Digital Whisper

גליון 149, אפריל 2023

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	יהונתן אלקבס, אמיל לוין, דר פיינשטיין ובן פרינטק

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורך

ברוכים הבאים לגליון ה-149 של DigitalWhisper!

וואו, כמה דברים היו החודש, רק מהימים האחרונים קשה לבחור על מה לדבר. ממש לפני ימים בודדים פורסם שהאקרים צפון קוריאנים נהלו במשך שנה שלמה מתקפת Supply Chain לאפליקציית ה-Desktop של 3CX, וקצת לפני כן פורסם ש-Apple שחררר עדכון לחולשה קריטית במספר רב של מוצריה שנעשה בה שימוש In The Wild. ה-Source של Twitter דלף לרשת, הסתבר שאפליקציה סינית מ-PlayStore של גוגל הריצה במשך תקופה 0day על מליוני מכשירים, ועוד כל כך הרבה אירועים... אך אנסה להתמקד ואבחר נושא אחד ☺

קצת לפני אמצע החודש, התפרסמה חולשה ב-Outlook המאפשרת למי שהשתמש בה לגנוב את פרטי ההזדהות לתיבת האימייל מבלי הצורך לבצע אינטרקציה עם המשתמש. החולשה הייתה בשימוש ע"י יחידות ב-GRU ומסתבר שהם הפעילו אותה על מטרות אירופאיות כבר מאמצע 2022 במדינות כמו רומניה, פולין, ירדן אוקראינה וטורקיה (ולפי החברה של Deep Instinct נראה, בדיעבד, שגם קבוצות תקיפה איראניות עשו בה שימוש עוד קודם לכן ב-2020).

היא זוהתה ככל הנראה כאשר הופעלה על מטרות באורקאינה ודווחה למיקרוסופט ע"י צוות ה-CERT האורקאיני (CERT-UA). החולשה לוגית והיא קיימת באיך ש-Outlook מפרסר חלק מה-Properties של מספר אובייקטים באימייל, ניתן לנצלה במספר דרכים, אחת מהן היא לשלוח לקורבן הזמנה לפגישה ב-Outlook ולהוסיף ל-Properties של ה-Reminder של הפגישה קובץ Sound שה-Outlook אמור להפעיל בעת פרסור קובץ הזימון (ספציפית, הפרמטר PidLidReminderFileParameter).

החולשה קיימת בכך שאם הנתיב של קובץ ה-Sound יצביע על נתיב UNC מרוחק, הנ"ל יגרום ל-Outlook להוציא בקשה SMB כדי להשיג את הקובץ, אך במידה ושרת ה-SMB יגיב בכך שיש צורך באותנטיקציה, הדבר יגרום לכך שה-Outlook ינסה לבצע אימות NTLM באופן כזה שניתן יהיה, בלא יותר מדי עבודה, לחלץ מה-NTLM Response המוצפן את ה-NTLM Hash של המשתמש. את ה-NTLM Hash ניתן לשבור וכך להשיג גישה לתיבת האימייל של הקורבן (במידה ונעשה שימוש בשרת Exchange שניתן לגשת אליו מחוץ לארגון), או להשתמש בו בדיוק כמו שהוא באמצעות טכניקת PassTheHash אם כבר יש דריסת רגל בארגון, או לשלב את פרטי ההזדהות שזה עתה התוקף השיג עם אחת מחולשות ה-PostAuth RCE שהתגלו בשרתי OWA לאחרונה (כמו ProxyLogon למשל) במידה והארגון לא דאג לעדכן את שרתיו...

למי שרוצה להתעמק בנושא, ממליץ לקרוא את הפוסט ב-InfoSecWriteups על המכונה ב-TryHackMe.

אין ספק שהשגת פרטי ההזדהות לכמעט כל תיבת אימייל של ארגונים שונים במידה אותה אתה כרגע תוקף עם צבא זאת נקודה המהווה יתרון משמעותי. שלא לדבר על כך שדרך תיבות האימייל הללו ניתן להשיג פרטי VPN או סיסמאות למערכות אחרות בארגון וכך להשיג נכסים שונים.

רמת הפשטות של הפעלת המתקפה היא כמעט בלתי נתפסת (אם תעקבו אחר אופן ניצול החולשה ב-Writeup תראו שניתן לממש אותה ע"י ממשק ה-Outlook בלבד, ואין צורך בלהנדס קבצים בפורמט בינארי מורכב או להתעסק בכל מני Properties לא מתועדים בקרביים של ה-Outlook). וזה מטורף שגם ב-2023, ניתן להשיג יתרון צבאי כזה משמעותי על פני מדינות יריבות בלי הצורך להשקיע עשרות שנים בפיתוח טכנולוגיית טילאות לשיוט חוץ-אטמוספרי סופר מרוכבת. החולשה הזאת לא דורשת ידע עמוק ב-Internals של שום טכנולוגיה במערכת ההפעלה, ולמעשה, ההתנהגות של ה-Outlook במקרה הזה היא אפילו מאוד סבירה.

מיקרוסופט, ב-MSRC, סיווגו את החולשה כ-"Microsoft Outlook Elevation of Privilege", אבל זאת שטות גמורה, החולשה הזאת היא הרבה מעבר לכך, ומאפשרת הרבה מעבר לכך.

אז רוצו לעדכן את מה שצריך לעדכן. ושיהיה בהצלחה לכולנו!

וכמובן, לא נשכח את כל מי שהשקיע מזמנו הפרטי, תרם את שעות השינה שלו, וישב לכתוב לנו מאמרים לגליון החודש! תודה רבה ל**יהונתן אלקבס**, תודה רבה ל**אמיל לוין**, תודה רבה ל**דר פיינשטיין**, תודה רבה ל**בן פרינטק!**

קריאה נעימה,

אפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	4
Inside LSASS	5
Cisco Passwords	37
עוקבים אחרי ETW	Error! Bookmark not defined.
דברי סיכום	51

Inside LSASS

מאת יהונתן אלקבס

הקדמה

הבסיס של כל פתרונות אבטחת המידע מכיל 4 אבני יסוד - אימות, הצפנה, בידוד וניתוח התנהגותי. למעשה, כמעט כל מוצר\פתרון הגנה שתבחרו עובד באמצעות וריאציה כזו או אחרת של אותם אבני יסוד. מבט זריז על מגוון פתרונות אבטחה מתחומים שונים יבהיר את הנקודה; בין אם ברמה התקשורתית כדוגמת VLANs ו-FW, ברמה התשתיתית כדוגמת NTLM / Kerberos ובין אם ברמה האפליקטיבית כדוגמת TLS ו-JWT. למעשה, גם אם ננסה לאחד מספר פתרונות לכדי EPP מוגמר כדוגמת EDR/AV או נעלה שלב לרמה הרשתית כמוצר MDR/XDR השואב לוגים מכל פינה ברשת ושולח את כלל המידע לניתוח בענן על ידי מודלים מבוססי AI לזיהוי אנומאליות בזמן אמת - הבסיס הוא אותו בסיס.

LSASS הוא מרכיב קריטי במערכת ההפעלה Windows. לאחר ביצוע מוצלח של logon לעמדת קצה, כתלות בסוג האימות, LSASS עשוי להכיל במרחב כתובות הזיכרון שלו סיסמאות clear text, NTLM hashes ואף TGTs במידה ונעשה שימוש ב-Kerberos. כאמור, הכל כתלות במנגנון האימות שבוצע. התהליך אחראי על אכיפת מדיניות אבטחה, ניהול אימות והרשאות משתמשים ושמירה על אבטחת מידע רגיש.

עקב תפקידו המכריע באבטחת המערכת, בין אם בסביבה הלוקאלית ובין אם בסביבת הדומיין, LSASS מהווה יעד פופולארי עבור תוקפים המעוניינים לנצל נקודות תורפה ב-process עצמו כדי לקבל גישה למשאבים נוספים. במידה ותוקף צלח בהשגת dump של התהליך, המרחק אל פיצוח סיסמאות ותזוזה רוחבית אינו עָנֵף. במשך שנים מגנים ותוקפים היוו חתול ועכבר בנושא הנ"ל בדיוק בגלל הסיבות הללו.

מהצד האדום, פעולות כדוגמת memory injection, process hollowing, התקנת דרייברים זדוניים ושימוש בכלים המיירטים את בקשות האימות על ידי שימוש ב-Windows API כגון Mimikatz נוצרו ומימשו שיטות חדשות בכל עת שיצאו עדכונים וגרסאות מגננה חדשות עבור LSASS. מהצד הכחול, הפתרונות המובילים שמיקרוסופט הציגה בנושא היו Credential Guard ו-Protected Process Light.

הראשון משתמש בווירטואליזציה מבוססת חומרה כדי לבודד מידע רגיש באמצעות ה-Hypervisor והשני מגביל את היכולת של תוכנות צד שלישי לגבש אינטרקציה עם תהליכים מוגנים על ידי שימוש בחתימות.

במאמר הקרוב אגע בצד הכחול ובמאמר המשך אקדיש את תשומת הלב לאדום. לפיכך, לאורך המאמר אנחנו הולכים לדבר על המענה ההגנתי ולהציג כיצד רכיביו עזרו לעצב ולאבטח את ארכיטקטורת המערכת

הנוכחית של LSASS (ותהליכים נוספים). אבל, בסופו של יום חשוב להבין כי הם מתבססים על אותם 4 אבני יסוד שהזכרנו בפתיח.

אתנחתא לפני שמתחילים: ניתן להבין לפי מספר ה-Buzzwords וראשי התיבות שנזרקו עד כה בהקדמה שהמאמר מיועד לאסכולה מתקדמת, עם זאת אעשה את המירב כדי להסביר בצורה הברורה ביותר עבור אלו שעדיין לא שם.

לפני שמתחילים

סייבר ואבטחת מידע לא נועדו לייצר כסף, אלא לחסוך ממנו. אני יודע שזה משפט קצת מוזר לשמוע בתחילת מאמר טכני על הגנת תהליכים בסביבת Windows אבל זו פרספקטיבה אלמנטרית שחשוב להבין. המודל העסקי של (כמעט) כל ארגון הוא למקסם את אחוזי הרווח והכוח של החברה. מתוך הבנה שנפילה למתקפת סייבר עלולה להשבית את סביבת הייצור והפרודאשן (Ransomware כדוגמה), להסגיר פטנטים רגישים עקב ריגול תעסוקתי ולזעזע את שווי מניות ההחברה - צמצום משטח תקיפה וניטור רשתי הם הסיבה מדוע מועסקים מיישמים, מגנים וחוקרי סייבר רבים.

המאמר הנוכחי מגיע במטרה להציג מענה לסט מאוד ספציפי של ווקטורי תקיפה - הוצאת מידע רגיש מתהלכים בפלטפורמת Windows בעת תקיפה אקטיבית לשם תזוזה רוחבית. כבר עכשיו חשוב לי לעצור ולומר כי פרקטיקות כדוגמת (**JIT**) Just-in-Time Admin, (**JEA**) Just Enough Admin המיישמים את אלמנט ההקשחה הבסיסי של Least Privileges <u>לא יוצגו כלל במאמר</u>. רעיון זה הוצג בצורה לא רעה (וטכנית) במאמר [Blue Hands-on BloodHound](). ממליץ מאוד את הקריאה למי שעוד לא יצא.

במסגרת המאמר כעת אנחנו הולכים לשים דגש על הפתרונות ש-Microsoft הציגה בשביל להגן בצורה מלאה ואיכותית יותר על תהליכים רגישים בסביבת Windows ובמאמר המשך כיצד תוקפים מצליחים (או לא) לעקוף אותם. זה כנראה לא יפתיע אתכם שקיימים מספר רב של מנגנונים Built in לאבטחת מבני נתונים **בזיכרון** של תהליכים מסוגים שונים בפלטפורמת Windows.

הפתרונות הרלוונטיים בנושא הם:

- Security Reference Monitor (SRM)
- Protected Process Light (PPL)
- AppContainer Isolation, Session 0 Isolation & Secure Desktop
- Windows Resource Protection (WRP)
- Windows Memory Management
- Windows Integrity Control (WIC)
- Mandatory Integrity Contorl (MIC)

- Isolated User Mode (IUM) by Hyper-V
- Virtual Secure Mode (VSM) / Virtual Base Security (VBS)

בהחלט רשימה לא קצרה (וכנראה שיש עוד). אני לא הולך לנסות להסביר את כולם במאמר הקרוב מ-2 סיבות עיקריות - אחת, על כל אחד מהם אפשר לכתוב מאמר שלם, ושתיים, אין לי את הידע לכך. עם זאת, אנחנו כן הולכים להכיר לעומק את אלו שקשורים בצורה ישירה ל-LSASS (חתן המאמר הנוכחי). קרי, רוב המאמר יתמקד 2 בפתרונות מהרשימה - PPL & VBS.

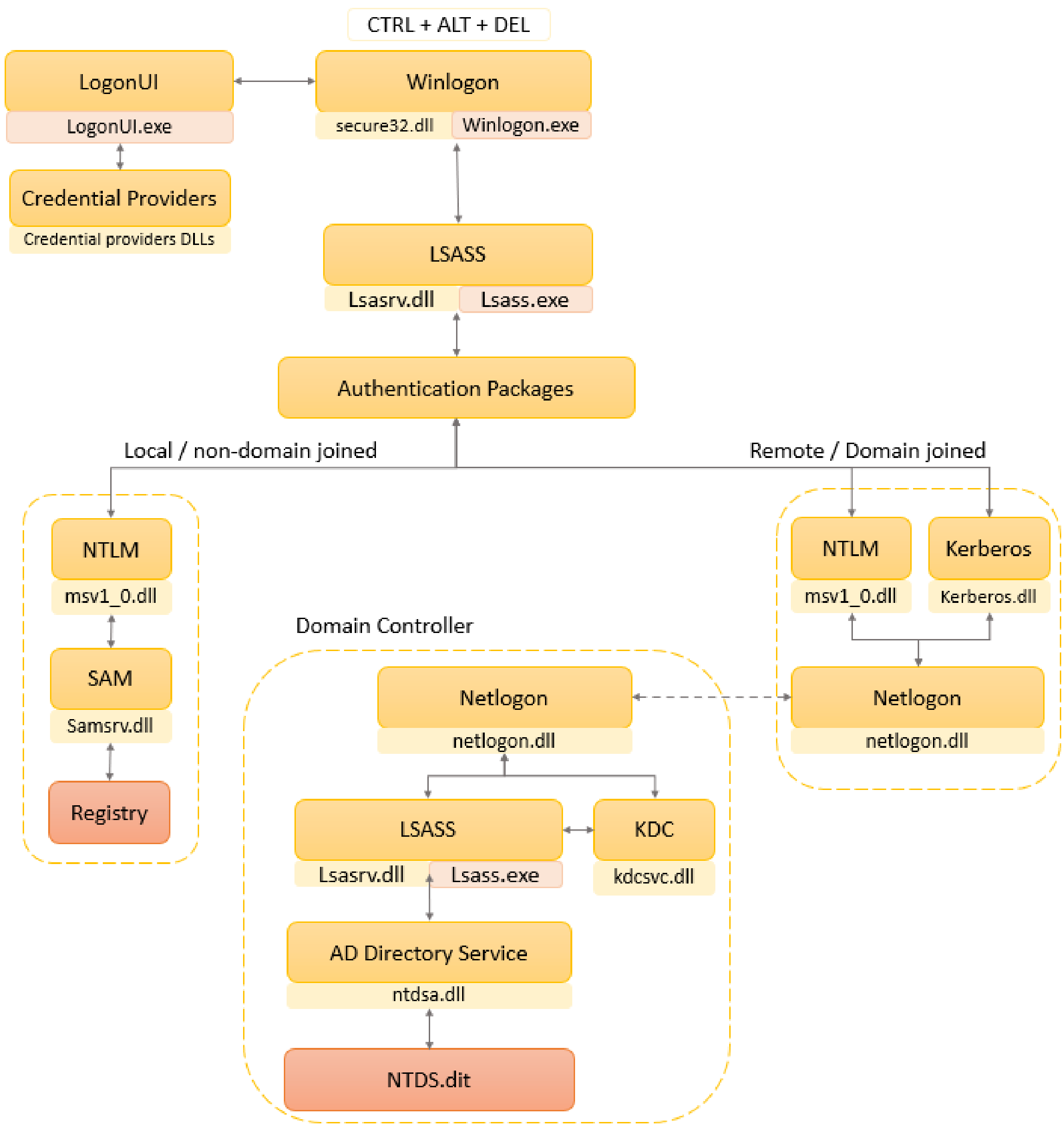
הערה: רוב החומר התיאורטי והרעיונות שאציג במסגרת המאמר כבר הוצגו בדוקומנטציה הרשמית של מייקרוסופט ובספר Windows Internals 7th edition . מטרת המאמר אינה לשכתב אותם אלא לאגד את המידע הרלוונטי לנושא בליווי מחקר בסביבת דומיין שנבנה לצורך הדגמה והסבר המאמר.

על רגל אחת - Windows Credentials Lifecycle Internals

בשביל להבין את מורכבות הבעיה אנחנו צריכים קודם לקחת צעד אחורה ולהבין בצורה מלאה את שיטות האימות וההרשאה ב-Windows. למזלי, נכתבו במסגרת המגזין מאמרים מעולים בנושאים אלו בסביבת הדומיין אשר מציגים לפרטי פרטים את אופן פעולת הפרוטוקולים הלוקחים חלק בכל הסיפור. שני מאמרים מופתיים שאני מאוד ממליץ לקרוא בנושא הם 1-st Step to Tame a Kerberos: Know Your Enemy ו- 2nd Step to Tame a Kerberos: Hit It Where It Hurts מאת עדי מליאנקר ונתנאל כהן שמסבירים כיצד תהליכים ענפים כדוגמת Kerberos ו-NTLM עובדים בסביבה דומיינית וכיצד תוקפים אוהבים לנצלם (המאמר השני מציג ליטרלי כמעט כל מתקפה שקיימת!).

תודות לצמד המאמרים (והחוקרים) הנ"ל, לא ניכנס לעומק ההבנה של פרוטוקולי NTLM ו-Kerberos. עם זאת, מחזור החיים של ה-Credentials בסביבה ה-Windows-ית החל מהתחברות משתמש ועד התנתקותו מהמערכת הוא נושא ששווה להציג (ולהבין) פעם אחת בצורה מלאה. לכן, על מנת לגבש תסריט צלול לכיצד פרטי משתמש עוברים במערכת ההפעלה לשם אימות והרשאה, ניכנס במספר עמודים הקרובים למבנה תהליכי האימות ב-Windows.

אני מאמין בקפיצה ישר למים ולכן בלי עיכובים נוספים - **ארכיטקטורת תהליכי האימות ב-Windows:**



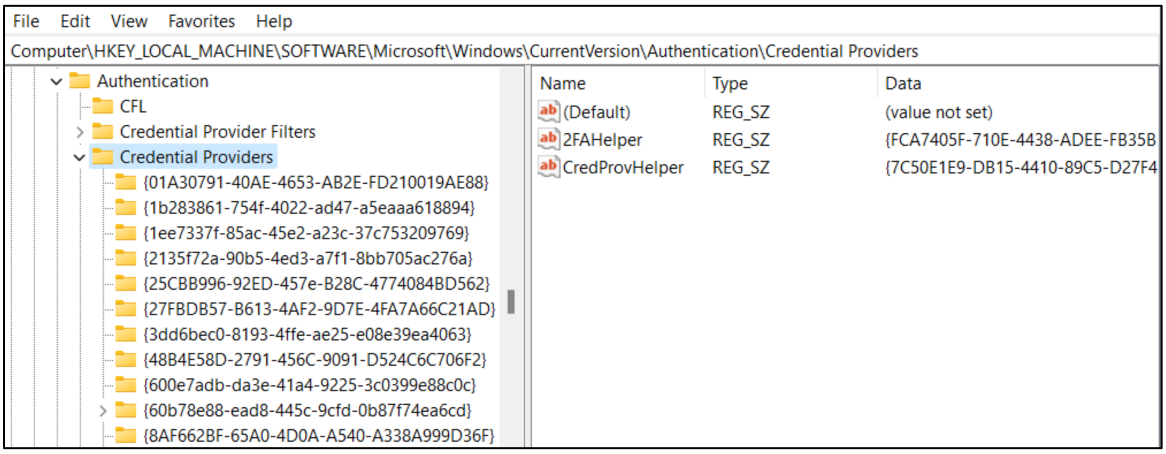
מה אנחנו רואים פה? בפסקה אחת: התחברות אינטראקטיבית לוקאלית מתבצעת על ידי תהליך כניסה (WinLogon) אשר מריץ את תהליך ממשק משתמש לכניסה (LogonUI) בשביל לקבל ממנו את נתוני היוזר באמצעות ה-Credential providers דרך Secure32.dll. משם, על ידי העברת הפרטים לתהליך LSASS, שימוש בחבילת אימות אחת או יותר, מתבצעת בדיקה במסדי הנתונים של SAM וActive Directory על פרטי המשתמש והאם הוא ראשי להיכנס למערכת. במידה, וכן, Winlogon יוצר סביבת desktop ומכניס את המשתמש אליה. ה-Local Security Authority (LSA) היא תת מערכת ב-Windows שאחראית על האימות וההתחברות של משתמשים בסביבה הלוקאלית ובדומיין והיא זו שמנהלת את כל התהליך.

זה היה הסבר על תהליך ההתחברות ב-Windows במינימום מילים. נקודה שחשוב להבין היא שתהליך אימות המשתמש לא מתבצע בקומפוננטה אחת אלא בשיתוף פעולה והעברת הנתונים של המשתמש בין מספר רכיבים במערכת ההפעלה (מיוצג על ידי חצים דו-כיווניים באיור).

הסבר קצת יותר טכני שמציג את הסיפור המלא למי שמתעניין:

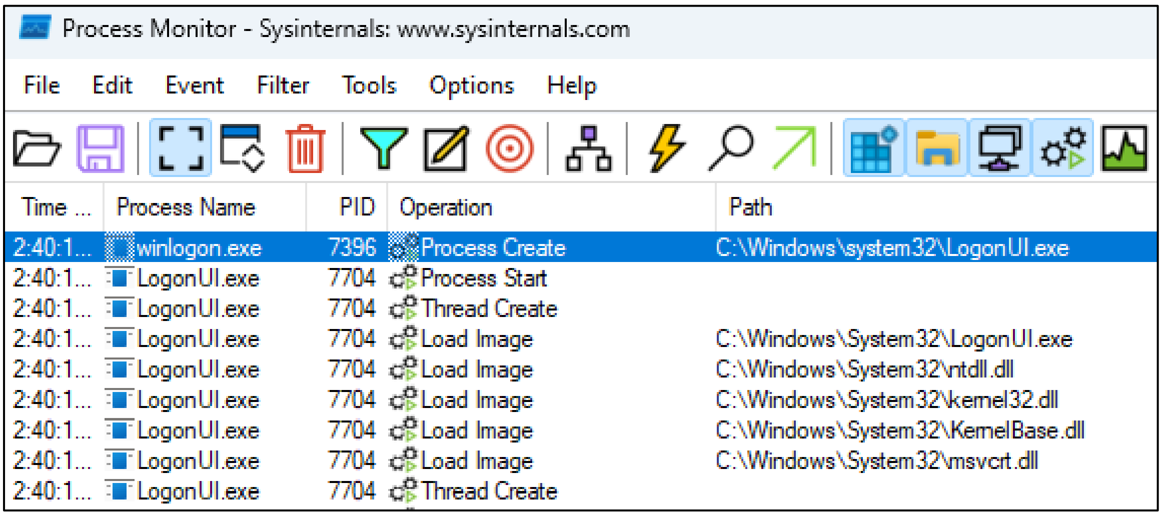
Winlogon.exe הוא תהליך מאומת (trusted) שאחראי לניהול אינטראקציות משתמש הקשורות להתחברות, הרצת תהליכי מערכת, יצירת סביבות עבודה (Desktops) ואבטחה. במילים אחרות, הוא עושה מלא דברים, אחד מהם זה לתאם את כלל ה-flow בתהליך ה-logon של היוזר מול הממשקים (הפנימיים והחיצוניים) של מערכת ההפעלה. כך למשל, על מנת לקלוט את פרטי המשתמש (שם + סיסמה) הוא קורה אל LogonUI.exe או נועל את העמדה במידה ובוצעו יותר נסיונות חיבור כושלים ממספר הפוליסה. בצורה דומה, הוא אחראי על כך שאף תהליך לא מאומת לא יקבל גישה אל סיסמת המשתמש.

אבל אין זה אומר שהוא פועל לבד. למעשה, הוא מסתמך (בצורה די עיוורת) על credential providers שמותקנים על המערכת שישיגו לו את פרטי המשתמש. **credential providers** הם 'ברוקרים' לתהליך האימות אשר מיוצגים על ידי אובייקטי COM בתוך DLLs ומספקים שיטות אימות שונות כדוגמת face recognition או smartcard. ישנם 21 credential providers דיפולטיביים ב-registry על התקנה חדשה של Windows 11:



Winlogon נחשב תהליך קריטי, במידה והוא קורס - כל המערכת תקרוס. לכן, בשביל להגן על מרחב הכתובות שלו מטעויות (תמימות ופחות תמימות) של credential providers מתבצע שימוש בתהליך שונה לטעינתם ולהצגת ממשק ה-logon למשתמשים - **LogonUI.exe**. אם מסיבה מסויימת הקוד של אחד ה-credential providers קרס, Winlogon פשוט יאתחל את logonUI.exe מחדש והוא בתורו יחל את תהליך איסוף נתוני המשתמש (ופעולות נוספות).

באמצעות Process Monitor ניתן לראות את רצף התהליך וטעינת קבצי DLL למימוש האימות בהמשך:



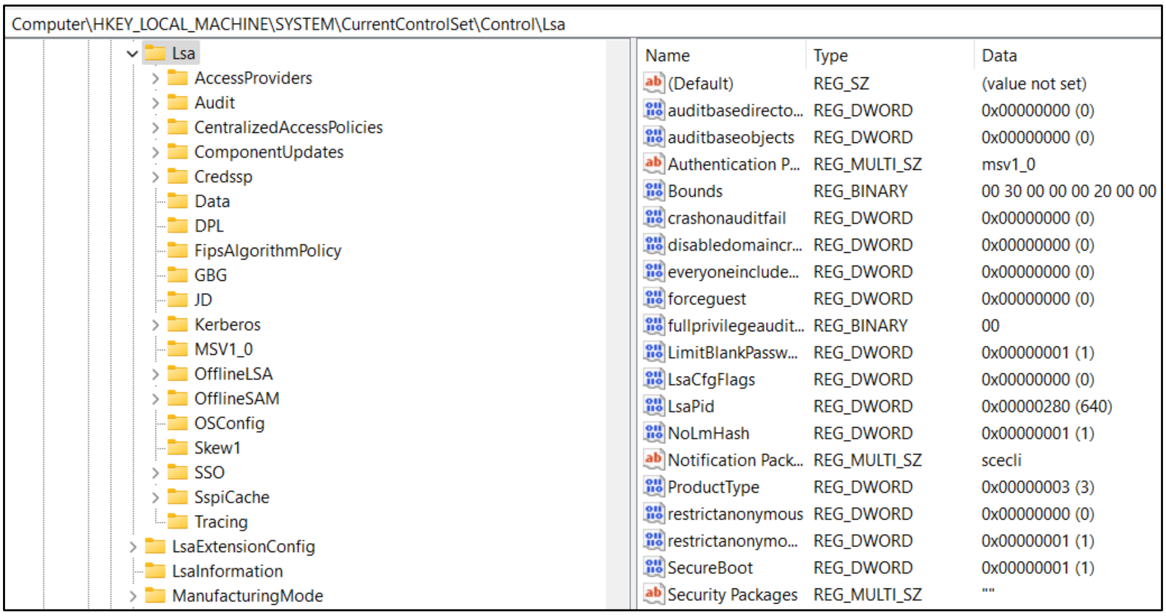
אז איך נראה תהליך ה-logon המלא ב-Windows?

תהליך ההתחברות למכונת קצה מתחיל כאשר המשתמש לוחץ על Ctrl+Alt+Del (רצף הנקרא SAS בדוקומנטציה), הקשה זו מטריגרת את Winlogon להתחל את תהליך LogonUI אשר קורא להשגת שם משתמש וסיסמה מה-credential providers שבו נעשה שימוש והעברבם אל Winlogon בצורה מוצפנת. במקביל, Winlogon יוצר רשומת התחברות לוקאלית ייחודית **SID** עבור היוזר ומשתמש בה בשביל להקצות desktop למשתמש (מאופיין במקלדת, עכבר, מסך וכדו').

לאחר מכן, Winlogon קורא אל תהליך **lsass.exe** בשביל לאמת את המשתמש ומעביר לו את ה-SID. LSASS בתורו מפעיל את ברוקר האימות (שנקבע לפי פוליסות בדומיין, תמיכה\מגבלות של תוכנת הקליינט וכו'), אלו הן **חבילות האימות**. חבילות אימות הן קבצי DLL המבצעים בדיקות אימות בזמן תהליך logon אינטראקטיבי. כשמדובר בהתחברות אינטראקטיבית לדומיין, לרוב נראה את **Kerberos.dll** נטען למרחב הכתובות של lsass.exe.

במידה והמחשב לא חלק בדומיין (התחברות לוקאלית) ו\או אין למכונה תקשורת ל-DC, החבילה שתטען ותנהל את אופרציית ה-challenge-response אל מול המשתמש היא **msv1_0.dll** (אשר מממש את פרוטוקול האימות של NTLM).

אם כי שתי אלו הן חבילות האימות הסטנדרטיות לביצוע אימות אינטראקטיבי ב-Windows, חשוב להבין שהן לא חבילות האימות היחידות שקיימות, ניתן לראות זאת ברגיסטרי:



במידה וחבילת האימות הצליחה לאמת את היוזר, LSASS מחזיר קריאה אל Winlogon אשר מצרף את ה-SID של המשתמש אל ה-token של תהליך ההתחברות למכונה ומפעיל logon shell בשם המשתמש.

חבילת האימות של MSV1_0 לוקחת את פרטי ה-logon של המשתמש ושולחת את שם היוזר עם ה-hash של הסיסמה שלו אל ה-SAM הלוקאלי (בהתבסס על שירות **Samsrv.dll**) בשביל לקבל את נתוני המשתמש - קבוצות שהיוזר חבר בהן, הגבלות וכד'. לאחר קבלת הנתונים, MSV1_0 בודקת שלא קיימות הגבלות (כגון שעות התחברות, מכונות קצה ועוד) בנוגע לאותו משתמש. במידה וקיים cache של התחברות דומיינית (קרי המשתמש התחבר לעמדה בדומיין לפני ועדיין קיימת רשומה לגבי ה-session שלו) MSV1_0 יגש לנתונים שקיימים בו באמצעות הפונקציונאליות של LSASS לשמירת סודות.

בגלל לא מעט חולשות ארכיטקטורה בפרוטוקול NTLM (ובמימוש שלו ב-MSV1_0) הפרוטוקול שמתבצע בה עיקר השימוש בדומיין הוא Kerberos 5. השוני המהותי הוא העובדה שלהבדיל מ-NTLM, חבילת האימות של קרברוס משתמשת בסט אלגוריתמים מתקדם יותר, חותמת על הודעות חשובות ומערבת צד שלישי לביצוע האימות - **שירות KDC שרץ ב-DC**. לכן, לוגים להתחברות ירשמו הן בעמדת הקצה והן ב-DC (ולאחר מכן גם בשרת אליו מעוניינים לפנות כמובן). הספרייה שתהליך lsass.exe ב-DC מריץ בזיכרון שלו למימוש פרוטוקול קרברוס היא **kdcsvc.dll**.

מהאיור אפשר לראות כי LSASS רץ גם במכונת הקצה וגם בצד שרת. כלומר, בשני הצדדים, עושים שימוש ב-netlogon כברוקר לתקשורת בין תהליכי LSASS (על ידי שימוש בפורט מיוחד הנקרא ALPC, עליו נסביר בפרק אחר) וכך ניתן להעביר את פרטי האימות של המשתמש מצד אל צד.

DLL נוסף שנטען לטובת האימות הוא **Ntdsa.dll** שאחראי על ניהול ממשק ה-API מול **NTDS.dit** (מסד הנתונים של ה-AD), הוא מאפשר פונקציונאליות רבה על עץ המידע שמאפיין את מבנה הדומיין. Ntdsa.dll מאפשר לרכיבי המערכת ליצור, לשנות ולמחוק אובייקטי AD, לשלוח שאילתות בנוגע נתוני AD ולבצע רפליקציות בין DC-ים.

לאחר אימות היוזר, ה-Kdcsvc מחזיר credentials דומיינים אל LSASS והוא בתורו מחזיר את תוצאת האימות (הצליח או לא) אל העמדה בה ה-logon התרחש. LSASS מוסיף לאובייקט שמייצג את משתמש את ה-SIDs שהיוזר מקושר אליהם בנוסף (קבוצות), מסתכל במסד הפוליסות הלוקאלי LGPO של המכונה (אשר מתעדכן אל מול פוליסות הדומיין ב-DC מעת לעת) ולפי כל רשומות ה-SIDs שהיוזר מקושר אליהם מקבל access token שמייצג את ההרשאות ומשאבים שהוא זכאי אליהם. לאחר יצירת ה-token, LSASS משכפל אותו, יוצר handle שניתן להעביר אל Winlogon (ביחד עם הודעה שהאימות התרחש בהצלחה) וסוגר את ה-handle של עצמו.

LSASS מעביר אל Winlogon את הפרטים הבאים - קישוריות ל-access token, LUID ל-session ההתחברות (כל session ב-Windows מקבל מזהה ייחודי לוקאלי כשהוא נוצר בהתאם לחבילת האימות באמצעותה הוא נוצר) ומידע על פרופיל המשתמש. ניתן לבחון את ה-sessions הפעילים בעמדת הקצה וב-DC באמצעות LogonSessions ו-Process Explorer בחבילת Sysinternals. כך למשל נוכל להסתכל על 2 רשומות לדוגמה במכונת הקצה WIN-CLIENT3 (בצד שמאל) וכדי להשלים את התמונה, נוכל לראות את ה-token-ים בתהליך LSASS שרץ ב-DC מחזיק אצלו באמצעות ה-handles (בצד ימין).

```
PS C:\Users\domain_admin\Desktop\SysinternalsSuite> .\logonsessions64.exe

LogonSessions v1.41 - Lists logon session information
Copyright (C) 2004-2020 Mark Russinovich
Sysinternals - www.sysinternals.com


[0] Logon session 00000000:000003e7:
    User name:    JON\WIN11-CLIENT3$
    Auth package: Negotiate
    Logon type:   (none)
    Session:      0
    Sid:          S-1-5-18
    Logon time:   2/19/2023 8:34:22 AM
    Logon server:
    DNS Domain:   JON.NET
    UPN:          WIN11-CLIENT3$@JON.NET
[8] Logon session 00000000:00085304:
    User name:    JON\domain_admin
    Auth package: Kerberos
    Logon type:   Interactive
    Session:      1
    Sid:          S-1-5-21-1403467943-1367565381-54031474-1108
    Logon time:   2/19/2023 8:34:41 AM
    Logon server: DC01
    DNS Domain:   JON.NET
    UPN:          domain_admin@JON.NET
```

WIN-CLIENT3

Token	NT AUTHORITY\SYSTEM:19871f
Token	NT AUTHORITY\SYSTEM:3e7
Token	NT AUTHORITY\SYSTEM:3e7
Token	JON\Administrator:623fa
Token	JON\Administrator:623fa
Token	JON\Administrator:623fa
Token	JON\Administrator:623fa
Token	JON\Administrator:623fa
Token	JON\Administrator:623fa
Token	NT AUTHORITY\SYSTEM:3e7
Token	NT AUTHORITY\SYSTEM:19871f
Token	NT AUTHORITY\SYSTEM:1c0bff8
Token	JON\WIN11-CLIENT3$:1e37359
Token	JON\WIN11-CLIENT3$:1e3dfbd
Token	JON\domain_admin:1ed2c92
Token	JON\domain_admin:1e9e6ee
Token	JON\WIN11-CLIENT3$:1e37359
Token	JON\domain_admin:1e9e6ee
Token	NT AUTHORITY\SYSTEM:1bde728
Token	JON\domain_admin:1ed2c92

DC

המספר שמיוצג ב-Logon session (שמאל) ובסיומת כל יוזר (ימין) הוא ה-LUID של token המשתמש.

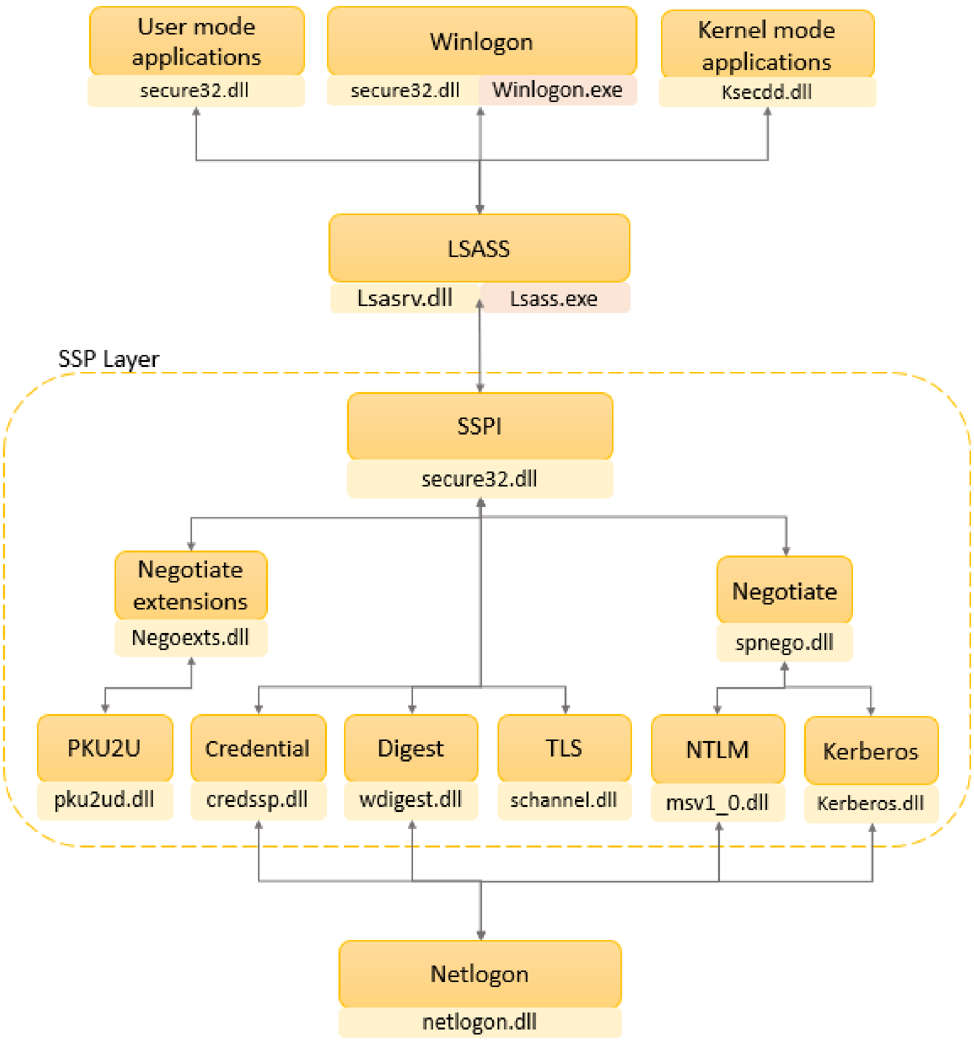
קבוצה חשובה שאי אפשר לא להציג כאשר מדברים על ארכיטקטורת האימות של Windows הם כלל ה-**Security Support Provider**, SSP בקצרה ו'ספקי אבטחה' בעברית שבורה. אפליקציות ב- Windows Server יכולות לאמת משתמשים על ידי שימוש בממשקים של SSPs בצורה אבסטקטית - ב-**SSPI**. SSP מיוצגים על ידי ספריות DLL שתומכת בספפיקציה של ממשק ה-SSPI. לכן, מפתחים אינם צריכים להבין את המורכבות של פרוטוקולי אימות ספציפיים כדוגמת Kerberos/NTLM או לבנות פרוטוקולי אימות באפליקציות שלהם *(כל מי שחושב שלהמציא את הגלגל מחדש זה פתרון טוב עתיד ללא מעט עליות לא צפויות בדרכו)*.

בדוקומנטציה שלה, Microsoft מתייחסת אל תשתית ה-SSPI כבסיס המרכזי לאימות במערכת ההפעלה. כל יישומי המערכת ב-kernel mode ואפליקציות user mode נדרשות לעבור דרכה על מנת לאמת משתמשים. SSPI זמין דרך מודול **Secur32.dll**, שהוא API המשמש להשגת שירותי אבטחה משולבים לאימות, שלמות ההודעות ופרטיות ההודעות. הוא מהווה שכבת אבסטרקציה בין פרוטוקולים ברמת האפליקציה לפרוטוקולי אבטחה.

מכיון שיישומים שונים דורשים דרכים שונות לזיהוי או אימות משתמשים ודרכים שונות להצפנת נתונים בזמן שהם עוברים ברשת, SSPI מספק דרך לגשת לספריות DLL המכילות שיטות אימות ופונקציות הצפנה שונות. מכאן גם ניתן להבין את <u>השוני</u> הטרמינולוגי בין SSPs אל חבילות אימות (Packages Authentication) שהוצגו בתרשים הקודם - SSPs הם קטגוריה כללית של רכיבי אבטחה ב-Windows שמספקים פונקציונליות קריפטוגרפית ושירותי אבטחה.

כלומר, בעברית פשוטה, SSPI מאפשר קריאה אל פונקציות פנימיות בכל ספק אבטחה בהתאם לסוג הפרוטוקול שנבחר לאופרציית האימות.

אם לעקוב אחרי ארכיטקטורת ה-Credentials שאיתה פתחנו את הפרק, הרי שניתן לייצג את **ארכיטקטורת ממשק SSPI** בצורה הבאה:



כפי שמוצג באיור, ה-SSPI ב-Windows מספק מכניזם שמכיל tokens לאימות על פני ערוצי התקשורת הקיימים (sockets, קריאות RPCs וכדו') בין מחשב הלקוח לשרת. כאשר יש צורך לאמת שני מחשבים או מכשירים כדי שיוכלו לתקשר ביניהם בצורה מאובטחת, הבקשות לאימות מנותבות ל-SSPI, אשר בתורו משלים את תהליך האימות, ללא קשר לפרוטוקול הרשת הנמצא כעת בשימוש.

כיצד זה קורה? כגוף אבסטרקטי, האובייקטים ש-SSPI מחזיר הם בינאריים והשרת והלקוח לא משנים אותם כלל בעת המעבר בין הבקשות. כך ניתן להעביר אותם לשכבת SSPI בצד שני לפירסור וניתוח. בצורה כזו, תשתית ה-SSPI מאפשרת לאפליקציות שונות להשתמש במודלי אבטחה שונים הזמינים במחשב (או ברשת) מבלי לשנות את הממשק לאותה מערכת אבטחה.

משפט מגניב שעוזר להבין באנגלית פשוטה את הנושא: **SSPI is used to Kerberize applications**.

בחבילות ה-SSP אפשר לראות את msv1_0.dll ואת Kerberos.dll שהרחבנו עליהם מקודם, על הבחירה מול מי לבצע את האימות מבין השניים (כאמור כתלות בחוקות דומיין ובתמיכה של מכונת הקצה) אחראית ספריית **Spnego.dll** כאשר דיפולטיבית היא תבחר בקרברוס.

מדוע TLS מוצג ומה הוא עושה שם? ובכן, קריפטוגרפיה של מפתח ציבורי היא פרקטיקה שימושית ובעיקר מאוד **אלסטית**. דיברתי על כך לא מעט במאמר על תעודות דיגיטליות לפני כמה שנים. הפלטפורמה של Windows ובפרט Windows server מאפשרת אימות מבוסס web. העברת זהויות על גבי האינטרנט, אימות זהויות אל מול שרתי web מאובטחים, שימוש ב-Azure Active Directory וחיבורו לסביבה הלוקאלית הן דוגמאות פרקטיות לכך. **Schannel.dll** משתמש במודל client-server בשביל לממש את היישומים השונים של TLS בגרסאותיו השונות.

בנושא אחר, **Wdigest.dll** היא גם חלק מחבילות האימות המוצעות. למרות ש-wdigest הוא פרוטוקול ישן, לא רלוונטי ופגיע. הסיבה העיקרית לכך שהוא לא מומלץ לשימוש היא מכיוון שהוא מעביר סיסמאות ו-credentials בתצורת clear text או MD5. הוא תומך web (על ידי ביצוע RPC לשרת) אז אולי הוא עדיין שם בשביל תמיכה לאחור ב-IIS או Internet Explorer (שהספיק גם להיות deprecated). בכל מקרה, למרות הדוקומנטציה של Microsoft בנושא, כשפתחתי את רשימת ה-DLLs שתהליך lsass.exe קורא להן Wdigest.dll הופיע במלוא הדרו.

שתי SSPs שפחות רלוונטיות לטובת המאמר הן CredSSP ו-PKU2U אבל אנחנו כבר פה אז נרחיב עליהן בקצרה בכל זאת. CredSSP מאפשרת Single Sign-on (SSO) למשתמש. מכירים את זה שאתם פותחים session ב-RDP ויש לכם גישה למשאבים שלכם גם שם? ובכן, CredSSP מאפשרת לאפליקציות לבצע דיליגציה ל-credentials שלכם ממחשב הלקוח לשרת היעד.

כחלק מתשתית ה-SSPI, חברת Microsoft מאפשרת תמיכה ב-extension SSPs, כלומר, הרחבות לתשתית האימות עצמה. אתם עובדים בסביבה היברידית של Azure AD ורוצים להזדהות באמצעות ה-online ID שיש לכם ולא עם משתמש רשתי\ לוקאלי? אתם יכולים באמצעות PKU2U. תאמת שיש מלא אופציות מגניבות בנושא אז למי שרוצה להרחיב שווה לעבור על הדוקומנטציה.

אעצור כאן את ההסבר על אימות והרשאות ב-Windows. אנשים (שהם בבירור לא אני) יכולים לכתוב אינספור מאמרים על הנושא הספציפי הזה בלבד, אז ללא ספק יש עוד מלא חומר שאפשר ללמוד אבל גם ככה סטיתי מנושא המאמר יותר ממה שתכננתי.

מתחילים

אוקיי לקח לנו רק 12 עמודים לעבור את ההקדמה אבל הבנו ש-LSASS הוא תהליך חשוב בכל הקשור לאימות לוקאלי ודומייני. מה שכן, בסופו של דבר הוא רק חלק קטן ואינטגרלי מנושא רחב משמעותית, אז **למה להתעמק דווקא בו?** ובכן, כפי שכל הפרק הקודם מציג בצורה טובה, lsass.exe הוא בהחלט חלק מתמונה רחבה מאוד אבל הוא מרכז בתוכו המון מידע רגיש, פוליסות וסודות של sessions אקטיביים במערכת.

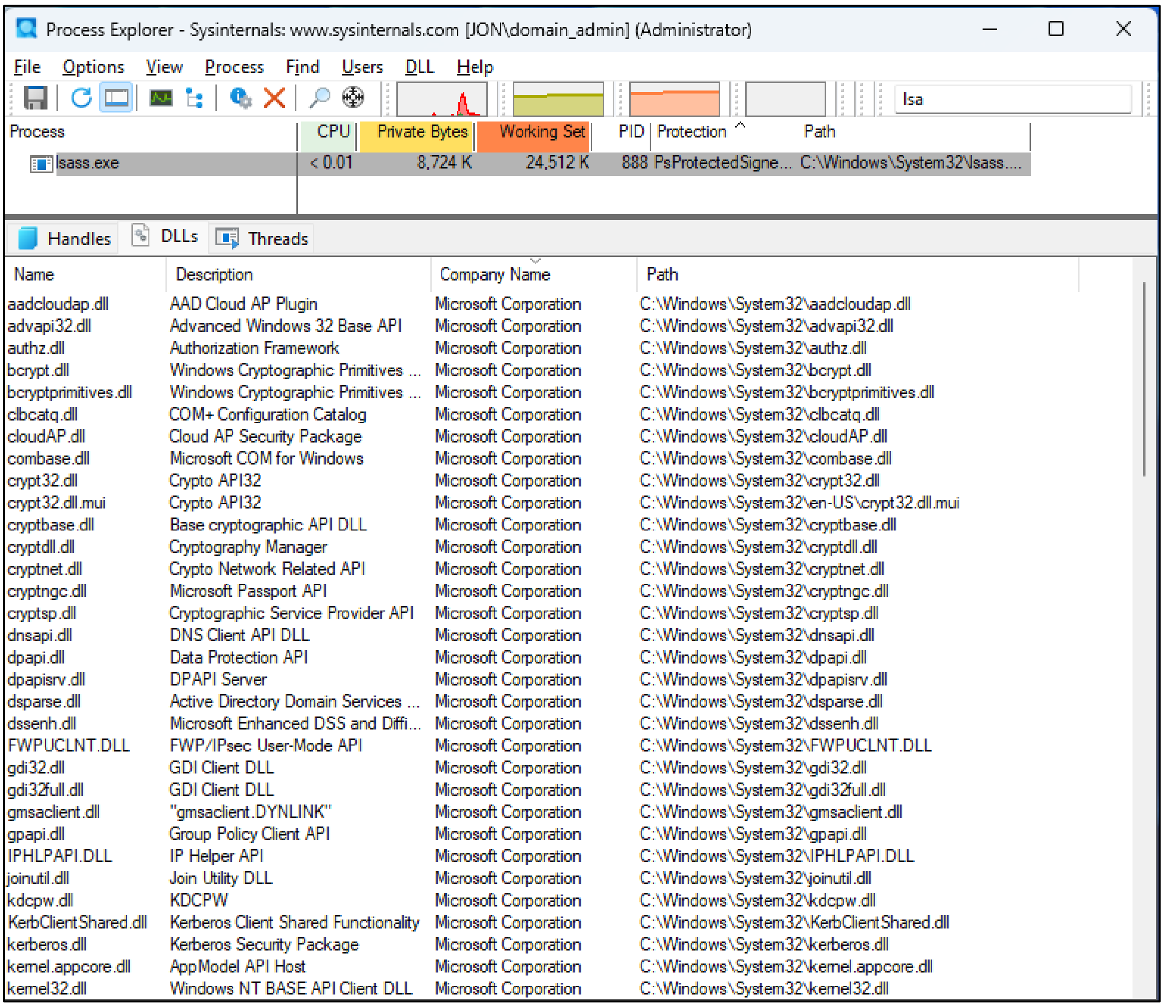
שאלה שחייבת להישאל היא **מדוע LSASS עושה את זה?** למה מייקרוסופט בחרו לכתוב תוכנה ששומרת את כל הסודות באופן מתמשך בזיכרון? התשובה, כמו בכל פתרון מבוסס מנגנון cache - חווית משתמש ואיכות חיים. בין אם זו גישה לקבצי רשת דרך SMB, הורדת מיילים באמצעות שרת Exchange, קבלת כתובת IP דרך שרת DHCP, נגישות אל Sharepoint - את כל אלו המשתמש מסוגל לבצע <u>seamlessly</u> ללא הצורך בהזדהות מול השרתים ע"י מנגנון SSO אל רוב השירותים בהם. זה אחד מהפיצ'רים שהופך את השימוש בדומיין **לפרקטי**.

אז אילו פעולות של משתמש אקטיבי יגרמו לטירגור יצירת LSA sessions שישמרו את פרטיו כ- LSA credentials בזכרון של LSASS?

- התחברות לוקאלית (local session) והתחברות מרוחקת (RDP session) במכונת קצה.
- כל משימה שהמשתמש יריץ באמצעות RunAs.
- כל הרצה של Windows service אקטיבי על מכונת הקצה.
- הרצה של תהליך אוטומטציה (בעל הרשאות) כזה או אחר - משימה מתוזמנת או סקריפט מבוסס batch job.
- שימוש בכלים אדמיניסטראטיביים מרוחקים (כדוגמת TeamViewer, PowerShell, VNC) לשם ביצוע פעולה (בתצורה לוקאלית) על מכונת קצה.

LSASS הוא תהליך עצום עם לוגיקה מאוד מורכבת. מספיק מבט קצר על מספר ה-DLLs וה-Handles שהוא מרכז בתוכו בשביל להבין את גודל הסיפור שאנחנו ניצבים מולו.

באמצעות Process Explorer נוכל לראות את המאפיינים האלו הן בעמדת הקצה והן ב-DC עצמו:



אלה רק שליש מסך ה-DLLs. סה"כ באותו רגע היו:

עמדה	ספריות DLLs	Handles
WIN11-CLIENT01	112	275
DC01	139	981

הערה: מספר ה-handles ב-DC נמוך יחסית מכיוון שבאותו רגע חיו ברשת רק 2 מחשבי קצה עם 2 יוזרים פעילים. בסביבה אמיתית המספר גבוה משמעותית.

נקודה מעניינת ששווה להציג היא העובדה שאנחנו יכולים לראות את כל ה-DLLs של lsass.exe באמצעות Process Explorer אבל אם ננסה לבצע את אותה פעולה באמצעות Listdll (כלי נוסף בחבילת sysinternals) לא נצליח. מדוע זה אם שניהם רצים על ידי אותו משתמש ובאותם הרשאות? שמרו את השאלה, נוכל לענות עליה אחרי הפרקים הבאים.

```
Administrator: Windows PowerShell

PS C:\Users\domain_admin\desktop\SysinternalsSuite> .\Listdlls64.exe lsass.exe

Listdlls v3.2 - Listdlls
Copyright (C) 1997-2016 Mark Russinovich
Sysinternals

Error opening lsass.exe(888):
Access is denied.
```

התמונה הגדולה

ברוב מערכות הפעלה מרובות משתמשים, האפליקציות מופרדות ברמת הריצה שלהן מקוד מערכת ההפעלה (אשר רץ בהרשאות גבוהות). המעבד של המחשב מודע לסטאטוס של כל תהליך שניכנס אליו ובהתאם מאפשר (או לא) גישה אל משאבי המערכת. על תהליך שמורשה לגשת ללב משאבי המערכת (הן בהיבטי תוכנה והן בהיבטי חומרה) נאמר כי הוא בעל הרשאות **kernel-mode** ועל תהליך המוגבל בפעולות ובממשקים העומדים לרשותו נאמר **user-mode**. בטרמינולוגיה הזו נשתמש לא מעט בפרק הבא.

בדומה למערכות UNIX גם Windows מבוסס על **מערכת הפעלה מונוליטית** בהקשר לכך שרוב הקוד של מ"ה והדרייברים משותף במרחב הזיכרון של הקרנל. קונספט שחשוב להבין הוא שלמרות שתהליכים חיים **זה לצד זה** במרחב כתובות הקרנל ותיאורטית יש לכל אחד מהם גישה ל-data structures של ראהו, זה ממש לא מקובל למשוך מידע בצורה הזו (קרי, על ידי פנייה ישירה לכתובת הזיכרון). במקום זאת, מתבצע שימוש בממשק API פורמאלי להעברת פרמטרים ובשביל לגשת\לשנות מבני נתונים.

אבל (וזה אבל חשוב) עובדה זו מציבה תהליכים רגישים כדוגמת LSASS בבעיה מכיוון שכעת כל יצרן *מדפסת שאפאחד מעולם לא שמע עליו* יכול להגיש את הדרייבר שלו ולקבל גישה לאותו מרחב כתובות "מוגן" בקרנל. כמובן שזה לא כל כך פשוט אבל הבנתם את הרעיון.

את המאמר פתחנו במשפט חצי פילוסופי שמטרתו הייתה להכליל את תצורות האבטחה (4 אבני יסוד) כאשר אנו ניגשים לחפש פתרון לבעיה אבטחתית. כאשר מייקרוסופט התמודדו עם הבעיה של מרחב הכתובות המשותף הם בחרו ב-2 פתרונות: **בידוד** (באמצעות Credential Guard) **ואימות** (חתימה על

תהליכים רגישים באמצעות Protected Process Light). בפרקים הקרובים אסביר על שני הנושאים האלו ובסופם נראה בדיוק כיצד Microsoft מפעילה את הצמד בשביל להגן על LSASS.

מאבטחים רק את מי שצריך - Protected Process Light

תהליכים רגישים ב-Windows רצים ברמת context מיוחדת הידועה בשם "Protected Process". תהליכים אלה מיועדים להיות מאובטחים יותר מתהליכים "רגילים"; מגוון האופרציות שהם יכולים לבצע דל משמעותית ויש עליהם מגבלות נוספות שלא חלות על תהליכים אחרים. העובדה שלא היינו רוצים ששירות Microsoft Defender (MsMpEng.exe) ירוץ בצורה דומה אל שירות ה-Fax (fxssvc.exe) היא ההמחשה הכי פשוטה לצורך הנ"ל.

Well, מודל אבטחת המידע של Windows מתבסס על העובדה שכל תהליך הרץ עם token בעל הרשאות **debug** (כמו זה שיש למשתמש Administrator) יוכל לבקש (ולקבל) גישה לכל תהליך הרץ במ"ה. הוא יוכל לקרוא ולכתוב לזיכרון התהליך, להזריק קוד, להשעות ולחדש ריצה של threads ולתשאל מידע על תהליכים אחרים. בצורה כזו תהליכים כמו Process Explorer ו-Tasks Manager עובדים בשביל למנף את הפונקציונאליות שלהם עבור המשתמש. ברור לנו שמכניזם זה מתנגש חזיתית עם מודלי אבטחת מידע נוספים בפלטפורמה.

המקור של Protected Process (PP) התחיל לפני די הרבה זמן עוד ב-Winows Vista / Server 2008 אבל שם כל המנגנון שתיארתי בפסקה מעל היה הרבה יותר הָדוּק ומנע פונקציונאליות רבה ברמת מערכת ההפעלה והמשתמש. למעשה המטרה שלשמה הוא הגיע לא הייתה בכלל להגן על credentials אלא יותר בכיוון של למנוע מהמשתמש לגנוב מידע מדיסקים עם זכויות יוצרים באמצעות נגן DVD פירטי.

לכן, ב-Windows 8.1 / Server 2012 R2 הוצג מנגנון חדש - Protected Process Light (PPL). מטרתו הייתה פשוטה - שחרור חלק מהמגבלות על אילו קבצי DLL מותרים לטעינה לתהליך מוגן והצגת מכניזם חתימה ברמות שונות עבור קובץ ה-executable הראשי.

ב-PPL קוד הרץ ב-user-mode (ואפילו אם הוא רץ בהרשאות גבוהות) לא יכול לחדור באמצעות הזרקת threads או השגת מידע על ה-DLLs שאותם תהליכים מוגנים טוענים. עם זאת, המודל של PPL מוסיף מימד חדש לכל הסיפור - attribute values. בסופו של יום, **עולם אבטחת המידע מתכנס לאמון**. בדיוק כמו בעולם ה-web המבוסס על TLS בו אנחנו (קרי, הדפדפנים שלנו) חייבים לסמוך על DigiCert Global Root CA בעיניים עצומות בשביל לעשות עסקים באינטרנט, כך גם בעולם התשתיות והדומיין - **חייבת להיות נקודת מוצא של אמון** - Root of Trust; חלק מאותה נקודה הם התהליכים המוגנים.

בעוד שב-PP נעשה שימוש במכניזם בינארי (חתום \ לא חתום) ב-PPL מתבצע שימוש במודל עם 8 רמות. PPL ברמה אחת יכול לפתוח כל תהליך באותה רמת חתימה (או רמה נמוכה יותר) עם גישה מלאה אבל

מוגבל מאוד במידה והתליך המוגן עם חתימה ברמה גבוהה יותר. הטבלה הבאה מסכמת את הרעיון בצורה טובה:

Open for Full Access

More Secure

Signer Name (PS_PROTECTED_SIGNER)	Level	Used For
PsProtectedSignerWinSystem	7	System and minimal processes (including Pico processes).
PsProtectedSignerWinTcb	6	Critical Windows components. PROCESS_TERMINATE is denied.
PsProtectedSignerWindows	5	Important Windows components handling sensitive data.
PsProtectedSignerLsa	4	Lsass.exe (if configured to run protected).
PsProtectedSignerAntimalware	3	Anti-malware services and processes, including third party. PROCESS_TERMINATE is denied.
PsProtectedSignerCodeGen	2	NGEN (.NET native code generation).
PsProtectedSignerAuthenticode	1	Hosting DRM content or loading user-mode fonts.
PsProtectedSignerNone	0	Not valid (no protection).

PPL חתומים על ידי ישויות שונות במערכת ההפעלה עם רמת Trust שונה; כאשר 0 מציין את הרמה הנמוכה ביותר של חתימה (אין חתימה) ו-7 מייצג את הרמה הגבוהה ביותר (תהליכי מערכת). אם נסתכל בדוקומנטציה של Microsoft בנושא מבנה נתונים של PS_PROTECTION נוכל לראות כיצד אותה רמת הגנה (קרי, חתימה) מיוצגת ב-EPROCESS בקרנל:

```
typedef struct _PS_PROTECTION {
    union {
        UCHAR Level;
        struct {
            UCHAR Type   : 3;
            UCHAR Audit  : 1;                // Reserved
            UCHAR Signer : 4;
        };
    };
} PS_PROTECTION, *PPS_PROTECTION;
```

למרות שהוא מיוצג כמבנה (struct), כל המידע מאוחסן בשתי nibbles של byte בודד אחד. 3 הביטים הראשונים מייצגים enum של סוג החתימה (None, Light, Full), כלומר האם התהליך PP או PPL או כלל לא מוגן. ארבעת הסיביות האחרונות מייצגות את enum של סוג החתימה. הייצוג של אותם enum-ים מוגדר בצורה הבאה:

```
typedef enum _PS_PROTECTED_TYPE {

    PsProtectedTypeNone = 0,

    PsProtectedTypeProtectedLight = 1,

    PsProtectedTypeProtected = 2

} PS_PROTECTED_TYPE, *PPS_PROTECTED_TYPE;
```

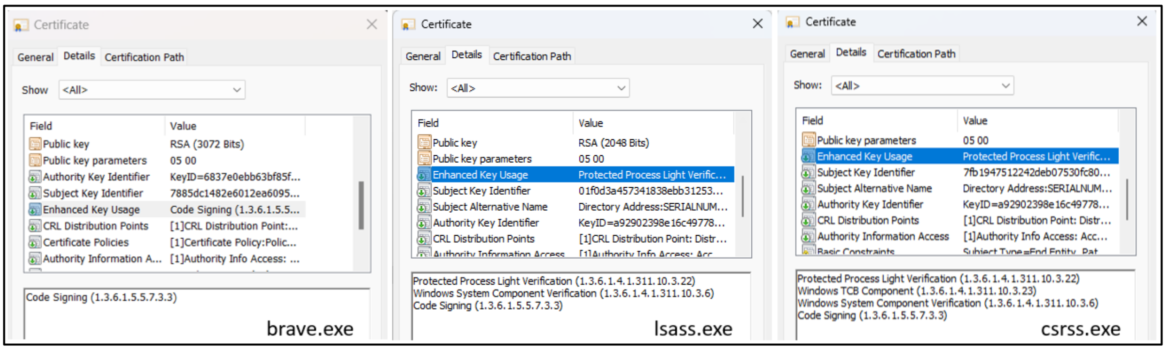
```
typedef enum _PS_PROTECTED_SIGNER {
   PsProtectedSignerNone = 0,       // 0
   PsProtectedSignerAuthenticode,   // 1
   PsProtectedSignerCodeGen,        // 2
   PsProtectedSignerAntimalware,    // 3
   PsProtectedSignerLsa,            // 4
   PsProtectedSignerWindows,        // 5
   PsProtectedSignerWinTcb,         // 6
   PsProtectedSignerWinSystem,      // 7
   PsProtectedSignerApp,            // 8
   PsProtectedSignerMax             // 9
} PS_PROTECTED_SIGNER, *PPS_PROTECTED_SIGNER;
```

כאמור הצגתי כי בטלה הראשית ישנן 8 רמות של "חתימות" בסדר עולה. כל רמה מוגדרת באמצעות קומבינציה של שני הערכים הללו (PS_PROTECTED_TYPE + PS_PROTECTED_SIGNER):

Protection level	***Value***	***Signer***	***Type***
PS_PROTECTED_SYSTEM	0x72	WinSystem (7)	Protected (2)
PS_PROTECTED_WINTCB	0x62	WinTcb (6)	Protected (2)
PS_PROTECTED_WINDOWS	0x52	Windows (5)	Protected (2)
PS_PROTECTED_AUTHENTICODE	0x12	Authenticode (1)	Protected (2)
PS_PROTECTED_WINTCB_LIGHT	0x61	WinTcb (6)	Protected Light (1)
PS_PROTECTED_WINDOWS_LIGHT	0x51	Windows (5)	Protected Light (1)
PS_PROTECTED_LSA_LIGHT	0x41	Lsa (4)	Protected Light (1)
PS_PROTECTED_ANTIMALWARE_LIGHT	0x31	Antimalware (3)	Protected Light (1)
PS_PROTECTED_AUTHENTICODE_LIGHT	0x11	Authenticode (1)	Protected Light (1)

נקודה שחשוב להבין היא שלכל (!) תהליך של מ"ה יש תעודה חתומה של מייקרוסופט. במידה ומדובר באפליקציה (כגון דפדפן Brave.exe) החתימה תהיה על ידי CA authority כמו DigiCert. בתעודה, תחת השדה של **Enhanced Key Usage** (EKU) מופיע עבור כל תהליך מוגן כיצד הוא חתום ומה ההרשאות שלו.

על ידי שימוש ב-Properties על קובץ ה-exe של התהליך עצמו או באמצעות Task manager האהוב ניתן לצפות בכל החתימות הדיגיטליות של התהליכים. לשם הדוגמה נסתכל על 3 תהליכים (brave.exe, lsass.exe ו-csrss.exe) ברמות חתימה שונות, קל לראות את ההבדל בשדה EKU:



בנוסף, ה-protection level של התהליך גם תשפיע על אילו DLLs הוא מורשה לטעון (בשביל למנוע DLL hijacking). כיצד זה קורה? כאשר תהליך מתחיל לרוץ נשמר לו ב-EPROCESS שדה של SignatureLevel וכאשר הוא בא לטעון DLL מתבצעת בדיקה ב-lookup table כלשהי לבדיקה האם ה-DLL הוא ב-signature level המתאים. זה נבדק על ידי מודל ה-Code Integrity באותה צורה בה ה-executable הראשי נבדק. ניתן לראות באמצעות משתמש חזק את התהליכים המוגנים באמצעות Process Explorer תחת עמודת "Protection":

Process	CPU	Private Bytes	Working Set	PID	Protection	Path
System	< 0.01	44 K	2,364 K	4		
Registry		5,808 K	34,320 K	92		[A device attached to the syst...
Memory Compression		500 K	70,884 K	2948		[A device attached to the syst...
SgmBroker.exe		5,048 K	8,520 K	10116	PsProtectedSignerWinTcb	[Access is denied.]
wininit.exe		1,312 K	6,312 K	704	PsProtectedSignerWinTcb-Light	[Access is denied.]
smss.exe		1,092 K	1,124 K	440	PsProtectedSignerWinTcb-Light	[Access is denied.]
services.exe		5,716 K	13,244 K	848	PsProtectedSignerWinTcb-Light	[Access is denied.]
csrss.exe		1,832 K	5,532 K	628	PsProtectedSignerWinTcb-Light	[Access is denied.]
csrss.exe	< 0.01	2,072 K	22,796 K	712	PsProtectedSignerWinTcb-Light	[Access is denied.]
svchost.exe		1,692 K	7,104 K	1328	PsProtectedSignerWindows-Light	[Access is denied.]
svchost.exe		2,508 K	9,928 K	9896	PsProtectedSignerWindows-Light	[Access is denied.]
SecurityHealthService.exe		3,632 K	14,764 K	8128	PsProtectedSignerWindows-Light	[Access is denied.]
lsass.exe		9,028 K	26,004 K	900	PsProtectedSignerLsa-Light	[Access is denied.]
NisSrv.exe		3,564 K	10,488 K	7488	PsProtectedSignerAntimalware-Light	[Access is denied.]
MsMpEng.exe	2.16	237,300 K	152,020 K	3724	PsProtectedSignerAntimalware-Light	[Access is denied.]
WUDFHost.exe	< 0.01	2,648 K	8,404 K	972		C:\Windows\System32\WUD...
WmiPrvSE.exe		11,968 K	27,056 K	4224		C:\Windows\System32\wbem...
winlogon.exe		2,320 K	10,288 K	772		C:\Windows\System32\winlog...

אם בתור משתמש חלש ננסה לראות את ה-DLLs של תהליך מוגן לא נמצא כלום, למעשה לא נוכל לראות שהתהליך מוגן בכלל. זה בגלל של Process Explorer עצמו עושה שימוש ב-APIs של user-mode על מנת לתשאל את המודלים שטעונים כרגע ובשביל זה נדרשת רמת גישה שלא מותרת על תהליכים מוגנים (במידה ואתם רצים תחת משתמש חזק ההרצה מתבצעת בצורה אחרת).

השיטות הפופולאריות בעבר להוצאת מידע מ-LSASS היו לטעון דרייבר או LSA plug-ins זדוניים, להצמיד אליו debugger ולהתבונן בקריאות שהוא מבצע. בדיוק כאן זורח השימוש ב-lsass.exe כ-PPL. לא ניתן לדבג PPL בזמן ריצה בכלל. שיטה נוספת שקצת יותר קשה לתת לה מענה היא לטעון דרייבר ישן (וחתום) שידוע כי יש בו פגיעות.

אוקיי אוקיי יונתן השתכנענו שאנחנו רוצים ש-lsass.exe יוגדר כ-PPL, כיצד עושים את זה? אני אחלק את התשובה שלי ל-2 - אחת ברמת מחשב בודד באמצעות ה-registry או Local Group Policy (מיוחס אל ל-Windows 11 בלבד) והשנייה ברמת הדומיין באמצעות פוליסת GPO שאפיץ לכל מחשבי הקצה. נתחיל:

הגדרת PPL במחשב בודד דרך ה-Registry

- **אוּנוֹ** - ניכנס למפתח שאחראי על LSA בריגסטרי:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa`

- **דוֹס** - נגדיר את הערכים הבאים במפתח בצורה הבאה:
  - "RunAsPPL"=DWORD: 1 במידה ובא לנו שיתבצע שימוש במשתנה תלוי UEFI.
  - "RunAsPPL"=DWORD: 2 במידה ולא בא לנו UEFI.

  אני ממליץ לעשות שימוש ב-UEFI (וגם ב-Secure Boot) כי אם לא אז ת'כלס שום דבר לא מונע מתוקף שהשיג גישה לעמדה לשנות בעצמו את ערכי הרגיסטרי.

- **טְרֶס** - לבצע reboot.
  - אלטרנטיבית, אפשר באמצעות CLI של PS בצורה הבאה:

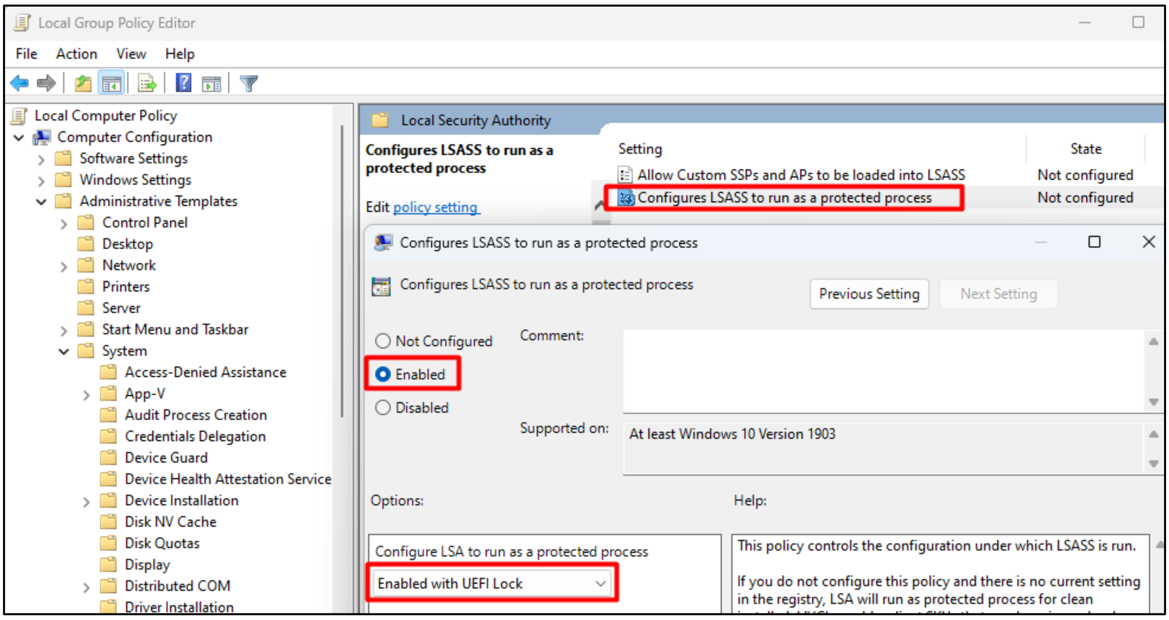
```
reg add “HKLM\SYSTEM\CurrentControlSet\Control\Lsa" /v "RunAsPPL" /t REG_DWORD
/d 1 /f
```

הגדרת PPL במחשב בודד באמצעות LGPO

נפתח את gpedit.msc ניגש אל:

Computer Configuration → Administrative Templates → System → Local Security Authority

ונגדיר את החוק "Configures LSASS to run as a protected process" בצורה הבאה:



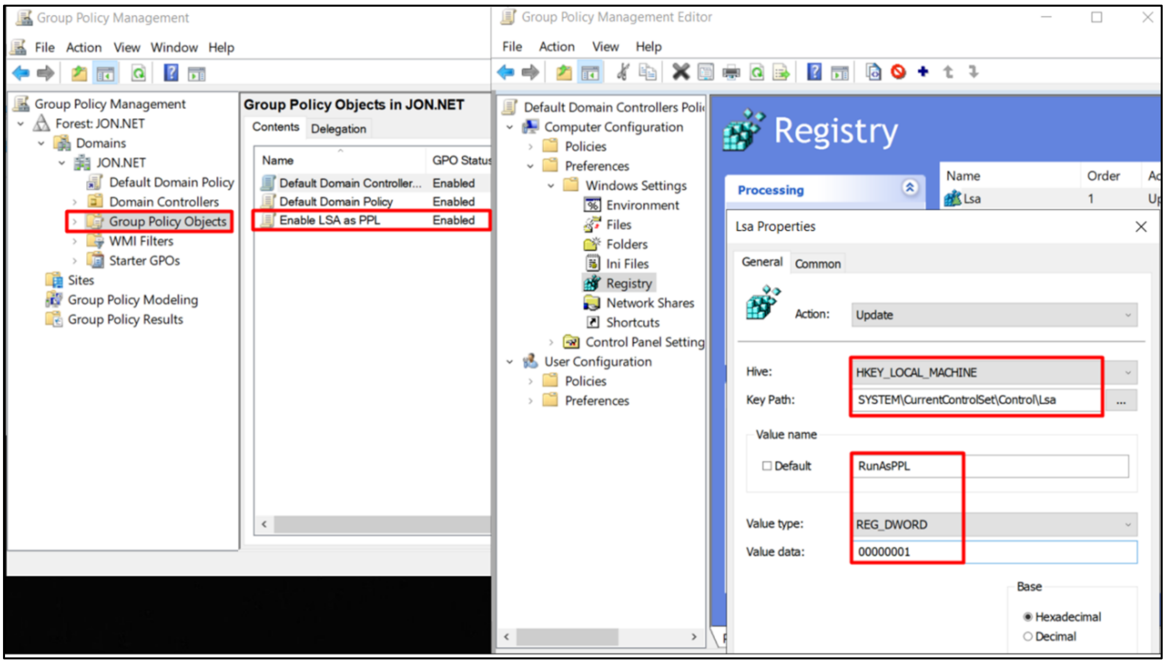
ריסטרט אחד וסיימנו.

הגדרת PPL בדומיין באמצעות GPO

אם הבנתם את השלב הקודם אזי כנראה שלא תהיה לכם בעיה להבין את השלב הנוכחי כי הוא למעשה הצעד השני באינדוקציה. כלומר, כעת פשוט נגדיר באמצעות חוקת GPO את השינוי ערכים ברגיסטרי של כלל המחשבים.

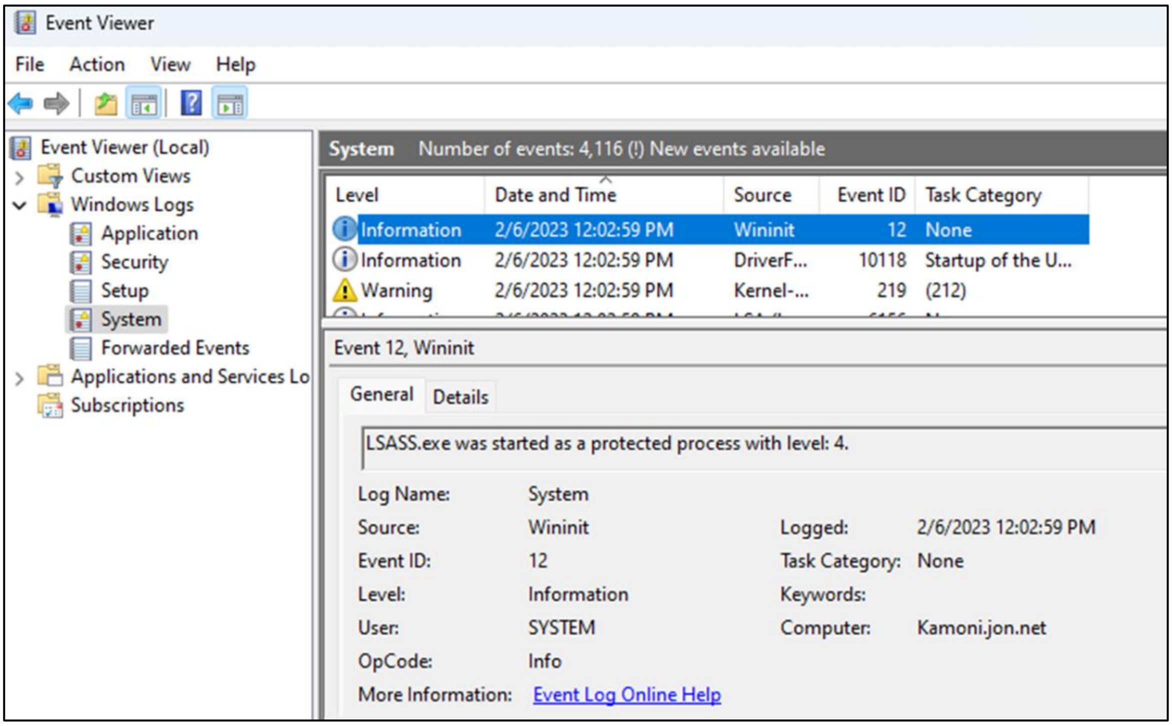
ב-DC שאחראי על ניהול האובייקטים בדומיין נפתח את קונסולת Group Policy Management וניצור חוקת GPO חדשה שמקושרת ברמת הדומיין (או אל OU) שמכילה את כלל המחשבים (אפשר גם להתלבש על חוקה שכבר קיימת אליהם אבל אנחנו אוהבים סדר אז ניצור אחת חדשה).

לאחר מכן נערוך אותה כך שתשנה את הערכים ברגיסטרי בהתאם למה שהראינו בהסבר מקודם. תמונה שווה אלף מילים אז נסתפק בתמונה הבאה:



אם המכונה מופעלת עם Secure Boot הערך עצמו יהיה מאוחסן בקושחה ולכן אותו ערך יהיה פעיל ללא קשר לשינויים ברגסטרי עצמו.

אם הגדרתם את השימוש בו ואתם רוצים לבדוק ש-LSA Protection עובד, ניתן לברר אם הוא עולה באמצעות Event Viewer:



אפשר לראות כי ה-level protection שניתן ל-lsass.exe היא 4 (PsProtectedSignerLsa לפי הטבלה הראשית ממקודם). במאמר המשך אציג יותר לעומק כיצד Mimikatz עוקפת את מנגנון PPL (אמל"ק - אחת הדרכים היא טעינת דרייבר חתום [mimidrv.sys] ואז באמצעות ה-service שלו הורדת ה-PPL של Lsass.exe. דרך אגב זה לא עובד יותר כי מייקרוסופט גנזו את התעודה שלו [דה..] אבל יש כלים אחרים שעושים שימוש בתוכנות כמו procexp64.exe בשביל לבצע משהו דומה).

אבל רגע רגע! לפני שאתם רצים להטמיע את השינויים האלו בכל הסביבה הארגונית שלכם, קיים שיקול שחייבים לקחת אותו בחשבון והוא **האם אתם משתמשים במודל אימות של צד שלישי?** בפרק על SSPI הסברנו שקיימים Authentication modules שניתן להוסיף.

במידה והארגון שלכם עושה שימוש באחד כזה, הוא **חייב** להיות חתום דיגיטלית עם החתימה של Microsoft (לא נתון הכי טריוויאלי).

כמו כן, חשוב לציין כי ב[דוקומנטציה] של Microsoft בנושא אבטחת LSA הם מציינים בנוסף גם כמה פעולות לביצוע Audit וניטור אחר פעולות חשודות (כמו טעינת דרייברים אל lsass.exe). אני חושב שזה חסר טעם להעמיק בנושא במאמר כי כנראה לכל מערכת SIEM כבר יש 100 חוקות בנושא.

Third-party PPLs

ניתן להגדיר כי תהליכים נוספים (שלא נכתבו על ידי מייקרוסופט) יהיו גם חתומים בתצורת PPL. תוכנות anti-malware (AM) ותוכנות המבצעות דיפלוימנט בהיקף רחב הן המחשה טובה לצורך הנ"ל. לרוב, מוצר AM יהיה מורכב מ-3 מרכיבים:

- קרנל דרייבר שמפרש את בקשות ה-I/O למערכת הקבצים ו\או הרשת, ומיישם מנגנוני חסימה בהתאם ל-traffic שהוא קולט.
- שירות user-space (לרוב שרץ תחת משתמש עם הרשאות חזקות) שיכול לערוך את חוקות הדרייבר, לקבל הודעות מהדרייבר בנוגע ל-events ולתקשר עם שרת לוקאלי או האינטרנט.
- תהליך GUI ב-user-mode שמתקשר עם המשתמש להעברת מידע ומאפשר למשתמש לקבל החלטות. כלומר, אותו פאנל גדול כזה עם pie charts שמעדכן תמונות סטאטוס שכולנו מכירים.

הסיבה להוספתם אל משפחת ה-PPL ברורה - גם במצב בו משתמש זדוני השיג הרשאות גבוהות, הוא לא יוכל להפסיק את ריצת ה-AM ו\או להזריק לה קוד (לפחות כל עוד הוא לא עושה שימוש באקספלויטים ברמת הקרנל). כמובן שעל מנת לקבל את האישור של מייקרוסופט הם נדרשים לעבור תהליך אימות מנהלתי כאשר בסופו שדה ה-EKU בתעודה יכיל את שדה ה-PPL של:

PS_PROTECTED_ANTIMALWARE_LIGHT (0x31)

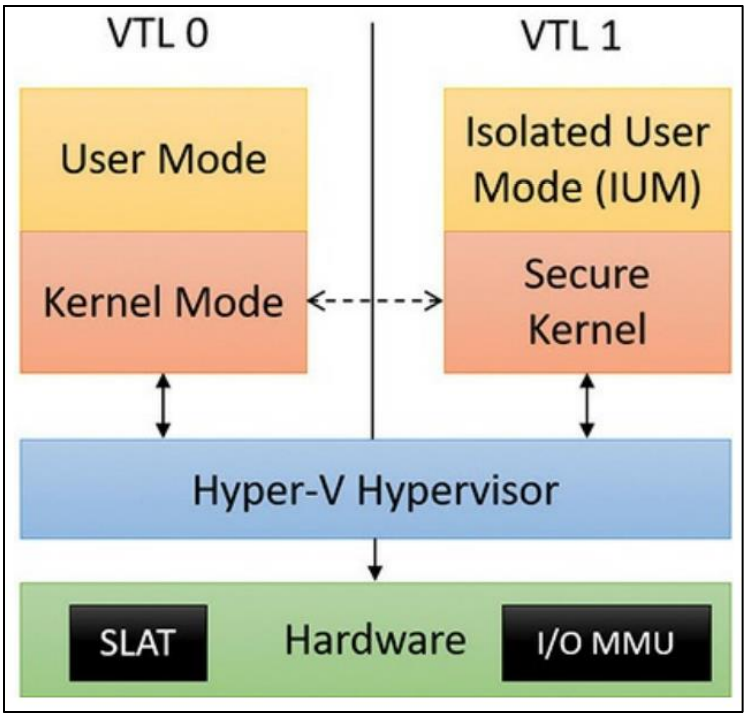
שכאמור נמצא ברמה אחת לפני האחרונה. ולכן גם במידה והשתלטו על תוכנת ה-AM, ההרשאות שלה יהיו יחסית נמוכות ביחס ל-PPLs אחרים (לשם דוגמה, היא לא תהיה מסוגלת לבצע dump אל lsass.exe).

חלוקה לשני עולמות - Virtual Trust Levels

על מנת להסביר על Credential Guard אנחנו צריכים לעשות עיקוף קטן ולהסביר קודם על Virtual Trust Levels (רמות אמון וירטואליות בעברית שבורה) בקרנל(ים) של Windows. ה-Hyper-V של מיקרוסופט הוא חלק בלתי נפרד מתהליך עליית מערכת ההפעלה ואף נטען לפני הקרנל (לעומת מימושים אחרים בהם ה-Hypervisor רץ כולו בקרנל). למעשה, חלקכם אולי יופתעו לגלות כי עקב היותו Hypervisor type 1, מערכת ההפעלה Windows שכולנו מכירים רצה כמכונה וירטואלית בצורה דומה לכך שבאמצעות WSL2 ניתן להריץ מכונת Ubuntu על גבי התשתית של Hyper-V. ליתר דיוק, כל מחיצה ב-Hyper-V מהווה **יחידה לוגית המיישמת מכונה וירטואלית מבודדת** משאר המכונות. לכל מחיצה יש מעבד וירטואלי שמנהל את ריצת התהליכים - מה שגורר ניהול context switch ברמת המעבד האמיתי של ה-Hyper-V עצמו. הגמישות שרמות אבסטרקציה מאפשרות הוא אינסופי.

המאמר Hyper-V Architecture for Security Researcher מאת דני אודלר בגליון 114 מדבר בדיוק על זה. שווה מאוד את הקריאה למי שרוצה להבין לעומק כיצד Hyper-V עובד. רמות אמון וירטואליות מדברות על אימפלמנטציה של 2 שכבות אבסטרקציה שה-Hypervisor מיישם עבור eco-system של קוד ברמות אבטחה ואמון שונות - **VTL1** & **VTL0**. בעברית פשוטה, מערכת ההפעלה והרכיבים שלה רצים ב-VTL0 בעוד שטכנולוגיות כדוגמת VBS (Virtual Based Security) רצות ב-VTL1 עם יכולות שונות לחלוטין. שום קוד ב-VTL0, אפילו אם הוא של הקרנל עצמה, לא יוכל להשפיע על התוכנות שרצות תחת VTL1 מכיוון שהוא לא מסוגל להיות מודע לקיומן.

קוד שרץ ב-VTL0 ישאר ב-"מימד" הזה ויוכל להשפיע רק על תהליכים שרצים באותו מימד. מצד שני, חשוב להבין ש-VTL1 מכיל את קוד הקרנל משלו שרץ ב-processor mode עם הרשאות גבוהות (ring 0 ב-x86/x64) וגם, בצורה דומה, אזור user-mode (שנקרא **IUM** - Isolated user mode) קיים. ניתן לחשוב על VTLs כפתרון אורתוגונאלי אל רמות ההשראות שבמעבד: kernel mode ו-user mode קיימים **בתוך כל VTL** וה-hypervisor מנהל את ההרשאות לאורך כל ה-VTLs.



[מקור - Window internals]

המון מילים שוברות שיניים נאמרו עד כה אז נקפוץ ישר לפרקטיקה בשביל להמחיש את הנושא - הקרנל המאובטחת של VTL1 מכילה בינאריים משלה ויושבת בדיסק בקובץ **securekernel.exe**. בצורה דומה, שחקן נוסף ששווה להזכיר בכל הקשור ל-VTL1 הוא **Iumdll.dll** שמקביל אל Ntdll.dll (ב-VTL0) ומאפשר ביצוע system calls עבור הקרנל של VTL1.

נקודה ששווה לתת עליה את הדגש היא שלפי השרטוט ניתן לראות כי אין אפשרות בכלל לתקשר בין kernel mode ב-VTL0 אל user-mode ב-VTL1 (בעל הרשאות גבוהות יותר). עם זאת, גם הצד ההפוך נכון - אי

אפשר לגעת עם קוד מבוסס kernel mode ב-VTL1 בקוד user mode ב-VTL0 מכיוון שהחוקיות של משתמש (ring 3) לא יכול לגעת בקוד קרנל (ring 0) נשמרת.

דרך טובה להבין את כל הבאלגן הזה בטרמינולוגיה היא להסתכל על כל עיניין של ה-VTLs **בפרספקטיבה של בידול ולא מפרספקטיבה של כוח.** אפליקציית user-mode ב-VTL1 לא יותר חזקה מקוד אפליקציה או דרייבר של VTL0 אלא היא מבודדת ממנו.

למעשה, שם אלטרנטיבי בדוקומנטציה ל-secure kernel של VTL1 הוא "**proxy kernel**". כלומר, לא רק שיישומי VTL1 אינם חזקים יותר מיישומי VTL0, במקרים רבים יש להם הרבה פחות יכולות. מה הכוונה? ובכן, מכיוון שהקרנל של VTL1 לא מיישמת מגוון שלם של יכולות מערכת, היא הלכה למעשה בוחרת ידנית אילו system calls היא תעביר לקרנל של VTL0. כל סוג של פעולות I/O, כולל קבצים, רשת ו-registry אסורים לחלוטין. מה עם פעולות גרפיקה? פפפפפ הצחקתם, אינן באות בחשבון; לשום דרייבר לא מותר לתקשר ולבנות GUI. קיצר, ב-VTL1 הכל מתבצע ומתנהל בתצורה סטטית.

ה-secure kernel, על ידי ריצה גם ב-VTL1 וגם מהיותה kernel mode למעשה בעלת רמת אמון גבוהה מאוד. משם כך, יש לה גישה מלאה לזיכרון ולמשאבים של VTL0. הקרנל, באמצעות ה-hypervisor, יכולה להגביל את הגישת מערכת ההפעלה של VTL0 אל מרחב כתובות זיכרון על ידי מינוף תמיכת חומרת ה-CPU. פעולה זאת נקראת בדוקומנטציה Second Level Address Translation, או **SLAT** בקיצור *(אם תשאלו אותי היו צריכים לשנות את הביטוי אל Second Level Unconstrained Translation, ללא ספק ראשי תיבות היו מעניינים יותר)*. SLAT הוא *(היא?)* הבסיס של טכנולוגיית Credential Guard שמאפשרת לשמור סודות במרחב כתובות כזה. נרחיב על Credential Guard בפרק הבא ונראה כיצד הוא משתמש באותה וירטואליזציה בשביל להגן על LSASS.

כדי למנוע מדרייברים רגילים למנף התקני חומרה ולגשת ישירות לזיכרון, נעשה שימוש בחומרה נוספת המכונה I/O memory management unit (MMU), אשר עושה וירטואליזציה לגישה לזיכרון עבור התקנים. זה יכול לשמש כדי למנוע התקני דרייברים להשתמש בגישה ישירה לזיכרון (DMA), כדי לגשת ישירות ל-hypervisor או לאבטח את אזורי הזיכרון הפיזיים של הקרנל. בכך למעשה יעקפו את SLAT מכיוון שלא מעורב זיכרון וירטואלי כלל בתהליך.

מכיוון שתהליכי user-mode הפועלים ב-VTL1 מבודדים, קוד זדוני פוטנציאלי עשוי לנסות לבצע system calls (שיאפשרו לו לחתום על סודות ובכך להיראות לגיטימי), וכן עלול לגרום לאינטראקציות זדוניות עם תהליכי VTL1 אחרים או עם ה-secure kernel. ככזה, רק מחלקה מיוחדת של קבצים בינאריים חתומים במיוחד, הנקראת **Trustlets**, מותרת להפעלה ב-VTL1. לכל Trustlet יש מזהה וחתימה ייחודיים, ולקרנל יש ידע מקודד של אילו Trustlets נוצרו עד כה. בצורה כזו, אי אפשר ליצור Trustlets חדשים ללא גישה לקרנל (שכאמור רק מייקרוסופט יכולה לשחק איתו), ולא ניתן לבצע patch אל Trustlets קיימים בשום צורה (מכיוון שהפעולה תבטל את החתימה).

Windows Defender Credential Guard

אם בעבר היה ניתן לשמור את סודות המשתמש רק בזכרון LSASS כעת על ידי שימוש בוירטואליזציה המצב קצת שונה. כאשר Credential Guard מופעל במכונת Windows מעתה ואילך יהיו 2 תהליכי LSASS - הרגיל, lsass.exe, זה שאנחנו מכירים ותהליך מבודד נוסף, **lsaiso.exe**, אשר רץ בתוך מכונה וירטואלית של Hyper-V. כלומר תוקף **יצטרך לעבור את ה-hypervisor** במידה והוא רוצה לגשת לנתונים של המשתמש (ולעשות dump ל-credentials) - וזו משימה אחרת לגמרי מאשר תקיפה של תהליך בודד.

מכיוון ש-LSAIso מסוגל לתקשר רק עם מספר בינאריים מאוד מצומצם של מערכת ההפעלה שחתומים על ידי Microsoft בלבד והוא לא משתמש בשום דרייבר חיצוני, משטח התקיפה עליו מצטמצם משמעותית.

כמו כן, כאשר מפעילים את Credential Guard ניתן לבצע SSO רק עם Kerberos. זה עתיד ליצור לא מעט אי-נוחות מכיוון שחבילות אימות אחרות כגון NTLMv1, WDigest ו-CredSSP (עליהן הסברנו מקודם מפרק של SSPI) נחסמות לשמירת Credentials ולכן לא ניתן לבצע SSO באמצעות הפרוטוקולים שהן מייצגות (אבל כן ניתן להזדהות באמצעותן). בנוסף, פרוטוקול Kerberos לא מאפשר יותר מעבר של unconstrained delegation וגם במצב PKINIT הוא ישתמש ב-RSA במקום ב-Diffie-Hellman. אך עם זאת, אימות מבוסס תעודות (כמו פרוטוקולים EAP-TLS ב-802.1x) מתאפשר ו-Credential Guard לא חוסם כלל.

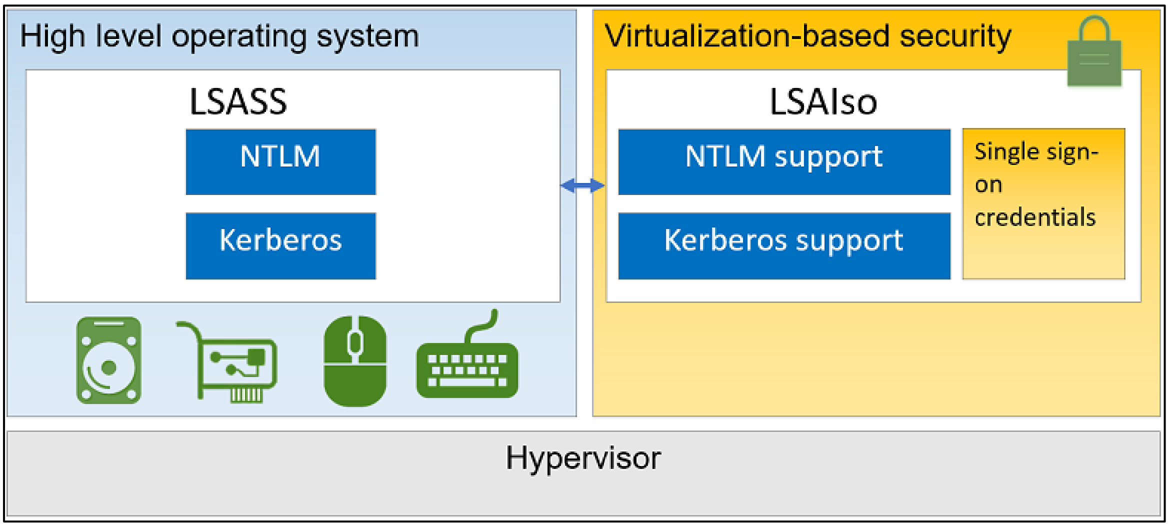
מכניזם נוסף ששווה להרחיב עליו בכמה מילים הוא **Device Guard**. בעוד ש-Credential Guard מתעסק ברובו בהגנת ה-Credentials של המשתמש (ומשם גם מגיע השם שלו), Device Guard מגיע במטרה להגן על כלל מערכת ההפעלה.

הוא מבוסס מנגנון whitelisting של יישומים מבוססי חומרה כדי להגן על devices (ומשם מגיע השם שלו) מפני תוכנות זדוניות ואיומים אחרים. הוא משתמש בשילוב חוקות מבוססות תוכנה בשביל להבטיח שרק אפליקציות מורשות יוכלו לרוץ במערכת ההפעלה ועם Secure Boot ו-VBS

הערה: בחלק מהדוקמנטציות מתייחסים אל VBS בתור VSM. למרות שקיים שוני טרמינולוגי קטן בין השניים, במהלך המאמר אתייחס אליהם בצורה זהה תחת השם VBS מכיוון שהוא השם הכללי של הטכנולוגיה.

דוגמה פרקטית ל-Device Guard היא היכולת שלו למנוע טעינה לזכרון של DLL-ים לא חתומים או untrusted ובכך למנוע DLL injection.

תמונה מפורסמת שחוזרת כמעט בכל בלוג אבטחת מידע שמתעסק ב-Credential Guard לקוחה מה[דוקומנטציה](#) של Microsoft בנושא:



[מקור - Microsoft]

מדובר באיור הממחיש ב-high level כיצד מתבצע הבידוד של LSASS באמצעות ה-hypervisor. בפרק הבא כאשר נדבר על נקודתית על הפתרונות, ניכנס יותר לעומק של המנגנון וכיצד הוא "מגן באמת" על נתוני המשתמש. אז כיצד מגדירים את השימוש ב-Credential Guard? בדומה ליישום ב-PPL, גם כאן אני אחלק את התשובה שלי ל-2 - אחת ברמת מחשב בודד באמצעות ה-registry או Local Group Policy והשנייה ברמת הדומיין באמצעות פוליסת GPO שאפיין לכל מחשבי הקצה.

חשוב לציין כבר מעכשיו שהחל מ-Windows 11 22H2 המנגנון [מאופשר אוטומטית](#). אירוני בהתחשב בכך שכבר שנים לא מעט חוקרים (כולל כותב שורות אלו) [מסתבכים להפעיל אותו במכונה וירטואלית](#). אז כיצד ניתן להפעיל אותו?

הגדרת Credential Guard במחשב בודד דרך ה-Registry

כאמור מנגנון האבטחה הנ"ל עושה שימוש ב-VBS ולכן ראשית נדרש לודא כי קודם Device Guard מאפשר שימוש בו ורק אז להגדיר את Credential Guard.

1. ניכנס לנתיב שאחראי על Device Guard ברגיסטרי:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\DeviceGuard`

ונוסיף שני מפתחות:

- הראשון `"EnableVirtualizationBasedSecurity"=DWORD: 1` בשביל לאפשר VBS.
- השני `RequirePlatformSecurityFeatures` גם מסוג DWORD. במידה ואנחנו רוצים רק שימוש ב-Secure Boot ניתן לו את הערך "1". במידה ואנחנו מכוונים גם ל-Secure Boot וגם אל DMA Protection ניתן לו את הערך "3".

2. ניכנס לנתיב שאחראי על LSA ברגיסטרי:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa

ונוסיף ערך DWORD בשם LsaCfgFlags אשר יכול לקבל שלושה ערכים:

- "0" - בשביל לבטל את Credential Guard
- "1" - בשביל לאפשר את Credential Guard עם UEFI lock
- "2" - בשביל לאפשר את Credential Guard ללא UEFI lock

אישית אני ממליץ לאפשר VBS ו-Secure Boot אבל לא UEFI lock למקרה ויהיה צורך לשנות את ההגדרה ממחשב מרוחק (ללא גישה ל-firmware). מצד שני, אני גם ממליץ לא להקשיב למאמרים רנדומאליים באינטרנט ותמיד לעשות את המחקר לכדאיות של כל שינוי בעצמכם. אלטרנטיבית, אפשר פשוט להגדיר את הכל ברגיסטרי דרך CLI של PS בצורה הבאה:

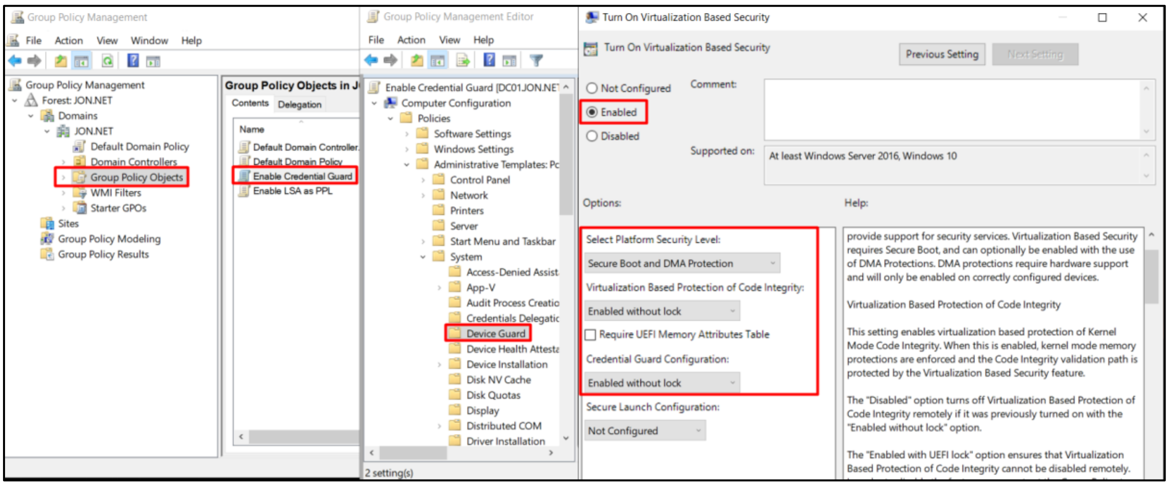
```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\DeviceGuard" /v
"EnableVirtualizationBasedSecurity" /t REG_DWORD /d 1 /f

reg add "HKLM\SYSTEM\CurrentControlSet\Control\DeviceGuard" /v
"RequirePlatformSecurityFeatures" /t REG_DWORD /d 3 /f

reg add "HKLM\SYSTEM\CurrentControlSet\Control\DeviceGuard" /v "Locked" /t REG_DWORD /d
0 /f
```

הגדרת Credential Guard בדומיין באמצעות GPO

כעת נגדיר באמצעות חוקת GPO את השינוי ערכים ברגיסטרי של כלל המחשבים בדומיין. ב-DC שאחראי על ניהול האובייקטים בדומיין יש לפתוח את קונסולת Group Policy Management וליצור חוקת GPO חדשה שמקושרת ברמת הדומיין (או לא OU) שמכילה את כלל המחשבים. כמו שכבר אמרתי מקודם בהסבר על PPL, תמונה שווה אלף מילים אז נסתפק בתמונה:



לאחר מכן כל שנשאר הוא לעדכן את חוקות ה-GPO על ידי הרצת הפקודה:

```
gpupdate /force
```

על מנת לבדוק ש-Credential Guard אכן מוגדר במחשב ניתן להריץ את פקודת PowerShell הבאה:

```
(Get-CimInstance -ClassName Win32_DeviceGuard -Namespace
root\Microsoft\Windows\DeviceGuard).SecurityServicesRunning
```

כאשר הפלט שמתקבל הוא "1" - Credential Guard מאופשר ובמצב ריצה. ובמידה הפלט הוא "0" - אז Credential Guard לא עובד.

הפרטים הקטנים - איך הכל מנגן ביחד

דיברנו על מלא נושאים שונים החל מכיצד נראה תהליך אסיפת ה-credentials ב-Windows, אילו פרוססים משתתפים בתהליך ומהם מנגנוני ההגנה שאמורים להגן עליהם. אבל לא הסברנו כיצד זה קורה - כיצד PPLs ושימוש ב-Credential Guard באמת מצליחים *(והאם הם באמת מצליחים?)* להגן על LSASS. ובכן, זה הזמן לחבר את כל הנושאים האלו ביחד.

אחרי תהליך logon מוצלח לעמדה, תהליך lsass.exe יחזיק בזיכרון לעיתים את **סיסמת המשתמש** בתצורת clear text, עם זאת במידה ומדובר באימות מבוסס NTLM הוא יחזיק בתוכו את ה-hash של סיסמת המשתמש (נקרא גם **NTOWF** - NT one-way function) אשר מבוסס על MD4. בצורה דומה כאשר תהליך ה-Logon מבוסס על Kerberos, תהליך lsass.exe יכיל בכתובות הזיכרון שלו את מבני הנתונים של **TGT** (Ticket Granting Ticket). כאשר הודעת KRB_AS_REP חוזרת היא מכילה את ה-TGT חתום על ידי סיסמת היוזר krbtgt ביחד עם מפתח session שחתום על ידי ה-hash של סיסמת המשתמש שהגיש את הבקשה. זה מדוע במסגרת מתקפת AS-REP roasting ניתן להשיג את ה-hash של היוזר (ולשבור אותו כנגד מילון ב-offline). לאחר מכן, המשתמש יכול להגיש את ה-TGT שוב ל-KDC בשביל לקבל TGS מוצפן באמצעות ה-service hash שאליו הוא רוצה להזדהות. כך, השרת יוכל לקלף את ההצפנה ולאמת את ה-TGS. כאן נכנסת מתקפת kerberoast שפועלת על אותו מנגנון על מנת לשבור את ה-hash של ה-service account. ממליץ על המאמר [2nd Step to Tame a Kerberos: Hit It Where It Hurts](#) לאלו המעוניינים להבין טוב יותר את הנושא.

בשביל להגן על סיסמת המשתמש, על ה-TGT ועל ה-NTOWF, נעשה בפרוסס של LSASS שימוש ב-**Credential Guard**. כיצד הוא עוזר להגן על כל סוג credential?

<u>הגנה על סיסמה</u> - סיסמת המשתמש מוצפנת באמצעות מפתח סימטרי (AES) בשביל לאפשר SSO באמצעות פרוטוקולים כמו RDP. מכיוון ששירותים אלו עושים שימוש בסיסמת clear text היא חייבת להישמר בזיכרון התהליך - מה שמאפשר לתוקפים לבצע code injection, שימוש בכלי debugger, שיטות אקספלוטציה נוספות ופיענוח. Credential Guard לא יכול לשנות את אופן היישום של אותם פרוטוקולים לא בטוחים. לכן, "הפתרון" היחיד שהוא מספק במקרה של סיסמאות clear text הוא **לבטל את הפונקציונאליות של SSO** (כמובן שזה לא כיף ומכריח ביצוע re-authenticate). זו גם הסיבה מדוע

מייקרוסופט ממליצים לבטל לחלוטין את השימוש בסיסמאות ולעבור לעשות שימוש במזהים ביומטרים כמו Windows Hello שעובד עם תווי פנים\ טביעת אצבע. בצורה כזו, כאשר מתבצע אימות ניתן להוריד את משטח התקיפה משמעותית מפני מתקפות keyloggers, kernel sniffing/hooking tools ואפליקציות user-mode שמבצעות spoofing. כמו כן, שימוש ב-MFA חזק כמו YubiKey או Smartcard בצמוד ל-PIN יכול לעשות את העבודה.

לפי STIG, על מנת לבטל את השימוש ב-Wdigest כשיטת אימות ניתן לשים ב-registry תחת:

KEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest

את הערך UseLogonCredential עם DWORD שווה 0.

על מנת להגן על NTOWF או TGT שמאוחסנים ב-LSASS נדרש להגן על LSASS בשלמותו - כאן נכנסים לתמונה תהליכי PPL עליהם דיברנו מקודם. אם התהליך חתום ומוגן, תוקף פוטנציאלי לא יוכל לגשת אליו. עם זאת, חשוב להבין כי אופציה זו מספקת הגנה מפני תהליכים זדוניים במרחב ה-user-mode; היא ממש לא מספקת הגנה מפני מתקפות kernel או מתקפות user-mode שמנצלים חולשה ב-kernel drivers (שכאמור קיימים לא מעט שניתן לטעון, ארחיב על הנושא במאמר אחר).

ובכן, Credential Guard מבטל את משטח התקיפה הנ"ל בכך שהוא טוען תהליך נוסף - **lsaiso.exe** אשר רץ בתור Trustlet בתוך VTL1. לכן, LSASS למעשה לא מחזיק את הסודות של המשתמש - אלא lsaiso מחזיק ואליו התוקף לא יכול בכלל לגשת כי הוא נמצא ב-"מימד קוד" אחר.

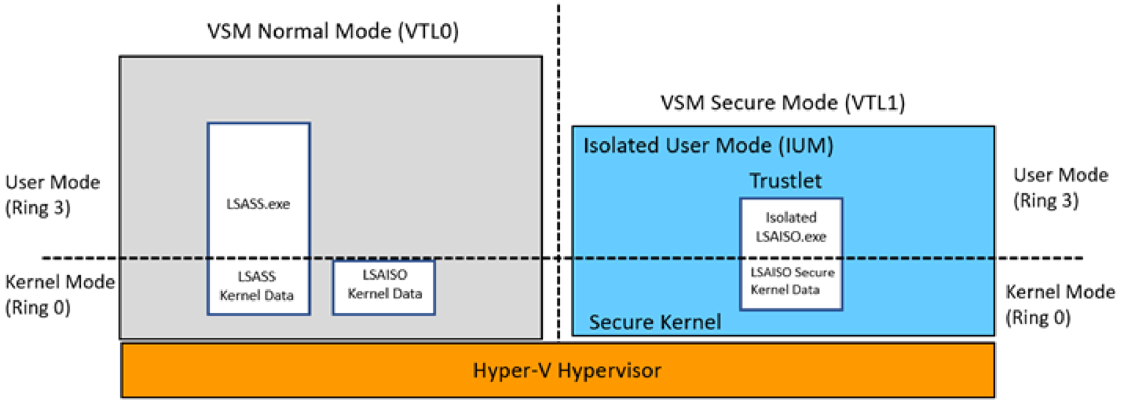
ל-VTL1 יש משטח תקיפה מינימלי (קטן משמעותית לעומת VTL0 שמוכרת לנו), מכיוון שאין לה את ליבת ה-"NT" הרגילה המלאה, וגם אין לה מנהלי התקנים או גישה ל-I/O של חומרה מכל סוג שהוא. ככזו, LSA מבודד (שהוא VTL1 Trustlet), אינו יכול לתקשר ישירות עם KDC.

זוהי עדיין אחריותו של lsass.exe, המשמש כיישום ופרוטוקול proxy על מנת לתקשר עם ה-KDC כדי לאמת את המשתמש ולקבל את ה-TGT (ואת המפתח) וה-NTOWF. LSASS גם אחראי על התקשורת עם שרת אליו אנחנו מזדהים באמצעות ה-ticket הרלוונטי (TGS).

אם עקבתם אחרי הפסקה האחרונה, כנראה שתפסתם את הבעיה העקרונית בארכיטקטורה של התהליך הזה - ה-TGT והמפתח/NTOWF שלו עוברים באופן זמני דרך LSASS במהלך האימות, וה-TGT והמפתח שלו זמינים *איכשהו* ל-LSASS להפקת service ticket.

זה מוביל לשתי שאלות:

- איך LSASS שולח ומקבל את הסודות מ-LSA המבודד?
- כיצד נוכל למנוע מתוקף להתחקות בצורה לגיטימית? הרי חייבת להיות תקשורת בין lsaiso.exe אל lsass.exe. מה מונע מתוקף לבצע מתקפות MITM?

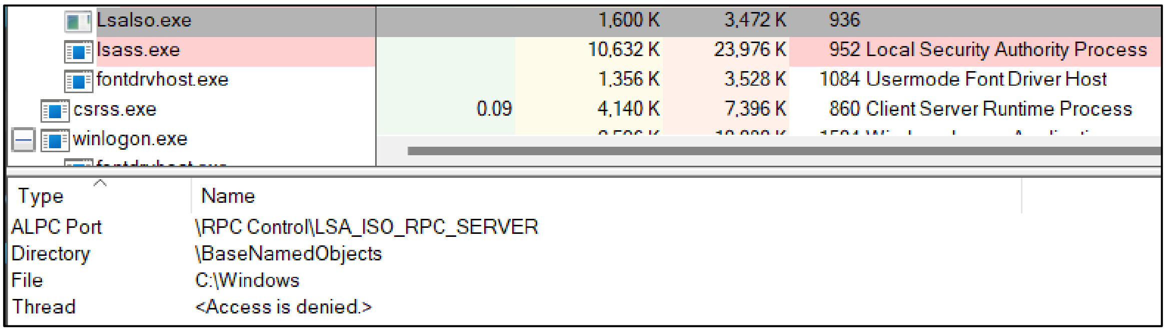


[מקור - Microsoft]

לגבי השאלה הראשונה, התשובה היא **Advanced Local Procedure Call** (ALPC). מדובר במכניזם של תקשורת העושה שימוש בסט קריאות מערכת שתומכות בביצוע RPC לוקאליות על ידי ncalrpc מרחוק. אופן העבודה של הפרוטוקול הוא על ידי פתיחת פורטים מבוססי ALPC שמאפשרים תקשורת בצורה דינאמית בין תהליכים ברמות שונות של אמון.

על ידי שימוש ב-ALPC ה-secure kernel מסוגלת לעביר את קריאות המערכת אל הקרנל הרגילה. לאחר מכן, ה-user-mode של VTL1 מיישם תמיכה ב-RPC על גבי פרוטוקול ALPC אשר מאפשר לאפליקציות VTL1 ואפליקציות VTL0 לתקשר ביניהן בדיוק כמו שהיו עושות עם RPC לוקאלי.

למעשה אם נפתח את ה-Process Explorer שוב נוכל לראות את תהליך lsaiso.exe עם ה-handle ופורט ALPC שלו (אלו למעשה מתקשרים עם lsass.exe):



בשביל לענות על השאלה השנייה (MITM) נדרש לערב קצת קריפטוגרפיה ומודל של Challenge-Response בכל הסיפור. מניעת מתקפת MITM בין ה-KDC ופרוטוקול ה-LSA המבודד דומה במיוחד לאופן פעולת SSL/TLS ואופן פעולת האינטרנט (כבר ציינתי כמה פעמים שרשמתי מאמר קצרצר על כך).

למרות ש-LSASS מתנהג כמו proxy, הוא למעשה רק רואה ומעביר **תעבורה מוצפנת** בין ה-KDC ותהליך ה-LSA המבודד. שני הצדדים מגבשים תקשורת מבוססת על session key (מפתח סימטרי) אשר נמצא רק ב-VTL1 וב-KDC. ה-KDC מצפין את אותו session key עם מפתח פרטי שרק לו יש, בצורה הזו ה-KDC יכול לענות עם TGT והמפתח שלו אחרי שהוא הצפין את התעבורה עם ה-session key של ה-LSA.

המודל הנוכחי הנ"ל מסוגל להגן אפילו על אימות מבוסס NTLM, כיצד? כאשר המשתמש מבצע logon עם סיסמת plain text, LSA שולח אותה אל תהליך ה-LSA המבודד, אשר בתורו מצפין את הסיסמה עם ה-session key שלו ומחזיר את ה-credentials אל lsass.exe. לאחר מכן, כשנדרש לבצע את ה--Challenge Response תהליך LSASS שולח את ה-NTLM challenge ביחד עם ה-credential המוצפן אל ה-LSA המבודד. הוא בתורו מפענח את התכולה, מוודא שהיא נכונה ושולח את ה-NTLM response על בסיס ה-challenge בתמורה.

המודל הנוכחי לא מושלם. למעשה, אפשר לחלק את פגיעותיו ל-4 קטגוריות של מתקפות:

- במידה והמכונה כבר נמצאת תחת שליטת התוקף, ניתן ליירט את הסיסמאות plain text בזמן שהן מוקלדות (Keylogger) ו\או בזמן שהן נשלחות לתהליך ה-LSA המבודד (במידה ו-LSASS נתון לשליטת התוקף).
- מכיוון שאין ל-NTLM מנגנון anti-replay ניתן לתפוס את ה-NTLM response ולענות באמצעותו שוב על אותו ה-challenge. לחילופין, במידה ותוקף הצליח להשתלט על LSASS לאחר ביצוע ה-logon הוא יוכל לתפוס את ה-credentials המוצפנים ולשלוח אותם ל-LSA המבודד בשביל ליצור NTLM response חדש עבור NTLM challenges אקראיים. עם זאת, שיטת התקיפה הזו רלוונטית רק עד ביצוע reboot מכיוון שבזמן עליית המערכת ה-LSA המבודד יוצר session key חדש.
- במקרה של התחברות Kerberos-ית, ניתן ליירט ה-NTOWF (שכאמור לא מוצפן) ואז לעשות בו שימוש מחדש - בדיוק כמו במתקפה סטנדרטית של pass-the-hash. כמובן שגם פה נדרשת השתלטות מבעוד מועד על המכונה.
- Downgrade - תוקף עם גישה פיזית למכונה יכול לבטל את Credential Guard (חשוב לזכור שבסה"כ מדובר בהגדרה ב-registry במידה ולא מוגדר UEFI lock) ואז לעשות שימוש במודל אימות legacy. בקו מחשבה דומה, ניתן לבטל את הגדרת ה-Protected Process ולבצע Reboot למכונה. כמובן ששימוש ב-Secure Boot ו-UEFI ימנע את שינוי ה-Registry.

סיכום

נקודה שקצת הוזנחה במאמר היא ההיסטוריה של כל הסיפור. מאיפה הגיע הצורך להוסיף מגננות על מגננות, לבודד תהליכי אימות ולהכיל חתימה על כל דבר שזז? התשובה היא שפשוט עבר יותר מידי זמן. סט האיומים שפרוטוקולים כמו WDigest ו-NTLM גדלו לתוכם פשוט לא תואם אל ה-design שלהם (ובטח ובטח של AD בכלליותו). אם בעבר להעביר hash של הסיסמה עצמה נחשב לבטוח, הרי שהיום כרטיס מסך GeForce RTX 4090 של Nvidia יכול לפצח [300 מליארד](#) (!) גיבובי NTLM בשנייה. פרוטוקולים שנבנו בשנות ה-90 פשוט לא תוכננו לכזו רמה של עמידות.

מסיבה זו בדיוק הגיעו מנגנוני האבטחה PPL ו-Credential Guard לחיזוקו של LSASS וכלל מערכת ההפעלה. ותאמת, הם שמים את Microsoft במצב די טוב. Mimikatz כבר לא רלוונטי (לפחות לא לבדו) וללא ספק קשה היום משמעותית לחלץ מידע מ-LSASS. אין זה אומר שאלו יספיקו בשביל למנוע מתוקף להתפשט ברשת אבל הוא יצטרך להשתמש באמצעים מתקדמים (ומסובכים) בשביל לעקוף אותם, מה שבסופו של דבר **יגדיל את הסיכוי שלו להתגלות**.

ישנם כלים התקפיים שמשכו תשומת לב ברמת החידוש והיצירתיות שלהם. בין אם זה ניצול מנגנון פנימי של התהליך או סביבתו ובין אם זה גילוי\המצאת וקטור תקיפה חדש ברמת התשתית. משפט זה מוביל אותנו בצורה טובה לסיום המאמר וכהכנה להמשך.

במאמר הבא אני מתכוון להיכנס לחלק מהכלים הבולטים לתקיפת LSASS, להריץ אותם בסביבת דומיין ולהסביר ברמה טכנית כיצד הם פועלים. רשימת הכלים הנוכחית (אשר צפויה להתעדכן) הינה: [Mimikatz](#) (כמובן), [PPLdump](#), [NanoDump](#), [RToolZ](#), [PPLcontrol](#), [EDRSandblast](#), [pypykatz](#).

אשמח לנצל את הבמה ולומר שבמידה ואתם משתמשים/מכירים כלי **קוד-פתוח** שמצליח לחלץ נתונים חרף כל מנגנוני ההגנה של Microsoft ולא נמצא פה ברשימה - תשלחו לי כדי שאעבור על הקוד שלו ואולי נציג אותו במאמר הבא ☺

על הכותב

[יהונתן אלקבס](#), חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב סוקולנטים, קפה וטיולים.

Cisco Passwords

מאת אמיל לוין ודר פיינשטיין

הקדמה

במהלך בדיקות חדירות, אין זה דבר נדיר למצוא קובץ קונפיגורציה של רכיב Cisco.

גיבוי של הקונפיגורציה יכול להימצא על מחשב כלשהו ברשת או על שרת שיתוף קבצים. בנוסף, תוכנות מסוימות כמו PuTTY יוצרות קובץ console log שיכול להכיל חלקים של קונפיגורציות של הרכיב אליו מתחברים.

לפעמים מבוצעים גיבויים של הקונפיגורציות בפרוטוקולים העוברים ברשת ב-Clear Text, כמו FTP ו-TFTP. כך ניתן להשיג קונפיגורציות של הרכיב באמצעות הסנפה של התקשורת. כמובן שפעמים רבות ניתן פשוט להתחבר לרכיב Cisco ולהשיג את הקונפיגורציה שלו. בדרך כלל, כשתוקף מצליח להתחבר לרכיב Cisco, הפקודה הראשונה שהוא יריץ היא *show running-config* על מנת לקבל את הקונפיגורציה הנוכחית והעדכנית של הרכיב. התוקף בד"כ מעוניין במידע רגיש המופיע בקונפיגורציה כמו סיסמאות שמורות ו-SNMP Community Strings. בנוסף, תוקף עשוי להשתמש בהגדרות התקשורת על מנת להתפשט ברשת.

במאמר זה נתמקד בסוגי המשתמשים, מנגנוני שמירת הסיסמאות, בדרכים למצוא סיסמאות ובדרכים לפצח את הסיסמאות המאוחסנות באופי מקומי על הרכיב.

משתמשים ברכיבי סיסקו

קיימים שני סוגי משתמשים בהם משתמשים להתחברות וניהול רכיבי Cisco.

1. דומייני
2. מקומי

בארגונים לרוב נשתמש במשתמשים דומיינים. במצב זה, קל יותר לנהל הרשאות, לאכוף התנהלות ולנטר על פעולותיהם.

משתמשים דומיינים

על מנת לחבר את רכיבי התקשורת לדומיין נשתמש בפרוטוקול שנקרא TACACS+ (או קודמו- Radius הפרוץ). פרוטוקול זה מאפשר תקשורת מול שרת שנקרא שרת AAA.

AAA הוא מושג המתאר את היכולות הבאות:

Authentication - זיהוי ואימות זהות המשתמש. (בד"כ באמצעות שם משתמש וסיסמה או מפתחות הצפנה).

Authorization - ניהול הרשאות המשתמש. כלומר, לאחר שהמשתמש עובר את תהליך האימות, איזה פקודות מותר לו להריץ ולאיזה מידע מותר לו לגשת. ההרשאות יכולות להיקבע לפי זמן ביצוע הפעולה, סוג המשתמש, קבוצת משתמשים אליה המשתמש שייך, מיקומו של המשתמש, שיטת החיבור של המשתמש וכו'.

Accounting - ניטור אחר פעולות המשתמש כגון מצב הרכיב בזמן ביצוע הפעולה, זמן ההתחברות לרכיב, מידע אליו המשתמש ניגש וכו'.

שרת ה-AAA מסונכרן מול ה-DC ומאפשר ניטור ואכיפה של התחברויות משתמשים ושל פעולותיהם על בסיס המשתמש הדומייני וההרשאות הדומייניות.

במאמר זה לא נתייחס למשתמשים הדומיינים, מפני שסיסמאותיהם אינן מאוחסנות על הרכיב, אלא רק למשתמשים המוגדרים מקומית.

משתמשים מקומיים

לרוב נשתמש במשתמשים מקומיים כמשתמשי גיבוי. שימוש נוסף הוא היכולת להגדיר interfaces ברכיב כך שעל מנת להתחבר באמצעותם, יש להזין שם משתמש מקומי ואת הסיסמה שלו.

ניתן להגדיר את הרכיב כך שתחילה הרכיב ינסה לאמת את המשתמש מול שרת ה-AAA. אם הוא לא מצליח לתקשר עם השרת הוא ידרדר לנסות לאמת את המשתמש כמשתמש מקומי. אם התקשורת עם שרת ה-AAA תקינה אך השרת לא מזהה את המשתמש, תיווצר שגיאת Authentication Failed ותהליך האימות יעצור (לא יהיה ניסיון אימות כמשתמש מקומי).

כתוצאה מהגדרת שרתי AAA לא הרמטית ו\או קיום משתמשי גיבוי מקומיים עם הרשאות גבוהות (אשר נתונים למדיניות סיסמאות שונה ולרוב מתירנית יותר מהדומיין), קיים הצורך לוודא שהסיסמאות המוגדרות למשתמשים המקומיים עומדות במדיניות הרצויה.

שיטות לשמירת סיסמאות

קיימות שלוש שיטות לשמירת סיסמאות ברכיבי Cisco - בצורה גלויה, בצורה מוצפנת סימטרית ובצורה מגובבת.

שמירת סיסמאות בצורה גלויה לא באמת שומרת על הסיסמאות אלא יותר מאחסנת אותן. אין סיבה להשתמש בשיטה זו כי כיום כמעט כל מערכת תומכת בהצפנה או בגיבוב.

שמירת סיסמאות בצורה מוצפנת (סימטרית) נחשבת טובה אך הבעיה היא שיש צורך לשמור את מפתח ההצפנה במקום מסוים. אם תוקף מגיע למפתח ההצפנה, הוא יכול לגלות את כל הסיסמאות המוצפנות. אלגוריתם ההצפנה בו Cisco משתמשים הוא AES-128.

שמירת סיסמאות בצורה מגובבת נחשבת השיטה הטובה ביותר לשמירת סיסמאות. פונקציית גיבוב קריפטוגרפית (Cryptographic Hash Function) היא פונקציה חד-כיוונית הממירה קלט באורך כלשהו לפלט באורך קבוע וידוע מראש.

פונקציית גיבוב קריפטוגרפית מתוכננת כך שכל שינוי בקלט יגרום לשינוי משמעותי בפלט. מנגנון זה מקשה על מציאת חוקיות מסוימת בפלט שתאפשר לגלות את הקלט. בנוסף, ניתן להוסיף לקלט ערך רנדומלי שנוצר במהלך יצירת הסיסמה. ערך זה נקרא Salt ומטרתו היא להקשות על מניית הסיסמה מהפלט. אלגוריתמי גיבוב בהם Cisco משתמשים הם SHA-256, MD5, SCRYPT ו-PBKDF2-SHA256.

בהמשך המאמר, נסביר בצורה יותר מפורטת על השימושים במושגים שהוזכרו כאן, הן לשמירת הסיסמאות והן לפיצוח הסיסמאות השמורות באופן מגובב.

סוגי סיסמאות

בנתבי / מתגי Cisco קיימות מספר אפשרויות לשמירת הסיסמאות של משתמשים מקומיים. הסיסמאות נשמרות בצורה גלויה (סוג 0), מוצפנת (סוגים 6 ו-7) ומגובבת - Hashed (סוגים 4, 5, 8, 9 ו-14) . יש לשים לב שסוג 0 נשמר בתור password וכל סוג אחר של סיסמה נשמר בתור secret. ברחבי האינטרנט ניתן למצוא מדריכים כלליים על סוגי הסיסמאות, אך חלקם חמקמקים יותר מאחרים. נבצע כאן סקירה מרוכזת על כולם:

Type 0

הסיסמה שמורה בצורה גלויה. פורמט סיסמה כזו הוא:

```
username <USERNAME> password 0 <PASSWORD>
```

לדוגמה:

```
username bob password 0 P@ssw0rd
```

Type 4

נעשה שימוש ב-SHA-256 לשמירת הסיסמה. פגמים משמעותיים במימוש האלגוריתם הופכים את ה-Hash לחלש יותר מה-MD5 של סוג 5. צוות ה-Design של Cisco דרש שימוש באלגוריתם PBKDF2-SHA256 עם 80bit של Salt ו-1,000 איטרציות. בפועל יש רק פעולה אחת של SHA256 על הסיסמה, בלי Salt. לכן אינו מומלץ והוא אף Deprecated החל מ-IOS 15.3.

פורמט סיסמה כזו הוא:

```
username <USERNAME> secret 4 <HASH>
```

לדוגמה:

```
username bob secret 4 g1rTD89b38NIXbGJse.zLc7Cega1TBTlKQNvYDh9Qo6
```

Type 5

הסיסמה שמורה כ-MD5 Hash של הסיסמה עם Salt. בצורה זו לא ניתן להשיג את הסיסמה המקורית מהתוכן של קובץ הקונפיגורציה (פרט למתקפות מילון או לשבירת אלגוריתם ה-Hash).

פורמט סיסמה כזו הוא:

```
username <USERNAME> secret 5 $1$<SALT>$<HASH>
```

לדוגמה:

```
username bob secret 5 $1$w1Jm$bCt7eJNv.CjWPwyfWcobP0
```

Type 6

הסיסמה שמורה לאחר הצפנת AES עם מפתח באורך 128 bit. פורמט זה משמש למצבים בהם נחוצה סיסמה השמורה בצורה הפיכה כמו מפתח הצפנה ל-VPN.

פורמט סיסמה כזו הוא:

```
username <USERNAME> password 6 <ENCRYPTED-PASSWORD>
```

לדוגמה:

```
username bob password 6 fZbe^WdXO`^O[YF`XLCfBV\BK`hMge]HF
```

Type 7

הסיסמה שמורה בצורה מוצפנת אך חלשה, שכן ניתן לפענח בקלות (פחות משנייה) את הסיסמה באמצעות כלים אינטרנטיים. אלגוריתם זה מסתמך על צופן vigenere ההפיך. בשל מנגנון הצופן, פורמט זה לשמירת הסיסמה מהווה רק קידוד של הסיסמה. לכן, יש להתייחס לסיסמה זו כ-obfuscated ולא כמוצפנת.

פורמט סיסמה כזו הוא:

```
username <USERNAME> password 7 <OBFUSCATED-PASSWORD>
```

לדוגמה:

```
username bob password 7 08116C5D1A0E550516
```

Type 8

נעשה שימוש ב-PBKDF2-SHA256, המשתמש באלגוריתם המתקן את הפגמים של סוג 4. נעשה שימוש ב-Salt של 80bit וביצוע של 20,000 איטרציות בחישוב. סוג זה נתמך החל מגרסה (3)15.3.

פורמט סיסמה כזו הוא:

```
username <USERNAME> secret 8 $8$<SALT>$<HASH>
```

לדוגמה:

```
username bob secret 8 $8$dsYGNam3K1SIJO$7nv/35M/qr6t.dVc7UY9zrJDWRVqncHub1PE9UlMQFs
```

Type 9

נעשה שימוש באלגוריתם SCRYPT (משתמש ב-SHA-256 בין היתר). ה-Hash כולל Salt 80bit ומתבצעות 16,384 איטרציות בחישוב. כוח העיבוד הרב הדרוש לביצוע ה-Hash מקשה מאוד על ניסיונות Brute Force. סוג זה נתמך החל מגרסה (3)15.3 או 15.4. פורמט סיסמה כזו הוא:

```
username <username> secret 9 $9$<salt>$<hash>
```

לדוגמה:

```
username bob secret 9 $9$nhEmQVczB7dqsO$X.HsgL6x1il0RxkOSSvyQYwucySCt7qFm4v7pqCxkKM
```

Type 14

סוג סיסמה זה נקרא convoluted9. גם אם יצא לכם לקנפג רכיבי Cisco לא בטוח שנתקלתם בסוג הזה. מדובר ב-SCRYPT המבוצע על MD5 (בעצם 5 + 9 שווה 14 ☺) סוג סיסמה זה נתמך בגרסאות Cisco IOS XE Gibraltar 16.11.2 ומעלה. סיסמה מסוג זה יכולה להיווצר גם כאשר מתבצע שדרוג מהגרסאות Cisco IOS XE Fuji 16.9.x \ 16.10.x \16.11.x \ 16.12.x בפועל יתבצע אלגוריתם SCRYPT על ה-MD5 של כל סיסמה מסוג 5.

לדוגמה:

```
username bob secret 5 $1$dNmW$7jWhqdtZ2qBVz2R4CSZZC0
```

יומר ל:

```
username bob secret 9 $14$dNmW$QykGZEEGmiEGrE$C9D/fD0czicOtgaZAa1CTa2sgygi0Leyw3/cLqPY426
```

במידה ויבוצע downgrade לגרסה שלא תומכת בסוג סיסמה זה (גרסה נמוכה מ-Cisco IOS XE Gibraltar 16.11.2), המשתמש שעליו מוגדרת הסיסמה לא יעבוד. מצב זה יכול לגרום לנעילה מחוץ לרכיב.

מציאת הסיסמאות

כל סיסמאות המשתמשים המקומיים נמצאות בקונפיגורציה של הרכיב. לרכיב Cisco שתי קונפיגורציות:

אחת נדיפה בזכרון, הנקראת Running-Config. ואחת בלתי נדיפה בדיסק, הנקראת Startup-Config.

ניתן לבדוק בכל שלב את השינויים בין הקונפיגורציות בעזרת הפקודה:

```
Show archive config differences
```

חשוב לבדוק שאין שינויים שבוצעו (מופיע ב-Running Config) ולא נשמרו על הדיסק (ב-Startup Config). אם זה המצב, ניתן יהיה לבצע ריסטרט לרכיב וסיסמאות של משתמשים עלולות להשתנות, משתמשים יכולים להתווסף/להימחק וכו'. לכן, אם נרצה להיות יסודיים כשנחפש סיסמאות, נחפש ב-2 הקונפיגורציות.

כעת נעבור לאיסוף הסיסמאות.

קיימים שני סוגי סיסמאות:

- **סיסמת משתמש** - סיסמה שנועדה לחיבור של משתמשים לרכיב.
- **סיסמת Enable** - סיסמה לגישה לרמת הרשאות מסוימת המצוינת בשורת הקונפיגורציה של הסיסמה. לדוגמה, גישה למצב enable.

לצורך הדוגמה, ניעזר בקונפיגורציה הנוכחית, אותה נציג בעזרת הפקודה:

```
show running-config
```

בתוך הקונפיגורציה נוכל לייצא את השורות הרלוונטיות למשתמשים בעזרת ה-Regex הבא:

```
username (\w+) (?:password|secret) (\d+) (?: privilege (\d+))?(.*)
```

או סיסמת enable:

```
enable (?:password|secret) (?:level (\d+) )?(\d+)?(.*)
```

פיצוח האשים

כמו שהוזכר כבר, הסיסמאות נשמרות בצורה גלויה (סוג 0) או בצורה מוצפנת (סוגים 6 ו-7) או בצורה מגובבת / Hashed (סוגים 4, 5, 8, 9, 14).

את סוג 0 לא נצטרך לפצח כמובן, סוג 7 ניתן להפיכה בקלות ולא דורש כמעט משאבי חישוב. את סוג 6 מאוד קשה לפצח ולא נתייחס לסוג זה כאן.

לעומת זאת, כדי לפצח את הסיסמאות השמורות בצורה מגובבת, ידרשו יותר משאבי חישוב. במצב זה, נצטרך להשתמש במתקפה שנקראת Rainbow Table Attack. מתקפה זו משתמשת בטבלה המכונה Rainbow Table ומכילה סיסמאות שכבר גובבו בפונקצית גיבוב. פונקציית הגיבוב תהיה פונקציית הגיבוב שהשתמשו בה לגיבוב הסיסמה שנרצה לפצח. אם יש שימוש ב-Salt, נצטרך להשתמש ב-Salt בפונקציית הגיבוב שבעזרתה ניצור את הטבלה ולא נוכל להשתמש בטבלה מוכנה מראש. הסיסמאות בהן נשתמש בטבלה יהיו בד"כ סיסמאות נפוצות או סיסמאות שאנחנו יודעים שייתכן שהארגון משתמש בו. לאחר בניית הטבלה, ניתן להשוות את הסיסמה המגובבת לכל אחד מהערכים המגובבים בטבלה.

מכאן, כאשר אנחנו מנסים לפצח סיסמאות מגובבות, המגבלה היחידה שלנו היא הזמן. תיאורטית, עם מספיק זמן, נוכל להשוות את כל האפשרויות של הסיסמאות המגובבות מול הסיסמה המגובבת שאנחנו רוצים לפצח.

קיימות פונקציות גיבוב רבות ושונות כאשר ההבדלים הרלוונטיים לנו ביניהן הוא הזמן שלוקח לפונקציה להמיר את הקלט לפלט והזמן שלוקח להשוות בין שני פלטים. ככל שזמנים אלו ארוכים יותר, ייקח לנו יותר זמן ליצור את ה-Rainbow Table ויותר זמן להשוות את הסיסמאות.

איפה שניתן, נשתמש בכלים מוכנים לפיצוח סיסמאות.

Hashcat הוא, כפי שמפתחיו מצהירים, הכלי המהיר ביותר בעולם לפריצת סיסמאות. הוא יכול לרוץ על Linux, Windows, MAC OS X וכן, לרוץ על מגוון סוגי מעבדים ו-GPUs. בנוסף, הוא משתמש במגוון אופטימיזציות להשוואת ערכי Hash וערכים מוצפנים (בין היתר כאלה אשר עוקפות צווארי בקבוק חומרתיים, כמו רוחב הפס של PCI-E). הכלי Hashcat תומך בפונקציות גיבוב רבות ביניהן פונקציות LM hash של Windows, MD4, MD5 SHA, פורמטי Crypt שונים ללינוקס, MySQL ו-Cisco PIX.

John the Ripper היא תוכנת קוד פתוח, המשמשת לאימות נתונים ופיצוח סיסמאות מוצפנות. התוכנה נכתבה בשפת C ופותחה על ידי החוקר Solar Designer. ל-John the Ripper יכולת פיצוח סיסמאות מגובבות ומוצפנות בשיטות רבות כגון: DES, MD4, MD5, Blowfish, LM, BSD, ו-Kerberos.

לאחר שהכנסנו את הסיסמאות בפורמט password:user לקובץ בו כל שורה מייצגת סיסמה אותה נרצה לפרוץ, כמו ש-Hashcat ו-John the Ripper אוהבים, נוכל להתחיל לפצח אותן.

Type 4

כאן John the Ripper הרבה יותר יעיל מ-Hashcat ולכן נשתמש בו:

```
john --format=Raw-SHA256 --wordlist=<wordlist> --fork=4 <Hashes.txt>
```

```
┌──(dw㉿kali)-[~/Desktop]
└─$ john --format=Raw-SHA256 --wordlist=/usr/share/wordlists/rockyou.txt -fork=4 pass4.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256 256/256 AVX2 8x])
Node numbers 1-4 of 4 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
P@ssw0rd         (bob)
1 1g 0:00:00:00 DONE (2023-03-05 09:39) 100.0g/s 204800p/s 204800c/s 204800C/s horoscope..00112233
Waiting for 3 children to terminate
2 0g 0:00:00:00 DONE (2023-03-05 09:39) 0g/s 12806Kp/s 12806Kc/s 12806KC/s !!nefty!!.abygurl69
4 0g 0:00:00:00 DONE (2023-03-05 09:39) 0g/s 12806Kp/s 12806Kc/s 12806KC/s !!peanut!!..*7¡Vamos!
3 0g 0:00:00:00 DONE (2023-03-05 09:39) 0g/s 12806Kp/s 12806Kc/s 12806KC/s !!push696!!.a6_123
Use the "--show --format=Raw-SHA256" options to display all of the cracked passwords reliably
Session completed

┌──(dw㉿kali)-[~/Desktop]
└─$ john --show --format=Raw-SHA256 pass4.txt
bob:P@ssw0rd

1 password hash cracked, 0 left
```

Type 5

```
hashcat -m 500 --username -O -a 0 <hashes.txt> <wordlist.txt>
```

```
$1$w1Jm$bCt7eJNv.CjWPwyfWcobP0:P@ssw0rd
Session..........: hashcat
Status...........: Cracked
Hash.Name........: md5crypt, MD5 (Unix), Cisco-IOS $1$ (MD5)
Hash.Target......: $1$w1Jm$bCt7eJNv.CjWPwyfWcobP0
Time.Started.....: Sun Feb 19 03:38:22 2023 (1 sec)
Time.Estimated...: Sun Feb 19 03:38:23 2023 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    11480 H/s (11.98ms) @ Accel:512 Loops:250 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 8196/14344385 (0.06%)
Rejected.........: 4/8196 (0.05%)
Restore.Point....: 6146/14344385 (0.04%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:750-1000
Candidates.#1....: gemini1 → thekillers
Started: Sun Feb 19 03:37:53 2023
Stopped: Sun Feb 19 03:38:24 2023
```

Type 7

ישנן שתי דרכים לבצע את הפענוח לסוג 7. הראשונה, בעזרת סקריפט פייתון קצר וחמוד:

```python
import random, re
xlat = [0x64, 0x73, 0x66, 0x64, 0x3b, 0x6b, 0x66, 0x6f, 0x41, 0x2c, 0x2e, 0x69,
0x79, 0x65, 0x77, 0x72, 0x6b, 0x6c, 0x64, 0x4a, 0x4b, 0x44, 0x48, 0x53, 0x55,
0x42, 0x73, 0x67, 0x76, 0x63, 0x61, 0x36, 0x39, 0x38, 0x33, 0x34, 0x6e, 0x63,
0x78, 0x76, 0x39, 0x38, 0x37, 0x33, 0x32, 0x35, 0x34, 0x6b, 0x3b, 0x66, 0x67,
0x38, 0x37]
def decrypt_type7(ep):
    dp = ''
    regex = re.compile('(^[0-9A-Fa-f]{2})([0-9A-Fa-f]+)')
```

```python
    result = regex.search(ep)
    s, e = int(result.group(1)), result.group(2)
    for pos in range(0, len(e), 2):
        magic = int(e[pos] + e[pos+1], 16)
        if s <= 50:
            # xlat length is 51
            newchar = '%c' % (magic ^ xlat[s])
            s += 1
        if s == 51: s = 0
        dp += newchar
    return dp
file_path = input()
f = open(file_path) # username:password format expected
for line in f:
    result = line.split(":")
    if(result[1]):
        print("Decrypted {}'s password: {}".format(result[0], decrypt_type7(result[1])))
```

והפלט:

```
C:\Users\User\Desktop\password7.txt
Decrypted bob's password: P@ssw0rd
```

הדרך השניה מאפשרת פענוח של הסיסמה על הרכיב אך מצריכה הרשאות לעריכת הקונפיגורציה (configure):

אחרי שנכנס ל-conf t (קיצור ל-Configure Terminal) נוכל ליצור key chain, בתוכו ליצור key ואז להכניס את הסיסמה של המשתמש בתור key-string 7:

```
Nati(config)#do sh run | inc username
username admin secret 5 $1$pB/i$VjTP3s4SdDJxbhDh60.zo.
username test password 7 111D1C1603
username emil password 7 070A2C4542
username dar password 7 12380446405858517C
Nati(config)#key chain test
Nati(config-keychain)#key 1
Nati(config-keychain-key)#key-string 7 12380446405858517C
Nati(config-keychain-key)#do sh key chain test
Key-chain test:
    key 1 -- text "Aa123456"
        accept lifetime (always valid) - (always valid) [valid now]
        send lifetime (always valid) - (always valid) [valid now]
```

מהיבט ההגנתי, ניתן לנטר על שימוש בפקודות אילו בעזרת הלוגים על הרכיב (ולשלוח אותם לשרת Syslog):

```
Feb 19 09:44:47.109: %SYS-5-CONFIG_I: Configured from console by admin on vty0 (90.210.1.254)
Feb 19 09:45:12.080: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:key *****
Feb 19 09:45:14.131: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:key *****
Feb 19 09:45:21.044: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:key-string *****
Feb 19 09:45:54.803: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:username dar password *****
Feb 19 09:45:54.803: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:!config: USER TABLE MODIFIED
Feb 19 09:45:59.662: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:username dar password *****
Feb 19 09:45:59.662: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:!config: USER TABLE MODIFIED
Feb 19 09:46:21.979: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:key *****
Feb 19 09:46:24.025: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:key *****
Feb 19 09:46:30.138: %PARSER-5-CFGLOG_LOGGEDCMD: User:admin  logged command:key-string *****
```

באמצעות מערכות AAA, ניתן גם לחסום ולנטר את ביצוע הפקודות על ידי משתמש דומיייני.

Type 8

hashcat -m 9200--username -O -a 0 <hashes.txt> <wordlist.txt>

```
$8$dsYGNam3K1SIJO$7nv/35M/qr6t.dVc7UY9zrJDWRVqncHub1PE9UlMQFs:cisco

Session..........: hashcat
Status...........: Cracked
Hash.Name........: Cisco-IOS $8$ (PBKDF2-SHA256)
Hash.Target......: $8$dsYGNam3K1SIJO$7nv/35M/qr6t.dVc7UY9zrJDWRVqncHub...UlMQFs
Time.Started.....: Sun Feb 19 03:46:19 2023 (10 secs)
Time.Estimated...: Sun Feb 19 03:46:29 2023 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     2608 H/s (9.13ms) @ Accel:256 Loops:512 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 24576/14344385 (0.17%)
Rejected.........: 0/24576 (0.00%)
Restore.Point....: 23552/14344385 (0.16%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:19968-19999
Candidates.#1....: tajmahal → 280789

Started: Sun Feb 19 03:46:06 2023
Stopped: Sun Feb 19 03:46:30 2023
```

Type 9

hashcat -m 9300 --username -O -a 0 <hashes.txt> <wordlist.txt>

```
$9$nhEmQVczB7dqsO$X.HsgL6x1il0RxkOSSvyQYwucySCt7qFm4v7pqCxkKM:cisco

Session..........: hashcat
Status...........: Cracked
Hash.Name........: Cisco-IOS $9$ (scrypt)
Hash.Target......: $9$nhEmQVczB7dqsO$X.HsgL6x1il0RxkOSSvyQYwucySCt7qFm...qCxkKM
Time.Started.....: Sun Feb 19 03:49:12 2023 (22 secs)
Time.Estimated...: Sun Feb 19 03:49:34 2023 (0 secs)
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:     1107 H/s (13.27ms) @ Accel:4 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests
Progress.........: 24512/14344385 (0.17%)
Rejected.........: 0/24512 (0.00%)
Restore.Point....: 24496/14344385 (0.17%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: crying → chloe01

Started: Sun Feb 19 03:48:46 2023
Stopped: Sun Feb 19 03:49:35 2023
```

Type 14

על מנת לפצח את ההאשים מסוג 14, נצטרך להפעיל MD5 עם ה-salt (שמופיע בקונפיגורציה) ואז Scrypt עם ה-salt (שמופיע בקונפיגורציה) על כל סיסמה ברשימת הסיסמאות שלנו. לכן, מניית סיסמאות מסוג זה תיקח משמעותית יותר זמן ממניית שאר סוגי הסיסמאות.

לשם כך, כתבנו PoC שמבצע את זה. הוא פחות יעיל מ-Hashcat או John The Ripper שמשתמשים בשיטות יעילות לפיצוח הסיסמאות. בכל מקרה, זה יותר טוב מכלום ולא מצאנו שום תיעוד למשהו דומה באינטרנט.

ה-PoC נעזר בפרויקט המצוין ciscoPWDhasher של BrettVerney שמבצע hashes מסוג 5 ו-9.

```
import hashlib
import scrypt
import base64
from passlib.hash import md5_crypt
import argparse
import multiprocessing
import functools
import time

# Translate Standard Base64 table to Cisco Base64 Table used in Type8 and TYpe 9
std_b64chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
cisco_b64chars = "./0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
b64table = str.maketrans(std_b64chars, cisco_b64chars)

def pwd_check(pwd):
  invalid_chars = r"?\""
  if len(pwd) > 127 or any(char in invalid_chars for char in pwd):
    return False
  return True

def type5(pwd, salt):
  return md5_crypt.using(salt_size=4, salt=salt).hash(pwd)

def type9(pwd, salt):
  # Create the hash
  pwd_hash = scrypt.hash(pwd.encode(), salt.encode(), 16384, 1, 1, 32)
  # Convert the hash from Standard Base64 to Cisco Base64
  pwd_hash = base64.b64encode(pwd_hash).decode().translate(b64table)[:-1]
  # Print the hash in the Cisco IOS CLI format
  password_string = f'{pwd_hash}'
  return password_string

def type14(word, type5_salt, type9_salt):
  word = word.rstrip()
  if pwd_check(word):
    return type9(type5(word, type5_salt), type9_salt), word
  return False

def parse_convoluted_password(pwd):
```

```
    _, _, type5_salt, type9_salt, hashed_pwd = pwd.split("$")
    return type5_salt, type9_salt, hashed_pwd


def main():
    parser = argparse.ArgumentParser(description="Tries to Crack Cisco Convoluted 9 Passwords")
    parser.add_argument("-p", "--password", required=True)
    parser.add_argument("-w", "--wordlist", required=True)
    parser.add_argument("-t", "--threads", type=int, default=1)
    args = parser.parse_args()



    pool = multiprocessing.Pool(args.threads)
    type5_salt, type9_salt, hashed_pwd = parse_convoluted_password(args.password)
    hashed_wordlist = []

    partial_process_line = functools.partial(type14, type5_salt=type5_salt,
                                             type9_salt=type9_salt)

    with open(args.wordlist) as words:
        results = pool.map(partial_process_line, words)
        for result in results:
            if result[0] == hashed_pwd:
                print(result[1])
                pool.terminate()
                break

if __name__ == "__main__":
    start_time = time.time()
    main()
    print("--- %.2f seconds ---" % (time.time() - start_time))
```

ביצועים של בדיקה מול מיליון סיסמאות (לצערנו ב-BASH צריך לעשות escaping ל-$):

10Threads:

```
(type14) [redacted] python3 type14.py -p \$14\$dNmW\$QykGZEEGmi
EGrE\$C9D/fD0czicOtgaZAa1CTa2sgygi0Leyw3/cLqPY426 -w 10-million-password-list-top-1
000000.txt -t 10
abcd
--- 720.67 seconds ---
```

5Threads:

```
(type14) [redacted] python3 type14.py -p \$14\$dNmW\$QykGZEEGmi
EGrE\$C9D/fD0czicOtgaZAa1CTa2sgygi0Leyw3/cLqPY426 -w 10-million-password-list-top-1
000000.txt -t 5
abcd
--- 959.69 seconds ---
```

טבלת פיצוח ההאשים

סוג	קושי פיצוח	מהירות מקסימלית	John the Ripper	Hashcat
0	instant	instant	n/a	n/a
7	instant	instant	n/a	n/a
4	easy	26.4 million per second	--format=Raw-SHA256	-m 5700
5	medium	1.2 million per second	--format=md5crypt	-m 500
8	hard	11.6 thousand per second	--format=pbkdf2-hmac-sha256	-m 9200
9	very hard	1.8 thousand per second	--format=scrypt	-m 9300
14	super hard (using our poc)	1.041 thousand per second (5t) 1.388 thousand per second (10t)	n/a	n/a

הקשחה וסיכום

כעת, כשאנחנו יודעים לאיזה משתמשים יש סיסמאות חלשות, נוכל להתחיל לטפל בבעיה. השלב הראשון הוא להבין באילו רכיבים ניתן לשדרג את הסיסמה כך שתישמר בצורה יותר מאובטחת.

אין שום סיבה להשתמש בסוגים 0, 4 ו-7, יש להחליף אותם מיידית. סוג 5 ניתן לשבירה בקלות יחסית ולכן עדיף להשתמש בו רק במקרים בהם 8 ו-9 לא זמינים.

גם במקרים בהם הסיסמה שמורה בצורה מאובטחת יחסית נצטרך לוודא שהסיסמה חזקה, שכן ההצפנה שלה לא משנה אם התוקף ינסה להתחבר עם הסיסמה "123456" ויצליח. בנוסף, שימוש בסיסמה חזקה יקשה על ביצוע מתקפת Rainbow Table Attack. לכן נמליץ להשתמש בכלים שנתנו לכם בסקירה על מנת לאכוף את מדיניות הסיסמאות הארגונית ולמנוע את התפשטות התוקף הבא.

ביבליוגרפיה

- Security Configuration Guide, Cisco IOS XE Gibraltar 16.12.x (Catalyst 9600 Switches) - Controlling Switch Access with Passwords and Privilege Levels
- Configuring Type 6 Passwords in IOS XE - Cisco Community
- Network Infrastructure Security Guide
- Cisco Password Types: Best Practices
- Cisco Password Cracking and Decrypting Guide - InfosecMatter
- Understand the AAA Authentication Login Default Local Group TACACS+ Command - Cisco
- Configure Basic AAA on an Access Server - Cisco
- Расшифровываем пароли, зашифрованные с помощью service password-encryption ~ Сетевые заморочки
- GitHub - BrettVerney/ciscoPWDhasher: A Python Cisco IOS, IOS-XE and NX-OS password hashing
- John the Ripper - ויקיפדיה
- Hashcat - ויקיפדיה

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 מנגנוניי הגנה ב-Windows ואיתם מוצרי הגנה צד שלישי כגון: פתרונות ניטור ובקרה (ArcSight, Splunk, Sysmon), 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:

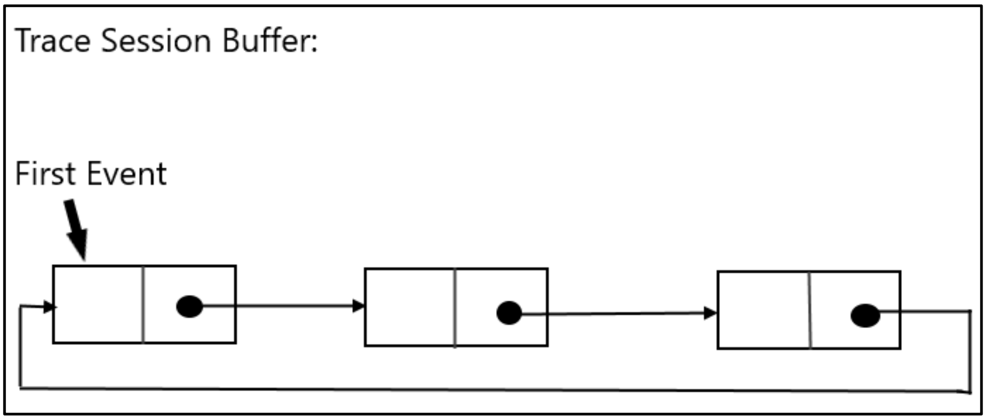
- C / C++ && WinAPI
- Windows Internals
- WinDbg && x64 Assembly

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 "בזבוז" 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 הציור שלי ב-Paint]

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

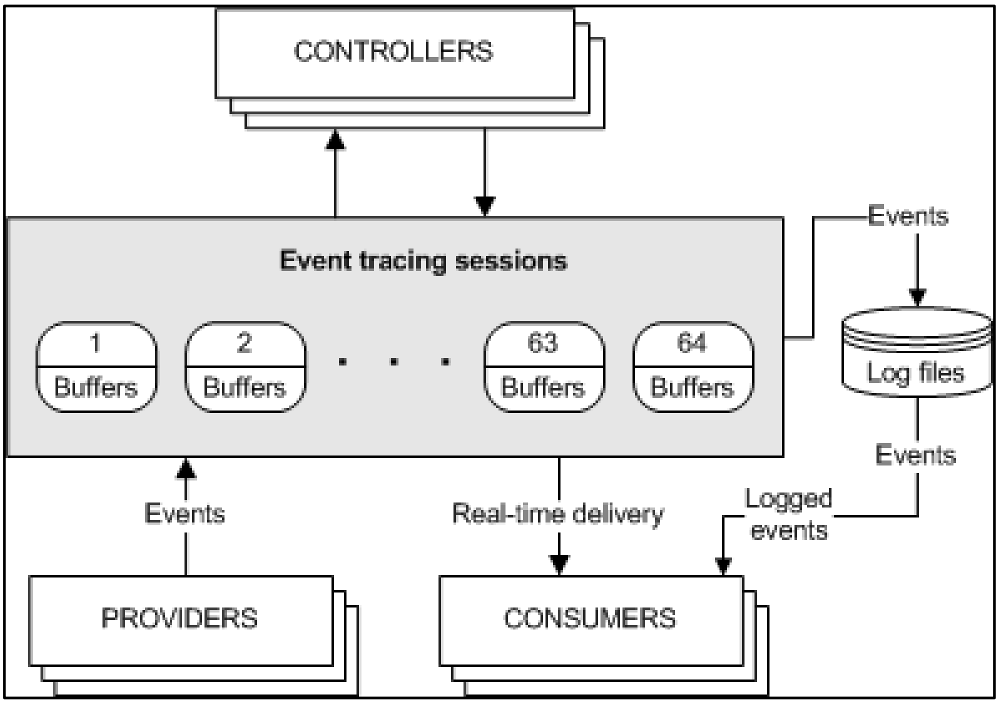
כבר עם התקנת Windows, מגיעים אלפי Providers המספקים מידע מפורט כמעט על כל אובייקט במערכת ההפעלה (הכולל מידע על ומה-Kernel, Application, User) והכל על מנת לספק תמונת מצב נוחה.

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.

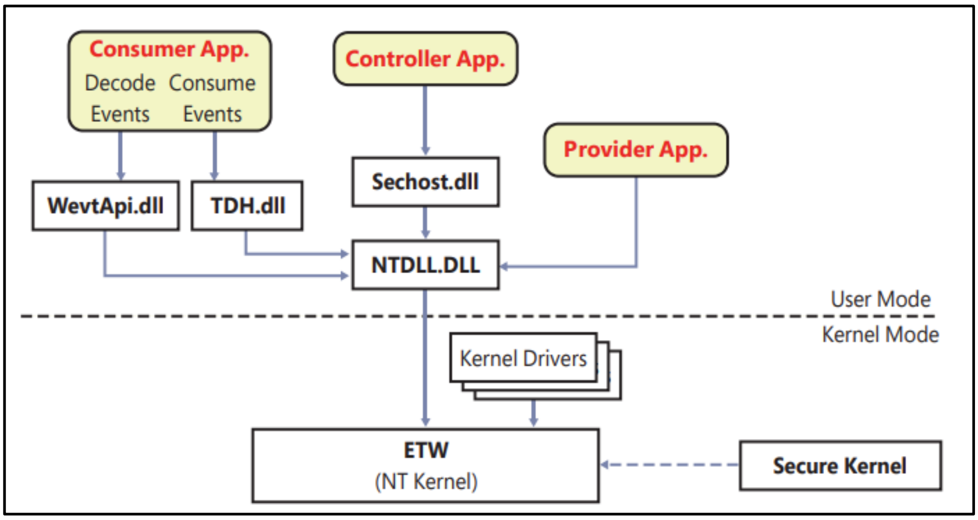
Provider - הגורם האחראי על יצירת הרישומים וכתיבתם ל-()Trace Session. כל תוכנה רשאית להירשם בתור Provider ולספק מידע על פי כל תבנית שתרצה. עקב המספר הרב של ה-Providers הקיימים במערכת ההפעלה, הם ידעו לספק מידע אך ורק כאשר אקטיביים ומקושרים ל-Trace Session. קיימים ארבעה סוגי Providers והם Manifest-Based ,MOF, WPP 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](https://) מ-User Mode ו-[EtwRegister](https://) 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 אחיו.

Controller - אחראי על כל הקשור במנגנון. ביצוע שינויים ב-Trace Sessions (יצירת Trace Session, עצירה והפעלה של Trace Session קיים, מחיקה והוספה של Provider ל-Trace Session קיים ושינויים בהגדרות השוטפות של ה-Trace Session). אז כן, הדרך הנוחה ביותר להתעסק עם המנגנון היא בעזרת [LOLBIN](https://) (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 אחראים 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 קונבציית 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
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:

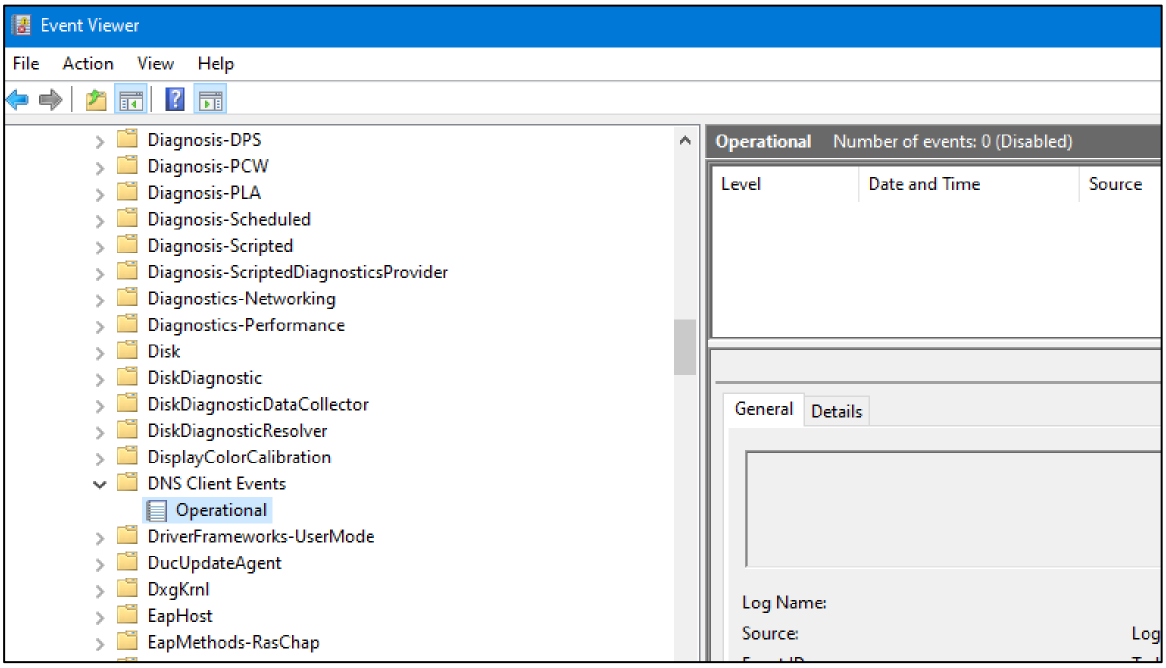
```
<channels>
    <importChannel chid="c1"
                   name="Microsoft-Windows-BaseProvider/Admin"
                   symbol="CHANNEL_BASEPROVIDER_ADMIN"
                   />

    <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 הקונבנציה 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="$(string.level.Informational)"> All informational content, including win:Warning </
    <level name="win:Verbose" symbol="WINEVENT_LEVEL_VERBOSE" value="5" message="$(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="0x10000000" />
        <keyword name="ut:PolicyTable" message="$(string.keyword_ut:PolicyTable)" mask="0x20000000" />
        <keyword name="ut:PerfCheckPoints" message="$(string.keyword_ut:PerfCheckPoints)" mask="0x40000000" />
        <keyword name="ut:RegistrationEvent" message="$(string.keyword_ut:RegistrationEvent)" mask="0x80000000" />
        <keyword name="ut:SendPath" message="$(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="0x1000000000" />
        <keyword name="ut:Authentication" message="$(string.keyword_ut:Authentication)" mask="0x2000000000" />
        <keyword name="ut:Configuration" message="$(string.keyword_ut:Configuration)" mask="0x4000000000" />
        <keyword name="ut:Global" message="$(string.keyword_ut:Global)" mask="0x8000000000" />
      </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="DnsServerForInt
    <event value="1002" symbol="DnsServerQueryChange" version="0" task="DnsServerQueryChange" level="win:Informational" template="DnsServerQueryCha
    <event value="1003" symbol="DnsServerValidationSuccess" version="0" task="DnsServerValidationSuccess" level="win:Informational" template="DnsSe
    <event value="1005" symbol="DnsServerValidationFailure" version="0" task="DnsServerValidationFailure" level="win:Error" template="DnsServerVali
    <event value="1007" symbol="DnsMissingPrimarySuffix" version="0" task="DnsMissingPrimarySuffix" level="win:Error" template="DnsMissingPrimarySu
    <event value="1008" symbol="DnsMissingPrimarySuffixSystem" version="0" task="DnsMissingPrimarySuffixSystem" level="win:Warning" keywords="ut:Dn
    <event value="1009" symbol="DnsNonMatchingSuffix" version="0" task="DnsNonMatchingSuffix" level="win:Error" template="DnsNonMatchingSuffixArgs"
    <event value="1010" symbol="DnsNonMatchingSuffixSystem" version="0" task="DnsNonMatchingSuffixSystem" level="win:Warning" keywords="ut:DnsAutoL
    <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:DnsAutoL
    <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="DnsAllSer
    <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="DnsNoServerConfigV4
    <event value="1020" symbol="DnsReadPolicyTable" version="0" task="DnsReadPolicyTable" level="win:Informational" keyword
```

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

מה עכשיו?

לפני שנכנס לבריכה 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 מציע, 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            {5BBB6C18-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 Manifests ו-Channels. 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: O:BAG:SYD:(A;;0x2;;;S-1-15-2-1)(A;;0x2;;;S-1-15-3-1024-3153509613-960666767-3724
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 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.

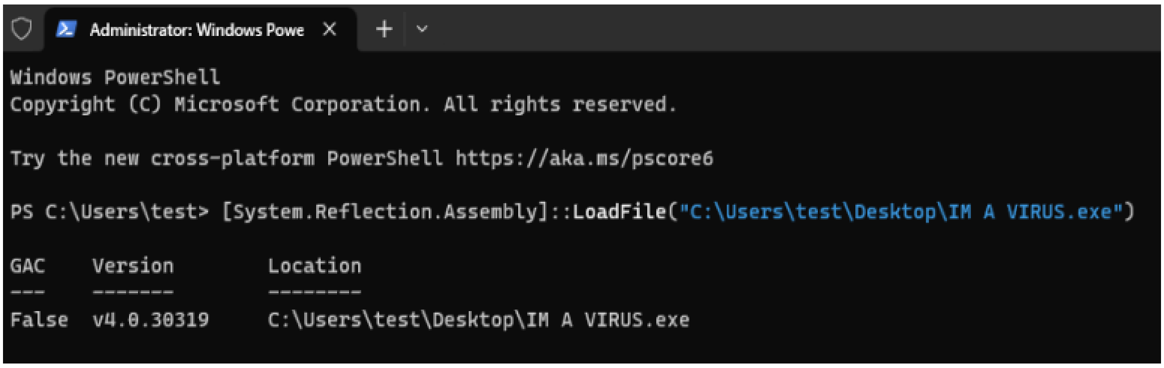
בסופו של דבר, הדרך הפשוטה שדיברתי עליה קודם לכן היא "wevtutil im <path to .man file>", כך נוכל לרשום manifest ובעצם לתת ל-wevtutil לעשות את כל עבודת ההתקנה בשבילנו.

מחבואים ב-CLR

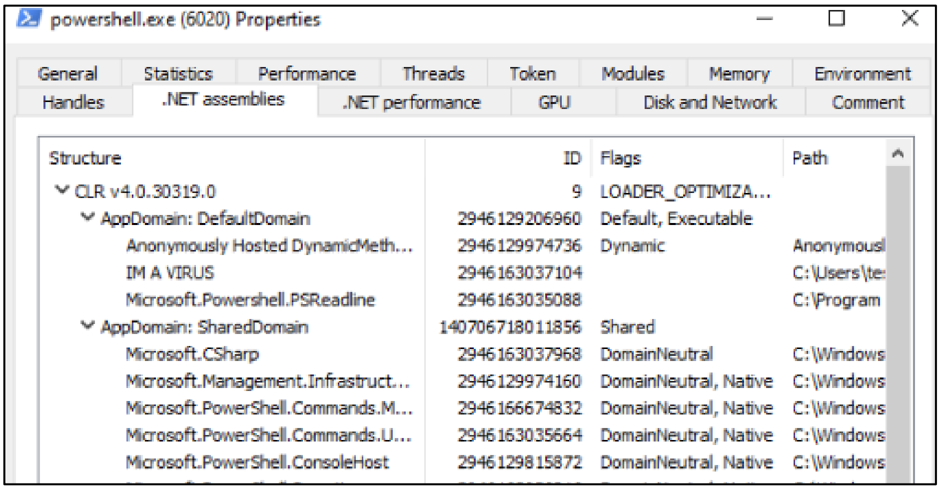
לפני שנתחיל, אני ממליץ בחום לקרוא את [המאמר המעולה](#) של ליאור פתיחה בגיליון 147 שיחסוך ממני להכיר בפניכם את המונחים הבסיסיים של .NET.

בתקופה האחרונה כלים רבים התחילו להשתמש במונח Patch ETW, [אבל מה זה בעצם?](#) הכל התחיל בעקבות מאמר 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:



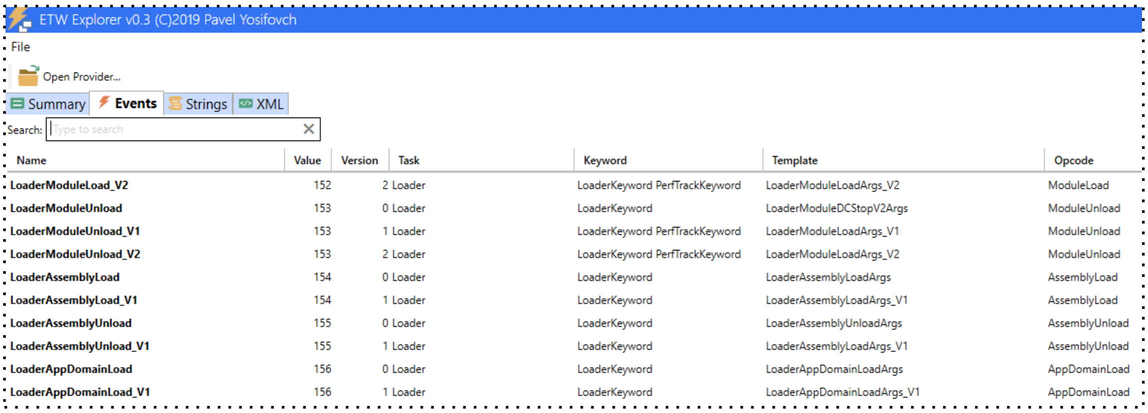
הפונקציה LoadFile, מתפקדת כ-Loader - מקבלת .NET שנמצא על הדיסק וטוענת אותו. בעזרת Process Hacker, נוכל לראות את ה-Assemblies הטעונים אל Process, ולמזלנו כל ה-Source Code שלו נמצא ב-Github. אז כן... הוא משתמש ב-ETW כדי לאסוף את הנתונים הללו. למטרת ההדגמה הוא יתפקד כ-Consumer לכל דבר:



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:


ETW Explorer v0.3 (C)2019 Pavel Yosifovch
File
Open Provider...
Summary | Events | Strings | XML
Search: Type to search

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 לכתיבת רישומים ל-Trace Session.

הפונקציה מקבלת את הארגומנטים הבאים:

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

```C++
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).

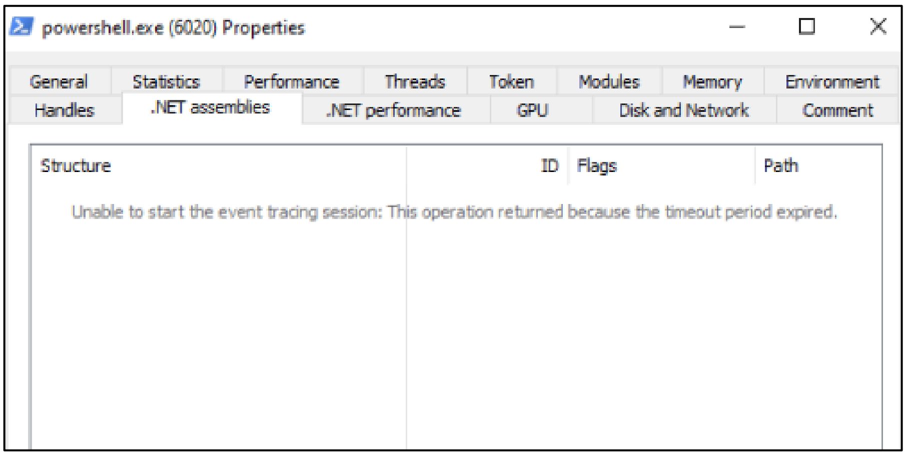
```
int MuteEvents(HANDLE hProc) {

    unsigned char EtwEventWriteArray[] = { 'E','t','w','E','v','e','n','t','W','r','i','t','e', 0x0 };

    void *EtwEventWrite = GetProcAddress(GetModuleHandle((LPCSTR)sNtdll), (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 ראיה על ה-Process.

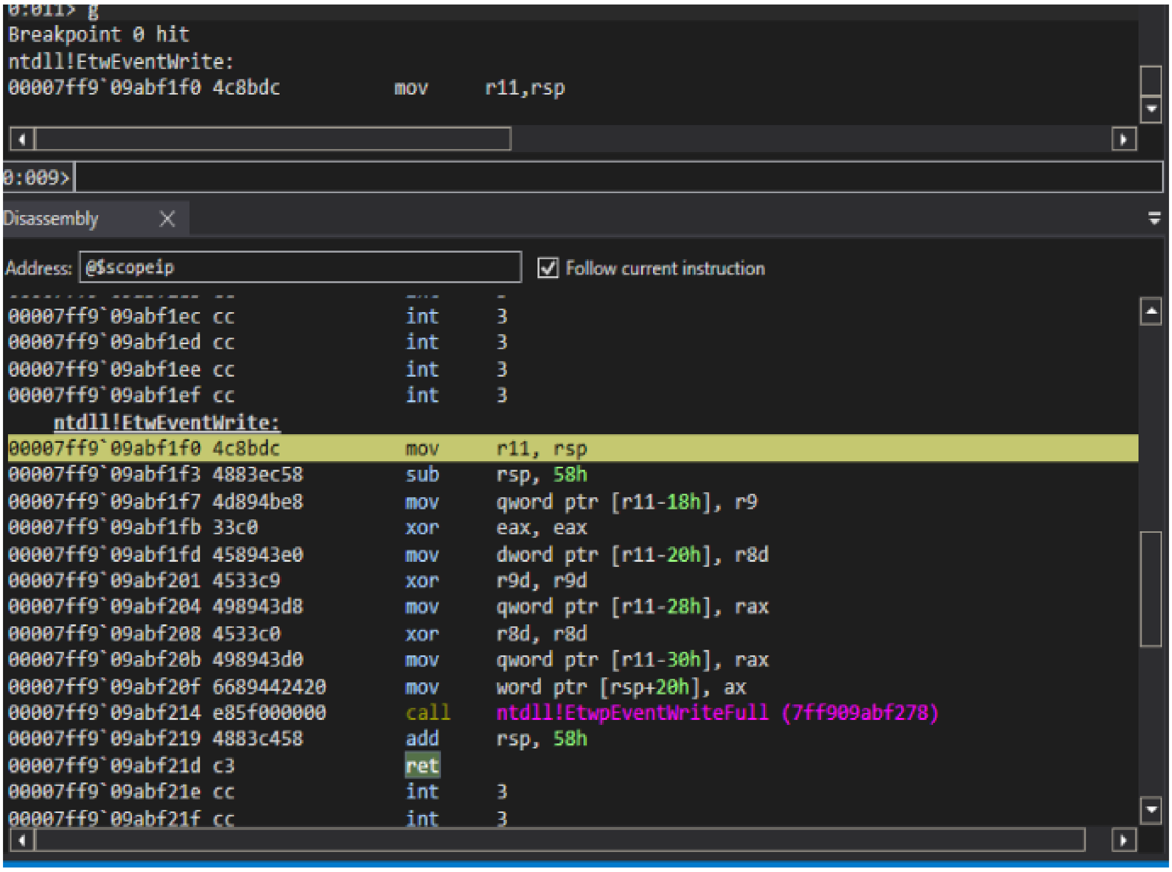
קיימות כמה שיטות לבצע את ה-Patch ולא בהכרח בפונקציה הזאת, אך קיימת גם האופציה לפלטר רישומים ולבצע מניעה בצורה כירורגית ובעצם להזרים חלק מהרישומים, נוכל להזרים אך ורקרישום ש ה-EventID שלו לדוגמא הוא 155 או כל דבר שקיים ב-UserData כמו שם ה-Assembly.

לצורך המחשה נפתח WinDbg ונשים BreakPoint על הפונקציה:

```
You can troubleshoot most symbol related issues by turning on symbol loading diagnostics (!sym noisy
You should also verify that your symbol search path (.sympath) is correct.
0:011> bp ntdll!etweventwrite
0:011> bl
     0 e Disable Clear  00007ff9`09abf1f0     0001 (0001)  0:**** ntdll!EtwEventWrite

0:011>
```

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



הערה: קונבנציית הקריאה לפונקציות ב-x64 בשונה מ-x86, 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

0:009>
```

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

LoaderAssemblyLoad_V1	154	1 Loader
LoaderAssemblyUnload	155	0 Loader
LoaderAssemblyUnload_V1	155	1 Loader
LoaderAppDomainLoad	156	0 Loader
LoaderAppDomainLoad_V1	156	1 Loader

על בסיס זה, נוכל ליצור Dll, שמבצע Hook לפונקציית EtwEventWrite ב-ntdll, מאותו הרגע הפונקציה תקבל פרמטרים ותקפוץ 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 Anti Malware-PPL.

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 בשבילה, 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 יריגש 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 במיוחד למוצרי הגנה המוכרים למיקרוסופט וחתומים על ידם, בואו נעביר מבט עליו.

המעקף שראינו במחקר של xpn 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 נוזקות ולא רק מול נוזקות 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_ALLOCVM_LOCAL
0x0000000000000002  KERNEL_THREATINT_KEYWORD_ALLOCVM_LOCAL_KERNEL_CALLER
0x0000000000000004  KERNEL_THREATINT_KEYWORD_ALLOCVM_REMOTE
0x0000000000000008  KERNEL_THREATINT_KEYWORD_ALLOCVM_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
0x0000000000100000  KERNEL_THREATINT_KEYWORD_SUSPEND_THREAD
0x0000000000200000  KERNEL_THREATINT_KEYWORD_RESUME_THREAD
0x0000000000400000  KERNEL_THREATINT_KEYWORD_SUSPEND_PROCESS
0x0000000000800000  KERNEL_THREATINT_KEYWORD_RESUME_PROCESS
0x0000000001000000  KERNEL_THREATINT_KEYWORD_FREEZE_PROCESS
0x0000000002000000  KERNEL_THREATINT_KEYWORD_THAW_PROCESS
0x0000000004000000  KERNEL_THREATINT_KEYWORD_CONTEXT_PARSE
0x0000000008000000  KERNEL_THREATINT_KEYWORD_EXECUTION_ADDRESS_VAD_PROBE
0x0000000010000000  KERNEL_THREATINT_KEYWORD_EXECUTION_ADDRESS_MMF_NAME_PROBE
0x0000000020000000  KERNEL_THREATINT_KEYWORD_READWRITEVM_NO_SIGNATURE_RESTRICTION
0x0000000040000000  KERNEL_THREATINT_KEYWORD_DRIVER_EVENTS
0x0000000080000000  KERNEL_THREATINT_KEYWORD_DEVICE_EVENTS
0x8000000000000000  Microsoft-Windows-Threat-Intelligence/Analytic

Value               Level                    Description
-------------------------------------------------------------------------------
0x04                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.

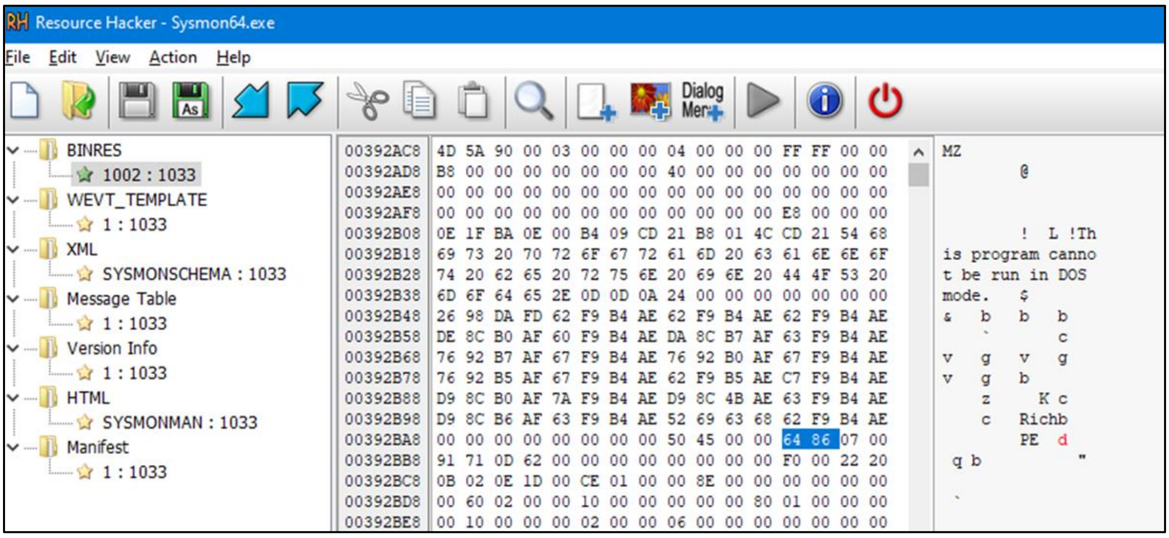
SlikETW

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 יכולת הזרמת המידע לשרתים צד שלישי מרוחקים ואפילו אינטגרציה עם חוקי YARA והתראה על בסיסם.

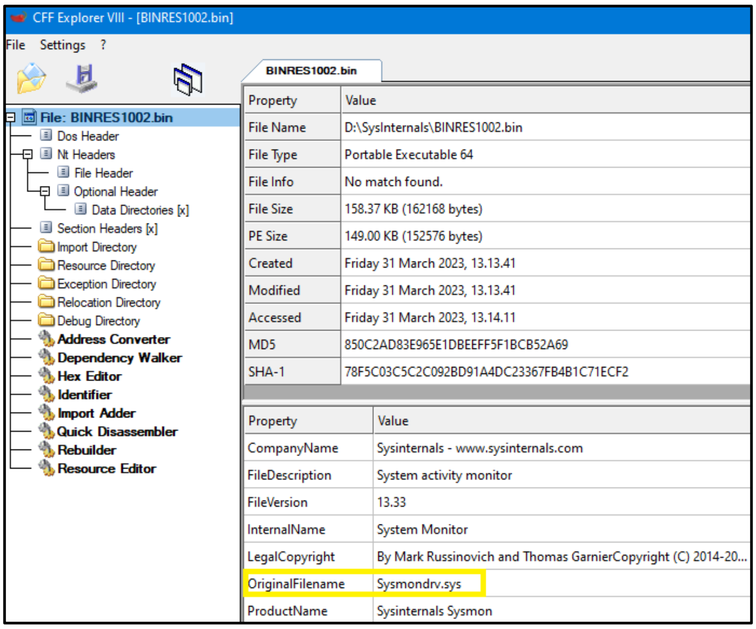
מאמר מעולה לצייד בסביבת AD עם SlikETW

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-x64 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 בתיקיית 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:



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 התקשורתיי 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 קונבציית 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.

```
0000074572FF530 80 84 61 F6 50 02 00 00 62 07 5F F6 50 02 00 00 ..a?P...b._?P...
0000074572FF540 80 84 61 F6 50 02 00 00 40 00 00 00 00 00 00 00 ..a?P...@.......
0000074572FF550 22 00 43 00 3A 00 5C 00 57 00 69 00 6E 00 64 00 ".C.:.\.W.i.n.d.
0000074572FF560 6F 00 77 00 73 00 5C 00 53 00 79 00 73 00 6D 00 o.w.s.\.S.y.s.m.
0000074572FF570 6F 00 6E 00 36 00 34 00 2E 00 65 00 78 00 65 00 o.n.6.4...e.x.e.
0000074572FF580 22 00 20 00 2D 00 6E 00 6F 00 6C 00 6F 00 67 00 ". .-.n.o.l.o.g.
0000074572FF590 6F 00 20 00 2D 00 61 00 63 00 63 00 65 00 70 00 o. .-.a.c.c.e.p.
0000074572FF5A0 74 00 65 00 75 00 6C 00 61 00 20 00 2D 00 6D 00 t.e.u.l.a. .-.m.
0000074572FF5B0 00 00 00 00 00 00 00 00 02 00 04 06 00 00 00 00 ................
0000074572FF5C0 63 01 00 50 00 00 00 00 00 00 00 00 00 00 00 00 c..P............
```

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 69 00 6E 00 64 00 6F 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 70 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 לתיקיית 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.
00000018732FCCE0 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 אנו רושמים את ה-Manifest:

```
0000018732FE2B0 22 00 43 00 3A 00 5C 00 57 00 69 00 6E 00 64 00  ".C.:.\.W.i.n.d.
0000018732FE2C0 6F 00 77 00 73 00 5C 00 73 00 79 00 73 00 74 00  o.w.s.\.s.y.s.t.
0000018732FE2D0 65 00 6D 00 33 00 32 00 5C 00 77 00 65 00 76 00  e.m.3.2.\.w.e.v.
0000018732FE2E0 74 00 75 00 74 00 69 00 6C 00 2E 00 65 00 78 00  t.u.t.i.l...e.x.
0000018732FE2F0 65 00 22 00 20 00 69 00 6D 00 20 00 22 00 43 00  e.". .i.m. .".C.
0000018732FE300 3A 00 5C 00 55 00 73 00 65 00 72 00 73 00 5C 00  :.\.U.s.e.r.s.\.
0000018732FE310 74 00 65 00 73 00 74 00 5C 00 41 00 70 00 70 00  t.e.s.t.\.A.p.p.
0000018732FE320 44 00 61 00 74 00 61 00 5C 00 4C 00 6F 00 63 00  D.a.t.a.\.L.o.c.
0000018732FE330 61 00 6C 00 5C 00 54 00 65 00 6D 00 70 00 5C 00  a.l.\.T.e.m.p.\.
0000018732FE340 4D 00 41 00 4E 00 46 00 34 00 34 00 39 00 2E 00  M.A.N.F.4.4.9...
0000018732FE350 74 00 6D 00 70 00 22 00 00 00 AA 7E 6B 01 00 00  t.m.p."...?~k...
```

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

```
Administrator: Windows Powe

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:          0xffffffffffffffff (Microsoft-Windows-Sysmon/Operational,0x1,0x2,0x4,0x8,0x10,0x20,0x40,0x80
,0x100,0x200,0x400,0x800,0x1000,0x2000,0x4000,0x8000,0x10000,0x20000,0x40000,0x80000,0x100000,0x200000,0x400000,0x
800000,0x1000000,0x2000000,0x4000000,0x8000000,0x10000000,0x20000000,0x40000000,0x80000000,0x100000000,0x200000000
,0x400000000,0x800000000,0x1000000000,0x2000000000,0x4000000000,0x8000000000,0x10000000000,0x20000000000,0x4000000
0000,0x80000000000,0x100000000000,0x200000000000,0x400000000000,0x800000000000,0x1000000000000,0x2000000000000,0x4
000000000000,0x8000000000000,0x10000000000000,0x20000000000000,0x40000000000000,0x80000000000000,0x100000000000000
,0x200000000000000,0x400000000000000,0x800000000000000,0x1000000000000000,0x2000000000000000,0x4000000000000000)
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 ל-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 User-Mode ETW, ולא צריך יותר מ-Logman ו-Wevtutil, עם ה-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 לSysmon 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

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-149 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב: למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה רבות כדי להביא לכם את הגליון.

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"T4lk1n' 8out a r3vo7u71on 5ounds like a wh15p3r"

הגליון הבא מתוכנן לצאת בסוף חודש אפריל.

אפיק קסטיאל,

31.03.2023