



---

## עוקבים אחרי ETW

מאת בן פרינטק

---

### הקדמה

מיקרוסופט, בעזרת מערכת ההפעלה האימתנית שלה, מצליחה להחזיק כבר תקופה ארוכה את התואר הנחשק, הלא הוא: "[מערכת ההפעלה הפופולרית ביותר](#)". נכון לחודשים האחרונים, עם אחוז מוחץ של 76% בקרב עמדות הקצה לשימוש ביתי ומשרדי בעולם.

עם השנים עולם אבטחת המידע נכנס לשיח הציבורי והעסקי, הדליק נורה אדומה גדולה ודחף חברות (ומיקרוסופט בפרט) להטמיע מנגנוני הגנה ב-Windows ואיתם מוצרי הגנה צד שלישי כגון: פתרונות ניטור ובקרה (Sysmon, Splunk, ArcSight), DLP, AV/EDR ו-SIEM אשר ידאג להציג את המידע מהסוכנים ויביאו לכל אדם או ארגון את האחיזה המיטבית והרחבה על עמדה וככלל על הארגון - ואיתו מגיעים רגעי הנחת.

נכון להיום, כמעט כל חברה המכילה סביבת עבודה מרובת עמדות ושרתים מנוהלת תחת Active Directory. סביבה זו נותנת לצוותי התפעול וההגנה את הדרך היעילה והנוחה ביותר לטפל בכל הבלאגן הזה הנקרא רשת ארגונית. כחלק משגרת ההגנה אשר מתפתחת מיום ליום אני מקווה ורוצה להאמין שכל חברה מטמיעה ניטור, תוכנת Antivirus (אך רק בגלל ש-Defender הבסיסי לא מנגיש ניהול רוחבי ונוח אז גם AV/EDR) ושרת האחראי להפצת עדכונים וניהול עמדות לטובת שליטה ובקרה אך בעיקר לצורך הגנה. הנתונים מוזרמים מהעמדות לשרת הניטור של הארגון המשחק כאן דמות מרכזית מעצם היותו הפחד הגדול של כל תוקף ומנגד עיני המגן. הרי כל עוד אין שריטות ועקבות אין תוקף ברשת 😊.

במאמר זה אציג את המנגנון Event Tracing for Windows או בקיצור ETW. איך הוא עובד, על מה הוא אחראי, מה אפשר לעשות איתו לטובה, מה אפשר לעשות איתו לרעה וכל מה שביניהם.

לפני שנמשיך הלאה, אני ממליץ בחום להכיר את הנושאים הבאים:

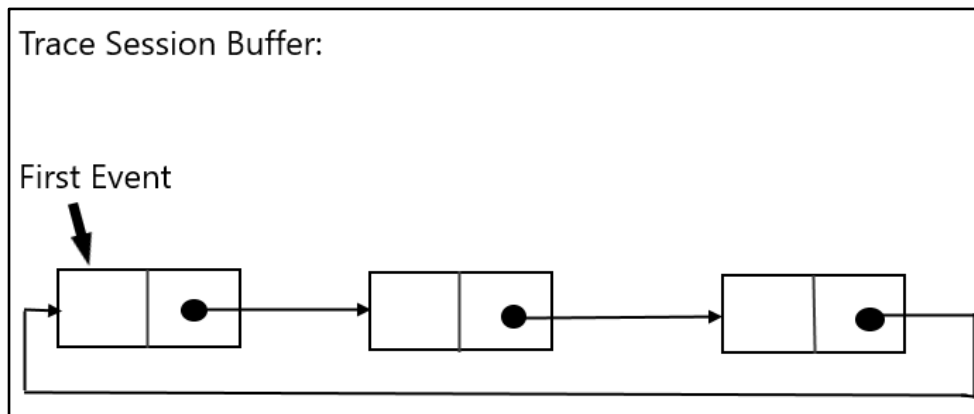
- C / C++ && WinAPI
- Windows Internals
- WinDbg && x64 Assembly

כך אצא מנקודת הנחה שהקורא מבין אותם ולא אתעכב עליהם במסגרת מאמר זה.

## Event Tracing for Windows

עם צאת Windows 2000 אי שם בדצמבר 99. הגיע לעולם מנגנון רישום מתוחכם ומהיר המספק מעקב על כלל מערכת ההפעלה. המנגנון נועד במקור לספק יכולות דיאגנוסטיקה, בקרה ומעקב על ביצועי המחשב מה-Kernel וגם מ-User Mode כאחד.

היתרון המרכזי של המנגנון, הוא היכולות לרשום אירועים מפורטים ללא "בזבז" משאבי המחשב. הרישומים מוזרמים לכל מי שחפץ בהם ונכתבים ל-Buffer מעגלי של השיחה (רשימה מקושרת מעגלית של רישומים), בעת יצירת Trace Session מגדירים את גודלו.



[תרשים המציג את ה-Buffer המדובר לעיל ואת יכולות הצירוף של Paint]

עוד נישה שרק הרימה למנגנון, היא הגמישות שלו. היכולות לבחור כירורגית אילו רישומים להציג מתוך Provider וסינון על בסיס הקריטריות שלהם, נותנת למשתמש ראייה מופתית.

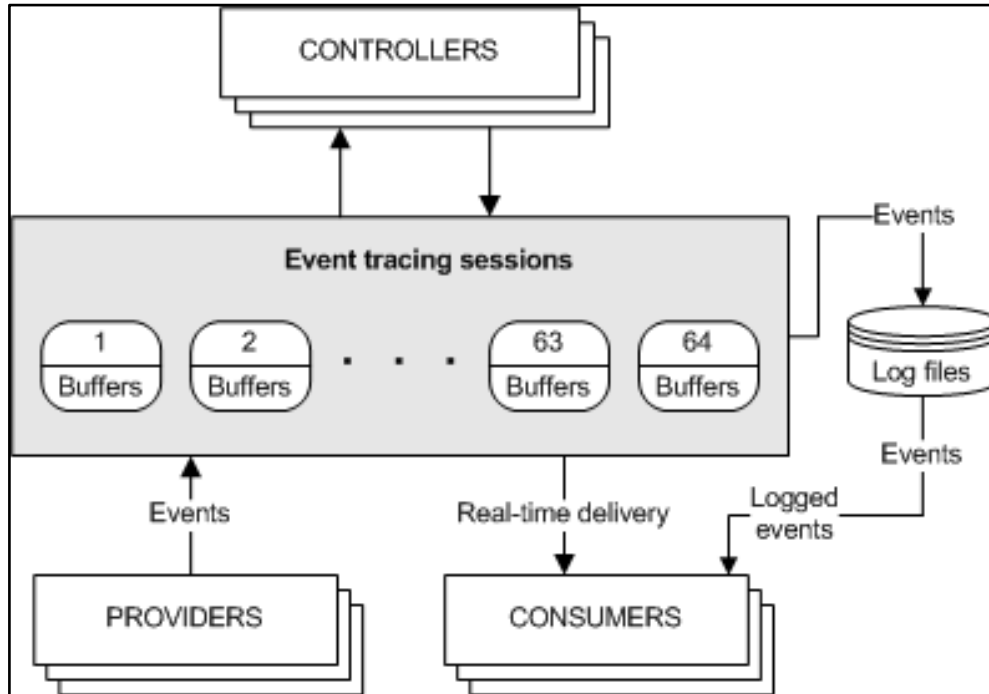
כבר עם התקנת Windows, מגיעים אלפי Providers המספקים מידע מפורט כמעט על כל אובייקט במערכת ההפעלה (הכולל מידע על ומה-User, Application, Kernel) והכל על מנת לספק תמונת מצב נוחה.

עד לתקופת Windows Vista, כל תהליך יכול היה להירשם לכמות בלתי מוגבלת של Providers ולקבל מידע מכל אחד מהם ללא הגבלה. אך כיום, כל תהליך במערכת ההפעלה יכול להירשם לעד כ-1,024 Providers. עם השנים, התבסס המנגנון והפך לחלק בלתי נפרד ממערכת ההפעלה ומ- Windows Performance Toolkit, סט כלים המיועד למתכנתים ולאנשי תשתיות.

בשנים האחרונות, הפך המנגנון לנתיב גילוי משמעותי המשפיע על כלי הגנה רבים (לא קשור כלל לתכנונים של מיקרוסופט), עזר לבסס אנליזות וכך להצביע על פעולות זדוניות בזמן אמת (אי שם ב-2012 מארק ראסינוביץ התחיל לדבר על סט כלים מזהיר ועוצמתי שפיתח עם חבריו במיקרוסופט, שמו Sysinternals. כן, הוא כבר אז השתמש במנגנון).

## ארכיטקטורה ומושגי יסוד

מבנה המנגנון מתבסס על כמה גורמים משמעותיים, התמונה הבאה ממחישה את החיבור בין כולם.



**Trace Session** - שיחה האחראית לקבל רישומים ולהעביר אותם לגורם המאזין. השיחה מאגדת את כל הגורמים במנגנון ומכילה הגדרות משלה, כגון: גודל ה-Buffer המקבל רישומים, שיטת הזרמת רישומים (Real-Time Or ETL File) למרות שקבצי ETL דומים במבנה ל-EVTX, נצטרך לבצע ניתוח שונה ולא להשתמש באותם כלי פרסור), אילו רישומים יקבל ה-Consumer מ-Provider מסוים (יכולות סינון) ואיזה Provider יספק מידע ל-Consumer המאזין. כיום, יכולים לעבוד במקביל עד כ-64 Trace Sessions, אך הגבלה זו ניתנת לשינוי ב-Registry עם יצירת DWORD בשם MaxSessionPerUser בנתיב:

`"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\WMI\Security"`

הנתיב הנל רק מזכיר לנו את WMI, אשר ביסס אותו עוד בתחילת דרכו של המנגנון אי שם ב-Windows 2000.

**Consumer** - אפשר להבין די בקלות מה חלקו במנגנון, תוכנה המעוניינת לקרוא / להאזין לרישומים תצטרך להיות מקושרת ל-Trace Session פעיל (גם אם היא מעוניינת להאזין לכמה במקביל, היא תוכל). Consumer שכל אחד מאיתנו מכיר הוא Event Viewer, ולמטרות דיאגנוסטיקה ודיבאגינג מקובל גם להזכיר את Resource Monitor ואת Windows Performance Analyzer.

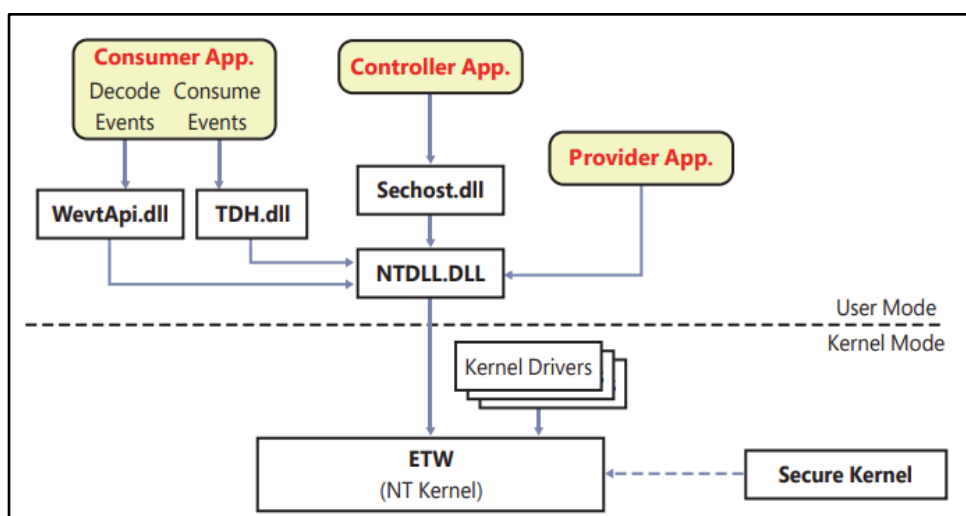
**Provider** - הגורם האחראי על יצירת הרישומים וכתובתם ל-Trace Session(). כל תוכנה רשאית להירשם בתור Provider ולספק מידע על פי כל תבנית שתוצה. עקב המספר הרב של ה-Providers הקיימים במערכת ההפעלה, הם ידעו לספק מידע אך ורק כאשר אקטיבים ומקושרים ל-Trace Session. קיימים ארבעה סוגי Providers והם MOF, WPP, Manifest-Based ו-TraceLogging. חשוב לציין, Provider יכול להיות תוכנה שרצה ביוזר מוד, דרייבר וגם DLL, כל Provider מותקן (בדרך פשוטה בעזרת Wevtutil.exe) במערכת ההפעלה עם GUID - Globally Unique Identifier משלו, על מנת לתקשר עם Provider נעשה זאת בעזרת [EventRegister](#) מ-User Mode ו-[EtwRegister](#) מקרנל מוד. אפשר למצוא כל Provider הנרשם במערכת ההפעלה, בנתיב ה-Registry הבא:

```
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\WINEVT\Publishers"
```

מחקר זה עסק ברוב מוחלט של Manifest Based Providers, לכן אתמקד בסוג זה ולא אפרט על אחרים.

**Controller** - אחראי על כל הקשור במנגנון. ביצוע שינויים ב-Trace Sessions (יצירת Trace Session, עזרה והפעלה של Trace Session קיים, מחיקה והוספה של Provider ל-Trace Session קיים ושינויים בהגדרות השוטפת של ה-Trace Session). אז כן, הדרך הנוחה ביותר להתעסק עם המנגנון היא בעזרת [LOLBIN](#) (טכניקת תקיפה אשר עושה שימוש זדוני בפונקציונאליות הטבעית של בינארים אשר כבר נמצאים על העמדה הנתקפת) הנקרא Logman, אם תרצו פחות פונקציונליות ופחות CLI תוכלו לבצע מניפולציות גם בעזרת Performance Monitor.

אז אסכם בקצרה: כיום, קיימים אלפי Providers המספקים מידע על הפעולות השוטפות במערכת ההפעלה (גם אם אין אחד שמספק את הרצוי, אמרתי זאת קודם אך בכל זאת, תמיד אפשר ליצור אחד בעזרת Manifest ו-Wevtutil). כל אחד הרוצה לקבל מידע מהם, יצטרך או לבצע פונקציות WinAPI המבצעות הרשמה והפעלה שבסופו של יום כל Controller משתמש בהם גם כן, כגון: StartTrace, מה שיבצע קישור לשיחה חדשה/קיימת או בדרך פשוטה יותר, להשתמש ב-Controller קיים.



## Manifest Based Providers

עד עכשיו הבנו איך אפשר ליצור Trace Session, אבל איך יוצרים Provider אתם שואלים? כמו שאמרתי קודם, Providers אחרים על יצירת מבנה הרישומים וגם על כתיבתם ל-Trace Session. כדי ש-Provider ידע לכתוב רישומים הוא צריך הסבר מפורט על מבנה הרישום, וכאן נכנס לתמונה ה-Instrumentation Manifest.

מסמך XML (מסתיים ב-man). על בסיס קונבנציות מיקרוסופט, המכיל את כל המידע שצריך ה-Provider - מבנה כל רישום והגדרותיו, איך לתת מענה ל-Consumer לסינון הרישומים, שם ה-Provider ועוד הגדרות גנריות שלו.

פאבל יוסיפוביץ עשה לנו עבודה קלה וכתב כלי בשם [EtwExplorer](#) המציג לנו את כל ה-Manifests Based Providers הללו בצורה די נוחה, אך בכל זאת, אנחנו נעשה זאת ידנית:). כדי להגיע להבנה עמוקה יותר, נעבור על כמה מאבני היסוד ב-Manifest של Provider לדוגמה, [winmeta.xml](#) ואיתו דוגמאות מ-MSDN.

ה-Element הראשון ב-XML, הוא <provider> אשר נועד לתת פרטים מזהים על Provider, מכיל את הפרטים הבאים:

```
XML
Copy
<instrumentationManifest
  xmlns="http://schemas.microsoft.com/win/2004/08/events"
  xmlns:win="http://manifests.microsoft.com/win/2004/08/windows/events"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  >
  <instrumentation>
    <events>
      <provider name="Microsoft-Windows-SampleProvider"
        guid="{1db28f2e-8f80-4027-8c5a-a11f7f10f62d}"
        symbol="PROVIDER_GUID"
        resourceFileName="<path to the exe or dll that contains the metadata resources>"
        messageFileName="<path to the exe or dll that contains the string resources>"
        message="$(string.Provider.Name)"
        . . .
      </provider>
    </events>
  </instrumentation>
```

- Name - כשמו כן הוא, כאן יהיה כתוב שמו
- GUID - כנל, מכיל את מזהה ה-Provider, ה-GUID שלו
- ResourceFileName - מכיל נתיב ל-EXE\DLL המכיל את מבנה הרישומים, לכן השימוש ב-Resource
- MessageFileName - מכיל נתיב ל-EXE\DLL - המשלים של ה-Resource, רק במבנה ברור למשתמש
- Symbol - סימבול מקושר ל-Provider
- Message - בתכלס, הוא לא הכרחי כ"כ... נועד להסביר ל-Consumer מה שמו. כאשר לא יהיה קיים ישתמשו ב-Name.



<channel> - כחלק מהגדרת ה-Provider נוכל לכתוב רישומים ל-channel. קיימים ארבעה סוגים והם: Admin, Operational, Analytic, ו-Debug. כל Channel נועד לספק חלוקה נוחה למשתמש על בסיס צורך ועניין:

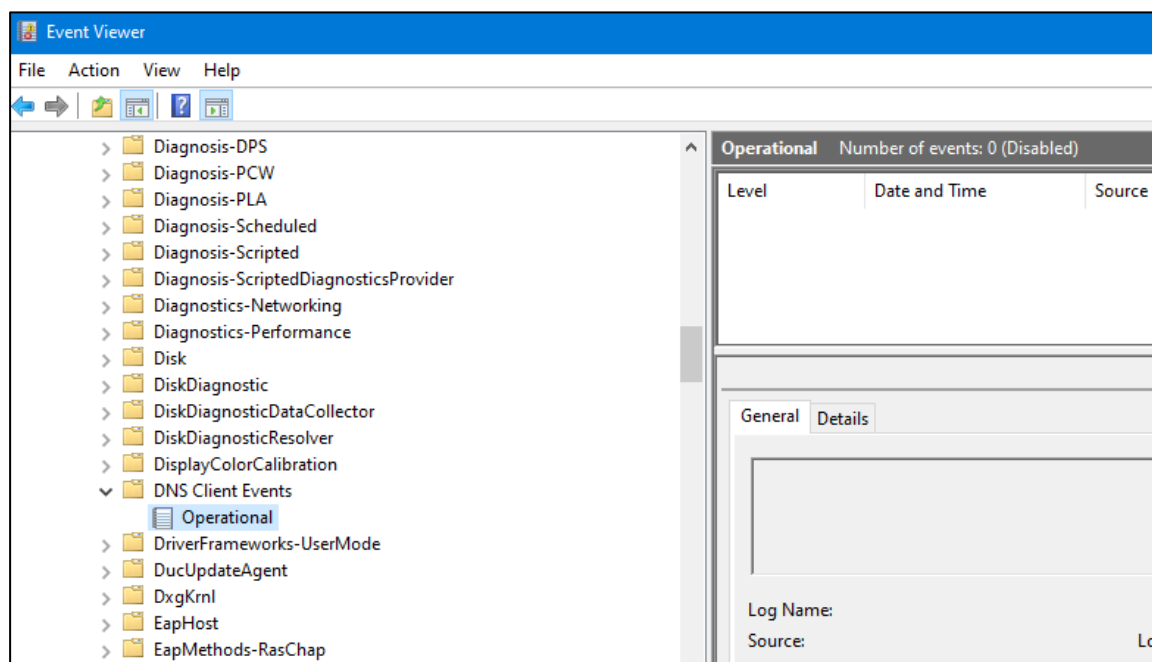
```
<channels>
  <importChannel chid="c1"
    name="Microsoft-Windows-BaseProvider/Admin"
    symbol="CHANNEL_BASEPROVIDER_ADMIN"
  />

  <channel chid="c2"
    name="Microsoft-Windows-SampleProvider/Operational"
    type="Operational"
    enabled="true"
  />
</channels>
```

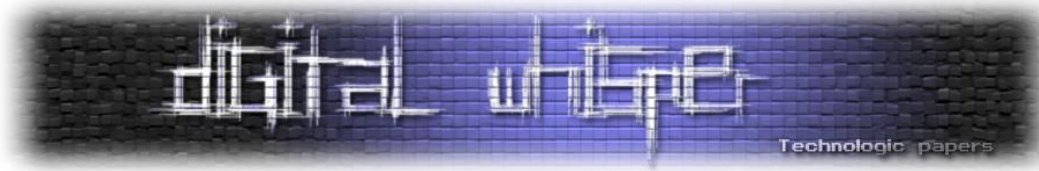
- **chid** - הוא ה-Channel ID, הסמל המייצג כל Channel. חייב להיות ייחודי רק לו.
- **name** - שמו של ה-Channel ואיתו מגיע שמו של ה-Trace Session.
- **type** - מסמל את אחד מארבעת סוגי ה-Channels.

כן, כל מי שתהה לעצמו, מונחים אלה מוכרים לנו מ-Event Viewer ואלה אותם Channels.

**הערה:** ברירת המחדל של Event Viewer היא להסתיר את Debug ו-Analytic אך תמיד אפשר להפעיל אותם ב-View Tab.



[Event Viewer - ב DNS-Operational Channel]



<levels> - אחראי על הגדרת דרגות החומרה של רישומי ה-Provider, בעזרת Element זה Consumer יכול לסנן רישומים על בסיס חומרתם בסדר עולה. כאשר יוצרים Manifest, נוכל להתבסס על הקונבציה המוכרת של מיקרוסופט אשר קיימת אצל כמה וכמה Providers, אך נוכל גם כן ליצור דרגות חדשות לגמרי.

כמו אצל ה-Channels, ההגדרות הללו גם בשימוש ה-Event Viewer, ובעזרתם אפשר לסנן על פי חומרת האירוע.

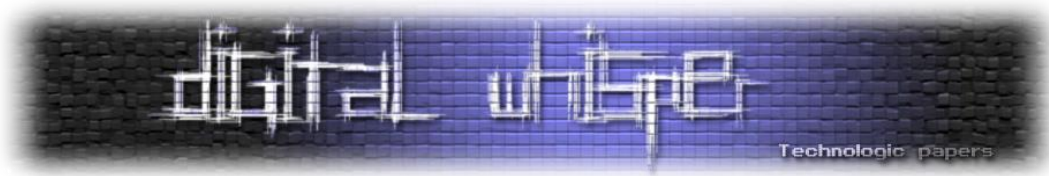
האפשרויות די גנריות ואני מאמין שגם די מוכרות לכולנו, Critical, Error, Warning, Information ו-Verbose:

```
<levels>
  <level name="win:LogAlways" symbol="WINEVENT_LEVEL_LOG_ALWAYS" value="0" message="$(string.level.LogAlways)"> Log Always </level>
  <level name="win:Critical" symbol="WINEVENT_LEVEL_CRITICAL" value="1" message="$(string.level.Critical)"> Only critical errors </level>
  <level name="win:Error" symbol="WINEVENT_LEVEL_ERROR" value="2" message="$(string.level.Error)"> All errors, includes win:Critical </level>
  <level name="win:Warning" symbol="WINEVENT_LEVEL_WARNING" value="3" message="$(string.level.Warning)"> All warnings, includes win:Error </level>
  <level name="win:Informational" symbol="WINEVENT_LEVEL_INFO" value="4" message="$(string.level.Informational)"> All informational content, including win:Warning </level>
  <level name="win:Verbose" symbol="WINEVENT_LEVEL_VERBOSE" value="5" message="$(string.level.Verbose)"> All tracing, including previous levels </level>
```

<templates> - נועד להציג מבנה לדוגמא של רישומים ל-Consumer. איך הנתונים אשר מגיעים בזמן ריצה יוצגו ברישום. לכן ה-Provider מציג תבנית לדוגמא של הרישום וכך ה-Consumer יכול לאכלס את הנתונים במקומם. מה שאומר ש-template יראה לנו איך הרישום הספציפי נראה ואילו נתונים נצטרך לשים בו, עד רמת סוג המשתנה שמתקבל:

```
<templates>
  <template tid="DnsNoServerConfigV4Args">
    <data name="Location" inType="win:UInt32" />
    <data name="Context" inType="win:UInt32" />
  </template>
  <template tid="DnsServerForInterfaceArgs">
    <data name="Interface" inType="win:UnicodeString" />
    <data name="TotalServerCount" inType="win:UInt32" />
    <data name="Index" inType="win:UInt32" />
    <data name="DynamicAddress" inType="win:UInt8" map="DnsIpTypeMap" />
    <data name="AddressLength" inType="win:UInt32" />
    <data name="Address" inType="win:Binary" length="AddressLength" />
  </template>
  <template tid="DnsServerQueryChangeArgs">
    <data name="Interface" inType="win:UnicodeString" />
    <data name="AddressLength" inType="win:UInt32" />
    <data name="Address" inType="win:Binary" length="AddressLength" />
  </template>
  <template tid="DnsServerValidationSuccessArgs">
    <data name="AddressLength" inType="win:UInt32" />
    <data name="Address" inType="win:Binary" length="AddressLength" />
```

[תבנית של רישומים מה-Provider, DNS-Client]



<keyword> - בעזרת Element זה נוכל למסגר רישומים ולייצג אותם כקבוצה. בעזרתו ה-Consumer

יכול לסנן גם כן, אך על בסיס תחום הגדרתו של ה-Provider. מבנה ה-Element נראה כך:

- Name - שם ייחודי ל-keyword
- Message - לרוב יסביר על ה-keyword
- mask - ייחודי bitmask אשר בעזרתו Consumer יכול לבחור אילו רישומים יקבל

```
<keywords>
  <keyword name="ut:GenericEvent" message="$(string.keyword_ut:GenericEvent)" mask="0x100" />
  <keyword name="ut:DnsAutoLogKeyword" message="$(string.keyword_ut:DnsAutoLogKeyword)" mask="0x100000000" />
  <keyword name="ut:PolicyTable" message="$(string.keyword_ut:PolicyTable)" mask="0x200000000" />
  <keyword name="ut:PerfCheckPoints" message="$(string.keyword_ut:PerfCheckPoints)" mask="0x400000000" />
  <keyword name="ut:RegistrationEvent" message="$(string.keyword_ut:RegistrationEvent)" mask="0x800000000" />
  <keyword name="ut:SendPath" message="$(string.keyword_ut:SendPath)" mask="0x100000000" />
  <keyword name="ut:ReceivePath" message="$(string.keyword_ut:ReceivePath)" mask="0x200000000" />
  <keyword name="ut:L3ConnectPath" message="$(string.keyword_ut:L3ConnectPath)" mask="0x400000000" />
  <keyword name="ut:L2ConnectPath" message="$(string.keyword_ut:L2ConnectPath)" mask="0x800000000" />
  <keyword name="ut:ClosePath" message="$(string.keyword_ut:ClosePath)" mask="0x100000000" />
  <keyword name="ut:Authentication" message="$(string.keyword_ut:Authentication)" mask="0x200000000" />
  <keyword name="ut:Configuration" message="$(string.keyword_ut:Configuration)" mask="0x400000000" />
  <keyword name="ut:Global" message="$(string.keyword_ut:Global)" mask="0x800000000" />
</keywords>
```

**הערה:** לכל Keyword קיים Bitmask מקושר אליו (מה שאפשר לראות - Mask), הערך נועד לסנן רישומים על ידי חיבור כל Bitmask לערך אחד כולל. אם נרצה לקבל רישומים רק על כמה רישומים ספציפיים, נוכל לקחת כמה Masks מכל מיני רישומים שונים לסכום אותם ולהעביר את הערך כאשר אנו יוצרים Trace Session.

<event> - ה-Element החשוב מכולם. על מנת שה-Provider ידע ליצור אותם, כל רישום חייב להיות מוגדר וכתוב ב-XML. מה שמייחד כל event הוא משתנה ה-Value שלו שברוב המקרים הוא מסמל את מספר הרישום (Event ID) ייחודי רק ל-event אחד בתוך Provider):

```
<events>
  <event value="1000" symbol="DnsNoServerConfigV4" version="0" task="DnsNoServerConfigV4" level="win:Informational" template="DnsNoServerConfigV4" />
  <event value="1001" symbol="DnsServerForInterface" version="0" task="DnsServerForInterface" level="win:Informational" template="DnsServerForInterface" />
  <event value="1002" symbol="DnsServerQueryChange" version="0" task="DnsServerQueryChange" level="win:Informational" template="DnsServerQueryChange" />
  <event value="1003" symbol="DnsServerValidationSuccess" version="0" task="DnsServerValidationSuccess" level="win:Informational" template="DnsServerValidationSuccess" />
  <event value="1005" symbol="DnsServerValidationFailure" version="0" task="DnsServerValidationFailure" level="win:Error" template="DnsServerValidationFailure" />
  <event value="1007" symbol="DnsMissingPrimarySuffix" version="0" task="DnsMissingPrimarySuffix" level="win:Error" template="DnsMissingPrimarySuffix" />
  <event value="1008" symbol="DnsMissingPrimarySuffixSystem" version="0" task="DnsMissingPrimarySuffixSystem" level="win:Warning" keywords="ut:DnsAutoLogKeyword" />
  <event value="1009" symbol="DnsNonMatchingSuffix" version="0" task="DnsNonMatchingSuffix" level="win:Error" template="DnsNonMatchingSuffix" />
  <event value="1010" symbol="DnsNonMatchingSuffixSystem" version="0" task="DnsNonMatchingSuffixSystem" level="win:Warning" keywords="ut:DnsAutoLogKeyword" />
  <event value="1011" symbol="DnsHostFileError" version="0" task="DnsHostFileError" level="win:Error" template="DnsMissingPrimarySuffixArgs" />
  <event value="1012" symbol="DnsHostFileErrorSystem" version="0" task="DnsHostFileErrorSystem" level="win:Error" keywords="ut:DnsAutoLogKeyword" />
  <event value="1013" symbol="DnsAllServersTimeout" version="0" task="DnsAllServersTimeout" level="win:Error" template="DnsAllServersTimeoutArgs" />
  <event value="1014" symbol="DnsAllServersTimeoutSystem" version="0" task="DnsAllServersTimeoutSystem" level="win:Warning" keywords="ut:DnsAutoLogKeyword" />
  <event value="1015" symbol="DnsServerTimeout" version="0" task="DnsServerTimeout" level="win:Informational" template="DnsAllServersTimeoutArgs" />
  <event value="1016" symbol="DnsNameError" version="0" task="DnsNameError" level="win:Informational" template="DnsAllServersTimeoutArgs" />
  <event value="1017" symbol="DnsAuthoritativeResponse" version="0" task="DnsAuthoritativeResponse" level="win:Informational" template="DnsAllServersTimeoutArgs" />
  <event value="1018" symbol="DnsLinkLocal" version="0" task="DnsLinkLocal" level="win:Informational" template="DnsLinkLocalArgs" />
  <event value="1019" symbol="DnsNoServerConfigV6" version="0" task="DnsNoServerConfigV6" level="win:Informational" template="DnsNoServerConfigV6" />
  <event value="1020" symbol="DnsReadPolicyTable" version="0" task="DnsReadPolicyTable" level="win:Informational" keyword="ut:Global" />
</events>
```

[כל Event מכריז על Attributes שלו (Value, Version) וכו'.] שיוך ל-level,template ומתאים ואם חלק מ-keyword





## מה עכשיו?

לפני שנכנס לברירה העמוקה הזו, נתחיל עם Collector הקיים במערכת ההפעלה, Logman. בעזרתו נוכל לשלוט בכל הנוגע במנגנון. מיצירת, הפעלת ושינוי הגדרות ה-Trace Session, עד תשאולי Providers שונים במערכת ההפעלה והתעסקות איתם.

בעזרת הדגל "-ets", נוכל לבצע תשאול ספציפי על Tracing Session. לדוגמא, עם הפקודה בתמונה מטה נוכל לראות את כל ה-Trace Sessions אשר פועלים:

```
PS C:\Users\test> logman.exe query -ets
```

Data Collector Set	Type	Status
Circular Kernel Context Logger	Trace	Running
Eventlog-Security	Trace	Running
DiagLog	Trace	Running
Diagtrack-Listener	Trace	Running
EventLog-Application	Trace	Running
EventLog-System	Trace	Running
LwtNetLog	Trace	Running
Microsoft-Windows-Rdp-Graphics-RdpIdd-Trace	Trace	Running
NetCore	Trace	Running
NtfsLog	Trace	Running
RadioMgr	Trace	Running
UBPM	Trace	Running
WdiContextLog	Trace	Running
WiFiSession	Trace	Running
EventLog-Microsoft-Windows-Sysmon-Operational	Trace	Running
umstartup	Trace	Running
UserNotPresentTraceSession	Trace	Running
COM	Trace	Running

נוכל לבצע תשאול ספציפי ולבקש פירוט על Trace Session לבחירתנו כך:

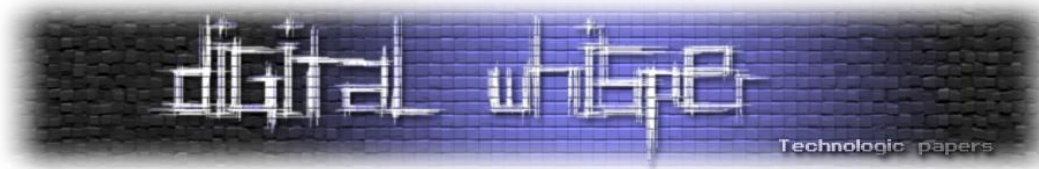
```
PS C:\Users\test> logman.exe query WiFiSession -ets
```

```
Name: WiFiSession
Status: Running
Root Path: C:\Windows\System32\LogFiles\WMI
Segment: Off
Schedules: On
Segment Max Size: 8 MB

Name: WiFiSession\WiFiSession
Type: Trace
Output Location: C:\Windows\System32\LogFiles\WMI\Wifi.etl
Append: Off
Circular: On
Overwrite: Off
Buffer Size: 80
Buffers Lost: 0
Buffers Written: 1
Buffer Flush Timer: 0
Clock Type: Performance
File Mode: File

Provider:
Name: {9CC9BEB7-9D24-47C7-8F9D-CCC9DCAC29EB}
Provider Guid: {9CC9BEB7-9D24-47C7-8F9D-CCC9DCAC29EB}
Level: 4
```

[מטה נוכל לראות חלק מה-Providers אשר מדווחים ל-Trace Session ואת הגדרות ה-Filter שלהם]



כפי שאמרתי קודם לכן, קיימים אלפי Providers במערכת ההפעלה (מבלי להזכיר את אלה שמגיעים עם התקנת Third Party Apps). לכן לרוב נרצה לראות רשימה של Providers, ומתוכה למצוא את האחד שיספק לנו את המידע הרלוונטי. בעזרת הפקודה הבאה, נציג את כל ה-Providers הקיימים במערכת ההפעלה:

```
PS C:\Users\Printac> logman query providers
```

Provider	GUID
.NET Common Language Runtime	{E13C0D23-CCBC-4E12-931B-D9CC2EEE27E4}
ACPI Driver Trace Provider	{DAB01D4D-2D48-477D-B1C3-DAAD0CE6F06B}
Active Directory Domain Services: SAM	{8E598056-8993-11D2-819E-0000F875A064}
Active Directory: Kerberos Client	{BBA3ADD2-C229-4CDB-AE2B-57EB6966B0C4}
Active Directory: NetLogon	{F33959B4-DBEC-11D2-895B-00C04F79AB69}
ADODB.1	{04C8A86F-3369-12F8-4769-24E484A9E725}
ADOMD.1	{7EA56435-3F2F-3F63-A829-F0B35B5CAD41}
Application Popup	{47BFA2B7-BD54-4FAC-B70B-29021084CA8F}
Application-Addon-Event-Provider	{A83FA99F-C356-4DED-9FD6-5A5EB8546D68}
ASP.NET Events	{AFF081FE-0247-4275-9C4E-021F3DC1DA35}
ATA Port Driver Tracing Provider	{D08BD885-501E-489A-BAC6-B7D24BFE6BBF}
AuthFw NetShell Plugin	{935F4AE6-845D-41C6-97FA-380DAD429B72}
BCP.1	{24722B88-DF97-4FF6-E395-DB533AC42A1E}
BFE Trace Provider	{106B464A-8043-46B1-8CB8-E92A0CD7A560}
BITS Service Trace	{4A8AAA94-CFC4-46A7-8E4E-17BC45608F0A}
Certificate Services Client CredentialRoaming Trace	{EF4109DC-68FC-45AF-B329-CA28254372}
Certificate Services Client Trace	{F01B7774-7ED7-401E-8088-B576793D7841}
Circular Kernel Session Provider	{54DEA73A-ED1F-42A4-AF71-3E63D056F174}
Classpnp Driver Tracing Provider	{FA8DE7C4-ACDE-4443-9994-C4E2359A9EDB}
Critical Section Trace Provider	{3AC66736-CC59-4CFF-8115-8DF50E39816B}

בדוגמא הבאה נבצע תשאול ל-Provider ספציפי ונראה אילו Keywords מעניינים ה-Manifest שלו מציג, שמו הוא "Security: NTLM Authentication". אפשר להבין לבד מה הוא מעניק ל-Consumer:

```
PS C:\Users\test> logman query providers "Security: NTLM Authentication"
```

Provider	GUID
Security: NTLM Authentication	{58BB6C18-AA45-49B1-A15F-085F7ED0AA90}

Value	Keyword	Description
0x0000000000000001	Error	Error Flag
0x0000000000000002	Warning	Warning Flag
0x0000000000000004	Init	Init Flag
0x0000000000000008	Misc	Misc Flag
0x0000000000000010	LogonSess	Logon Session Flag
0x0000000000000020	Leak	Leak Track Flag
0x0000000000000040	Lpc	Lpc Flag
0x0000000000000080	LpcMore	Lpc More Flag
0x0000000000000100	Api	Api Flag
0x0000000000000200	ApiMore	Api More Flag
0x0000000000000400	SKey	Session Keys Flag
0x0000000000000800	Nego	Negotiate Flag
0x0000000000001000	Updates	Updates Flag
0x0000000000002000	NtLmV2	NTLM V2 Flag
0x0000000000004000	Cred	Credentials Flag
0x0000000000008000	Version	Protocol Version Flag
0x0000000000010000	Target	Target Flag

PID	Image
0x00002090	C:\Windows\System32\svchost.exe
0x00002124	C:\Windows\System32\svchost.exe
0x000002a4	C:\Windows\System32\lsass.exe



קודם לכן, כשדיברנו על מבנה ה-Manifest, הזכרתי את wevtutil, בעזרתו אפשר בדרך פשוטה להתקין Manifest-Based Providers. הבינארי wevtutil נותן למשתמש יכולת לרשום, לשנות, לתשאל ולמחוק Channels-ו Manifests. כך נוכל לראות את כל ה-Manifests הרשומים:

```
PS C:\Users\test> wevtutil.exe el
AMSI/Debug
Analytic
Application
DirectShowFilterGraph
DirectShowPluginControl
Els_Hyphenation/Analytic
EndpointMapper
FirstUXPerf-Analytic
ForwardedEvents
HardwareEvents
```

[קיימים יותר מדי Manifests מותקנים/LogFiles ואין צורך אמיתי להציג את כולם]

גם כאן קיימת אפשרות לבקשה ספציפית יותר אשר תספק לנו מידע על סוג ה-Channel והרשאותיו, מי ה-Provider שלו, היכן נשמר קובץ הרישומים על הדיסק וסביבת ההפרדה שלו:

```
PS C:\Users\test\Desktop> wevtutil gl Microsoft-Windows-PowerShell/Operational
name: Microsoft-Windows-PowerShell/Operational
enabled: true
type: Operational
owningPublisher: Microsoft-Windows-PowerShell
isolation: Application
channelAccess: 0:BAG:SYD:(A;;0x2;;;S-1-15-2-1)(A;;0x2;;;S-1-15-3-1024-3153509613-960666767-37246
227-1950414635-4190290187)(A;;0xf0007;;;SY)(A;;0x7;;;BA)(A;;0x7;;;SO)(A;;0x3;;;IU)(A;;0x3;;;SU)
-33)(A;;0x1;;;S-1-5-32-573)
logging:
  logfileName: %SystemRoot%\System32\Winevt\Logs\Microsoft-Windows-PowerShell%4Operational.evtx
  retention: false
  autoBackup: false
  maxSize: 15728640
publishing:
  fileMax: 1
```

התכונה isolation אחראית על גישת הכתיבה על Trace Session. קיימות 2 גישות סטנדרטיות Application ו-System, וכל אחת מהן מכילות Security Descriptor ידוע מראש. אך בשביל לבצע הרשאות כתיבה משלנו נצטרך להשתמש בגישת Custom isolation. תכונת ה-channelAccess כתובה כ-SDDL [כאן](#), String, המספקת לנו מידע על הרשאות קריאה על ה-Trace Session. אם וכאשר ניצור Custom isolation תכונת ה-channelAccess תתייחס גם כן להרשאות כתיבה.

ETW ירש את אותו מודל האבטחה מ-WMI, דוקומנטציה ממיקרוסופט חסרה לכן נשלים על כך מהמחקרים של Geoff Chappell [פה](#).

בסופו של דבר, הדרך הפשוטה שדיברתי עליה קודם לכן היא "wevtutil im <path to .man file>", כך נוכל לרשום manifest ובעצם לתת ל-wevtutil לעשות את כל עבודת ההתקנה בשבילנו.

## מחבואים ב-CLR

לפני שנתחיל, אני ממליץ בחום לקרוא את [המאמר המעולה](#) של ליאור פתיחה בגיליון 147 שיחוסך ממני להכיר בפניכם את המונחים הבסיסיים של .NET.

בתקופה האחרונה כלים רבים התחילו להשתמש במונח Patch ETW, אבל מה זה בעצם? הכל התחיל בעקבות מאמרו של אדם צ'סטר (XPN), המציג איך אפשר למנוע מ-Process לדווח על רישומים ל-Provider של .NET. ולחסל את יכולותיו ב-User Mode.

Reflection.Assembly - טכניקה המעניקה לנו יכולת לטעון קוד C# מקומפל לזיכרון ה-Process, לא משנה אם הוא נמצא על הדיסק או מערך של בתים. על מנת להשתמש בשיטה זו, נצטרך לחיות ב-Process שקיים בו CLR או לטעון אותו בעזרת קוד ל-Process:

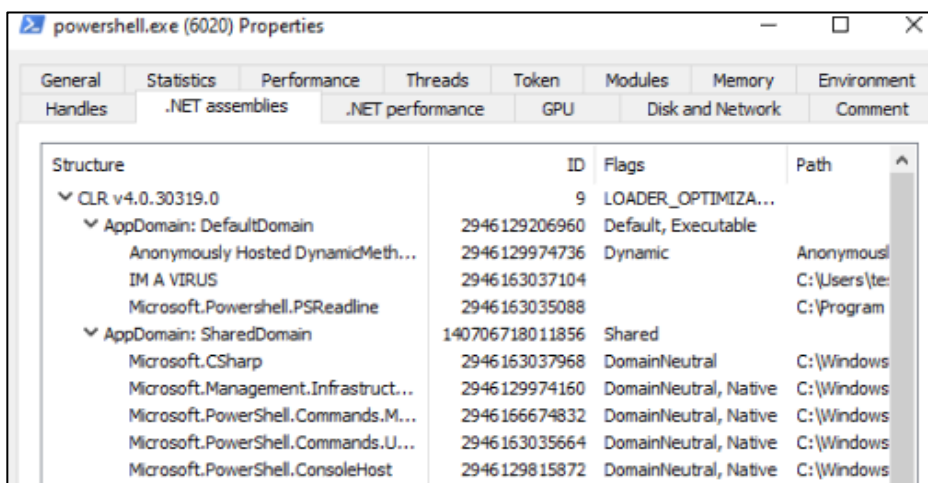
```
Administrator: Windows Power
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\test> [System.Reflection.Assembly]::LoadFile("C:\Users\test\Desktop\IM A VIRUS.exe")

GAC    Version      Location
----    -
False  v4.0.30319   C:\Users\test\Desktop\IM A VIRUS.exe
```

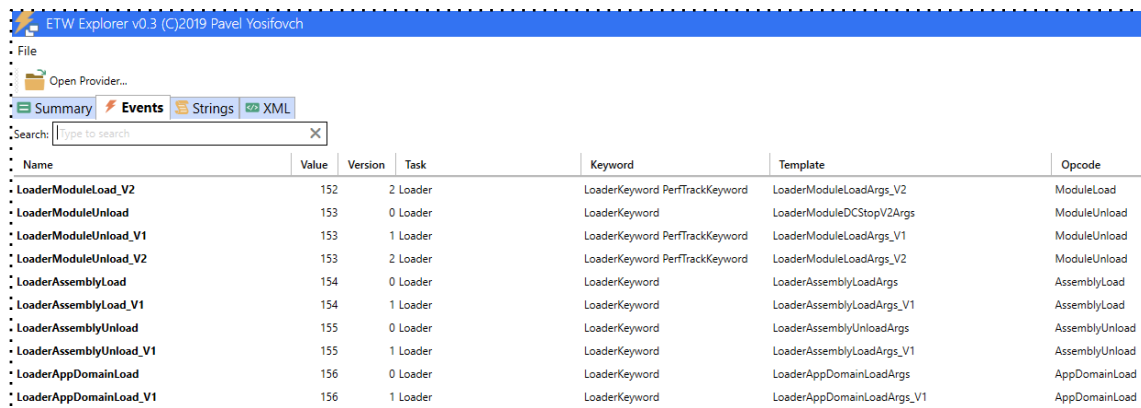
הפונקציה LoadFile, מתפקדת כ-Loader - מקבלת .NET שנמצא על הדיסק וטוענת אותו. בעזרת Process Hacker, נוכל לראות את ה-Assemblies הטעונים אל Process, ולמזלנו כל ה-Source Code שלו נמצא ב-Github. אז כן... הוא משתמש ב-ETW כדי לאסוף את הנתונים הללו. למטרת ההדגמה הוא יתפקד כ-Consumer לכל דבר:



הכלי הזדוני שלנו אכן נטען ו-Microsoft-Windows-DotNETRuntime-דאג לדווח על כך לכל מי שרוצה.

הגיע הזמן לקפוץ לבריכה העמוקה יותר, נשתמש בEtwExplorer על מנת לחקור את ה-Manifest של .NET. ולחקור אירועים מעניינים.

אחרי חיפוש קצר אפשר לשים לב לכמה רישומים שכאשר מוזרמים למקום הנכון יכולים להוות IOC רציני:



Name	Value	Version	Task	Keyword	Template	Opcode
LoaderModuleLoad_V2	152	2	Loader	LoaderKeyword PerfTrackKeyword	LoaderModuleLoadArgs_V2	ModuleLoad
LoaderModuleUnload	153	0	Loader	LoaderKeyword	LoaderModuleDCStopV2Args	ModuleUnload
LoaderModuleUnload_V1	153	1	Loader	LoaderKeyword PerfTrackKeyword	LoaderModuleLoadArgs_V1	ModuleUnload
LoaderModuleUnload_V2	153	2	Loader	LoaderKeyword PerfTrackKeyword	LoaderModuleLoadArgs_V2	ModuleUnload
LoaderAssemblyLoad	154	0	Loader	LoaderKeyword	LoaderAssemblyLoadArgs	AssemblyLoad
LoaderAssemblyLoad_V1	154	1	Loader	LoaderKeyword	LoaderAssemblyLoadArgs_V1	AssemblyLoad
LoaderAssemblyUnload	155	0	Loader	LoaderKeyword	LoaderAssemblyUnloadArgs	AssemblyUnload
LoaderAssemblyUnload_V1	155	1	Loader	LoaderKeyword	LoaderAssemblyLoadArgs_V1	AssemblyUnload
LoaderAppDomainLoad	156	0	Loader	LoaderKeyword	LoaderAppDomainLoadArgs	AppDomainLoad
LoaderAppDomainLoad_V1	156	1	Loader	LoaderKeyword	LoaderAppDomainLoadArgs_V1	AppDomainLoad

המחקר של XPN, הציג את ההתממשקות של clr.dll עם ETW ואיך הזרמת הרישומים עובדת אצלו. במהלך המחקר הגיע אל הפונקציה EtwEventWrite, המיוחצנת על ידי ntdll, ומשמשת Providers לכתובת רישומים ל-Trace Session.

הפונקציה מקבלת את הארגומנטים הבאים:

- **RegHandle** - מפונקציות הרישום Handle רישום ל-Provider
- **EventDescriptor** - מבנה יבש של הרישום
- **UserData** ו-**UserDataCount** - כמות הארגומנטים והמידע הכתוב ברישום בהתאם למבנה

```

C++
Copy

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

אחרי כמה שנים של ערפל, מיקרוסופט דאגו לספר לנו שהפונקציה פנימית למערכת ההפעלה ונקראת מתוך [EventWrite](#) הנמצאת ב-Advapi32.dll.

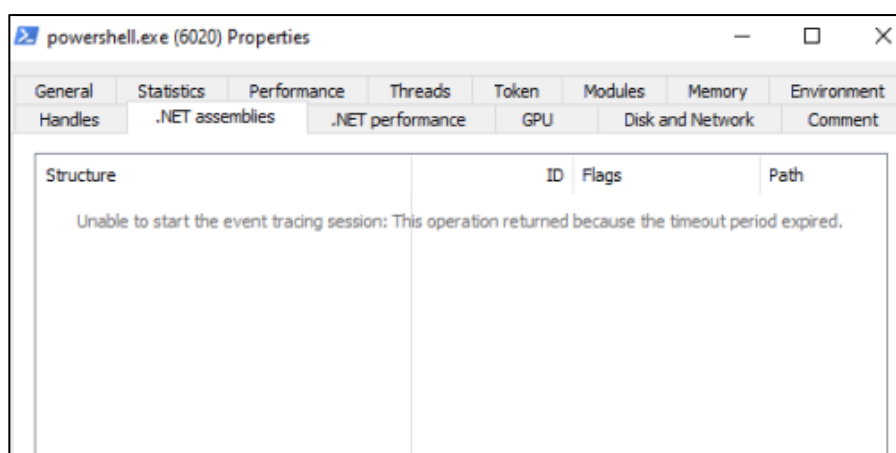
על מנת למנוע הזרמת רישומים, מבצעים Patch לפרולוג פונקציית EtwEventWrite. ה-Patch יכתוב בתחילת הפונקציה את אותם הבתים, הפונקציה תרוץ ובסופה ניקוי המחסנית יתבצע כשורה. השינוי מתבצע ב-ntdll הטעון לזיכרון שלו.



לצורך ההדגמה, כתבתי פרויקט קטן שמבצע את המשימה שלנו בצורה די בסיסית, הפונקציה המרכזית בו מקבלת Handle ל-Process הקורבן, מחפשת את כתובת EtwEventWrite ולאחר מכן מבצעת Patch עם ארבעת הבתים הללו כ-Inline Assembly (מתאים לסביבת x64).

```
int MuteEvents(HANDLE hProc) {  
    unsigned char EtwEventWriteArray[] = { 'E', 't', 'w', 'E', 'v', 'e', 'n', 't', 'W', 'r', 'i', 't', 'e', 0x0 };  
    void *EtwEventWrite = GetProcAddress(GetModuleHandle((LPCSTR)ntdll), (LPCSTR)EtwEventWriteArray);  
    // xor rax, rax; ret  
    char patch[] = "\x48\x33\xc0\xc3";  
  
    WriteProcessMemory(hProc, EtwEventWrite, (PVOID) patch, (SIZE_T) sizeof(patch) - 1, (SIZE_T *) NULL);  
    return 0;  
}
```

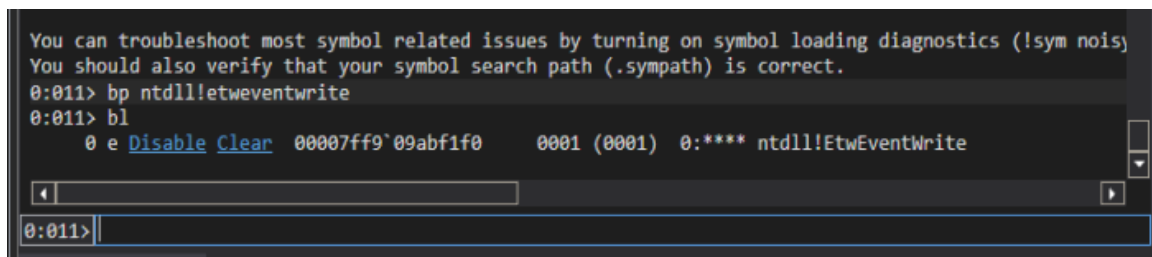
נבדוק מה המצב לאחר הרצה:

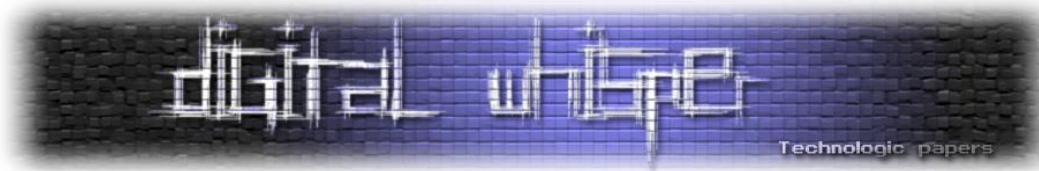


כמו שאפשר לראות, ה-Consumer שלנו - Process Hacker, אכן מאבד ראייה על ה-Process.

קיימות כמה שיטות לבצע את ה-Patch ולא בהכרח בפונקציה הזאת, אך קיימת גם האופציה לפלטר רישומים ולבצע מניעה בצורה כירורגית ובעצם להזרים חלק מהרישומים, נוכל להזרים אך ורק רישום ש-ה-EventID שלו לדוגמא הוא 155 או כל דבר שקיים ב-UserData כמו שם ה-Assembly.

לצורך המחשה נפתח WinDbg ונשים BreakPoint על הפונקציה:





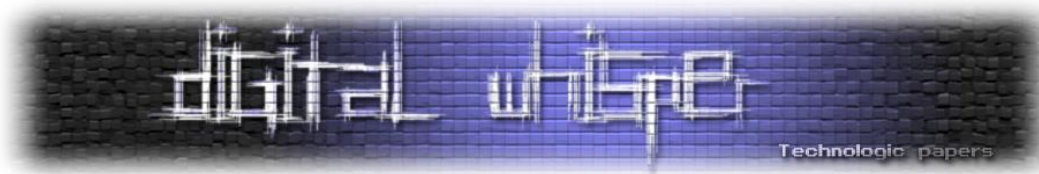
במקום לחכות עד שיגיע ה-Breakpoint, נוכל לפתוח Process Hacker ולהטריג את הדיווח, נראה שאכן הגענו לפונקציה:

```
0:011> g
Breakpoint 0 hit
ntdll!EtwEventWrite:
00007ff9`09abf1f0 4c8bdc      mov     r11, rsp
0:009>
Disassembly
Address: @$scope!p
Follow current instruction
-----
00007ff9`09abf1ec cc          int    3
00007ff9`09abf1ed cc          int    3
00007ff9`09abf1ee cc          int    3
00007ff9`09abf1ef cc          int    3
ntdll!EtwEventWrite:
00007ff9`09abf1f0 4c8bdc      mov     r11, rsp
00007ff9`09abf1f3 4883ec58    sub     rsp, 58h
00007ff9`09abf1f7 4d894be8    mov     qword ptr [r11-18h], r9
00007ff9`09abf1fb 33c0       xor     eax, eax
00007ff9`09abf1fd 458943e0    mov     dword ptr [r11-20h], r8d
00007ff9`09abf201 4533c9     xor     r9d, r9d
00007ff9`09abf204 498943d8    mov     qword ptr [r11-28h], rax
00007ff9`09abf208 4533c0     xor     r8d, r8d
00007ff9`09abf20b 498943d0    mov     qword ptr [r11-30h], rax
00007ff9`09abf20f 6689442420  word ptr [rsp+20h], ax
00007ff9`09abf214 e85f000000 call    ntdll!EtwpEventWriteFull (7ff909abf278)
00007ff9`09abf219 4883c458    add     rsp, 58h
00007ff9`09abf21d c3         ret
00007ff9`09abf21e cc          int    3
00007ff9`09abf21f cc          int    3
```

**הערה:** קובנציית הקריאה לפונקציות ב-64x64 בשונה מ-86x8, מתבצעת על ידי העברת ארבעת הפרמטרים הראשונים באוגרים ורק לאחר מכן במחסנית (מהפרמטר השמאלי ביותר לימני). מה שאומר שבשביל לראות את הפרמטרים שלנו נצטרך להסתכל על האוגרים R8, RDX, RCX ו-R9.

הפרמטר השני הוא זה שיעניין אותנו, והוא ה-EventDescriptor ובתוכו קיים Id שיבדיל לנו בין כל רישום:

```
C++ Copy
typedef struct _EVENT_DESCRIPTOR {
    USHORT    Id;
    UCHAR     Version;
    UCHAR     Channel;
    UCHAR     Level;
    UCHAR     Opcode;
    USHORT    Task;
    ULONGLONG Keyword;
} EVENT_DESCRIPTOR, *PEVENT_DESCRIPTOR;
```



כמו שאמרתי הוא עובר לפונקציה כפרמטר השני ולכן נסתכל על האוגר RDX בייצוג ה-Struct המתאים:

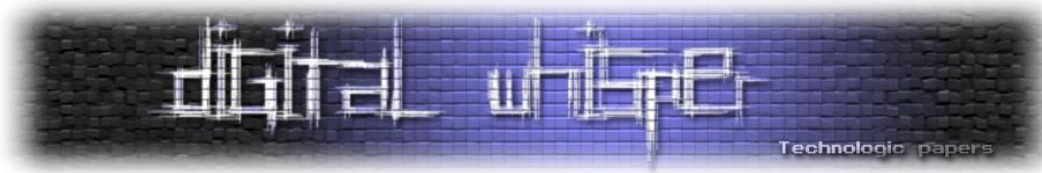
```
ntdll!EtwEventWrite:
00007ff9`09abf1f0 4c8bdc      mov     r11,rsp
0:009> dt _EVENT_DESCRIPTOR [rdx]
clrjit!_EVENT_DESCRIPTOR
+0x000 Id      : 0x9b
+0x002 Version : 0x1 ''
+0x003 Channel : 0 ''
+0x004 Level   : 0x4 ''
+0x005 Opcode  : 0x27 '''
+0x006 Task    : 2
+0x008 Keyword : 8
```

נבצע המרה מ-Hex ל-Decimal ונקבל 155, לאחר מכן נחזור ל-EtwExplorer ונראה שהוא באמת רישום תקף ב-Manifest:

LoaderAssemblyLoad	154	0 Loader
LoaderAssemblyLoad_V1	154	1 Loader
LoaderAssemblyUnload	155	0 Loader
LoaderAssemblyUnload_V1	155	1 Loader
LoaderAppDomainLoad	156	0 Loader
LoaderAppDomainLoad_V1	156	1 Loader
LoaderAppDomainUnload	157	0 Loader

על בסיס זה, נוכל ליצור Dll, שמבצע Hook לפונקציית EtwEventWrite ב-ntdll, מאותו הרגע הפונקציה תקבל פרמטרים ותקפוז אל אימפלמנטציה משלנו, ב-DLL שלנו נוכל לכתוב Switch Case-ים, שמוודאים על בסיס ה-Id ב-EventDescriptor איזה רישום להציג ואיזה לא וכך נוכל לבצע סינון על כל רישום זדוני שנרצה להסתיר.





## למה וכמה Threat-Intelligence Provider עוזר להגן

כמו שראינו קודם, קיימים כמות נכבדת של Providers, כמה מעניינים יכולים להיות:

- Windows Kernel Trace
- Microsoft-Windows-Kernel-Process
- Microsoft-Windows-DNS-Client

צר לי לבאס אך ה-Provider המעניין ביותר הוא לא אחד מאלה ולא לשם כך התכנסנו. ביום בהיר אחד ב-2005, מיקרוסופט הוסיפה פיצ'ר לגרסאות x64 של ווינדוס XP. הנקרא PatchGuard, מנגנון הנועד למנוע שינויים למבנים בקרנל, כגון עריכה של SSDT. פיצ'ר זה גרם לבלאגן גדול אצל חברות המפתחות אנטי-וירוסים אשר הסתמכו על Kernel Hooks וכתוצאה מכך גרם להם לעיוורון ב-Kernel.

כיום רוב יכולות ה-AV\EDR מבוססות על User-Mode Api Hooking, Minifilter Driver ו-Kernel Callbacks. עם כל היתרונות הללו מגיעים לא מעט חסרונות, מיקרוסופט מאפשרת לדרייברים לעקוב ולקבל התראות על כמה אירועים קריטיים בעזרת Kernel Callbacks.

איתו קיימות 3 בעיות עיקריות, הראשונה היא הכמות המוגבלת של דרייברים שיכולים להירשם, השנייה היא היכולות המוגבלות לפעולות הנוגעות לזיכרון והשלישית היא ה-Race Condition בפעולות כגון: חסימת קבצים. User-Mode Hooks היא הדרך היחידה לראות פעולות זיכרון ושימוש ב-WinApi בזמן ריצה, על ידי שינוי הפרולוג של הפונקציה וביצוע Hook (קפיצה ל-DLL של ה-EDR המוזרק לכל Process וכך יכול לבחון את הפרמטרים ולאבחן כל קריאה לפונקציה). כמובן שאפשר לעקוף דרך זאת די בקלות - בעזרת Unhooking או Direct Syscalls. אך למרות כל הפתרונות הללו, אנו נשארים עיוורים לכלל הפעולות ב-Kernel.

באוקטובר 2018, יצאה לעולם גרסת Windows 10 Build 1809 ואיתה הגיע Kernel Provider שבא לטפל בדיוק בבעיה הזאת, שמו הוא "Microsoft-Windows-Threat-Intelligence". ה-Provider מעניק יותר מדי מידע על כל אירוע - עד כדי פירוט על מצבי דפים בזיכרון, אך היופי האמיתי הוא הזמינות שלו אך ורק לתהליכים הרצים כ-Anti Malware Protected-Process-Light או בקצרה PPL-Anti Malware.

**הערה:** נוכל לפתוח WinDbg, ב-3 Offset של ה-PEB, קיים שדה סיביות ובו יתמקמו להם 8 Flags כביטים, אם וכאשר יופיע הערך 0x31 ה-Process אכן AntiMalware-PPL.

נקודת התורפה של כל הסיפור הזה הוא ה-User Mode סרוויס של ה-EDR וזה בדיוק הפתרון שבבילה, חשוב להגיד שלא כל אחד יכול לעשות מה שבא לו (למרות הכל אנחנו מדברים פה על Windows) - וכדי שהוא ירוץ כ-AntiMalware PPL, חייב [ELAM](#) דרייבר מותקן איתו שמכיל את החשוב מכל, חתימות ☺.

הצורך ב-PPL הכרחי. אם EDR\AV לא ירוץ כ-PPL כל ברנש עם הרשאות ירגיש חזק מספיק לשחק עם ה-Process המגן שלו. אני לא בא ואומר ש-PPL פותר הכל. הוא פשוט יעלה את דרגת הקושי במשחק הזה. אחרי שהבנו למה מיקרוסופט ייעדו את ETI במויחד למוצרי הגנה המוכרים למיקרוסופט וחתומים על ידם, בואו נעביר מבט עליו.

המעקף שראינו במחקר של חקא יעזור לנו מול Providers רבים, רק כאשר הם רצים ב-User Mode. אך מול ETI זה כבר סיפור אחר עקב ריצתו ב-Kernel.

יכולות ה-Provider מעניקות למאזין יכולות מעקב אחר כל פונקציה הנמצאת בשימוש תדיר בקרב מפתחי נזקות ולא רק מול נזקות User-Mode אלא גם מול קריאות של הפונקציות הללו מהקרנל כמו ב-DoublePulsar:

```
PS C:\Users\test> logman query providers "Microsoft-Windows-Threat-Intelligence"
Provider                                     GUID
-----
Microsoft-Windows-Threat-Intelligence      {F4E1897C-BB5D-5668-F1D8-040F4D8DD344}

Value      Keyword      Description
-----
0x0000000000000001  KERNEL_THREATINT_KEYWORD_ALLOCMV_LOCAL
0x0000000000000002  KERNEL_THREATINT_KEYWORD_ALLOCMV_LOCAL_KERNEL_CALLER
0x0000000000000004  KERNEL_THREATINT_KEYWORD_ALLOCMV_REMOTE
0x0000000000000008  KERNEL_THREATINT_KEYWORD_ALLOCMV_REMOTE_KERNEL_CALLER
0x0000000000000010  KERNEL_THREATINT_KEYWORD_PROTECTVM_LOCAL
0x0000000000000020  KERNEL_THREATINT_KEYWORD_PROTECTVM_LOCAL_KERNEL_CALLER
0x0000000000000040  KERNEL_THREATINT_KEYWORD_PROTECTVM_REMOTE
0x0000000000000080  KERNEL_THREATINT_KEYWORD_PROTECTVM_REMOTE_KERNEL_CALLER
0x0000000000000100  KERNEL_THREATINT_KEYWORD_MAPVIEW_LOCAL
0x0000000000000200  KERNEL_THREATINT_KEYWORD_MAPVIEW_LOCAL_KERNEL_CALLER
0x0000000000000400  KERNEL_THREATINT_KEYWORD_MAPVIEW_REMOTE
0x0000000000000800  KERNEL_THREATINT_KEYWORD_MAPVIEW_REMOTE_KERNEL_CALLER
0x0000000000001000  KERNEL_THREATINT_KEYWORD_QUEUEUSERAPC_REMOTE
0x0000000000002000  KERNEL_THREATINT_KEYWORD_QUEUEUSERAPC_REMOTE_KERNEL_CALLER
0x0000000000004000  KERNEL_THREATINT_KEYWORD_SETTHREADCONTEXT_REMOTE
0x0000000000008000  KERNEL_THREATINT_KEYWORD_SETTHREADCONTEXT_REMOTE_KERNEL_CALLER
0x0000000000010000  KERNEL_THREATINT_KEYWORD_READVM_LOCAL
0x0000000000020000  KERNEL_THREATINT_KEYWORD_READVM_REMOTE
0x0000000000040000  KERNEL_THREATINT_KEYWORD_WRITEVM_LOCAL
0x0000000000080000  KERNEL_THREATINT_KEYWORD_WRITEVM_REMOTE
0x0000000001000000  KERNEL_THREATINT_KEYWORD_SUSPEND_THREAD
0x0000000002000000  KERNEL_THREATINT_KEYWORD_RESUME_THREAD
0x0000000004000000  KERNEL_THREATINT_KEYWORD_SUSPEND_PROCESS
0x0000000008000000  KERNEL_THREATINT_KEYWORD_RESUME_PROCESS
0x0000000010000000  KERNEL_THREATINT_KEYWORD_FREEZE_PROCESS
0x0000000020000000  KERNEL_THREATINT_KEYWORD_THAW_PROCESS
0x00000000040000000  KERNEL_THREATINT_KEYWORD_CONTEXT_PARSE
0x00000000080000000  KERNEL_THREATINT_KEYWORD_EXECUTION_ADDRESS_VAD_PROBE
0x00000000100000000  KERNEL_THREATINT_KEYWORD_EXECUTION_ADDRESS_MMFL_NAME_PROBE
0x00000000200000000  KERNEL_THREATINT_KEYWORD_READWRITEVM_NO_SIGNATURE_RESTRICTION
0x00000000400000000  KERNEL_THREATINT_KEYWORD_DRIVER_EVENTS
0x00000000800000000  KERNEL_THREATINT_KEYWORD_DEVICE_EVENTS
0x00000000000000000  Microsoft-Windows-Threat-Intelligence/Analytic

Value      Level      Description
-----
0x004      win:Informational  Information

PID      Image
-----
0x00000000
```

**הערה:** Microsoft-Windows-Security-Auditing הוא ה-Provider האחראי על הזרמת רישומי ה-Security שאנו מכירים מ-Event Viewer. רק Trace Session אחד רשאי להשתמש בו במקביל, והיחיד שמשתמש בו היום הוא EventLog-Security. בדומה ל-ETI, גם הוא מצריך ELAM דרייבר לפני שנרשמים אליו. לצערי לא ארחיב על כך במסגרת מאמר זה אך למרחיבי עניין, בהינתן הרשאות System, קיימת דרך להאזין לו במקביל ולקבל ממנו רישומים ☺



לסיכום, בעזרת ETI, מיקרוסופט הוסיפה יכולות שלא רק ביססו את ראיית ה-EDR אלא העצימו אותו והביאו אותו לפיק יכולות זיהוי וגילוי מרהיבות. עכשיו כשאנחנו מכירים, אפשר להבין למה הוא כל כך מועיל וקריטי למוצרי הגנה.

### Keylogger with ETW

בעזרת "Microsoft-Windows-USB-USBPORT" נוכל לקבל מידע על כל פעולה המתבצעת מה-USB ולפרסר אותה עד רמת האותיות הנשלחות:

```
PS C:\Users\test> logman query providers "Microsoft-Windows-USB-USBPORT"
```

Provider	GUID
Microsoft-Windows-USB-USBPORT	{C88A4EF5-D048-4013-9408-E04B7DB2814A}

Value	Keyword	Description
0x0000000000000001	Diagnostic	
0x0000000000000002	PowerDiagnostics	
0x0000000000000004	PerfDiagnostics	
0x0000100000000000	ms:ReservedKeyword44	
0x0000200000000000	ms:Telemetry	
0x0000400000000000	ms:Measures	
0x0000800000000000	ms:CriticalData	
0x8000000000000000	Microsoft-Windows-USB-USBPORT/Diagnostic	

Value	Level	Description
0x04	win:Informational	Information
0x05	win:Verbose	Verbose

### WinShark

[Winshark](#) הינו פלאגין ETW ל-Wireshark אשר עזר לא מעט, ונוצר במיוחד בשביל זה. בעזרת libpcap מעניק ל-Wireshark התממשקות עם יכולות Consumer. יכולות כמו פלטור על בסיס Process או תקשורת בין Pipes, מעניקה יכולות חקירה מדהימות ועוזר בכל עולם החקירה ב-Windows.

### SlikETW

[SlikETW](#) הוא Wrapper שנכתב ב-C# נוח לשימוש הנועד לתת ממשק נוח למטרות מחקר. בעזרתו אפשר די בקלות ליצור Consumer בלי שום הסתבכות וסיבובים ב-MSDN. בנוסף קיימות לו יכולות הזרמת המידע לשרתים צד שלישי מרוחקים ואפילו אינטגרציה עם חוקי YARA והתראה על בסיסם.

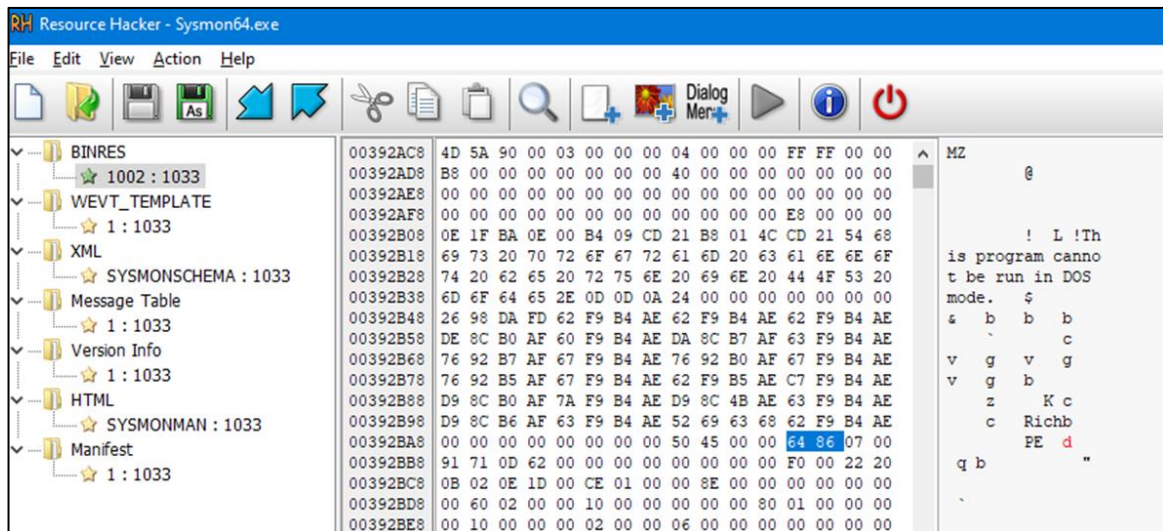
[מאמר מעולה לצייד בסביבת AD עם SlikETW](#)



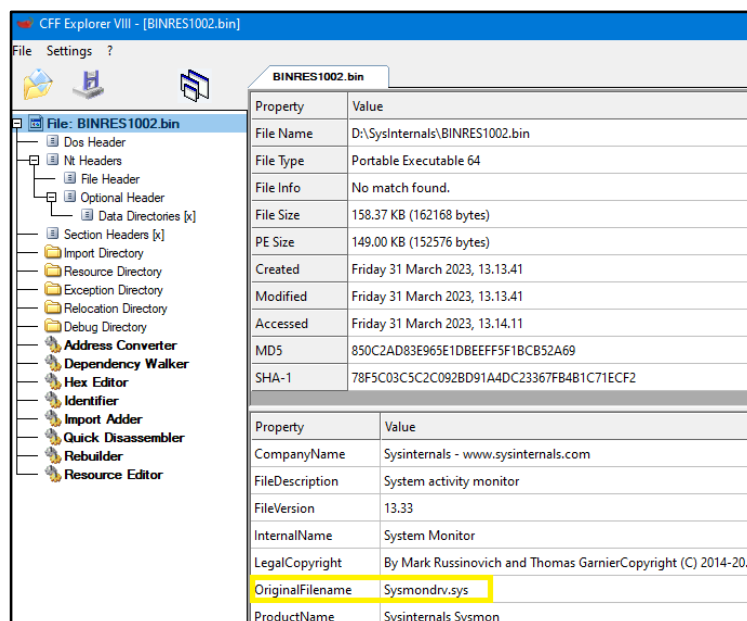
## Sysmon Internals

Sysmon לרוב משחק תפקיד משמעותי אצל צוותי הגנה רבים, מה שמסב את תשומת הלב של שחקנים רבים בעולם הסייבר ההתקפי. לכן רציתי לחקור אותו, ולהבין מה הוא עושה under the hood (את גרסת ה-64x שלו).

נתחיל מההתחלה, נפתח אותו ב-Resource Hacker ונראה שהוא מכיל לא מעט Resource-ים:



בעזרת ה-Magic נוכל לראות שה-Resource הראשון אכן PE כמצופה מ-resource שנמצא בתיקית BINRES, נמשיך קצת הלאה ל-FileHeader על מנת למצוא את Machine Field, המעיד על ארכיטקטורת ההרצה. בשביל לעבוד נוח יותר, נייצא את ה-Resource לקובץ על הדיסק ונזרוק אותו ל-CFF. לאחר שנייבא אותו ל-CFF נוכל לראות בבירור, שה-Resource הוא אכן ה-Driver של Sysmon שנטען בזמן התקנה:



עוקבים אחרי ETW

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

יכולות ה-Driver של Sysmon דומות לכל Driver של מוצר הגנה אחר, בעזרתו הוא נרשם ל-Kernel Callbacks וכ- Minifilter וכך מקבל עדכונים מה-Kernel. עקב חוסר היכולת לחיות בקרנל בזכות PatchGuard, מיקרוסופט הבינה שעשתה קצת בלאגן לעולם ההגנה והציגה את Kernel Callbacks, אשר לא מבצעים שינוי בזיכרון ה-Kernel ובטוחים לשימוש:

```
PsSetCreateProcessNotifyRoutineEx()
PsSetCreateThreadNotifyRoutine()
PsSetLoadImageNotifyRoutine()
```

הדרייבר נרשם כ-Minifilter - מה שמעניק לו יכולות האזנה על מערכת הקבצים, ונרשם ל-Callbacks לפונקציות שמנטרות פעולות מסוימות לפני שהן מתבצעות וגם אחרי.

Sysmon נרשם ל-Callbacks (ממש בעזרת הפונקציות שבתמונה) כאלה המעניקים לו התראה על יצירת Process, יצירת Thread, טעינה לזיכרון ועוד. על מנת לאסוף מידע תקשורתי Sysmon יוצר Trace Session עם Windows Kernel Trace כאשר רק net keyword קיים וככה אוסף את כל המידע התקשורתי שלו.

נחזור ל-Resource Hacker, בתיקיית WEVT\_TEMPLATE, נמצא טמפלייט של מבנה ה-EVTX שלו. בתיקיית XML, קיים Resource הנקרא SysmonSchema, וכמו שהוא נשמע הוא הסכמה הדיפולטית המגיעה עם Sysmon כאשר לא מביאים לו סכמת חוקים בתהליך ההתקנה הראשוני. Version Info ו-Manifest מספקים מידע על הגרסה שלו ובעיקר metadata. בתיקיית HTML קיים Resource, הנקרא sysmonman אשר נראה כמו קובץ XML (כמו שאמרנו קודם, קבצים המסתיימים ב-man. הם קבצי Instrumentation Manifest על בסיס קובציית מיקרוסופט).

כאשר Sysmon מותקן על עמדה הוא מבצע כמה פעולות קריטיות על מנת להתחיל את העבודה השוטפת שלו, לשם כך נעבור לחקירה דינאמית בזמן התקנה.

בשלב הראשון מתבצעת קריאת CreateFileW של Sysmon אל-C:\Windows (המעתיקה את ה-Exe), נשים bp על CreateProcessW. ה-bp הראשון שנעצור בו הוא יצירת Process חדש של Sysmon, הפרמטר השני ב-CreateProcessW מכיל את הארגומנטים והוא זה שמעניין אותנו לכן נסתכל על הפרמטר השני שמועבר דרך RDX.





אז Sysmon יוצר Process חדש עם ה-EXE בנתיב החדש עם הארגומנטים: -accept, -nologo, -m  
 נעביר מבט חוזר על ה-help של Sysmon:

```
Usage:
Install: Sysmon64.exe -i [<configfile>]
Update configuration: Sysmon64.exe -c [<configfile>]
Install event manifest: Sysmon64.exe -m
```

ואכן נראה שהוא משתמש ב-Manifest, אבל אנחנו עדיין מחפשים איך נוצר ה-Trace Session שלו.  
 נשים bp נוסף, כעת על הפונקציה FindResource, כדי לראות מתי הוא מחפש את SYSMONMAN.  
 נסתכל על הפוינטר ב-RDX, והנה הגענו אליו:

```
00007FF6AB2AC640 74 00 2D 00 57 00 09 00 0E 00 04 00 0F 00 77 00 t.-.w.i.n.d.o.w.
00007FF6AB2AC650 73 00 2D 00 53 00 79 00 73 00 6D 00 6F 00 6E 00 s.-.S.y.s.m.o.n.
00007FF6AB2AC660 00 00 00 00 00 00 00 00 53 00 59 00 53 00 4D 00 .....S.Y.S.M.
00007FF6AB2AC670 4F 00 4E 00 4D 00 41 00 4E 00 00 00 00 00 00 00 O.N.M.A.N.....
00007FF6AB2AC680 46 00 69 00 6E 00 64 00 52 00 65 00 73 00 6F 00 F.i.n.d.R.e.s.o.
```

נשים bp על CreateFile בכדי לראות אם הוא כותב אותו לדיסק, וכן, קובץ Temp נכתב לדיסק לתקיית  
 Temp עם שם MAN מגונרט:

```
00000018732FBE40 43 00 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00 C.:.\U.s.e.r.s.
00000018732FBE50 5C 00 74 00 65 00 73 00 74 00 5C 00 41 00 70 00 \.t.e.s.t.\.A.p.
00000018732FBE60 70 00 44 00 61 00 74 00 61 00 5C 00 4C 00 6F 00 p.D.a.t.a.\.l.o.
00000018732FBE70 63 00 61 00 6C 00 5C 00 54 00 65 00 6D 00 70 00 c.a.l.\.T.e.m.p.
00000018732FBE80 5C 00 4D 00 41 00 4E 00 44 00 39 00 42 00 32 00 \.M.A.N.D.9.B.2.
00000018732FBE90 2E 00 74 00 6D 00 78 00 00 00 00 00 00 00 00 00 ..t.m.p.....
```

נמשיך הלאה ונראה שאנחנו עוצרים שוב ב-CreateProcessW, ניקח את הארגומנט מ-RDX, עם יצירת  
 Process חדש של wevtutil.exe תוך שימוש בארגומנט um (שנועדה לעשות Uninstall) וה-Resource  
 שנכתב לתקיית Temp:

```
00000018732FCC90 71 CD 2F 73 18 00 00 00 11 00 00 00 F9 7F 00 00 q?/s.....?...
00000018732FCCA0 22 00 43 00 3A 00 5C 00 57 00 69 00 6E 00 64 00 ".C.:.\W.i.n.d.
00000018732FCCB0 6F 00 77 00 73 00 5C 00 73 00 79 00 73 00 74 00 o.w.s.\.s.y.s.t.
00000018732FCCC0 65 00 6D 00 33 00 32 00 5C 00 77 00 65 00 76 00 e.m.3.2.\.w.e.v.
00000018732FCCD0 74 00 75 00 74 00 69 00 6C 00 2E 00 65 00 78 00 t.u.t.i.l...e.x.
00000018732FCCF0 65 00 22 00 20 00 75 00 6D 00 20 00 22 00 43 00 e.". .u.m. ".C.
00000018732FCCF0 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00 5C 00 :.\U.s.e.r.s.\.
00000018732FCD00 74 00 65 00 73 00 74 00 5C 00 41 00 70 00 70 00 t.e.s.t.\.A.p.p.
00000018732FCD10 44 00 61 00 74 00 61 00 5C 00 4C 00 6F 00 63 00 D.a.t.a.\.L.o.c.
00000018732FCD20 61 00 6C 00 5C 00 54 00 65 00 6D 00 70 00 5C 00 a.l.\.T.e.m.p.\.
00000018732FCD30 4D 00 41 00 4E 00 44 00 39 00 42 00 32 00 2E 00 M.A.N.D.9.B.2...
00000018732FCD40 74 00 6D 00 70 00 22 00 00 00 AA 09 F9 7F 00 00 t.m.p."...?..?..
00000018732FCD50 10 CE 2F 73 18 00 00 00 00 00 00 00 00 00 00 00 .?/s.....
```

ניתן לו להמשיך לרוץ ונעצור שוב ב-CreateProcessW, נסתכל על RDX והפעם בעזרת wevtutil im רושמים את ה-Manifest:

```

00000018732FE2B0 22 00 43 00 3A 00 5C 00 57 00 69 00 6E 00 64 00 .C.:.\.W.i.n.d.
00000018732FE2C0 6F 00 77 00 73 00 5C 00 73 00 79 00 73 00 74 00 o.w.s.\.s.y.s.t.
00000018732FE2D0 65 00 6D 00 33 00 32 00 5C 00 77 00 65 00 76 00 e.m.3.2.\.w.e.v.
00000018732FE2E0 74 00 75 00 74 00 69 00 6C 00 2E 00 65 00 78 00 t.u.t.i.l...e.x.
00000018732FE2F0 65 00 22 00 20 00 69 00 6D 00 20 00 22 00 43 00 e.i.m. .i.m. .C.
00000018732FE300 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00 5C 00 :.\.U.s.e.r.s.\.
00000018732FE310 74 00 65 00 73 00 74 00 5C 00 41 00 70 00 70 00 t.e.s.t.\.A.p.p.
00000018732FE320 44 00 61 00 74 00 61 00 5C 00 4C 00 6F 00 63 00 D.a.t.a.\.L.o.c.
00000018732FE330 61 00 6C 00 5C 00 54 00 65 00 6D 00 70 00 5C 00 a.l.\.T.e.m.p.\.
00000018732FE340 4D 00 41 00 4E 00 46 00 34 00 34 00 39 00 2E 00 M.A.N.F.4.4.9...
00000018732FE350 74 00 6D 00 70 00 22 00 00 00 AA 7E 6B 01 00 00 t.m.p....?~k...
00000018732FE360 78 86 AA 7E 6B 01 00 00 78 86 AA 7E 6B 01 00 00 n.?~k n.?~k
    
```

הבנו ש-Sysmon אכן יוצר Trace Session ומתקין את ה-Provider שלו בעזרת wevtutil.

```

Administrator: Windows Power
PS C:\Users\test> logman query "EventLog-Microsoft-Windows-Sysmon-Operational" -ets

Name:                EventLog-Microsoft-Windows-Sysmon-Operational
Status:               Running
Root Path:            %systemdrive%\PerfLogs\Admin
Segment:              Off
Schedules:            On
Segment Max Size:     100 MB

Name:                EventLog-Microsoft-Windows-Sysmon-Operational\EventLog-Microsoft-Windows-Sysmon-Operational
Type:                 Trace
Append:               Off
Circular:              Off
Overwrite:            Off
Buffer Size:          64
Buffers Lost:         0
Buffers Written:      487
Buffer Flush Timer:   1
Clock Type:           System
File Mode:             Real-time

Provider:
Name:                 Microsoft-Windows-Sysmon
Provider Guid:        {5770385F-C22A-43E0-BF4C-06F5698FFBD9}
Level:                 255
KeywordsAll:          0x8000000000000000 (Microsoft-Windows-Sysmon/Operational)
KeywordsAny:          0xffffffffffff (Microsoft-Windows-Sysmon/Operational, 0x1, 0x2, 0x4, 0x8, 0x10, 0x20, 0x40, 0x80,
0x100, 0x200, 0x400, 0x800, 0x1000, 0x2000, 0x4000, 0x8000, 0x10000, 0x20000, 0x40000, 0x80000, 0x100000, 0x200000, 0x400000, 0x800000,
0x1000000, 0x2000000, 0x4000000, 0x8000000, 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x100000000, 0x200000000, 0x400000000, 0x800000000,
0x1000000000, 0x2000000000, 0x4000000000, 0x8000000000, 0x10000000000, 0x20000000000, 0x40000000000, 0x80000000000, 0x100000000000, 0x200000000000, 0x400000000000, 0x800000000000, 0x1000000000000, 0x2000000000000, 0x4000000000000, 0x8000000000000)
Properties:            65
Filter Type:           0

The command completed successfully.
    
```

[תמונה המציגה את ה-Trace Session של Sysmon]

קיימות מספר שיטות מוכרות להשתיק את Sysmon ולגרום לו להפסיק לדווח והנה כמה מהן:

- Unload של Driver של Sysmon
- עריכת ה-Schema ב-Registry
- טעינת דרייבר זדוני שמוציא את Sysmon מרשימת ה-Callbacks
- עצירת ה-Service



ברוב המקרים לא מכירים שיטות תקיפה מבוססות ETW User-Mode, ולא צריך יותר מ-Logman ו-Wevtutil, עם LOLBIN-ים אלה, נוכל להשתמש בשיטות כמו:

- מחיקת ה-Provider של Sysmon מה-Trace Session שלו
- שינוי גודל ה-Buffer של השיחה
- מחיקת ה-Provider ממערכת ההפעלה

עם קצת הסלמת הרשאות ל-System, נוכל לבצע בפקודה אחת בלבד עם logman לדוגמא, ולגרום ל-Sysmon להפסיק לדווח עד כיבוי והדלקה מחדש:

```
Logman update trace "EventLog-Microsoft-Windows-Sysmon-Operational" -p "Microsoft-Windows-Sysmon" -ets
```

## סיכום

כבר שנים ש-Windows פוסעת בצעדי ענק אל עבר הלא נודע. בעולם הסייבר אי אפשר להפסיק, כל יום מתפרסמת חולשה חדשה שצריך לדאוג לה ומנגנוני הגנה ומיטיגציות שכיף לעקוף.

עם ETW, מיקרוסופט ביססה יכולת שהיום מהווה לב אצל רוב מוצרי ההגנה בשוק ואצל לא מעט EXE-ים קיימים במערכת ההפעלה (אפילו אצל netsh!). בידיים הנכונות, עבודה של צייד יכולה להיות קלה מאוד. אז האם תוקף צריך להסס יותר? אולי כדאי להבין איך משתמשים במנגנון ואיך צוותי הגנה עובדים איתו? לבסס את יכולות התקיפה איתו? האם אנחנו צועדים לעידן של צוותי צייד מותממים? מי יודע.

מקווה שהצלחתי לשכנע אתכם ש-ETW אכן ראוי ללא מעט עיטורים, וגרמתי לכם לחקור ולגלות Provider-ים מעניינים ואיך אפשר להפיק מהם את המירב.

לכל דבר, מוזמנים לפנות אליי במייל:

[bprintac1@gmail.com](mailto:bprintac1@gmail.com)