

## Coreflood - ניתוח זיכרון ומתודולוגיות הדבקה

מאת דניאל קויפמן

### הקדמה

Coreflood היא נוזקה מסוג סוס טרויאני ו-Botnet אשר נוצרה על ידי קבוצה של האקרים מרוסיה ושחררה ב-2010. ה-FBI כללה ברשימת המערכות המודבקות שלה בערך "17 סוכניות ממשלה מקומיות, בנוסף לתחנת משטרה; שלושה נמלי תעופה; שני קבלני ביטחון; חמישה בנקים או מוסדות פיננסיים; כ-30 אוניברסיטאות או מכללות; 20 בתי חולים; ומאות עסקים נוספים." [1] אנחנו ניעזר ב-Volatility כדי לצלול לעומק. Volatility היא אוסף חינומי של כלים שכתובים בפיתוח, עבור חילוץ של שאריות ומזהים דיגיטליים מזיכרון תנודתי (RAM) או מתמונות זיכרון. טכניקות החילוץ מתבצעות באופן נפרד לחלוטין מהמערכת הנחקרת, אך מציעות נראות חסרת תקדים בהקשר למצב ההרצה של המערכת.

למה ניתוח זיכרון? הסיבה לכך שהיא שזיכרון תנודתי הוא בעל חשיבות קריטית בלעזור לנו להבין את המצב של המערכת המודבקת. זה ממש דומה למה שבלשים עושים, רק אם הפושע הוא וירוס. כשזה נוגע למתקפות נוזקה, זיכרון תנודתי לפעמים עשוי להיות המקור היחידי שיש לנו בחקר המתקפה. מגמות אחרונות בתחום מראות לנו שכמות לא מבוטלת של נוזקות שוכנות רק בזיכרון, מה שאומר שניתוח של זיכרון לא-תנודתי לא יתן לנו רמזים על הימצאות של נוזקה מסוימת.

"Malware can hide, but it MUST run" - SANS DFIR

### הכנה

תמונת הזיכרון שנעבוד איתה יכולה להימצא [כאן](#).

אני משתמש בסביבת העבודה-SIFT, שנוצרה על ידי ידידינו האהובים ב-SANS, שהיא למעשה כמו-KALI, אבל עבור התחום של ניתוח זיכרון.

## החקירה עצמה

אז בכדי להתחיל את החקירה, אני ארצה להציג רשימה פשוטה של תהליכים שרצו על המכונה כדי לראות אם אני אוכל לזהות תהליכים חשודים. אוכל לעשות זאת בעזרת הפקודה `pslist`.

Windows משתמשת במבנה נתונים מסוג double linked list של מבני `_EPROCESS` כדי לבצע מעקב אחר כל התהליכים הפעילים. הרשימה הזו שוכנת ב-Kernel. Volatility משתמשת בעובדה הזו ומחפשת אחר המצביע-`PsActiveProcessHeader` אשר מצביע להתחלה של רשימת מבני ה-`_EPROCESS` ב-Kernel. לאחר מכן, Volatility מבצעת אנומרציה על הרשימה הזו כדי להציג את התהליכים הרצים:

```
$ vol.py -f coreflood.vmem pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V) Name PID PPID Thds Hnds Sess Wow64 Start Exit
-----
0x810b1660 System 4 0 58 183 ----- 0
0xff2ab020 smss.exe 544 4 3 21 ----- 0 2010-08-11 06:06:21 UTC+0000
0xff1ecda0 csrss.exe 608 544 10 369 0 0 2010-08-11 06:06:23 UTC+0000
0xff1ec978 winlogon.exe 632 544 20 518 0 0 2010-08-11 06:06:23 UTC+0000
0xff247020 services.exe 676 632 16 269 0 0 2010-08-11 06:06:24 UTC+0000
0xff255020 lsass.exe 688 632 19 344 0 0 2010-08-11 06:06:24 UTC+0000
0xff218230 vmacthlp.exe 844 676 1 24 0 0 2010-08-11 06:06:24 UTC+0000
0x80ff88d8 svchost.exe 856 676 17 199 0 0 2010-08-11 06:06:24 UTC+0000
0xff217560 svchost.exe 936 676 10 272 0 0 2010-08-11 06:06:24 UTC+0000
0x80fbf910 svchost.exe 1028 676 71 1341 0 0 2010-08-11 06:06:24 UTC+0000
0xff22d558 svchost.exe 1088 676 5 80 0 0 2010-08-11 06:06:25 UTC+0000
0xff203b80 svchost.exe 1148 676 14 208 0 0 2010-08-11 06:06:26 UTC+0000
0xff1d7da0 spoolsv.exe 1432 676 13 135 0 0 2010-08-11 06:06:26 UTC+0000
0xff1b8b28 vmtoolsd.exe 1668 676 5 221 0 0 2010-08-11 06:06:35 UTC+0000
0xff1fdc88 VMUpgradeHelper 1788 676 4 100 0 0 2010-08-11 06:06:38 UTC+0000
0xff143b28 TPAutoConnSvc.e 1968 676 5 100 0 0 2010-08-11 06:06:39 UTC+0000
0xff25a7e0 alg.exe 216 676 6 105 0 0 2010-08-11 06:06:39 UTC+0000
0xff364310 wscntfy.exe 888 1028 1 27 0 0 2010-08-11 06:06:49 UTC+0000
0xff38b5f8 TPAutoConnect.e 1084 1968 1 61 0 0 2010-08-11 06:06:52 UTC+0000
0xff3865d0 explorer.exe 1724 1708 12 341 0 0 2010-08-11 06:09:29 UTC+0000
0xff3667e8 VMwareTray.exe 432 1724 1 49 0 0 2010-08-11 06:09:31 UTC+0000
0xff374980 VMwareUser.exe 452 1724 6 189 0 0 2010-08-11 06:09:32 UTC+0000
0x80f94588 wuauclt.exe 468 1028 4 134 0 0 2010-08-11 06:09:37 UTC+0000
0xff3ad1a8 IEXPLORE.EXE 2044 1724 10 366 0 0 2010-08-15 18:11:17 UTC+0000
0x80fdc368 logon.scr 124 632 1 15 0 0 2010-08-15 18:21:28 UTC+0000
0xff125020 cmd.exe 1136 1668 0 ----- 0 2010-08-15 18:24:00 UTC+0000 2010-08-15 18:24:00 UTC+0000
```

כפי שניתן לראות, נראה שרוב התהליכים הם רגילים, אך יש כמה שתופסים את העין. Internet Explorer, רץ, בנוסף ל-`cmd`, והדבר הנכון עבורי לבדוק הוא חיבורים יוצאים. אולי הנוזקה מוחבאת בתוך התהליך של הדפדפן ומשתמשת בעובדה שהדפדפן מבצע חיבורים כל הזמן כדי להחביא תקשורת Command & Control?

אז הפעולה הנכונה היא לבדוק חיבורים יוצאים. אם נראה שאין שום חיבורים יוצאים דרך הדפדפן, נהיה חייבים לחפש במקור אחר. השתמשתי בתוסף `connscan` כדי לבדוק חיבורים פעילים/שהושמדו.

“To find `_TCPT_OBJECT` structures using pool tag scanning, use the `connscan` command. This can find artifacts from previous connection that have since been terminated, in addition to the active ones.” - Volatility Documentation.



התוסף הזה עובד על ידי סריקה של הזיכרון הפיזי וניסיון למצוא את החתימה (TCPT) 0x54455054 וביצוע ניתוח של-28 הסיביות הבאות בתור ה-TCPT\_OBJECT המלא:

```
$ vol.py -f coreflood.vmem connscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P) Local Address Remote Address Pid
-----
0x00eda590 172.16.176.143:1058 65.54.81.209:80 2044
0x01079e70 172.16.176.143:1082 209.234.234.16:80 2044
0x0107c888 172.16.176.143:1059 4.23.40.126:80 2044
0x0108fcd8 172.16.176.143:1072 65.55.15.124:80 2044
0x010fa448 172.16.176.143:1065 65.55.253.21:80 2044
0x02214988 172.16.176.143:1092 65.54.81.14:80 2044
0x026c68a8 172.16.176.143:1074 65.55.15.243:80 2044
0x02ae4bb0 172.16.176.143:1073 65.55.15.123:80 2044
0x048b25f0 172.16.176.143:1085 65.55.149.119:80 2044
0x04a045f8 172.16.176.143:1057 65.54.81.49:80 2044
0x04a04e70 172.16.176.143:1095 69.43.160.145:80 2044
0x04a4a4a0 172.16.176.143:1084 12.120.180.24:80 2044
0x04be2558 172.16.176.143:1079 65.54.81.22:80 2044
0x05536e70 172.16.176.143:1090 65.54.81.14:80 2044
0x05802340 172.16.176.143:1062 65.55.18.18:80 2044
0x05c9e200 172.16.176.143:1067 65.54.81.14:80 2044
0x05deea30 172.16.176.143:1068 65.54.81.14:80 2044
0x06015ab0 172.16.176.143:1053 207.46.170.10:80 2044
0x0605f208 172.16.176.143:1086 202.89.231.60:80 2044
0x06125538 172.16.176.143:1083 65.54.81.79:80 2044
0x0623a438 172.16.176.143:1066 96.6.41.210:80 2044
0x06450720 172.16.176.143:1077 65.55.149.121:80 2044
0x064509f0 172.16.176.143:1063 64.4.18.73:80 2044
0x06497a68 172.16.176.143:1075 65.55.15.124:80 2044
0x067bd218 172.16.176.143:1070 65.54.81.209:80 2044
0x07c17be0 172.16.176.143:1060 65.55.239.161:80 2044
sansforensics@siftworkstation: ~/Downloads
```

זה נראה כאילו המכונה ביצעה תקשורת לגיטימית, מכיוון שכל התקשורת נוצרה על-ידי PID מספר 2044, שהוא IEXPLORE.EXE. כדי לוודא שאין שום תקשורת זדונית, אני אשתמש בתוסף socks, אשר סורק את הזיכרון עבור ADDRESS\_OBJECT.



על ידי סריקה של הזיכרון עבור האובייקט הזה, אנחנו יכולים לקבל מידע על-Sockets פתוחים/שהושמדו:

```
-----
0x01096470      2044    1085      6 TCP           0.0.0.0      2010-08-15 18:11:24 UTC+0000
0x010caa00      2044    1087      6 TCP           0.0.0.0      2010-08-15 18:11:33 UTC+0000
0x010d7e98      2044    1053      6 TCP           0.0.0.0      2010-08-15 18:11:19 UTC+0000
0x010f7400      2044    1066      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x0110d3a8      2044    1054      6 TCP           0.0.0.0      2010-08-15 18:11:19 UTC+0000
0x0111d928      2044    1068      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x01120c40         4      445      17 UDP          0.0.0.0      2010-08-11 06:06:17 UTC+0000
0x01131930     1088    1025      17 UDP          0.0.0.0      2010-08-11 06:06:38 UTC+0000
0x01134008         4         0      47 GRE          0.0.0.0      2010-08-11 06:08:00 UTC+0000
0x01134e98      2044    1074      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x0115f128      936     135      6 TCP           0.0.0.0      2010-08-11 06:06:24 UTC+0000
0x029d9580     1148    1900     17 UDP          127.0.0.1    2010-08-15 18:24:00 UTC+0000
0x02daad28      216    1026      6 TCP           127.0.0.1    2010-08-11 06:06:39 UTC+0000
0x037c0768      2044    1077      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x043d4ac8      2044    1055      6 TCP           0.0.0.0      2010-08-15 18:11:19 UTC+0000
0x044197b0         4     138     17 UDP          172.16.176.143 2010-08-15 18:09:23 UTC+0000
0x048617a0     1088    1061     17 UDP          0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x04861ab8      2044    1081      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x04864a08      2044    1059      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x04866478      2044    1060      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x0486ee98         4     139      6 TCP           172.16.176.143 2010-08-15 18:09:23 UTC+0000
0x04a074d8         4     137     17 UDP          172.16.176.143 2010-08-15 18:09:23 UTC+0000
0x04a4be98         4    1033      6 TCP           0.0.0.0      2010-08-11 06:08:00 UTC+0000
0x04a96a78      2044    1075      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x04b591a8      2044    1082      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x04b59998      2044    1070      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x04b5e880      2044    1057      6 TCP           0.0.0.0      2010-08-15 18:11:20 UTC+0000
0x04be3c08      2044    1056      6 TCP           0.0.0.0      2010-08-15 18:11:20 UTC+0000
0x04be7008         4     445      6 TCP           0.0.0.0      2010-08-11 06:06:17 UTC+0000
0x04c2d698      2044    1063      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x0573f710      2044    1069      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x058844a0     1088    1078     17 UDP          0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x05b95668      2044    1073      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x05c162c8      2044    1058      6 TCP           0.0.0.0      2010-08-15 18:11:20 UTC+0000
0x05ca0cd8      2044    1089      6 TCP           0.0.0.0      2010-08-15 18:11:33 UTC+0000
0x05ce53d0      2044    1079      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x05dee200     1148    1900     17 UDP          172.16.176.143 2010-08-15 18:09:24 UTC+0000
0x05e344c0      2044    1080      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x05f44008      688     500     17 UDP          0.0.0.0      2010-08-11 06:06:35 UTC+0000
0x05f48008     1028    123     17 UDP          127.0.0.1    2010-08-15 18:24:00 UTC+0000
0x0605f008      2044    1086      6 TCP           0.0.0.0      2010-08-15 18:11:24 UTC+0000
0x061253c8      2044    1084      6 TCP           0.0.0.0      2010-08-15 18:11:24 UTC+0000
0x06237b70      688         0    255 Reserved  0.0.0.0      2010-08-11 06:06:35 UTC+0000
0x066f1498      2044    1083      6 TCP           0.0.0.0      2010-08-15 18:11:23 UTC+0000
0x069012b8      2044    1067      6 TCP           0.0.0.0      2010-08-15 18:11:21 UTC+0000
0x069d5250      688    4500     17 UDP          0.0.0.0      2010-08-11 06:06:35 UTC+0000
0x06b1ac00      2044    1052     17 UDP          127.0.0.1    2010-08-15 18:11:19 UTC+0000
sansforensics@siftworkstation: ~/Downloads
$
```

כפי שאנחנו יכולים לראות, יש מספר חיבורים מוזרים שניגשים ל-PID מספר 2044, או הדפדפן שלנו. הגיע הזמן לשלוף את האקדחים הגדולים, כי זה נראה שמצאנו נוזקה שמתחבאת בתהליך הזה.

הפקודה הראשונה שאשתמש בה תהיה-malfind. הפקודה הזאת מוצאת קוד מוזרק בתוך זיכרון של תהליך. היא עושה זאת על ידי חיפוש של אזורי זיכרון מוקצה (על ידי בדיקה של ה-VAD), ובודקת אם יש להן מזהים של קוד הרצה שלא ממופה לשום קובץ על הדיסק.



השיטה שבה איזור זיכרון נבדק בניסיון למצוא קוד מוזרק היא על ידי בדיקה של ה-VAD עבור הרשאות-page, כמו execute. אם לחלק בזיכרון יש הרשאות הרצה, והוא לא ממופה לשום קובץ הרצה בדיסק, זאת אזהרה ברורה עבור מה שעשוי להיות הזרקת קוד:

```

Process: IEXPLORE.EXE Pid: 2044 Address: 0x7ff80000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 45, PrivateMemory: 1, Protection: 6

0x000000007ff80000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x000000007ff80010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x000000007ff80020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0x000000007ff80030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

0x000000007ff80000 0000          ADD [EAX], AL
0x000000007ff80002 0000          ADD [EAX], AL
0x000000007ff80004 0000          ADD [EAX], AL
0x000000007ff80006 0000          ADD [EAX], AL
0x000000007ff80008 0000          ADD [EAX], AL
0x000000007ff8000a 0000          ADD [EAX], AL
0x000000007ff8000c 0000          ADD [EAX], AL
0x000000007ff8000e 0000          ADD [EAX], AL
0x000000007ff80010 0000          ADD [EAX], AL
0x000000007ff80012 0000          ADD [EAX], AL
0x000000007ff80014 0000          ADD [EAX], AL
0x000000007ff80016 0000          ADD [EAX], AL
0x000000007ff80018 0000          ADD [EAX], AL
0x000000007ff8001a 0000          ADD [EAX], AL
0x000000007ff8001c 0000          ADD [EAX], AL
0x000000007ff8001e 0000          ADD [EAX], AL
0x000000007ff80020 0000          ADD [EAX], AL
0x000000007ff80022 0000          ADD [EAX], AL
0x000000007ff80024 0000          ADD [EAX], AL
0x000000007ff80026 0000          ADD [EAX], AL
0x000000007ff80028 0000          ADD [EAX], AL
0x000000007ff8002a 0000          ADD [EAX], AL
0x000000007ff8002c 0000          ADD [EAX], AL
0x000000007ff8002e 0000          ADD [EAX], AL
0x000000007ff80030 0000          ADD [EAX], AL
0x000000007ff80032 0000          ADD [EAX], AL
0x000000007ff80034 0000          ADD [EAX], AL
0x000000007ff80036 0000          ADD [EAX], AL
0x000000007ff80038 0000          ADD [EAX], AL
0x000000007ff8003a 0000          ADD [EAX], AL
0x000000007ff8003c 0000          ADD [EAX], AL
0x000000007ff8003e 0000          ADD [EAX], AL

```

כפי שאנחנו יכולים לראות, אנחנו לא מקבלים יותר מדי מידע שימושי מהפקודה. יכול להיות שהנוזקה לא טענה במפורש קובץ הרצה מסוים לזיכרון, אלא רק הזריקה shellcode. יכול להיות גם שהנוזקה מחקה את תחילית ההרצה מהזיכרון כדי להימנע מזיהוי על ידי קריאה לפונקציה [VirtualFree](http://VirtualFree) על ה-ImageBase של ה-DLL המוזרק. בכל מקרה, אנחנו לא יכולים לדעת בוודאות כי אין פה למעשה ראיות חזקות לפעילות זדונית.



הפקודה הבאה שאני אשתמש בה היא: **apihooks**:

“This finds IAT, EAT, inline style hooks, and several special types of hooks. For Inline hooks, it detects CALLs and JMPs to direct and indirect locations... and calls to unknown code pages in kernel memory.” - Volatility documentation.

כדי להבין בדיוק מה קורה פה, אנחנו צריכים לעשות חזרה קטנה על מה זה IAT.

כאשר קובץ הרצה נטען לראשונה, ה-Windows loader אחראי על קריאה של מבנה ההרצה של הקובץ וטעינה של הקובץ לזיכרון. בנוסף לכך, הוא גם טוען את כל קבצי ה-DLL שהקובץ משתמש בהן וממפה אותם ל-Process address space.

קובץ ההרצה גם מדווח על כל הפונקציות שהוא זקוק להן מכל-DLL. מכיוון שהכתובות של הפונקציות הן לא סטטיות, היה נדרש לפתח מנגנון שמאפשר לכתובות האלו להשתנות מבלי לשנות את כל הקוד המקומפל בזמן ההרצה.

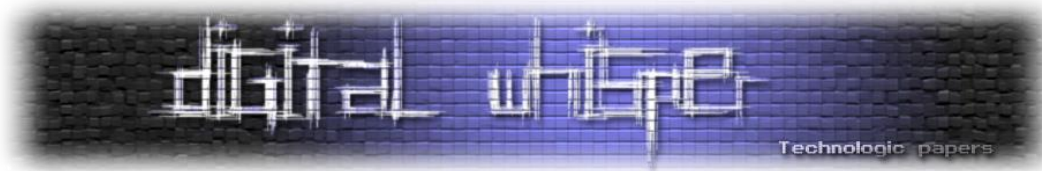
זה הושג על ידי שימוש ב-IAT(Import address table). זוהי למעשה טבלת הפונקציות המיובאות, אשר ממולאת על ידי ה-Windows loader כאשר קבצי ה-DLL נטענים.

מבנה הזיכרון הזה יכול לקבל שינויים מנוזקה כדי לגרום לקובץ ההרצה לקרוא לפונקציות אחרות במקום אלו שהוא באמת התכוון לקרוא להן.

אני רוצה להשתמש ב-**apihooks** ספציפית על התהליך של הדפדפן כי אני כבר חושד שבוצע בו שינוי מסוים בהתבסס על הפלט של-**sockscan**. מכיוון שהפלט של הפקודה ארוך מדי, אני שומר אותו לקובץ טקסט:

```
*****
Hook mode: Usermode
Hook type: Import Address Table (IAT)
Process: 2044 (IEXPLORE.EXE)
Victim module: WINHTTP.dll (0x4d4f0000 - 0x4d548000)
Function: kernel32.dll!LoadLibraryA
Hook address: 0x7ff82a50
Hooking module: <unknown>

Disassembly(0):
0x7ff82a50 51          PUSH ECX
0x7ff82a51 e80aefffff  CALL 0x7ff81960
0x7ff82a56 84c0       TEST AL, AL
0x7ff82a58 7414       JZ 0x7ff82a6e
0x7ff82a5a 8b442408   MOV EAX, [ESP+0x8]
0x7ff82a5e 8b0d0054fa7f  MOV ECX, [0x7ffa5400]
0x7ff82a64 8b512c     MOV EDX, [ECX+0x2c]
0x7ff82a67 50        PUSH EAX
*****
```



זוהי רק תוצאה אחת מתוך רבות של יירוטים (hooks) לפונקציות שונות. כולם קוראים לאותה כתובת - 0x7ff82a6e. בדוגמה הזו, אנחנו יכולים לראות שהנוזקה יירטה את- [LoadLibraryA](#) בקובץ-WINHTTP.dll.

ניתן להשתמש ב-LoadLibrary כדי לטעון מודול לתוך מרחב הזיכרון של התהליך ולהחזיר-handle שניתן להשתמש בו ב-GetProcAddress כדי לקבל את הכתובת של פונקציית ה-DLL.

אנחנו צריכים להבין מה הנוזקה מנסה לבצע פה. כדי לעשות זאת, אנחנו צריכים לצלול אל תוך פיסת הקוד שהנוזקה מנסה להריץ. לכן, אני אשתמש בתוסף-volshell. בעזרתו ניתן לקפוץ לאיזורי זיכרון, לראות מה נמצא בתוכם, ולבצע-disassembly או קריאה עליהם.

```
$ vol.py -f coreflood.vmem volshell
Volatility Foundation Volatility Framework 2.6.1
Current context: System @ 0x810b1660, pid=4, ppid=0 DTB=0x319000
Welcome to volshell! Current memory image is:
file:///home/sansforensics/Downloads/coreflood.vmem
To get help, type 'hh()'
>>> cc(pid=2044)
Current context: IEXPLORE.EXE @ 0xff3ad1a8, pid=2044, ppid=1724 DTB=0x6cc0320
>>> dis(0x7ff81960)
0x7ff81960 55          PUSH EBP
0x7ff81961 8bec         MOV EBP, ESP
0x7ff81963 83ec28      SUB ESP, 0x28
0x7ff81966 a10054fa7f  MOV EAX, [0x7ffa5400]
0x7ff8196b 8138270c0000 CMP DWORD [EAX], 0xc27
0x7ff81971 7607        JBE 0x7ff8197a
0x7ff81973 b001        MOV AL, 0x1
0x7ff81975 e969010000  JMP 0x7ff81ae3
0x7ff8197a 8b0d6875fa7f MOV ECX, [0x7ffa7568]
0x7ff81980 8139270c0000 CMP DWORD [ECX], 0xc27
0x7ff81986 7653        JBE 0x7ff819db
0x7ff81988 8b156875fa7f MOV EDX, [0x7ffa7568]
0x7ff8198e 0fbe4204    MOVSX EAX, BYTE [EDX+0x4]
0x7ff81992 85c0        TEST EAX, EAX
0x7ff81994 7507        JNZ 0x7ff8199d
0x7ff81996 32c0        XOR AL, AL
0x7ff81998 e946010000  JMP 0x7ff81ae3
0x7ff8199d 8b0d6875fa7f MOV ECX, [0x7ffa7568]
0x7ff819a3 83c104      ADD ECX, 0x4
0x7ff819a6 51          PUSH ECX
0x7ff819a7 ff1550e0f97f CALL DWORD [0x7ff9e050]
0x7ff819ad 8945f8      MOV [EBP-0x8], EAX
0x7ff819b0 837df800    CMP DWORD [EBP-0x8], 0x0
0x7ff819b4 740a        JZ 0x7ff819c0
0x7ff819b6 8b55f8      MOV EDX, [EBP-0x8]
0x7ff819b9 52          PUSH EDX
0x7ff819ba ff154ce0f97f CALL DWORD [0x7ff9e04c]
0x7ff819c0 a10054fa7f  MOV EAX, [0x7ffa5400]
0x7ff819c5 8138270c0000 CMP DWORD [EAX], 0xc27
0x7ff819cb 7607        JBE 0x7ff819d4
0x7ff819cd b001        MOV AL, 0x1
0x7ff819cf e90f010000  JMP 0x7ff81ae3
0x7ff819d4 32c0        XOR AL, AL
0x7ff819d6 e908010000  JMP 0x7ff81ae3
0x7ff819db 8b          DB 0x8b
0x7ff819dc 0d          DB 0xd
0x7ff819dd 68          DB 0x68
0x7ff819de 75fa        JNZ 0x7ff819da
>>> █
```



לאחר שנכנסתי ל-volshell, ביצעתי שימוש בפקודה-cc(context) כדי להיכנס לקונטקסט זיכרון של תהליך מסוים. השתמש ב-PID של הדפדפן בתור פרמטר לפקודה.

\*\*\*\*\*

Hook mode: Usermode  
Hook type: Import Address Table (IAT)  
Process: 2044 (IEXPLORE.EXE)  
Victim module: WINHTTP.dll (0x4d4f0000 - 0x4d548000)  
Function: kernel32.dll!LoadLibraryW  
Hook address: 0x7ff82ac0  
Hooking module: <unknown>

Disassembly(0):  
0x7ff82ac0 51 PUSH ECX  
0x7ff82ac1 e89aeffff CALL 0x7ff81960  
0x7ff82ac6 84c0 TEST AL, AL  
0x7ff82ac8 7414 JZ 0x7ff82ade  
0x7ff82aca 8b442408 MOV EAX, [ESP+0x8]  
0x7ff82ace 8b0d0054fa7f MOV ECX, [0x7ffa5400]  
0x7ff82ad4 8b5130 MOV EDX, [ECX+0x30]  
0x7ff82ad7 50 PUSH EAX

\*\*\*\*\*

הכתובת של ה-hook address מראה לנו את הכתובת שיוטרה, והכתובת מתחת מראה לנו את הכתובת שהנוזקה רוצה שהקוד שלנו ילך אליה.

בואו נסכם את כל מה שקרה עד כה:

- ביצענו שימוש ב-volatility כדי למצוא קוד זדוני שרץ על המכונה.
- התחלנו עם פעולות בסיסיות כמו לבצע בדיקה של תהליכים ותקשורת.
- עברנו לבדוק האם הנוזקה מתחבאת על ידי הזרקת קוד להתהליכים תמימים וגילינו מספר דברים מעניינים.

אז כדי להבין מה הקוד הזדוני מבצע, הדבר הראשון שאני רוצה לעשות זה להשתמש ב-vaddump הפקודה הזאת מאפשרת לי לבצע-dump על חלקת זיכרון. אך יש בעיה קטנה: מקודם השתמשנו ב-malfind שלמעשה צריכה להראות לנו מודולים שהוזרקו לזיכרון. לצערנו, לא הצלחנו למצוא שום מודול כזה.

בדרך כלל, כאשר מודול נטען לזיכרון, יש לו דבר שנקרא-MZ header. מכיוון שבמקרה שלנו אין לנו את ה-MZ header, זה סימן טוב שהמודול עבר שינוי מסוים שלא היה אמור לקרות בהרצה תקינה. למרות זאת, אנחנו יכולים להשתמש בפקודה-impscan. שמאפשרת לנו לשלוף מידע על קריאות-API שבוצעו על ידי הקוד לכיוון מודולים אחרים כדי להבין מה בעצם הקוד המוזרק עושה.





Impscan עובד על ידי סריקה של הזיכרון עבור פונקציות כמו-JMP/CALL. קריאות מסוג זה לפעמים ישלפו את כתובת הפונקציה מתוך ה-IAT. מה שבעצם אפשר לעשות עם-volatility, זה לבדוק מה יש בתוך ה-IAT ולאילו פונקציות הוא מצביע.

אז בעצם כדי להשתמש בתוסף הזה אנחנו צריכים לספק את כתובת הבסיס שאליה המודול הזדוני טעון בזיכרון. כדי להשיג את המידע הזה, נשתמש ב-vaddump, וניתן לו בתור פרמטר את התהליך של הדפדפן (2044). נזכור גם שכתובת הזיכרון שאנחנו מחפשים היא 0x7ff81960.

בהתבסס על המידע הזה, אנחנו יכולים להבין מהפלט של-vaddump:

```
VAD node @ 0xff1fb390 Start 0x7ff80000 End 0x7ffadfff Tag Vads
Flags: CommitCharge: 45, PrivateMemory: 1, Protection: 6
Protection: PAGE_EXECUTE_READWRITE
```

שכתובת הבסיס של המודול הזדוני היא 0x7ff80000. עכשיו יש לנו בעצם את כל מה שאנחנו צריכים ואנחנו יכולים להתקדם לשלב הבא שבו אנחנו משתמשים בפקודה-impscan:

IAT	Call	Module	Function
0x7ff9e000	0x77dd77b3	ADVAPI32.dll	SetSecurityDescriptorDacl
0x7ff9e004	0x77dfd4c9	ADVAPI32.dll	GetUserNameA
0x7ff9e008	0x77dd6bf0	ADVAPI32.dll	RegCloseKey
0x7ff9e00c	0x77ddea4f	ADVAPI32.dll	RegCreateKeyExA
0x7ff9e010	0x77dfc123	ADVAPI32.dll	RegDeleteKeyA
0x7ff9e014	0x77ddede5	ADVAPI32.dll	RegDeleteValueA
0x7ff9e018	0x77ddd966	ADVAPI32.dll	RegNotifyChangeKeyValue
0x7ff9e01c	0x77dd761b	ADVAPI32.dll	RegOpenKeyExA
0x7ff9e020	0x77dd7883	ADVAPI32.dll	RegQueryValueExA
0x7ff9e024	0x77ddebe7	ADVAPI32.dll	RegSetValueExA
0x7ff9e028	0x77dfc534	ADVAPI32.dll	AdjustTokenPrivileges
0x7ff9e02c	0x77e34c3f	ADVAPI32.dll	InitiateSystemShutdownA
0x7ff9e030	0x77dfd11b	ADVAPI32.dll	LookupPrivilegeValueA
0x7ff9e034	0x77dd7753	ADVAPI32.dll	OpenProcessToken
0x7ff9e038	0x77dfc8c1	ADVAPI32.dll	RegEnumKeyExA
0x7ff9e03c	0x77dd778e	ADVAPI32.dll	InitializeSecurityDescriptor
0x7ff9e044	0x7c809c28	kernel32.dll	SetEvent
0x7ff9e048	0x7c81082f	kernel32.dll	CreateThread
0x7ff9e04c	0x7c80aa66	kernel32.dll	FreeLibrary
0x7ff9e050	0x7c801d77	kernel32.dll	LoadLibraryA
0x7ff9e054	0x7c809750	kernel32.dll	TlsGetValue
0x7ff9e058	0x7c809bf5	kernel32.dll	TlsSetValue
0x7ff9e05c	0x7c80e016	kernel32.dll	DuplicateHandle
0x7ff9e060	0x7c809919	kernel32.dll	GetCurrentThread
0x7ff9e064	0x7c80e00d	kernel32.dll	GetCurrentProcess
0x7ff9e068	0x7c9105d4	kernel32.dll	HeapAlloc

עכשיו אנחנו יכולים לראות את הרשימה היפה של הפונקציות הטעונות. אנחנו גם יכולים לראות את העמודה של-IAT אשר מראה לנו את הכתובת של האלמנט במערך ה-IAT. Impscan גם מראה לנו את השמות של הפונקציות הטעונות, מה שעשוי לעזור לנו להבין מה הקוד עושה לפי השם של הפונקציה.

אם נחזור קצת למעלה במאמר ונראה את הפלט שקיבלנו לאחר שימוש ב-volshell, אנחנו נראה שיש שם קריאה ל-0x7ff9e04c. בפלט של-impscan שקיבלנו עכשיו, אנחנו יכולים לראות שהכתובת הזאת משוייכת לפונקציה-LoadLibrary. אנחנו בעצם יכולים להסתכל על הפלט של-impscan ולהשוות אותו לקוד המוזרק כדי למצוא מה כל קריאת פונקציה עושה. בדוגמה שלנו, FreeLibrary משחררת את מודול ה-DLL, ובמידה ויש צורך בכך, מחסירה את מספר הייחוסים שלו. כאשר המספר הזה מגיע לאפס, המודול משוחרר ממרחב הכתובת של התהליך הקורא, וה-handle לא תקף יותר.

עכשיו יש לנו יכולת לעבור על הקוד המוזרק ולהבין את היכולת שלו. בעיה גדולה בשיטה הזו היא שזה לוקח המון זמן לעבור על כל הקוד כדי להבין את התמונה הגדולה. עקב כך, אנחנו יכולים פשוט להסתכל על השמות של הפונקציות ולנסות לדלות רמזים מהמידע הזה. כבר עלה בי חשד שמגיע מהפונקציות האלה:

IAT	Call	Module	Function
0x7ff9e000	0x77dd77b3	ADVAPI32.dll	SetSecurityDescriptorDacl
0x7ff9e004	0x77dfd4c9	ADVAPI32.dll	GetUserNameA
0x7ff9e008	0x77dd6b40	ADVAPI32.dll	RegCloseKey
0x7ff9e00c	0x77ddeaf4	ADVAPI32.dll	RegCreateKeyExA
0x7ff9e010	0x77dfc123	ADVAPI32.dll	RegDeleteKeyA
0x7ff9e014	0x77ddede5	ADVAPI32.dll	RegDeleteValueA
0x7ff9e018	0x77ddd966	ADVAPI32.dll	RegNotifyChangeKeyValue
0x7ff9e01c	0x77dd761b	ADVAPI32.dll	RegOpenKeyExA
0x7ff9e020	0x77dd7883	ADVAPI32.dll	RegQueryValueExA
0x7ff9e024	0x77ddebe7	ADVAPI32.dll	RegSetValueExA
0x7ff9e028	0x77dfc534	ADVAPI32.dll	AdjustTokenPrivileges
0x7ff9e02c	0x77e34c3f	ADVAPI32.dll	InitiateSystemShutdownA
0x7ff9e030	0x77dfd11b	ADVAPI32.dll	LookupPrivilegeValueA
0x7ff9e034	0x77dd7753	ADVAPI32.dll	OpenProcessToken
0x7ff9e038	0x77dfc8c1	ADVAPI32.dll	RegEnumKeyExA
0x7ff9e03c	0x77dd778e	ADVAPI32.dll	InitializeSecurityDescriptor
0x7ff9e044	0x7c809c28	kernel32.dll	SetEvent
0x7ff9e048	0x7c81082f	kernel32.dll	CreateThread
0x7ff9e04c	0x7c80aa66	kernel32.dll	FreeLibrary
0x7ff9e050	0x7c801d77	kernel32.dll	LoadLibraryA
0x7ff9e054	0x7c809750	kernel32.dll	TlsGetValue
0x7ff9e058	0x7c809bf5	kernel32.dll	TlsSetValue
0x7ff9e05c	0x7c80e016	kernel32.dll	DuplicateHandle
0x7ff9e060	0x7c809919	kernel32.dll	GetCurrentThread
0x7ff9e064	0x7c80e00d	kernel32.dll	GetCurrentProcess
0x7ff9e068	0x7c9105d4	kernel32.dll	HeapAlloc

אנחנו יכולים לראות שהנוזקה טענה מספק פונקציות כדי להשתמש ב-registry. כפי שאתם כבר כנראה יודעים, ה-registry מכיל מידע רגיש. יש לא מעט נזקות שמשמשות ב-registry כדי להשיג-persistence על ידי הוספה של קובץ הרצה לנתיב של-autorun שיופעל בהרצה הבאה של מערכת ההפעלה.

אנחנו נשתמש בתוסף-handles. הפקודה הזאת מאפשרת לנו להסתכל על ה-handles שבהם משתמשים תהליכים. מערכת ההפעלה של-Windows משתמשת באובייקטים כדי לייצג ולגשת למשאבי מערכת, כמו קבצים, רכיבים, מפתחות וכו'. למעשה, ניתן לגשת לאובייקט על ידי שימוש בטבלת ה-handles שנמצאת ב-kernel. פתיחה של אובייקט תגרור הוספה של מצביע לאובייקט בטבלת ה-handles.

נשתמש בפקודה-handles ונבצע סינון רק על מפתחות-registry:

```
0xe12e8958 2044 0x180 0x20019 Key USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe1051fb8 2044 0x184 0x20019 Key USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe12e4420 2044 0x188 0xf003f Key MACHINE\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER
0xe1068ec0 2044 0x18c 0x20019 Key USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe131abe0 2044 0x19c 0x20019 Key USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe15ff7d8 2044 0x1ac 0x20019 Key USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe1249db0 2044 0x1b8 0x20019 Key USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe10f5d98 2044 0x1bc 0x2001f Key USER\S-1-5-21-1614895754-436374069-839522115-500\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER\RUNMRU
0xe17a37f8 2044 0x1c0 0x20019 Key USER\S-1-5-21-1614895754-436374069-839522115-500\SOFTWARE\MICROSOFT\INTERNET_EXPLORER\TYPEDURLS
0xe10f5e00 2044 0x230 0xf003f Key MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK2\PARAMETERS\PROTOCOL_CATALOG9
0xe12e1978 2044 0x238 0xf003f Key MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK2\PARAMETERS\NAMESPACE_CATALOGS
0xe12db0e8 2044 0x250 0xf003f Key USER\S-1-5-21-1614895754-436374069-839522115-500\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER\FILEEXTS
0xe1332228 2044 0x26c 0x20019 Key MACHINE\SOFTWARE\MICROSOFT\WINDOWS\NT\CURRENTVERSION\DRIVERS32
0xe12e50a0 2044 0x288 0x20019 Key MACHINE\SOFTWARE\MICROSOFT\ACTIVE_SETUP\INSTALLED_COMPONENTS\{89820200-ECBD-11CF-8B85-00AA005B4383}
```

הרשימה עצמה תהיה ארוכה יותר, אבל חתכתי איזור שנראה לי מעניין.

אז קודם כל, אנחנו יכולים לראות מפתח שנגמר ב-"TYPEDURLS". המפתח הזה מקבל שימוש רגיל על ידי הדפדפן, אבל הוא גם יכול לקבל שימוש על ידי נוזקה שתבצע קריאה על אתרים שאליהם נכנס הקורבן. זוהי טכניקת ריגול טובה, מכיוון שהקוד מוזרק לתוך התהליך של הדפדפן עצמו. מפתח דומה הוא-"RUNMRU" שרושם תוכנות שרצות כרגע, מה שיאפשר לתוקף לקבל גישה למידע הזה ולהבין את המערכת של הקורבן יותר טוב.

מה שעוד מעניין פה, זה שימוש במפתח של-"DRIVERS32". זה יכול לרמוז לנו על כך שהנוזקה ביצעה התקנה של התקן במערכת של הקורבן. ראיתי גם מספר מפתחות שקשורים לאבטחה של Internet Explorer. זאת יכולה להיות שיטה נוספת שמחלישה את אמצעי האבטחה על המחשב של הקורבן, וגורמת לו להיות פגיע ליותר סוגי מתקפות.

**סיכום עד כה:**

- עדיין לא ראינו ראיות חזקות לפעולות זדוניות ב-registry.
- לא ראינו עדיין שימוש במפתחות-custom registry.
- אנחנו כרגע רק יכולים לנחש מה הנוזקה עושה ב-registry.

בואו נמשיך.



קעת נחזור קצת אחורה, לכתובת **0x7ff80000**. אני רוצה לבצע-dump לחלקת הזיכרון הזאת כדי לבצע ניתוח מעמיק יותר.

```
sansforensics@siftworkstation: ~/Downloads
$ vol.py -f coreflood.vmem vaddump --pid=2044 -b 0x7ff80000 -D ./
Volatility Foundation Volatility Framework 2.6.1
-----
Pid      Process      Start      End      Result
-----
2044    IEXPLORE.EXE  0x7ff80000 0x7ffadfff ./IEXPLORE.EXE.485d1a8.0x7ff80000-0x7ffadfff.dmp
s
```

אנחנו יכולים להשתמש בפקודה-strings כדי לראות מחרוזות שמובנות אל תוך קובץ מסוים:

```
sansforensics@siftworkstation: ~/Downloads
$ strings IEXPLORE.EXE.485d1a8.0x7ff80000-0x7ffadfff.dmp > injection.txt
s
```

אם נפתח את קובץ הטקסט ונגלול בערך לאמצע, אנחנו נראה את הדבר הבא:

```
user32.dll
Progman
Internet Explorer_Server
cleanup
appinit_dlls
rundll32.exe
winlogon.exe

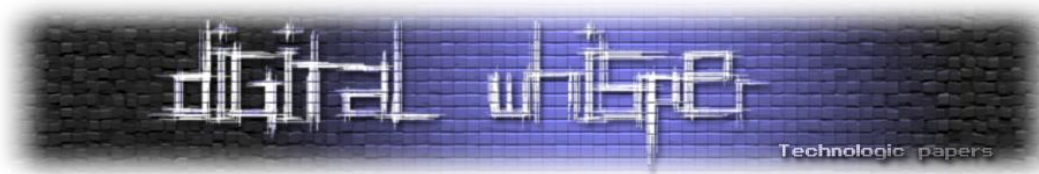
*.nhs.net/*
*.nhs.uk/*
*.hilton.*
*.yahoo.*
*.google.*
checkbox
password
text
submit
hidden
```

אז אנחנו יכולים בוודאות לראות כמה מחרוזות חשודות. בנוסף לכך, נראה שיש פה אפילו בדיקת-regex מסוימת שמחפשת אחרי כתובות אינטרנט ספציפיות, אולי כדי לגנוב חשבונות שקשורים לכתובות האלו?

דבר נוסף שאנחנו יכולים לראות זה חיפוש אפשרי אחר דפדפנים שמותקנים אצל הקורבן:

```
Global\
*\explorer.exe
*\intern*\iexplore.exe
*\firefox.exe
*\opera.exe
*\skype.exe
AFCORE
COM2PLUS_MessageWindowsClass
```

אז זה נראה שיש עוד כמה דפדפנים שהם מטרות להזרקת קוד. שימו לב לכך שאנחנו גם רואים את-explorer.exe כדי לבדוק את התאוריה הזו, נשתמש ב-**malfind**:



```

Process: explorer.exe Pid: 1724 Address: 0x1b20000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01b20000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x01b20010  00 00 b2 01 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x01b20020  10 00 b2 01 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x01b20030  20 00 b2 01 00 00 00 00 00 00 00 00 00 00 00 00  .....

0x01b20000  0000          ADD [EAX], AL
0x01b20002  0000          ADD [EAX], AL
0x01b20004  0000          ADD [EAX], AL
0x01b20006  0000          ADD [EAX], AL
0x01b20008  0000          ADD [EAX], AL
0x01b2000a  0000          ADD [EAX], AL
0x01b2000c  0000          ADD [EAX], AL
0x01b2000e  0000          ADD [EAX], AL
0x01b20010  0000          ADD [EAX], AL
0x01b20012  b201          MOV DL, 0x1
0x01b20014  0000          ADD [EAX], AL
0x01b20016  0000          ADD [EAX], AL
0x01b20018  0000          ADD [EAX], AL
0x01b2001a  0000          ADD [EAX], AL
0x01b2001c  0000          ADD [EAX], AL
0x01b2001e  0000          ADD [EAX], AL
0x01b20020  1000          ADC [EAX], AL
0x01b20022  b201          MOV DL, 0x1
0x01b20024  0000          ADD [EAX], AL
0x01b20026  0000          ADD [EAX], AL
0x01b20028  0000          ADD [EAX], AL
0x01b2002a  0000          ADD [EAX], AL
0x01b2002c  0000          ADD [EAX], AL
0x01b2002e  0000          ADD [EAX], AL
0x01b20030  0000          ADD [EAX], AL

```

כאן יש לנו למעשה חלקת זיכרון עם הרשאות של-EXECUTE\_READWRITE, שתמיד מעלות חשד בנוגע להזרקת קוד. עכשיו אני רוצה לבצע-dump לחלקת הזיכרון הזו ולראות אם נוכל למצוא עוד דברים. אולי נוכל למצוא את אותו הקוד שהוזרק ל-IEXPLORE.EXE? לצערי, נראה שלא כך המקרה מכיוון שהמשקל של איזור הזיכרון לאחר ה-dump הוא קטן מדי בהשוואה ל-dump של-IEXPLORE.EXE. חיפוש על ידי-strings גם לא הניב תוצאות מעניינות.

אם ניתן מבט על קובץ הטקסט, אנחנו נוכל לראות פלט שמיוצר כאשר מדובר בתקשורת- & Command Control (שבסוג תקשורת זה, הקורבן מקבל פקודות מתחנה מרוחקת).



לדוגמה:

```

Bytes transmitted: %d, received: %d
STATUS_DELAY
STATUS_RESOLVING
STATUS_CONNECTING
STATUS_ESTABLISHED
STATUS_SHUTDOWN
STATUS_LISTENING
Invalid command "%s" (%s) from %s
Restricted mode has been set; cannot perform "%s" command without administrative
privilege
Frozen mode has been set; cannot perform "%s" command
Unimplemented command %s
Unknown or ambiguous command %s
Administrative privilege is required for the command "%s"
Creating string table %s
Adding string #d (%s) to the table %s
Finalizing string table %s
Removing string table %s
Removing string #d (%s) from the table %s
String #d (%s) cannot be removed from the table %s
finalized string table %s:
!%d - %s

```

זה בוודאות נראה כמו פלט של פקודות שיוצרו עבור מישהו שמנסה להריץ פקודות על הקורבן. זה נראה כאילו התחנה השולטת תקבל שליטה מלאה על תהליכים שרצים ומופסקים על הקורבן.

אז עכשיו שיש לנו את המידע הזה, אני רוצה לנסות למצוא את מנגנון ה-persistence של הנוזקה, כלומר, איך היא שורדת אתחול של המחשב של הקורבן:

```

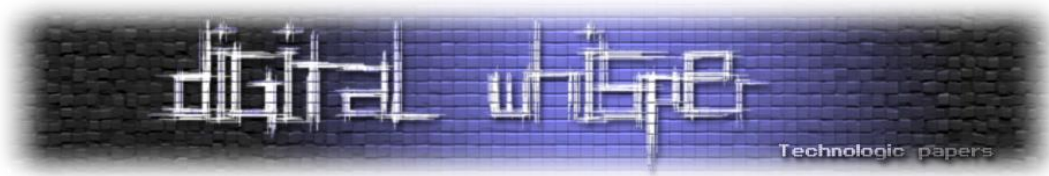
*32.dll
Software\Classes\CLSID\InprocServer32
ThreadingModel
Apartment
Software\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers
SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
Software\Microsoft\Windows NT\CurrentVersion\Windows
AppInit_DLLs
rundll32.exe ,init
#rundll

```

מפתח ה-registry בשם-RunOnce הוא מהלך קלאסי לקבל-persistence, בגלל שכל מה שנכתב למפתח הזה, מורץ פעם אחת כאשר מערכת ההפעלה עולה, ואז נמחק. הנוזקה יכולה לכתוב את עצמה כל פעם אל תוך המפתח הזה כל פעם שהמערכת רצה. שיטה נוספת שאפשר לראות פה היא שימוש ב-AppInit\_DLLs. זה למעשה מנגנון שמאפשר לרשימה של קבצי-DLLs להיטען אל תוך כל תהליך מצב משתמש במערכת. מנגנון זה יכול בקלות להיות מותקף על ידי הנוזקה על ידי רישום של כל ה-DLL שהנוזקה משתמשת בהן אל תוך מיקום המפתח:

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows

כפי שחשדנו כבר, המחרוזות שראינו עד כה נוגעות בעיקר בקבצי-DLL וב-registry.



זה יהיה רעיון טוב לבדוק מה אנחנו יכולים למצוא על קבצי ה-DLL בתיקה-system32:

```
C:\WINDOWS\system32\comsbap.dIl
SetEvent
```

ובכן ובכן ובכן. זה אפילו לא קובץ DLL!! , יש פה שימוש באות i גדולה.

כעת אשתמש בפקודה-filescan כדי לראות אם אני יכול לשלוף את הקובץ הזה מהזיכרון ולנתח אותו:

```
$ vol.py -f coreflood.vmem filescan | grep comsbap
Volatility Foundation Volatility Framework 2.6.1
0x0000000005c5aee0 1 0 R--r-- \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dat
0x0000000005c54be0 1 0 R--r-d \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
0x0000000005c54c78 1 0 R--r-- \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
sansforensics@siftworkstation: ~/Downloads
```

עכשיו, אנחנו יכולים להשתמש ב-dumpfiles כדי לשלוף את הקובץ עצמו מהזיכרון:

```
$ vol.py -f coreflood.vmem dumpfiles -Q 0x0000000005c54be0 -D ./
Volatility Foundation Volatility Framework 2.6.1
ImageSectionObject 0x05c54be0 None \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
DataSectionObject 0x05c54be0 None \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
```

על ידי שימוש בכלי-PEStudio, אנחנו יכולים לקבל מידע נוסף על הקבצים. יש לנו אינדיקציה שהקובץ עבר תהליך של-Packing כדי לגרום לו לעקוף פתרונות אנטי-וירוס:

property	value	value	value
name	UPX0	UPX1	UPX2
md5	n/a	109D2AE3563176ADC194E24...	DE4EA98BC575E09A3860480...
entropy	n/a	7.981	3.962
file-ratio (95.14%)	n/a	94.44 %	0.69 %
raw-address	0x00000400	0x00000400	0x00011400
raw-size (70144 bytes)	0x00000000 (0 bytes)	0x00011000 (69632 bytes)	0x00000200 (512 bytes)
virtual-address	0x10001000	0x1001C000	0x1002D000
virtual-size (184320 bytes)	0x0001B000 (110592 bytes)	0x00011000 (69632 bytes)	0x00001000 (4096 bytes)
entry-point	-	0x0002CC30	-
characteristics	0xE0000080	0xE0000040	0xC0000040
writable	x	x	x
executable	x	x	-
shareable	-	-	-
discardable	-	-	-
initialized-data	-	x	x
uninitialized-data	x	-	-
unreadable	-	-	-
self-modifying	x	x	-
virtualized	x	-	-
file	n/a	n/a	n/a

זה בעצם גורם לי להבין שהמודול הזה באמת זדוני.

## סיכום

ובכן, הצלחנו בעזרת הכוח של volatility לעקוב אחר מספר תהליכים שבהם הנוזקה השתמשה. כמובן שזה לא מתקרב אפילו לעומק המלא של הנוזקה הזו, אלא רק מעבר על הפונקציות הברורות יותר שבהן היא משתמשת.

ניתוח נוזקות וזיכרון זה תחום עצום עם עומק אין סופי ללמידה. ישנם לא מעט סטטיסטיקות ממגוון סוכנויות כאלה ואחרות שמראות לנו פעם אחר פעם כמה מקום נוזקות תופסות לנו בספקטרום אבטחת המידע. בעוד שיש כלים ופלטפורמות אוטומטיות מעולות לניתוח נוזקות, אני חושב שיש ערך עילאי להבנה בסיסית ואף מתקדמת של איך הדברים עובדים מתחת לפני השטח. על ידי שימוש בטכניקות ניתוח סטאטי (בחינה של קובץ ההרצה מבלי בחינה של ההוראות עצמן או ביצוע הנדסה לאחור) וניתוח דינאמי (בחינה של הנוזקה בזמן ההרצה או לאחר ההרצה) אנחנו נקבל תמונה הרבה יותר טובה עם סוגן וטיבן של הנוזקות שאנחנו עשויים להיתקל בהן.

## על הכותב

[דניאל קויפמן](#), בן 24, מהנדס מערכות SIEM ו-Detection, ובעל זיקה חזקה לעולמות ה-DFIR. ניתן ליצור קשר בלינקדאין, טוויטר (@KoifSec) או במייל: [koifmandani@gmail.com](mailto:koifmandani@gmail.com)

## בנוסף: מתודולוגיות זיהוי של שירותים זדוניים

אשמח להציג פה את תהליך העבודה הכללי שאני עוקב אחריו כאשר מדובר בניסיון זיהוי של שירותים זדוניים:

- מעבר על מרשם האירועים, בדגש על אירועים של 4697, 7040, 7036, 7034, 7035, 7045.
- מעבר על מפתחות-registry של שירותים:
  - HKLM\SYSTEM\CurrentControlSet\Services
  - HKLM\SYSTEM\ControlSet001\Services
  - HKLM\SYSTEM\ControlSet002\Services
- מעבר על רשומות 4688 של שירותים שנוצרו על ידי sc.exe/at.exe
- מעבר על מזהים של הרצה: shimcache, prefetch, amcache.
- מעבר על \$MFT/\$I30 עבור סימנים של יצירת קבצים זדוניים.
- שימוש ב-svcscan של volatility כדי לזהות שירותים לא מקושרים ולזהות שירותים רצים.



## בנוס: איך הזרקת-DLL עובדת? <sup>[2]</sup>

הזרקת-DLL היא טכניקת הזרקה שטוענת קובץ-DLL זדוני בתוך ההקשר של תהליך רץ לגיטימי. התהליך המרוחק מקבל מניפולציה על ידי פונקציות כמו-[CreateRemoteThread](#). כאשר התהליך הנגוע טוען את ה-DLL הזדוני, מערכת ההפעלה באופן אוטומטי קוראת לפונקציה-DllMain של הקובץ, אשר נקבעת על ידי היוצר של קובץ ה-DLL. הפונקציה מכילה את הקוד הזדוני, ויש לה את אותה הגישה למערכת כמו לתהליך שבתוכו היא רצה. השלבים שהתהליך הזדוני מבצע הם:

- הפעלה של הרשאות-debug (debug\_privilege) (SB\_DEBUG\_PRIVILEGE) אשר נותנת לה את האפשרות לקרוא ולכתוב בזיכרון של תהליכים אחרים, כאילו היא היתה-debugger.
- מציאה של ה-process ID על ידי שימוש בשם של התהליך. בדרך כלל זה מבוצע על ידי יצירת תמונת מערכת (snapshot) וחפוש אחר התהליך הספציפי על ידי שימוש בפונקציות כמו-Process32Next, Process32First, CreateToolhelp32Snapshot.
- פתיחה של התהליך הקורבן עם ההרשאות המתאימות, והשגה של-handle עבורו. ניתן לבצע זאת על ידי קריאה לפונקציה-OpenProcess עם לפחות אחד מהגישות הבאות: PROCESS\_CREATE\_THREAD, PROCESS\_VM\_WRITE, PROCESS\_VM\_OPERATION.
- השגה של הנתבי המלא של ה-DLL, אשר כבר קיים על הדיסק.
- שיוך של איזור זיכרון חדש בתוך מרחב הכתובות הוירטואלי של התהליך הקורבן, בגודל השם המלא של ה-DLL הזדוני (כולל הנתבי שלו). ניתן לבצע זאת על ידי שימוש ב-VirtualAlloc עם הרשאות של-PAGE\_READWRITE.
- כתיבה של השם המלא של ה-DLL את תוך איזור הזיכרון החדש על ידי שימוש ב-WriteProcessMemory אל תוך הכתובת ישירות.
- יצירה של תת-תהליך חדש בתוך ההקשר של התהליך המקורי אשר מריץ את הפונקציה-LoadLibrary, עם שימוש בתור פרמטר בנתבי המלא של קובץ ה-DLL. זה למעשה טוען את ה-DLL הזדוני וההזרקה מבוצעת. ראשית, נשלף-handle עבור kernel32.dll והכתובת של הפונקציה-LoadLibrary נשלפת ממנו. לאחר מכן, תת-התהליך נוצר על ידי שימוש ב-APIs כמו-CreateRemoteThread, NtCreateThreadEx, או RtlCreateUserThread. הפרמטרים החשובים עבורם הם ה-handle של התהליך הקורבן, המצביע (pointer) לכתובת של הפונקציה-LoadLibrary, והמצביע לכתובת איזור הזיכרון שמאחסנת את השם המלא של קובץ ה-DLL. זה אומר ש-LoadLibrary טוענת את קובץ ה-DLL. תת-התהליך מריץ את איזור הזיכרון הוירטואלי של התהליך הקורבן באופן מיידי לאחר היצירה שלו.
- ניקוי על ידי שחרור הזיכרון המוקצה וסגירה של ה-handle לתהליך הקורבן ולתת-התהליך, על ידי שימוש ב-VirtualFree ו-CloseHandle, בהתאמה.