור: http://behttps://code.google.com/p/volatility/wiki/CommandReference#printkey]

על ידי הפעלת הפקודה הבאה, ניתן לבדוק מה מוגדר מפתח ספציפי ב-Registry ומה עכשיו.

Svcscan - הצגת רשימת שירותי המערכת (Services) ומצב הריצה שלהם:

הפקודה:

```
python vol.py -f mem.dump dump --profile=Win7SP0x86 svcscan
```

תחזיר תוצאה לדוגמא:

```
Volatile Systems Volatility Framework 2.0
Record   Order Pid Name       DisplayName                    Type                      State          Path

[snip]

0x6ea738 0xf5 1148 WebClient  WebClient                      SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\system32\svchost.exe -k LocalService
0x6ea7c8 0xf6 1028 winmgmt    Windows Management Instrumentation SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6ea858 0xf7 ---- WmdmPmSN   Portable Media Serial Number Service SERVICE_WIN32_SHARE_PROCESS SERVICE_STOPPED --------
0x6ea8e8 0xf8 ---- Wmi        Windows Management Instrumentation... SERVICE_WIN32_SHARE_PROCESS SERVICE_STOPPED --------
0x6ea970 0xf9 ---- WmiApSrv   WMI Performance Adapter        SERVICE_WIN32_OWN_PROCESS   SERVICE_STOPPED --------
0x6eaa00 0xfa ---- WS2IFSL    Windows Socket 2.0 Non-IFS Service... SERVICE_KERNEL_DRIVER       SERVICE_RUNNING \Driver\WS2IFSL
0x6eaa90 0xfb 1028 wscsvc     Security Center                SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6eab20 0xfc 1028 wuauserv   Automatic Updates              SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6eabb0 0xfd 1028 WZCSVC     Wireless Zero Configuration    SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6eac40 0xfe ---- xmlprov    Network Provisioning Service   SERVICE_WIN32_SHARE_PROCESS SERVICE_STOPPED --------
0x6eacd0 0xff ---- lanmandrv  lanmandrv                      SERVICE_KERNEL_DRIVER       SERVICE_RUNNING \Driver\lanmandrv
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference#svcscan]

על ידי הפעלת הפקודה הבאה, ניתן לבדוק אילו שירותי מערכת (Services) מופעלים ומהו נתיב ההפעלה של כל Service.

כאמור, מלבד ניתוח נתונים אודות מערכת ההפעלה, ל-Volatility יש מספר פלאגינים לטובת איתור קודים זדוניים הקיימים במערכת, נציג מספר דוגמאות:

Apihooks - איתור Hook-ים ברמת User/Kernel Mode.

הפקודה:

```
python vol.py -f mem.dump --profile=Win7SP0x86 -p PID apihooks
```

דוגמא לפלט:

```
Name                              Type      Target                              Value
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!GetProcAddress         0x0 0x7ff82360 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!LoadLibraryW           0x0 0x7ff82ac0 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!CreateFileW            0x0 0x7ff82240 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!LoadLibraryA           0x0 0x7ff82a50 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       ADVAPI32.dll!RegSetValueExW         0x0 0x7ff82080 (UNKNOWN)
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference22]

על ידי הפעלת הפקודה הבאה, אנו בודקים האם ישנם תהליכים במערכת אשר בוצע להם apihook, מה שיכול להצביע על תוכנה זדונית במערכת. בדוגמא הנ"ל, ניתן לראות כי ב-IAT (Import Address Table) של התהליך IEXPLORE, בוצעו Hook-ים למספר כתובות של פונקציות.

Malfind - איתור DLL וקוד מוזרק בזכירון של תהליכים ברמת ה-User Mode ע"י חיפוש תווים באיזורים חשודים.

הפקודה:

```
python vol.py -f mem.dump malfind -p PID
```

תוצאה לדוגמא:

```
Volatile Systems Volatility Framework 2.0
Name                 Pid    Start      End        Tag    Hits Protect
explorer.exe           1724 0x01600000 0x01600FFF VadS      0      6
(MM_EXECUTE_READWRITE)
Dumped to: hidden_dumps/explorer.exe.4a065d0.01600000-01600fff.dmp
0x01600000   b8 35 00 00 00 e9 cd d7 30 7b b8 91 00 00 00 e9    .5......0{......
0x01600010   4f df 30 7b 8b ff 55 8b ec e9 ef 17 c1 75 8b ff    O.0{..U......u..
0x01600020   55 8b ec e9 95 76 bc 75 8b ff 55 8b ec e9 be 53    U....v.u..U....S
0x01600030   bd 75 8b ff 55 8b ec e9 d6 18 c1 75 8b ff 55 8b    .u..U......u..U.
0x01600040   ec e9 14 95 bc 75 8b ff 55 8b ec e9 4f 7e bf 75    .....u..U...O~.u
0x01600050   8b ff 55 8b ec e9 0a 32 bd 75 8b ff 55 8b ec e9    ..U....2.u..U...
0x01600060   7d 61 bc 75 6a 2c 68 b8 8d 1c 77 e9 01 8c bc 75    }a.uj,h...w....u
0x01600070   8b ff 55 8b ec e9 c4 95 4b 70 8b ff 55 8b ec e9    ..U.....Kp..U...
```

```
Disassembly:
01600000: b835000000                    MOV EAX, 0x35
01600005: e9cdd7307b                    JMP 0x7c90d7d7
0160000a: b891000000                    MOV EAX, 0x91
0160000f: e94fdf307b                    JMP 0x7c90df63
01600014: 8bff                          MOV EDI, EDI
01600016: 55                            PUSH EBP
01600017: 8bec                          MOV EBP, ESP
01600019: e9ef17c175                    JMP 0x7721180d
0160001e: 8bff                          MOV EDI, EDI
01600020: 55                            PUSH EBP
explorer.exe        1724 0x015D0000 0x015F5FFF VadS      0      6
(MM_EXECUTE_READWRITE)
Dumped to: hidden_dumps/explorer.exe.4a065d0.015d0000-015f5fff.dmp
0x015d0000   4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00    MZ..............
0x015d0010   b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00    ........@.......
0x015d0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x015d0030   00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00    ................
0x015d0040   0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68    ........!..L.!Th
0x015d0050   69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f    is program canno
0x015d0060   74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20    t be run in DOS
0x015d0070   6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00    mode....$.......
```

לא נעבור על הפלט לעומק, אך ניתן לראות כי תחת התהליך Explorer.exe, בכתובת 0x015d0000 (איזור עם הרשאות MM_EXECUTE_READWRITE), קיים קוד בינארי (על פי ה-Header) שככל הנראה הוזרק עליו מתוך קוד זדוני המנסה לבצע פעולות תחת הרשאותיו.

Psxview - איתור תהליכים "בלתי נראים" במערכת:

הפקודה:

```
python vol.py -f mem.dump psxview
```

דוגמא לפלט:

Offset	Name	Pid	pslist	psscan	thrdproc	pspcid	csr_hnds	csr_list
0xff1b8b28	vmtoolsd.exe	1668	1	1	1	1	1	0
0x80ff88d8	svchost.exe	856	1	1	1	1	1	0
0xff1d7da0	spoolsv.exe	1432	1	1	1	1	1	0
0x810b1660	System	4	1	1	1	1	0	0
0x80fbf910	svchost.exe	1028	1	1	1	1	1	0
0xff2ab020	smss.exe	544	1	1	1	1	0	0
0xff3667e8	VMwareTray.exe	432	1	1	1	1	1	0
0xff247020	services.exe	676	1	1	1	1	1	0
0xff217560	svchost.exe	936	1	1	1	1	1	0
0xff143b28	TPAutoConnSvc.e	1968	1	1	1	1	1	0
0x80fdc648	1_doc_RCData_61	1336	0	1	1	1	1	0

[מקור: https://code.google.com/p/volatility/wiki/CommandReference#psxview]

על ידי הפעלת הפקודה, ניתן לבדוק האם ישנם תהליכים מוסתרים במערכת. הבדיקה נעשית על ידי השוואה בין מספר מקורות, כגון: PsActiveProcessHead (מצביע לרשימה מקושרת של התהליכים המוגדרים במערכת), רשימה מקושרת של csrss וכו'. בדוגמא הנ"ל, ניתן לראות כי התהליך

1_doc_RCData_61 קיבל את הערך 0 בעמודה pslist (הרשימה המקושרת הכוללת את רשימת התהליכים במערכת אינה כוללת תהליך זה, כלומר אין הצבעה ל-EPROCESS של התהליך) ולכן התהליך חשוד כתוכנה זדונית.

Userassist - איתור התהליכים האחרונים שרצו במערכת:

הפקודה:

```
python vol.py -f mem.dump userassist
```

דוגמא לפלט:

```
REG_BINARY    %windir%\system32\displayswitch.exe :
Count:           13
Focus Count:     19
Time Focused:    0:05:20.500000
Last updated:    2010-03-09 19:49:20

0000   00 00 00 00 0D 00 00 00 13 00 00 00 60 CC 05 00    ............`...
0010   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0020   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0030   00 00 80 BF 00 00 80 BF FF FF FF FF EC FE 7B 9C    ..............{.
0040   C1 BF CA 01 00 00 00 00                            ........

REG_BINARY    %windir%\system32\calc.exe :
Count:           12
Focus Count:     17
Time Focused:    0:05:40.500000
Last updated:    2010-03-09 19:49:20

0000   00 00 00 00 0C 00 00 00 11 00 00 00 20 30 05 00    ............ 0..
0010   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0020   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0030   00 00 80 BF 00 00 80 BF FF FF FF FF EC FE 7B 9C    ..............{.
0040   C1 BF CA 01 00 00 00 00                            ........
                        ........
```

[במקור: http://gleeda.blogspot.co.il/2011/04/volatility-14-userassist-plugin.html]

על ידי הפעלת הפקודה הבאה, ניתן לראות אילו תהליכים הורצו על ידי המשתמש במחשב, כאשר למעשה המידע הזה מאוכסן ב-registry. בדוגמא הנ"ל, ניתן לראות כי המשתמש הפעיל את התוכנות calc.exe ו-displayswitch.exe. הרחבה מעניינת על הנושא, ניתן לקרוא בבלוג של חוקר אבטחת המידע Didier Stevens, בפוסט:

http://blog.didierstevens.com/programs/userassist/

Bioskbd - קריאת ה-Keyboard buffer של ה-BIOS.

הפקודה:

```
python vol.py -f mem.dump bioskbd
```

על ידי הפעלת הפקודה הבאה, ניתן לראות (במקרים מסויימים) מקשים שהוקלדו ונשמרו באזור ה-BIOS הנמצא בזיכרון. עובד רק כאשר משתמשים ב-BIOS שאכן שומר את המידע ב-Buffer הנ"ל. לפי הבלוג של המפתחת, הפקודה תעבור על הביוסים של HP, Intel ו-Lenovo.

השגת סיסמאות ה-Login של המשתמשים במערכת:

במידה ובידינו קיים Snapshop של VM, או ממש זיכרון של מכונה שעלינו לחקור, נרצה להתחבר אליה על מנת לאמת את ממצאינו, או בכדי לקחת דגימות חיות מהמערכת (לדוגמא - כאשר נרצה לחקור תולעת שמתנהגת באופן מסויים רק כאשר מפעילים טריגר כזה או אחר). לא תמיד נוכל להשיג את הסיסמאות של אותה מערכת. Volatility מאפשרת לנו להגיע מאוד קרוב לכך ובמקרים לא מעטים אף להכניס אותנו פנימה.

יש בידינו Snapshop של מכונה וירטואלית במצב Lock, ראשית - נרצה לבדוק את סוג מערכת ההפעלה (את זאת אגב, נוכל לעשות בקלות ע"י ניסיון הפעלת ה-Snapshop בעזרת VMware ובדיקה כיצד נראה מסך ה-Logon...). כמו שראינו בתחילת המאמר, בעזרת Volatility נוכל לעשות זאת בעזרת הפקודה Imageinfo. לאחר מכן, נשלוף את רשימת ה-Hive-ים הקיימים ב-Registry במערכת:

```
Python vol.py -f mem.dump --profile=OS-Version hivelist
```

פלט לדוגמא, יראה כך:

```
Volatile Systems Volatility Framework 2.1_alpha
Virtual            Physical           Name
------------------ ------------------ ----
0xfffff8a001053010 0x000000000b1a9010 \??\C:\System Volume Information\Syscache.hve
0xfffff8a0016a7420 0x0000000012329420 \REGISTRY\MACHINE\SAM
0xfffff8a0017462a0 0x00000000101822a0
\??\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT
0xfffff8a001abe420 0x000000000eae0420
\??\C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT
0xfffff8a002ccf010 0x0000000014659010
\??\C:\Users\testing\AppData\Local\Microsoft\Windows\UsrClass.dat
0xfffff80002b53b10 0x000000000a441b10 [no name]
0xfffff8a00000d010 0x000000000ddc6010 [no name]
0xfffff8a000022010 0x000000000da51010 \REGISTRY\MACHINE\SYSTEM
0xfffff8a00005c010 0x000000000dacd010 \REGISTRY\MACHINE\HARDWARE
0xfffff8a00021d010 0x000000000cd20010 \SystemRoot\System32\Config\SECURITY
0xfffff8a0009f1010 0x000000000aa1a010 \Device\HarddiskVolume1\Boot\BCD
0xfffff8a000a15010 0x000000000acf9010 \SystemRoot\System32\Config\SOFTWARE
0xfffff8a000ce5010 0x0000000008c95010 \SystemRoot\System32\Config\DEFAULT
0xfffff8a000f95010 0x000000000c2b4010 \??\C:\Users\testing\ntuser.dat
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference22#hivelist]

המטרה שלנו היא להשיג את המיקום בזיכרון של ה-Hive-ים של ה-SYSTEM ושל ה-SAM. במקרה שלנו:

ה-Hive של ה-SYSTEM נמצא בכתובת:

```
0xfffff8a000022010
```

וה-Hive של ה-SAM נמצא:

```
0xfffff8a0016a7420
```

לא נרחיב על זה יותר מדי, אך למי שלא מכיר, ה-Registry במערכת מחולק ל-Hive-ים שונים, כל Hive תפקידו לשמור מידע מסוג שונה. יש Hive לכל משתמש, ויש Hive כללי לכלל המשתמשים. ב-SAM (קיצור של Security Accounts Manager) מערכת ההפעלה שומרת בין היתר HASH (מסוג LM ו-NTLM) של סיסמאת המשתמשים.

לאחר שהשגנו את הפרטים הנ"ל, נריץ את הפקודה Hashdump באופן הבא:

```
python vol.py hashdump -f image.dd -y [SYSTEM Hive Addr] -s [SAM Hive
Addr]
```

פלט לדוגמא:

```
Administrator:500:08f3a52bdd35f179c81667e9d738c5d9:ed88cccbc08d1c18bcded
317112555f4:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
HelpAssistant:1000:ddd4c9c883a8ecb2078f88d729ba2e67:e78d693bc40f92a53419
7dc1d3a6d34f:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:8bfd47482583168a0
ae5ab020e1186a9:::
ASPNET:1004:2b5f618079400df84f9346ce3e830467:aef73a8bb65a0f01d9470fadc55
a411c:::
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference23#hashdump]

זה לא נושא המאמר, ולכן לא נרחיב על כך, אך ניתן לראות כי המבנה של הפלט הינו שם המשתמש, לאחר מכן הגדרות הקבוצה אליה הוא שייך, ואז שתי מחרוזות בנות 32bit כל אחת. הראשונה הינה סיסמאת המשתמש שמורה כ-LM והשניה היא אותה הסיסמה שמורה כ-NTLM. קיימים כיום כלים רבים ולא מעט טכניקות המאפשרות את שבירת ה-Hash-ים הנ"ל בעזרת Rainbow Tables וכו'.

סיכום

ניתוח זיכרון חי (live memory analysis) מאפשר לקבל מידע רב על הנעשה במחשב שלנו ועל ידי כך לזהות האם ישנן תוכנות זדוניות שמערכות ההגנה הסטנדרטיות לא זיהו.

במאמר זה, נעשה שימוש ב-Volatility (https://code.google.com/p/volatility) על מנת להדגים מה ניתן לחקור תוך שימוש במידע הקיים בזיכרון. יש לציין כי ישנן פקודות רבות שלא הוזכרו במאמר אך יכולות לעזור רבות בניתוח.

מקורות ומאמרים לקריאה נוספת

- https://code.google.com/p/volatility/wiki/CommandReference23
- http://www.aldeid.com/wiki/Volatility
- http://volatility-labs.blogspot.com
- http://gleeda.blogspot.com
- http://www.behindthefirewalls.com/2013/07/zeus-trojan-memory-forensics-with.html
- https://blogs.sans.org/computer-forensics/files/2012/04/Memory-Forensics-Cheat-Sheet-v1_2.pdf