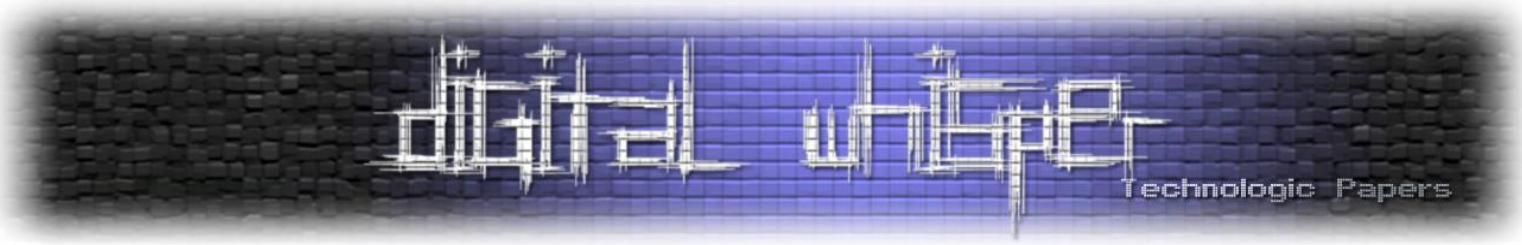


Digital Whisper

גליון 105, אפריל 2019

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	כסיף דקל, רונן שוסטין, יהודה כורסיה, אייל איטקין, אור צינגיסר וגלעד זינגר

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

דבר העורכים

היום, בדיוק לפני 20 שנה (31/03/1999) הסרט הראשון מטרילוגיית הסרטים The Matrix יצא לאקרנים. באותה התקופה הייתי בן 11, וממש התחלתי את צעדיי הראשונים בתחום ההאקינג (עד כמה שניתן לכנות התחברות לערוצי IRC בשרתים שכוחי אל, חיפוש קטעי קוד לא מובנים בבוטי XDCC או במנועי חיפוש כגון Astalavista.box.sk, PacketStormSecurity.com, קמפולם והרצתם כנגד פלטי nmap ולצפות לטוב - האקינג). ומלבד צפייה בסרטים שהיו אז, כגון Hackers, או Wargames כמעט ולא שמעתי כלום על התחום (על הסרטים Tron ו-Sneakers שמעתי מספר שנים מאוחר יותר).

הייתי אז רק בן 11, אך למיטב זכרוני השיח מסביב כמעט ולא הזכיר את המושגים "האקר" או "האקינג" (שלא נדבר על "סייבר"). הדבר הקרוב ביותר למגמת סייבר בבית הספר היה חוג QBasic שאליו הייתי רשום, והאינטרנט היה הרבה פחות נגיש וזמין מהיום (הן מבחינת ההתחברות אליו והן מבחינת הנגישות למידע הקיים בו, פחות השתמשו בגוגל ויותר במנועי חיפוש שחיפשו מתוך רשימות סטטיות). המודעות לאבטחת מידע כמעט ולא הייתה קיימת: כמעט ולא השתמשו ב-Firewall-ים ואותם קטעי קוד שקמפלנו עבדו הרבה מאוד זמן גם אחרי שעדכונים לאותן החולשות פורסמו. התעניינתי מאוד בתחום ההאקינג, ואף הצלחתי להשיג מקבץ טקסטים שהעיד על כך שקיימת סוג של סצינת האקינג מקומית, אך גם מתוך הטקסטים וגם מהשיחות בערוצי ה-IRC ניתן היה לראות שמדובר במתי מעט. האקינג פשוט לא היה משהו שדיברו עליו.

אחרי שהסרט מטריקס יצא לאור, הייתה לי הרגשה שמשהו מעט השתנה (אך בהחלט יכול להיות שמדובר בחוויה אישית בלבד). מלבד ההנאה בצפייה בסרט, ולהתפלסף על העלילה שלו עם חברים, הייתה בי ההרגשה הפנימית שלא סתם יוצרי הסרט בחרו בהאקר להיות הדמות הראשית (נכון שזה מסתדר אחלה עם כל העלילה וכו', אבל כנראה לא חשבתי על זה לעומק). הרגשתי שהבחירה לכך נבעה מאותה הרגשה שאני הרגשתי: באותן שנים (כך לפחות חשבתי) כמעט ואף אחד לא הבין איך באמת האקרים פועלים. גם כשאותם קטעי קוד סתומים שהרצתי פעלו לי, הייתה לי כמעט אפס הבנה מה עשיתי ואיך בדיוק זה קרה, וההרגשה הייתה שיש כאן איזשהו סוג של קסם שבמקרה הצלחתי להחשף להבנה של איך לבצע אותו - אבל לא למה הוא עובד.

במהלך העלילה, ניאו - גיבור הסרט - נחשף לעוד ועוד רבדים של "המציאות" בה הוא חי, הרבה מהסרט מתרחש במסדרונות "מאחורי הקלעים" של העולם. זאת בדיוק הייתה הרגשתי כלפי תחום ההאקינג: תחום אשר דורש לקלף שכבה ועוד שכבה מעל פני המציאות כדי להבין בדיוק איך משהו עובד, ורק אחרי ההבנה הזו ניתן לבצע את אותם "קסמים". שלל הניתוחים שיצאו על הסרט תמיד חשפו בפני עוד טפח ונתנו לי זווית חדשה על הסרט, בין אם אלו הרפרנסים השונים המוזכרים בו, אם ההקבלות לכל מני

פילוסופיות דתיות שונות ואם היו אלו סתם בדיחות קטנות שיוצרי הסרט שתלו פה ושם, כולן תמיד הזכירו לי שיש עוד רובד ועוד שכבה שאפשר לקלף.

בשנים שחלפו לאחר מכן יצאו עוד סרטי המשך, משחקים למחשב ולקונסולות אחרות ברוח הסרט, ועוד. פחות התחברתי אליהן (למרות שבילדות שלי הייתי יחסית Gamer, לא שיחקתי בהם אף פעם ולא הרגשתי את הצורך לנסות), ולמרות שנהנתי מסרטי ההמשך - שמרתי אמונים לסרט הראשון. המשכתי להרגיש כך למרות שבסרטי ההמשך יש לא מעט קריצות לתחום ההאקינג. העלילה בסרט הראשון הרגישה לי ממוקדת וסגורה יותר, גם האפקטים, והדיאלוגים הרגישו לי אמיתיים יותר, ובכלליות האווירה הרגישה לי טובה יותר.

מאז, הייתי רוצה להאמין, התבגרתי רבות (בכל זאת, עברו 20 שנה...), וגם זווית המבט שלי על תחום ההאקינג התבגרה לא מעט. אך את אותו ניצוץ של רגש שצץ בי כשראיתי לראשונה את הסרט, אני נושא בליבי עד היום. ובמחשבה לאחור - גם עם כמות הסרטים הרבה שיצא לי לראות ב-20 השנים האחרונות, אני חושב שלא ראיתי סרט טוב יותר ממנו.

הגליון הנוכחי מוקדש לסרט הנפלא הזה, לכל אותם האקרים בודדים שמסתובבים אי שם ברחבי הסייברספייס ולכל הכותבים שלנו - שבזכותם, ילדים בני 11 היום יכולים להשיג חומר איכותי בעברית ואינם צריכים להסתובב בכל מני פינות מפוקפקות באינטרנט ☺

וכמובן איך לא - נעבר לתודות: בזמן שכולנו בזבזנו את הזמן החודש, יש מי שניצל אותו בצורה נבונה וישב לכתוב מאמר למגזין, כדי שיהיה לכם גם הפעם איך להתחיל החודש על רגל ימין ☺, אז תודה רבה ל**כסיף דקל**, תודה רבה ל**רונן שוסטין**, תודה רבה ל**יהודה כורסיה**, תודה רבה ל**אייל איטקין**, תודה רבה ל**אור צינגיסר** ותודה רבה ל**גלעד זינגר**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

מבוא ל-Ghidra - חלק א'

מאת כסיף דקל ורונן שוסטין

הקדמה

בתחילת החודש, ארגון הביון האמריקאי, ה-NSA, שחרר את דרקון ה-Ghidra, כלי חינמי רב עוצמה שפותח במשך שנים רבות על ידי חטיבת המחקר של ה-NSA כדי לסייע במחקר פנימי. הארגון החליט לשחרר את הכלי כפרויקט קוד פתוח, תחילה בצורה חלקית, אך בעתיד יעלה הקוד בצורה מסודרת ל-Github הרשמי ויהיה ניתן לבנות אותו מקוד המקור.

אז מה הוא למעשה Ghidra? הכלי מהווה Reverse Engineering Framework שלם, הנתמך בפלטפורמות רבות. הכלי כולל בתוכו פיצ'רים כגון Disassembler, Decompiler, Assembler, Graphing, Scripting בדומה ל-IDA Pro, וגם שלל פיצ'רים נוספים.

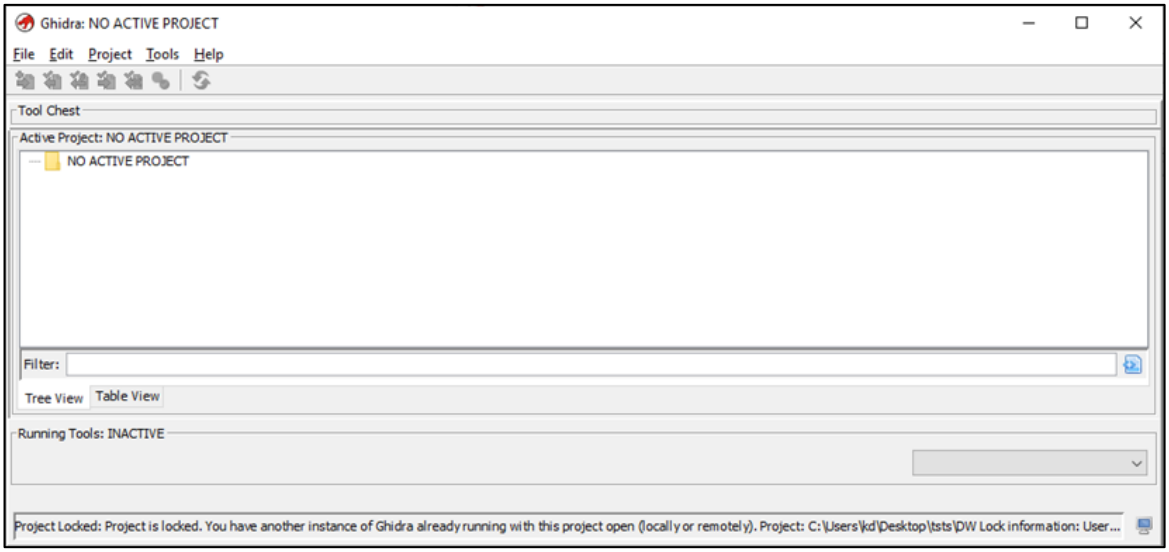
הכלי תומך במגוון רחב של Instruction Sets ומאפשר למשתמשים להוסיף Processors חדשים בקלות, דבר שכבר קרה, מספר פעמים, מהר מאוד עם שחרור הכלי. הכלי מאפשר סקריפטינג בשפות Java ו-Python וכמו כן במצב Headless CLI. עצם היותו כתוב ב-Java, שהיא שפה "ניידת", נתמך הכלי בכל מערכת הפעלה המאפשרת להריץ JVM.

הכלי אומץ במהרה על ידי הקהילה ונכון לרגע כתיבת המאמר, נכתבו לכלי מגוון רחב של פלאגינים, תמיכה ב-CPUs נוספים, מאמרים וסרטונים רבים.

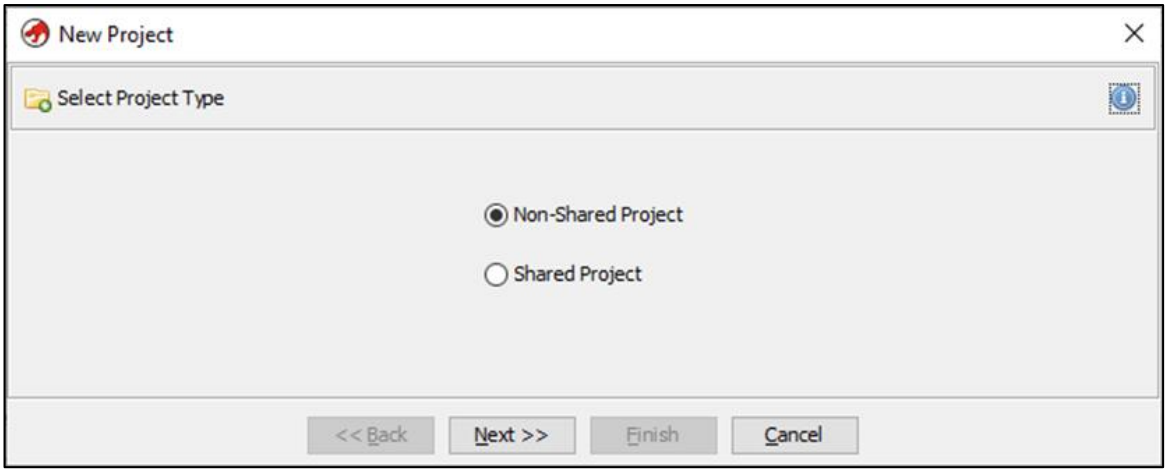
במאמר זה לא נעסוק ב-Reversing באופן כללי אלא נסקור את התכונות העיקריות וניתן טיפים כלליים לשימוש בכלי, הבה נתחיל.

הקמת פרויקט חדש

לאחר התקנת Ghidra ופתיחתה יופיע החלון הבא:



חלון זה נקרא Project Management, למעשה כל דבר הוא פרויקט ב-Ghidra, בשונה מ-IDA, לא ניתן להתחיל את העבודה רק על ידי טעינת קובץ Input חדש. כאשר תפתחו את התוכנה בפעם הראשונה ותרצו להקים פרויקט חדש, יופיע החלון הבא:



תחילה תצטרכו לבחור האם מדובר בפרויקט משותף. התוכנה מאפשרת עבודה משותפת על פרויקטים וסנכרון בין המשתמשים השונים תחת אותו פרויקט באמצעות שרת מרכזי, נבחר ליצור פרויקט רגיל ונמשיך.

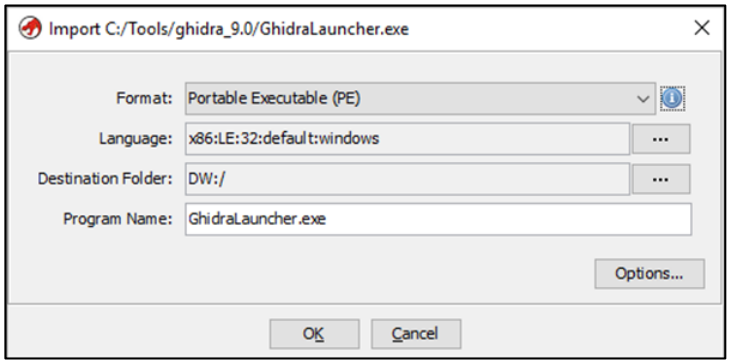
טעינת קבצים חדשים לפרויקט

במסך לאחר מכן תצטרכו לבחור שם לפרויקט ואת המיקום בדיסק בו הוא ישמר. לאחר יצירת הפרויקט נטען בינארי חדש באמצעות File ואז לחיצה על Import File או על ידי קיצור המקשים i, ניתן גם לגרור את

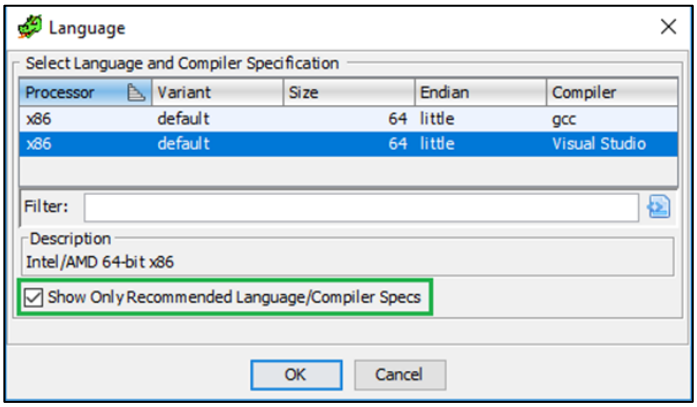
הקובץ ישירות לתוך החלון. נכון לרגע כתיבת המאמר, הכלי תומך בפורמטים הבאים Out-of-the-box (ללא פלאגינים חיצוניים):

- GZF Input Format
- Ghidra Data Type Archive Format
- XML Input Format
- Common Object File Format (COFF)
- Dalvik Executable (DEX)
- Debug Symbols (DBG)
- Executable and Linking Format (ELF)
- Java Class File
- MS Common Object File Format (COFF)
- Mac OS X Mach-O
- Module Definition (DEF)
- New Executable (NE)
- Portable Executable (PE)
- Preferred Executable Format (PEF)
- Program Mapfile (MAP)
- Relocatble Object Module Format (OMF)
- Old-style DOS Executable (MZ)
- Intel Hex
- Motorola Hex
- Raw Binary

וגם במגוון ארכיטקטורות CPU רחב, בעת טעינת קובץ חדש נקבל את המסך הבא:

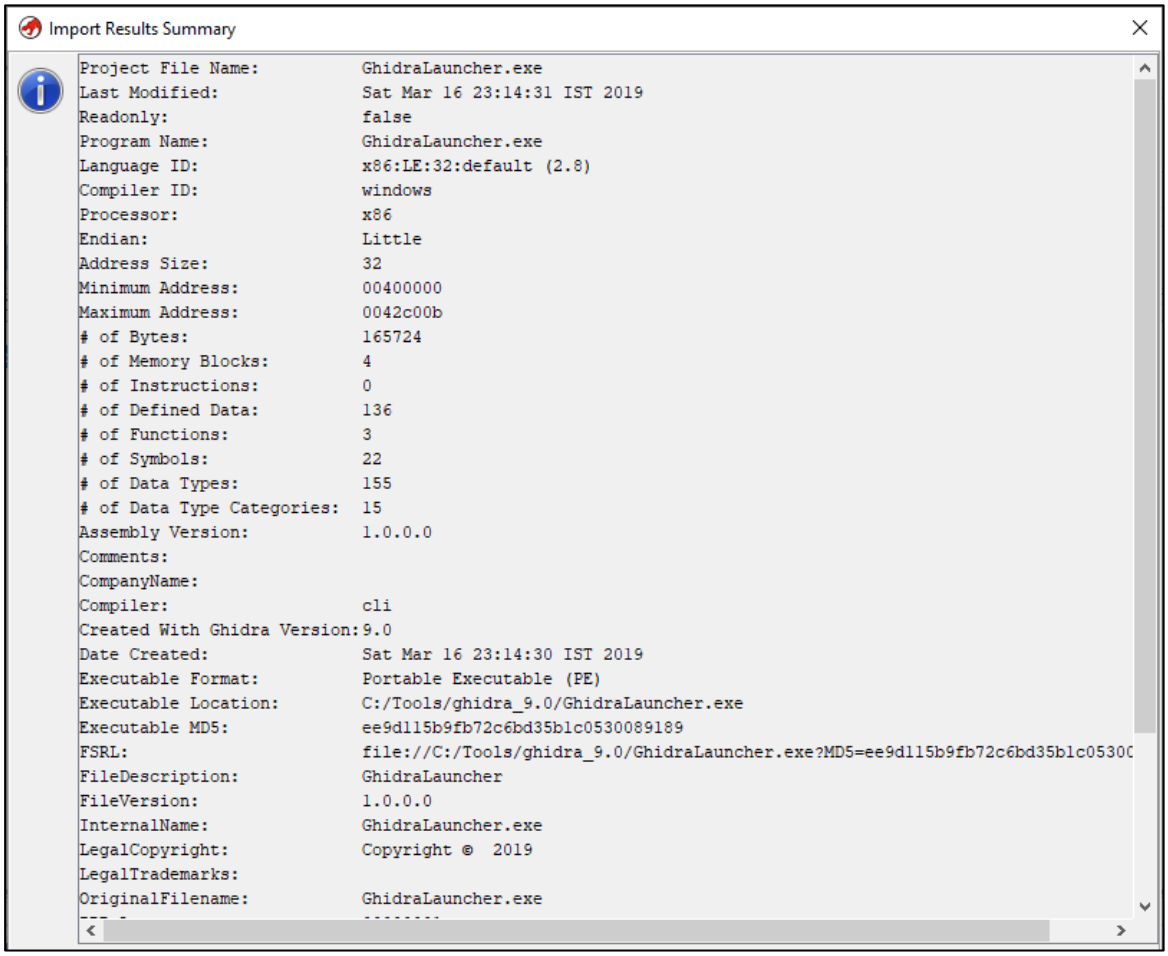


ניתן לראות ש-Ghidra כבר הבינה לבד באיזה קובץ מדובר (במידה וניתן לזיהוי) ואילו הגדרות להחיל עליו, אך במידה והנכם טוענים קובץ Raw מסוג כלשהו, למשל Firmware, או שצריך איזשהם כוונונים נוספים ניתן לעשות זאת על ידי לחיצה על כפתור שלוש נקודות ליד שדה ה-Language:

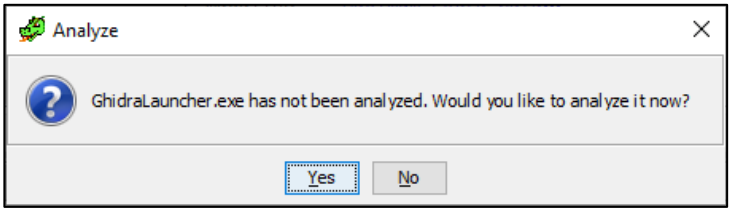


כאן ניתן לכוונן את בחירותיה של Ghidra לגבי הקובץ, כגון ארכיטקטורה (ביטול הסימון בעיגול הירוק יציג ארכיטקטורות נוספות אך לרוב לא תצטרכו לגעת בזה כאשר תתעסקו עם קבצים בעלי פורמט מוכר), סוג קומפיילר, סדר בתים (Endianness) וכדומה.

לאחר טעינת הקובץ, התוכנה תציג לנו סקירה מהירה של הקובץ:

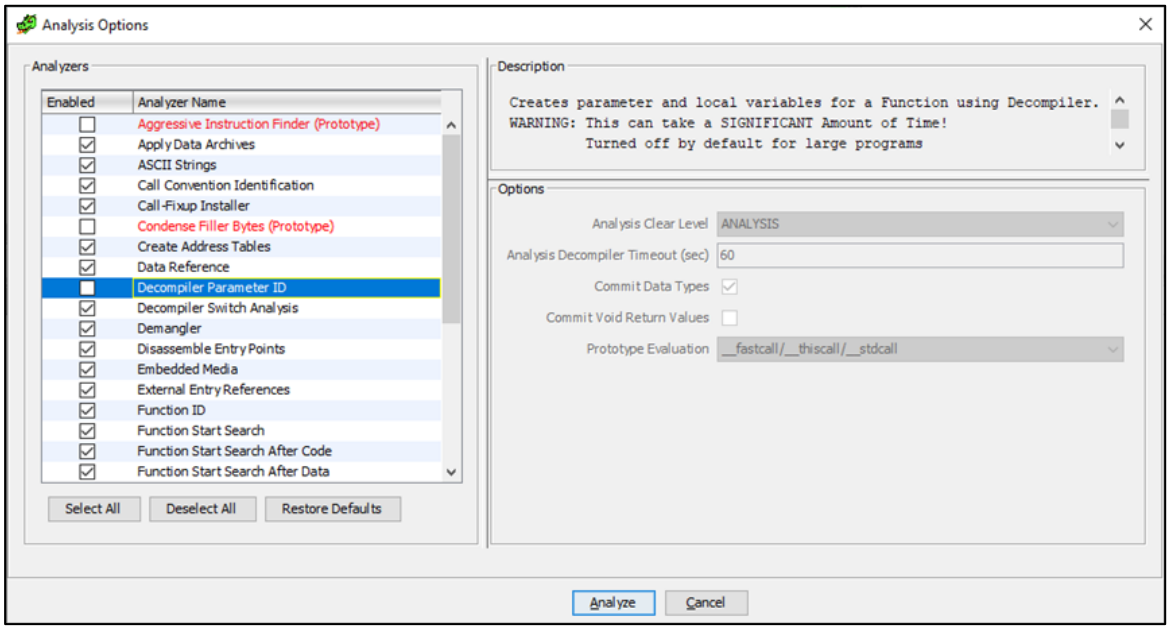


לחיצה כפולה על הקובץ שנטען או על הדרקון הירוק תפתח את מסך ה-CodeBrowser, מסך זה הינו המסך העיקרי איתו תעבדו בתוכנה ולמעשה משמש מין מסך ראשי לשאר המסכים. לאחר האנימציה המגניבה כמובן, נשאל אם ברצוננו להפעיל את ה-Auto Analysis כרגע:



כעת ייפתח חלון נוסף שבו Ghidra תאפשר לנו לבחור הגדרות נוספות לגבי האנליזה האוטומטית, הגדרות אלה לרוב גנריות אך יש כמה אשר מותאמות לארכיטקטורות מסוימות בלבד.

הכלי מגיע עם המון Analysis Tools, ביניהם חיפוש מחרוזות, טעינת Symbol Data ואפילו חיפוש של תמונות ומדיה נוספת. אנו ממליצים לסמן את האופציה הזו לקבלת תוצאה טובה (משופרת) יותר במסך ה-Decompiler שנראה בהמשך:



ניתן גם להפעיל את ה-Auto Analysis בשלב מאוחר יותר באופן ידני על ידי:

Analysis -> Auto-Analysis

בהפעלתו, לכל הפחות יתבצעו הדברים הבאים:

- התחלה ב-Entry Point
- ביצוע Disassembly
- יצירת פונקציות
- יצירת רפרנסים

כפי שתואר לעיל, ע"פ מה שסומן בהגדרות, כלי ה-Auto Analysis יבצע ניתוחים נוספים כגון:

- שימוש באופרנדים על מנת לייצר רפרנסים לקוד או Data היכן שהאופרנדים מצביעים
- יצירת Switch Statements ו-Function Signatures על בסיס מידע מ-Decompiler
- ביצוע Demangle ל-Mangled Symbols
- ניתוחים נוספים על בסיס סוג הקובץ הטעון ועוד

עבור קבצים רבים, הניתוח מסוגל להניב תוצאות טובות יותר בגישת "שכבות", כלומר, עדיף יהיה להפעיל תחילה ניתוח בסיסי בלבד, ולאחר מכן להריץ אפשרויות ניתוח אחרות באופן ידני.

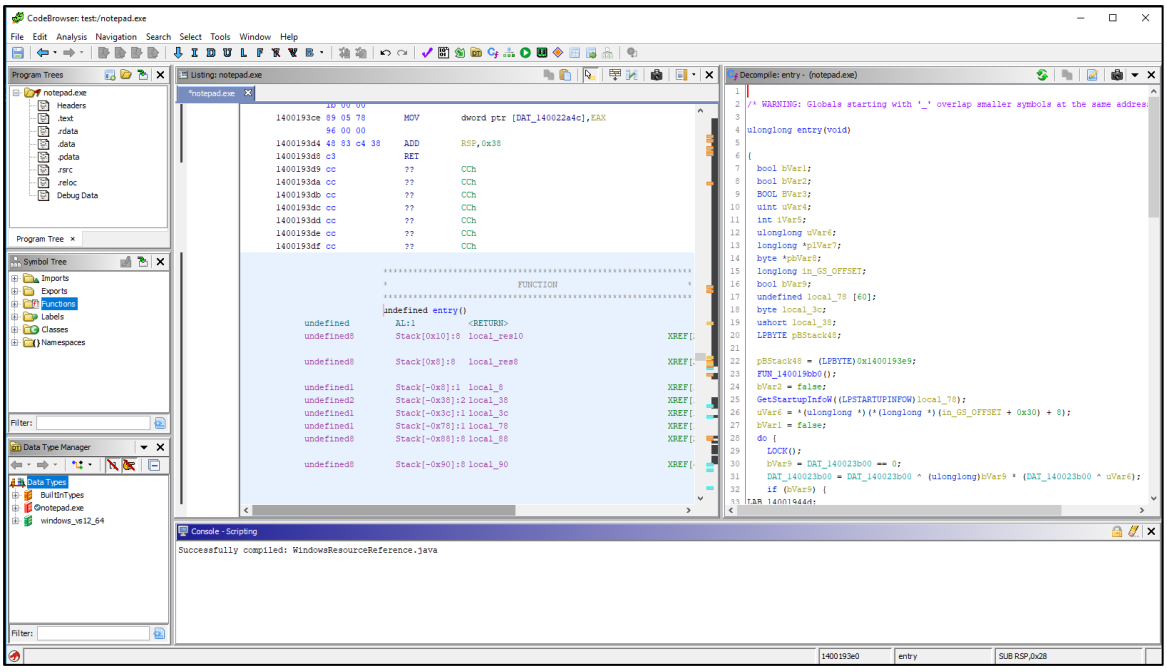
ניתוח בסיסי:

- יש לכבות כל ניתוח ספקולטיבי או מיוחד, כמו ניתוח המחסנית, בייחוד אם מדובר בקובץ גדול. ניתן יהיה להפעיל אפשרויות אלו שוב לאחר מכן, על כלל הקובץ או פונקציות מסוימות.
- באופן כללי, על מנת שניתן יהיה להבחין בין Code לבין Data עדיף להשאיר את ה-Analyzers הבאים דלוקים: Reference Analyzers, Data Analyzers, Function Analyzers.

ניתוח המשכי:

- ניתן להפעיל את ה-Auto Analysis מחדש עם אפשרויות אחרות או להפעיל כל אופציה בנפרד על ידי לחיצה על Analysis ואז One-Shot. ניתוחים נפוצים מומלצים:
  - Decompiler Parameter ID
  - Decompiler Switch Analysis
  - Stack Analyzers
  - Etc

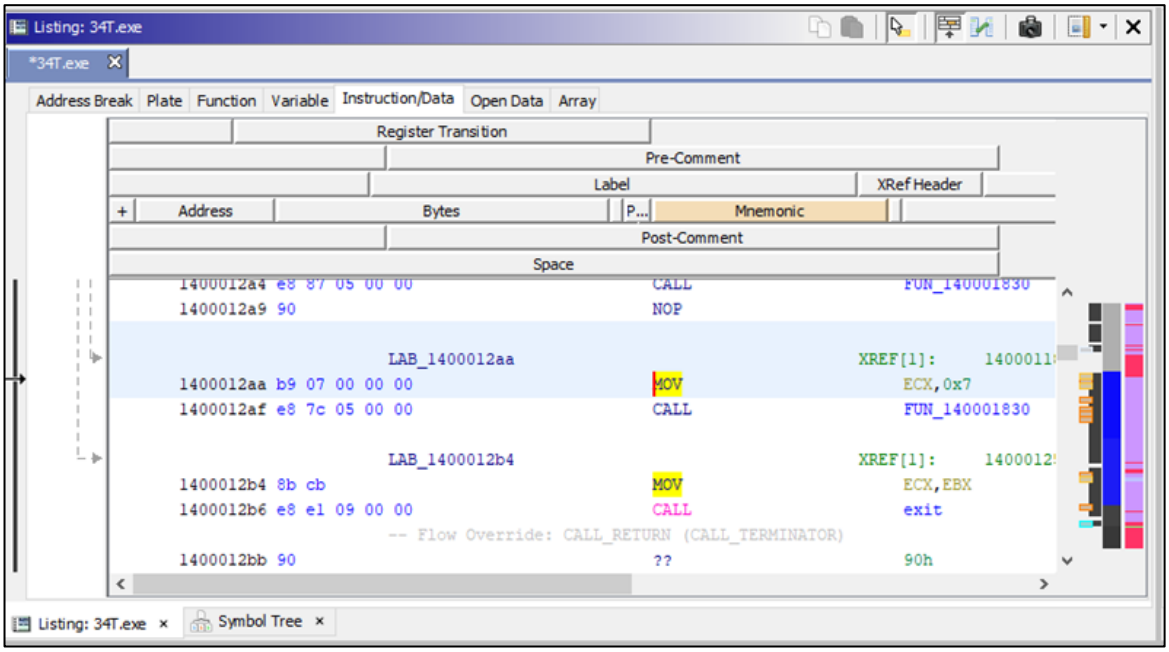
סביבת העבודה



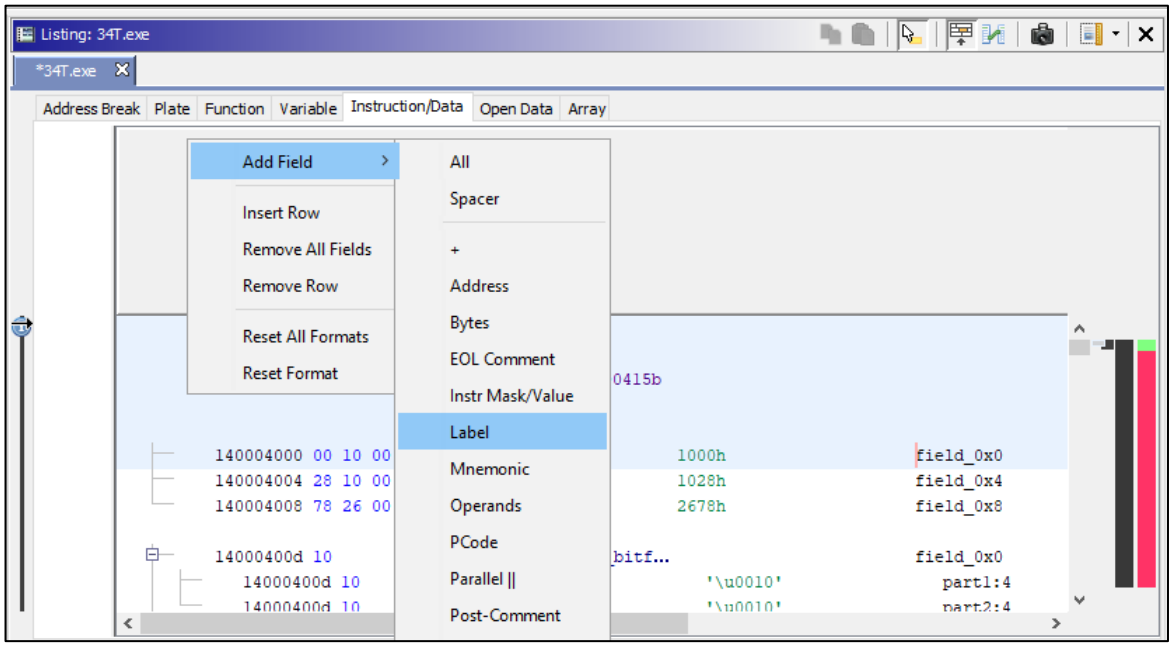
כך נראה החלון הראשי של Ghidra, שמו של חלון זה נקרא CodeBrowser. רוב הממשקים הויזואלים קיימים בתוך חלון זה, למשל: התפריטים הראשיים, חלונות ה-Listing, Decompiler, Graph, מנהל הטיפוסים, חיפוש מחרוזות, עורך Hex ועוד. רוב עבודת הניתוח תתבצע תחת חלון זה. הממשק מחולק לחלונות שאותם שניתן להזיז ולשנות. במהלך המאמר נסקור את הפיצ'רים הקיימים במערכת, נבין כיצד להשתמש בהם ונחלוק טיפים אשר ישפרו את חווית השימוש בתוכנה ואופן העבודה איתה. סביבת העבודה עמוסה בפיצ'רים ומכל טוב, ומפאת זאת לא נוכל לסקור את כולם אך נדבר על הדברים העיקריים והשימושיים.

חלון ה-Listing

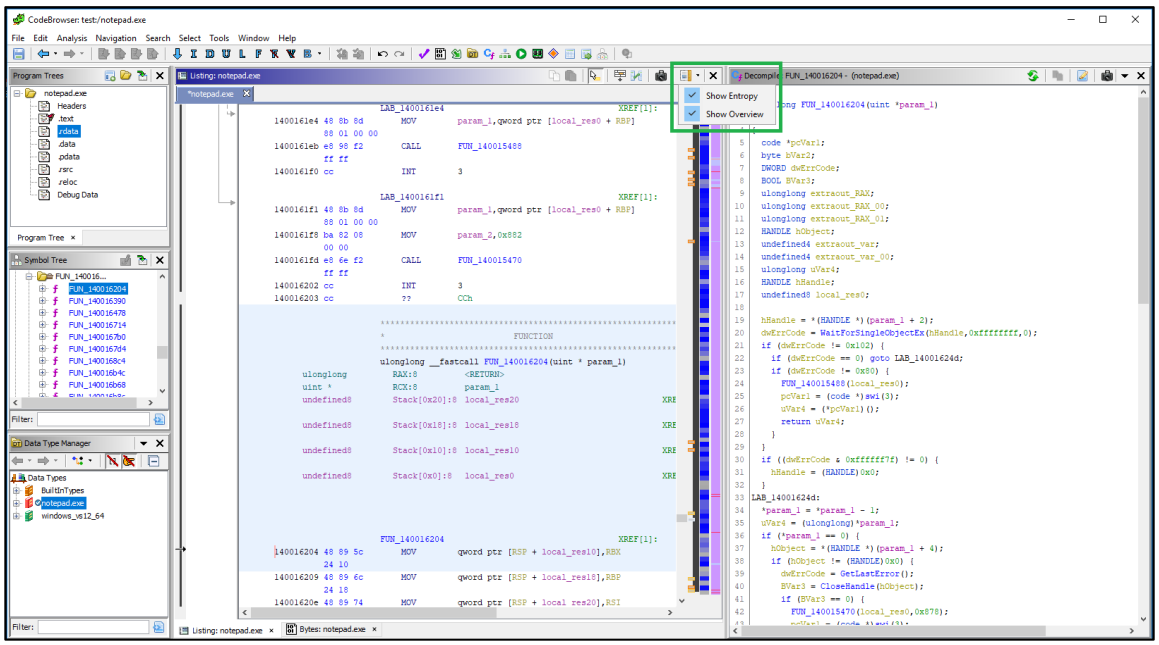
חלון זה הוא למעשה המקביל ל-IDA View-A במצב Flat ובדומה לו הוא מכיל ניתוח Disassembly, Data ואפילו דברים נוספים (בין היתר, אף הצגת מדיה). לחיצה על כפתור ה-  Edit תאפשר התאמה אישית של השדות בחלון:



ניתן להוסיף שדות נוספים על ידי לחיצה ימנית על משטח העורך ולחיצה על Add Field:



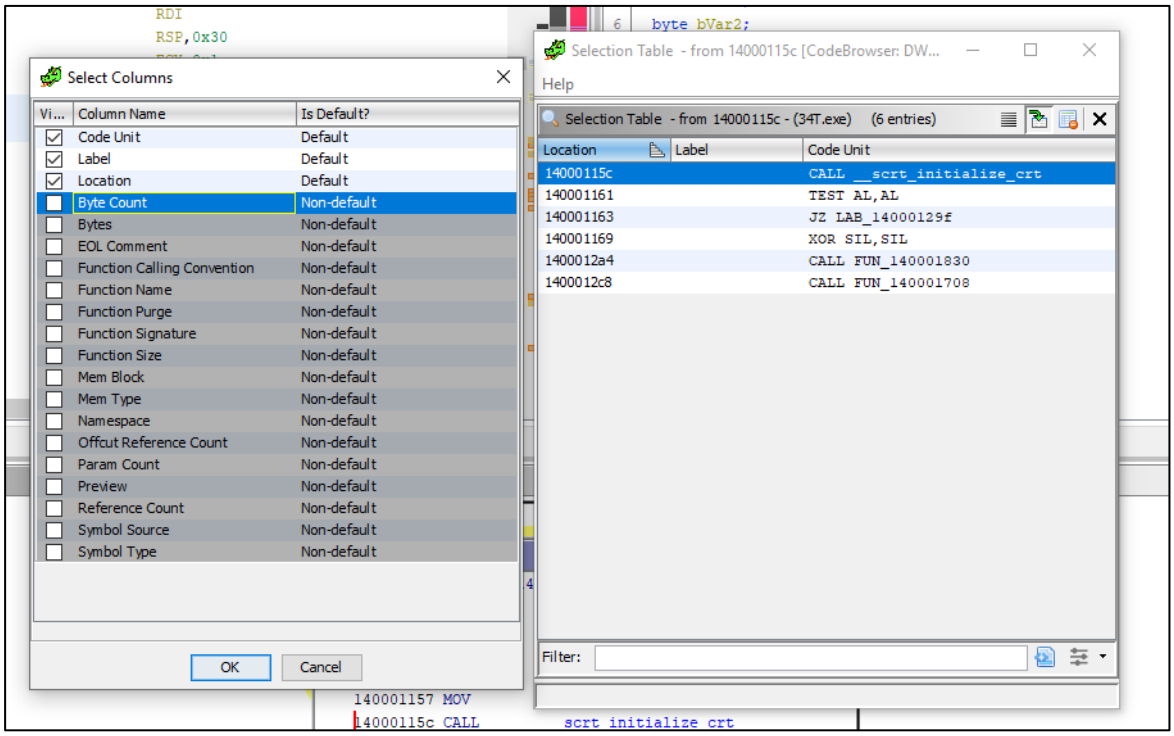
חלון ה-Listing מאפשר לקבל תצוגה ויזואלית של האנטרופיה בקובץ הנוכחי ו-Overview כללי של מפת הקובץ בדומה לפס העליון ב-IDA:



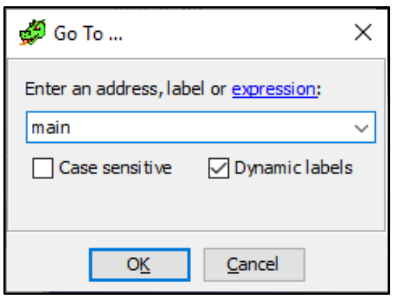
הכלי מאפשר לייצר טבלאות כמעט מכל אלמנט בתוכנה, על מנת להפעיל את הפיצ'ר סמנו את האלמנטים הרצויים ואז לחצו על:

Select -> Create Table From Selection

וכמובן שניתן להוסיף/להסיר עמודות כמו בכל טבלה בתוכנה, כך זה נראה:



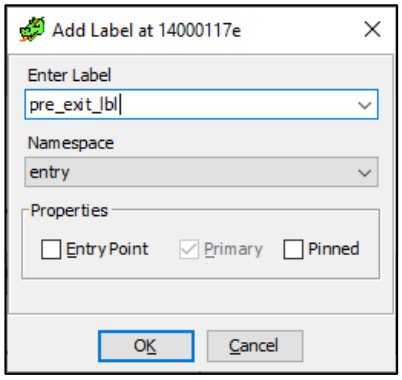
לאחר טעינת המידע הרצוי לטבלה ניתן לבצע עליה חיתוכים מתקדמים. על מנת "לקפוץ" אל Labels או כתובות ניתן ללחוץ על המקש 'G' בדומה ל-IDA:



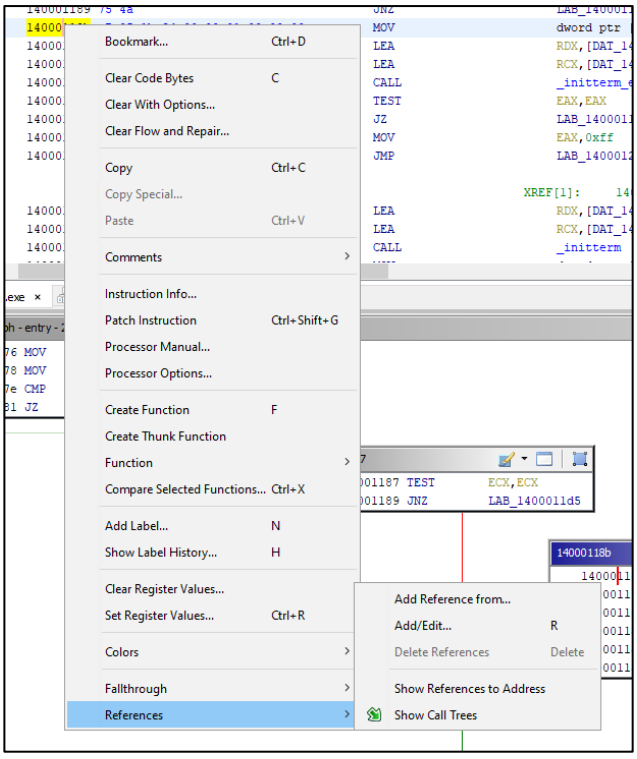
את כל קיצורי המקשים ניתן לשנות בנתיב ההגדרות הבא:

`Edit -> Tool Options -> Key Bindings`

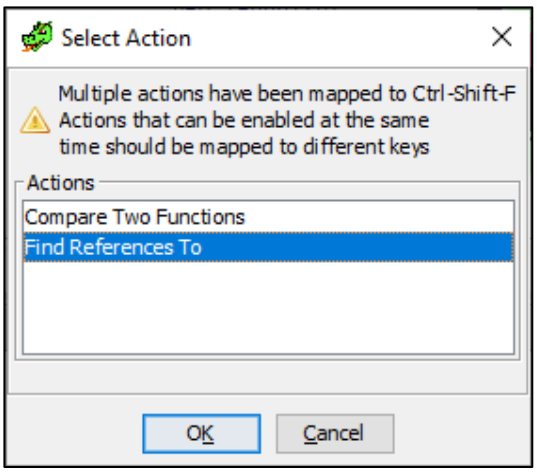
כדי לתת שם לפונקציה, Label, משתנה, או כל דבר אחר שאפשר לתת לו שם, לחצו על המקש 'L':



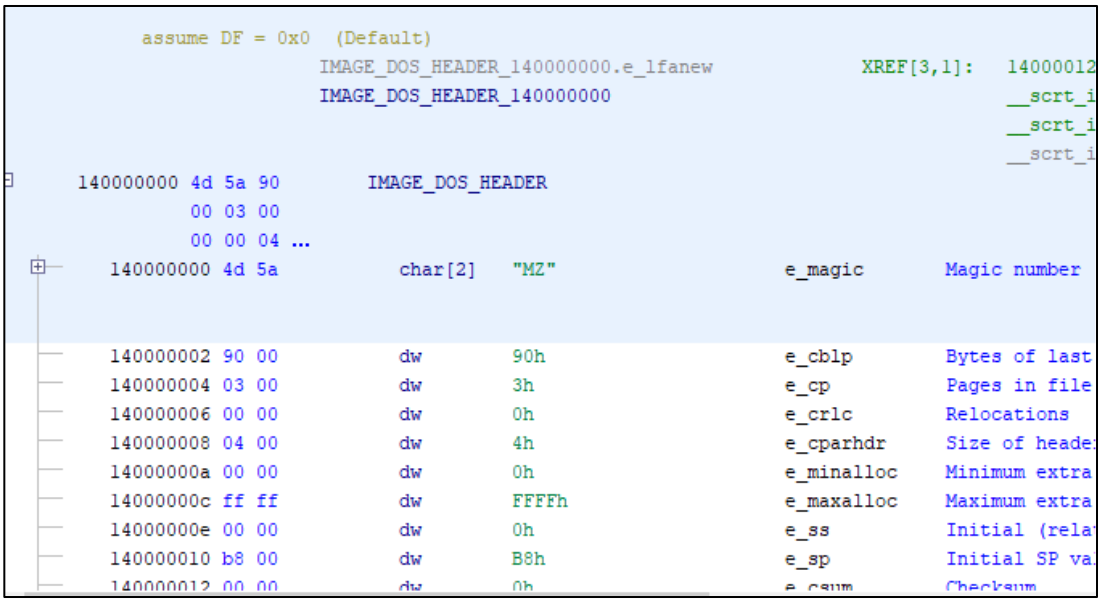
על מנת לחפש Cross References ניתן להשתמש בעכבר על ידי לחיצה ימנית על האובייקט הרצוי:



או על ידי קיצור במקלדת CTRL + SHIFT + F:

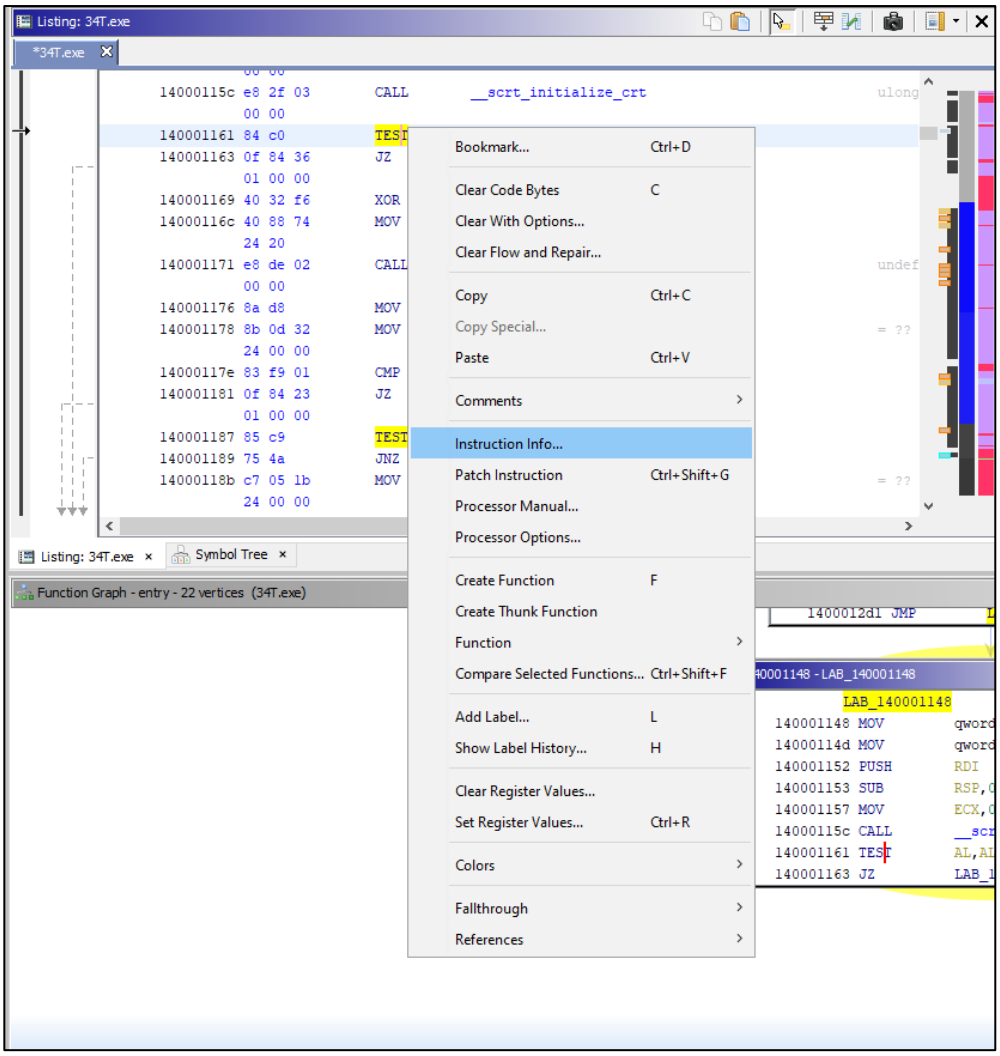


דבר נוסף ששווה להזכיר בהקשר של חלון ה-Listing הוא שהתוכנה מפרסרת באופן אוטומטי את הפורמט של הקובץ שנטען ומציגה את ה-Headers בחלון ה-Listing:

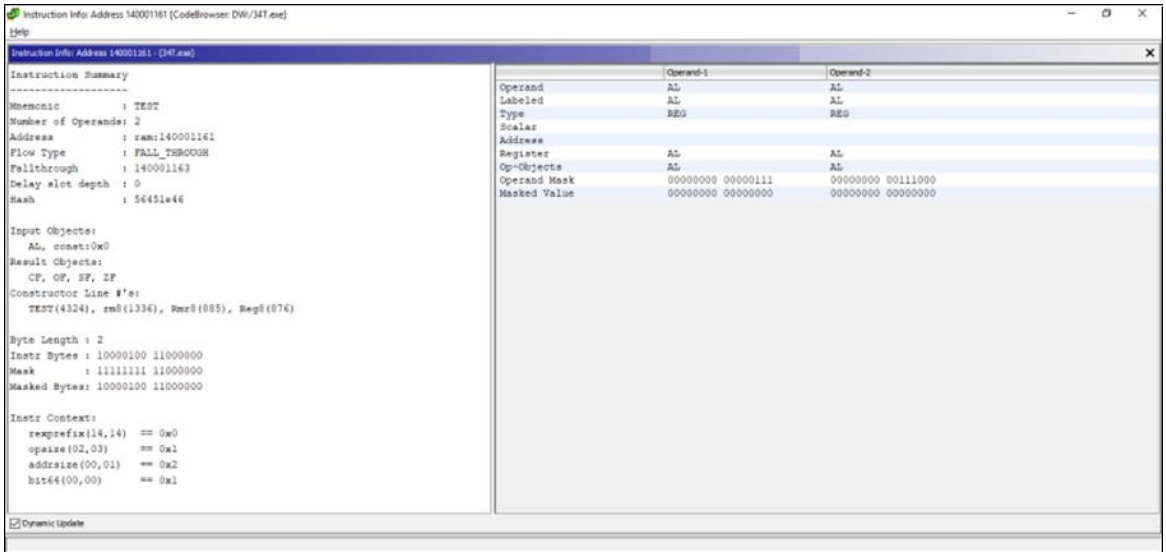


דבר שבהחלט יכול להיות שימושי.

במידה ואתם לא סגורים על פקודת מעבד מסוימת או צריכים תזכורת, תוכלו לסמן אותה וללחוץ על Instruction Info:



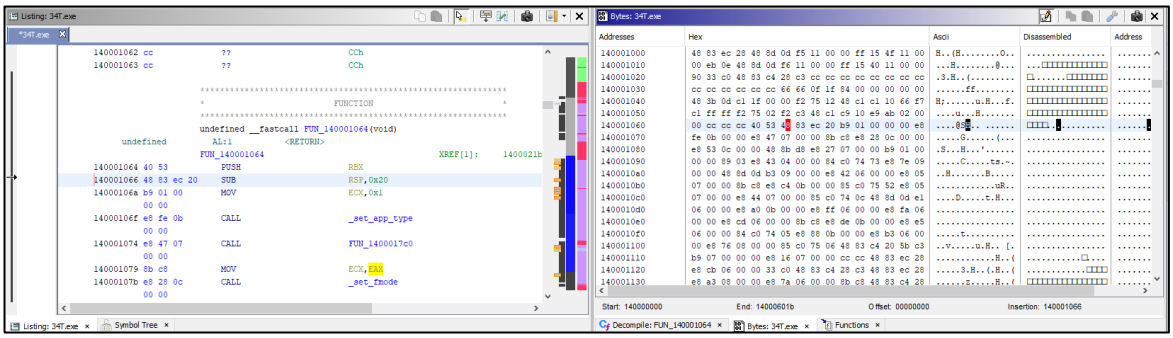
מיד ייפתח החלון החביב הבא אשר יציג פירוט שימושי מאוד אודות הפקודה:



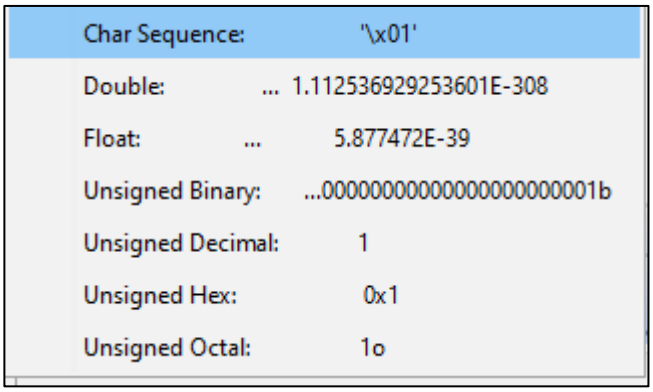
ניתן לערוך פקודות מעבד גם בחלון ה-Listing ובחלונות נוספים על ידי לחיצה ימנית בעכבר על הפקודה הרצויה ובחירה ב-Patch Instruction (או בקיצור המקשים CTRL + SHIFT + G):

```
140001000 48 83 ec 28      SUB     RSP,0x28
140001004 48 8d 0d         LEA     RCX,[0x140002200]
          f5 11 00 00
14000100b ff 15 4f         CALL    qword ptr [->API-MS-WIN-CRT-STDIO-L1-1-
          11 00 00
```

כעת נוכל לערוך את ה-Instruction כרצוננו, כמו כן ניתן לערוך חלקים במודול גם באמצעות Hexeditor על ידי לחיצה על הכפתור: 

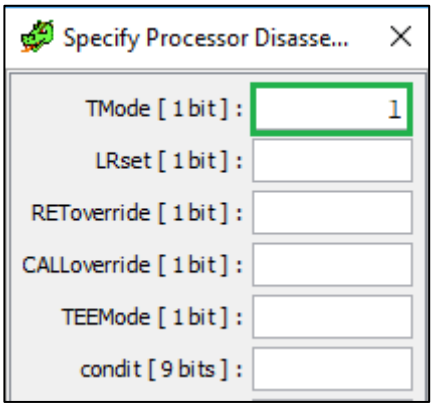


סימון ערך כלשהו בחלון ה-Listing ולחיצה על Convert בתפריט הלחצן הימני תציג בפניכם את החלון הבא:



לחיצה על אחת מן האופציות הנ"ל תחיל את השינוי בחלון ה-Listing.

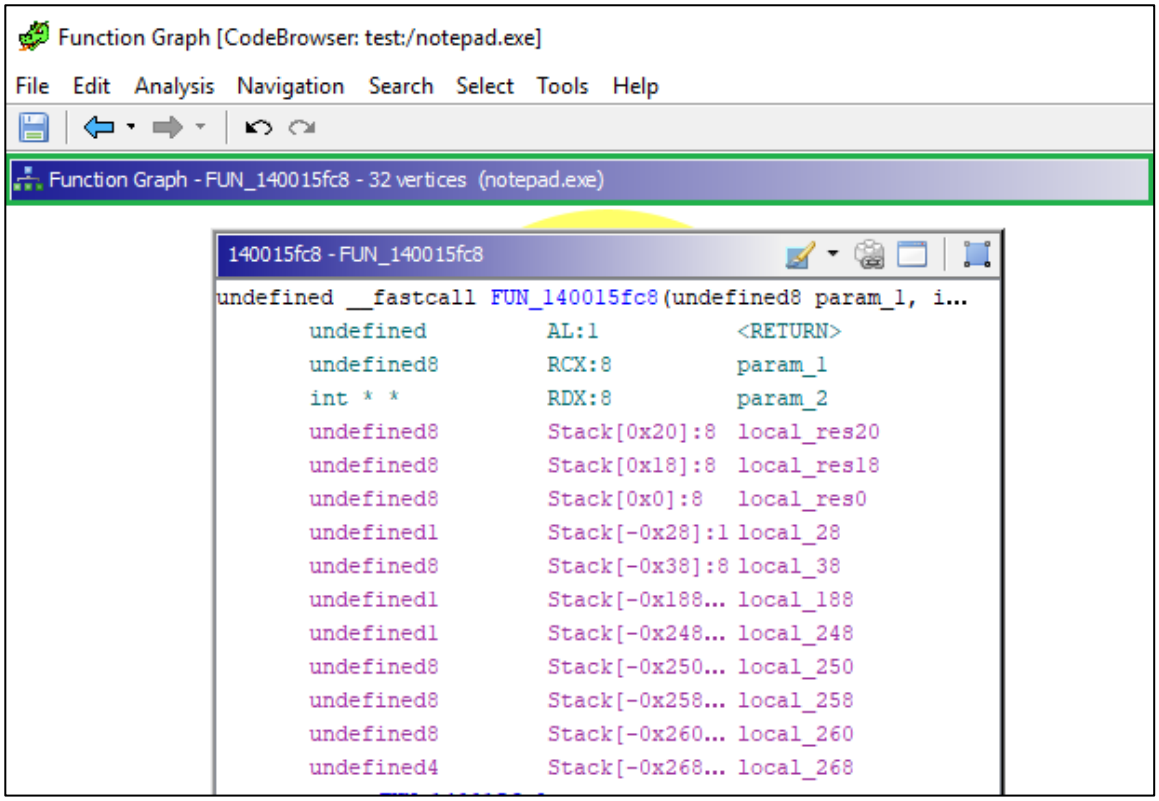
קיימים דברים פחות טריוויאלים בתוכנה כמו למשל איך לאכוף Disassembly במצב Thumb כאשר מתעסקים עם קבצי ARM. על מנת לבצע פעולה זו יש לבחור ב-"Processor Disassembly Options" בתפריט הלחצן הימני ואז להחיל את השינוי הבא:



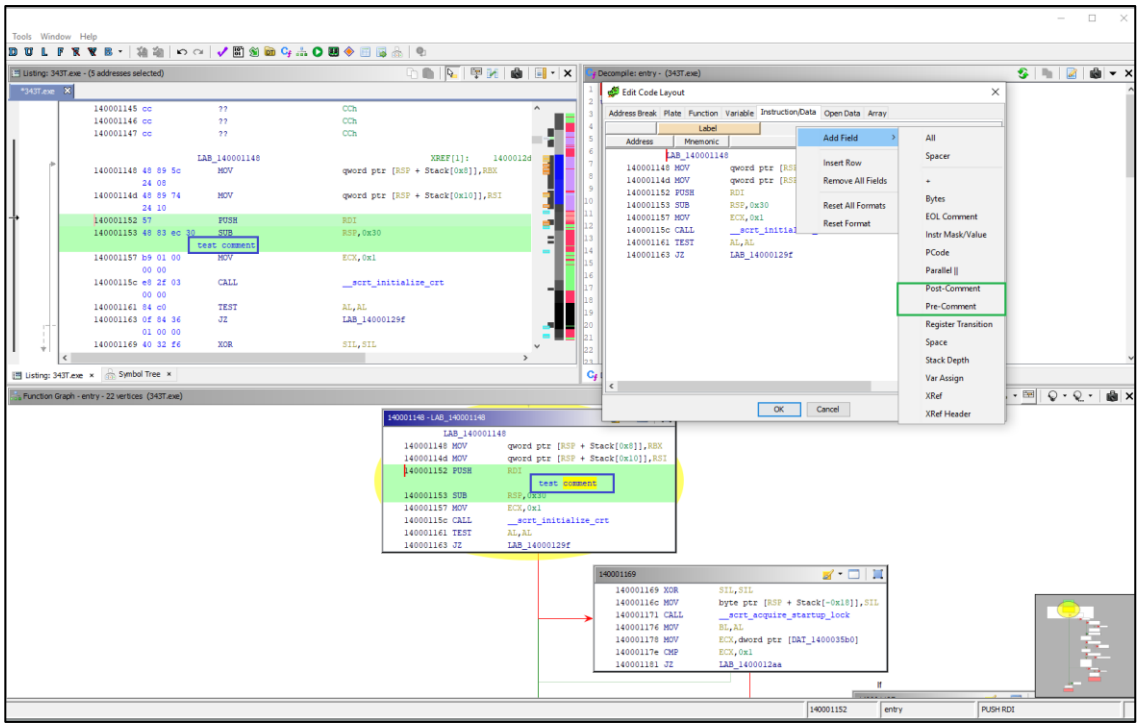
כפי שוודאי הבחנתם, תפריט הלחצן הימני מלא בפיצ'רים שימושיים. ככלל ב-Ghidra, רצוי להשתמש בלחצן הימני כאשר אתם מחפשים משהו, התשובה עשויה להיות שם.

חלון ה-Graph

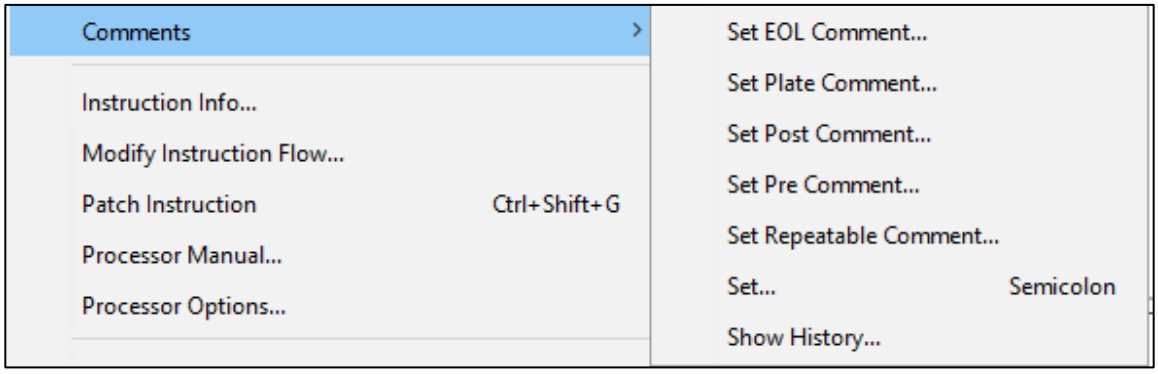
החלון אינו פעיל כברירת מחדל, ניתן לפתוח אותו על ידי לחיצה על  או בתפריט Window ואז לחיצה על Function Graph. החלון יפתח בנפרד, אך ניתן להצמיד אותו לחלון הראשי. יש לגרור את החלון הפנימי (עם הפס הכחול) ולא את המעטפת:



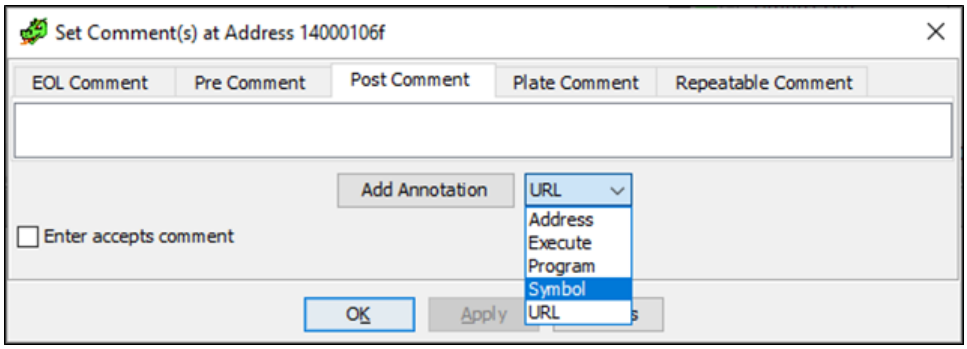
דבר נוסף שלא קיים כברירת מחדל הינו הצגת הערות בחלון ה-Graph. כפי שהוסבר לעיל, ניתן לערוך גם את חלון הגרף באמצעות הכפתור  , נלחץ על הכפתור ונבצע את הפעולות הבאות:



הערות ניתן להוסיף על ידי לחיצה ימנית על השטח בו ברצונכם להוסיף הערה, ואז על סוג ההערה הרצוי:

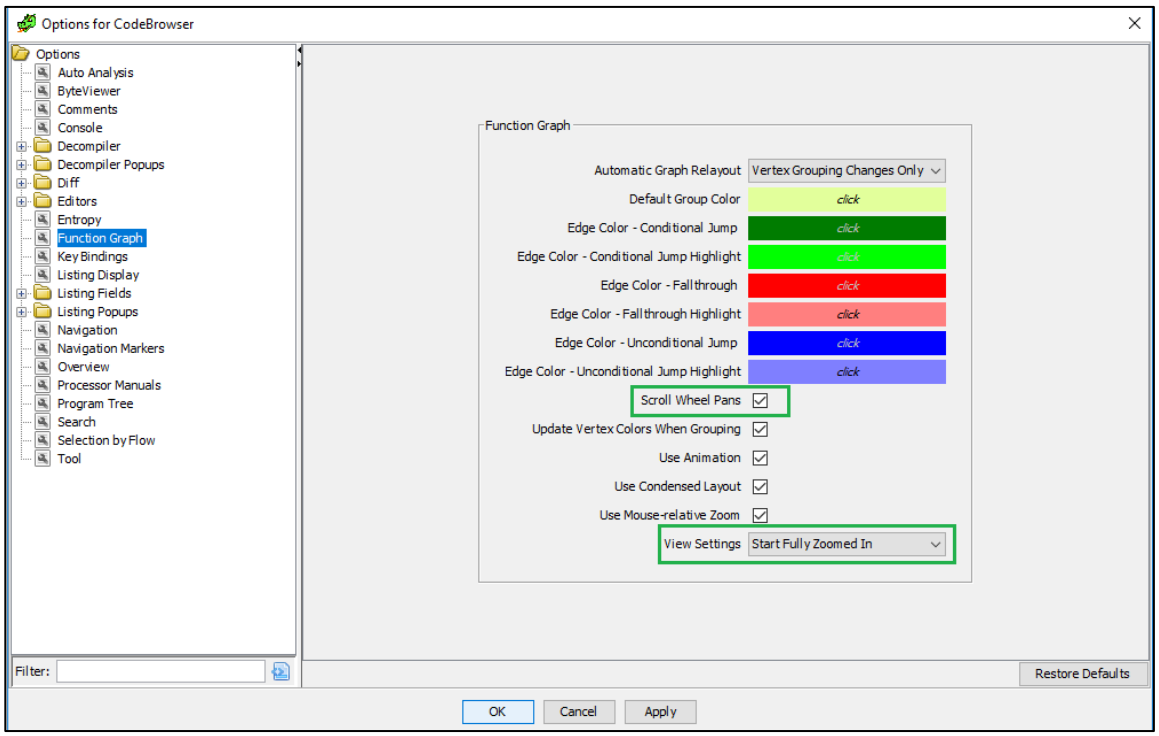


כעת יש לבחור את סוג ההערה הרצוי, פיצ'ר חשוב נוסף בעניין זה הוא שניתן להוסיף Symbol להערות על ידי:



כאשר ישתנה שם ה-Label, יתבצע עדכון באופן אוטומטית גם בהערה. על מנת לאפשר "מעבר נוח" למשתמשי IDA, ניתן לשנות את ההגדרות הבאות בנתיב:

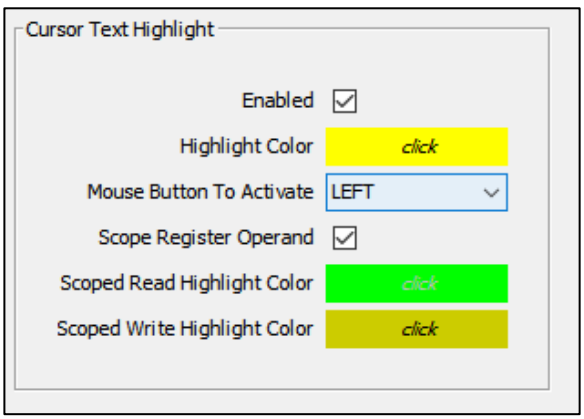
Edit -> Tool Options -> Function Graph



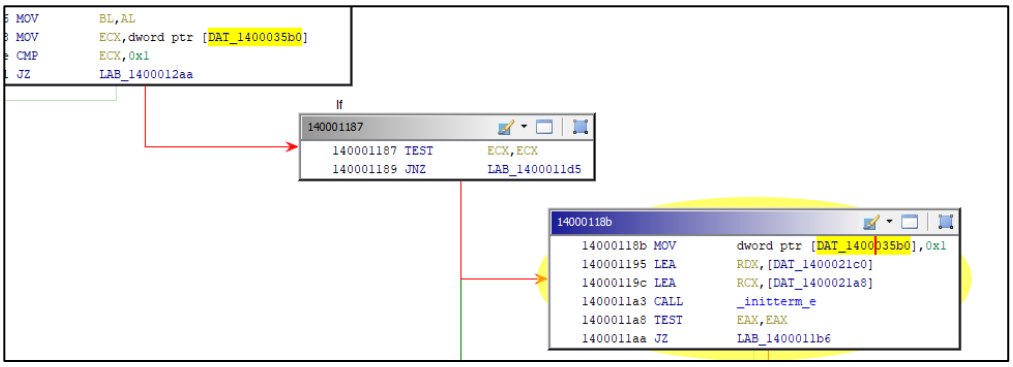
כפי שניתן להסיק לבד, שינוי ההגדרות הנ"ל יאפשר גלילה בגרף באמצעות הגלגלת בעכבר והגרף ייפתח במצב Zoomed in כברירת מחדל.

כמו כן, על מנת לבצע Highlight למשתנה, קבוע, אוגר או פקודה כלשהו יש ללחוץ על הלחצן האמצעי בעכבר (Scrolling button), ניתן לשנות גם הגדרה זו על מנת לאפשר IDA-like behavior על ידי ביצוע השינוי הבא:

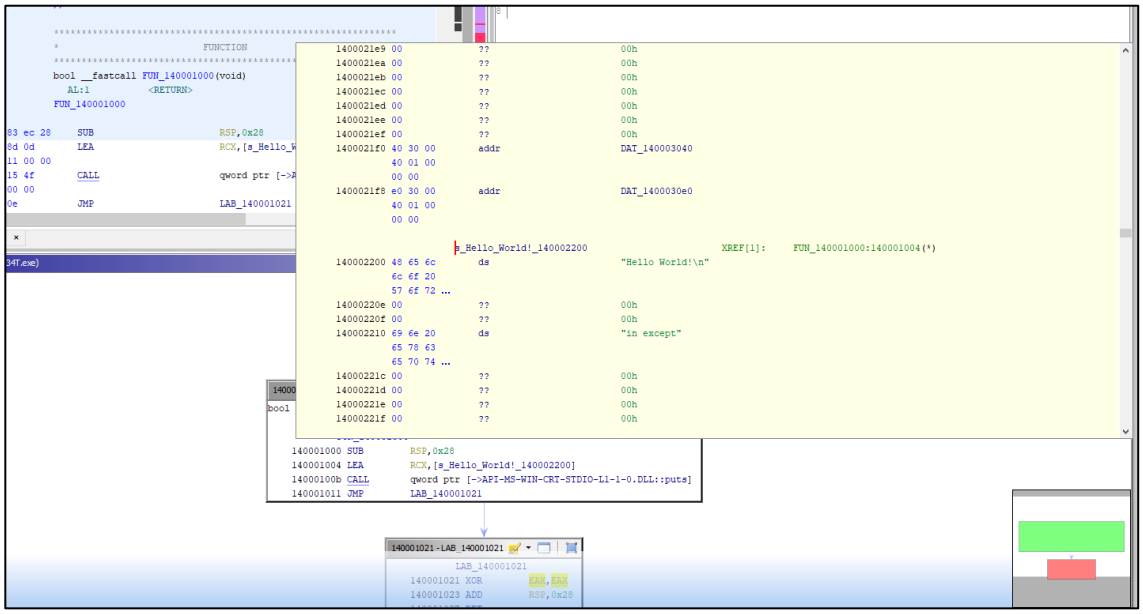
Edit -> Tool Options -> Listing Fields -> Cursor Text Highlight



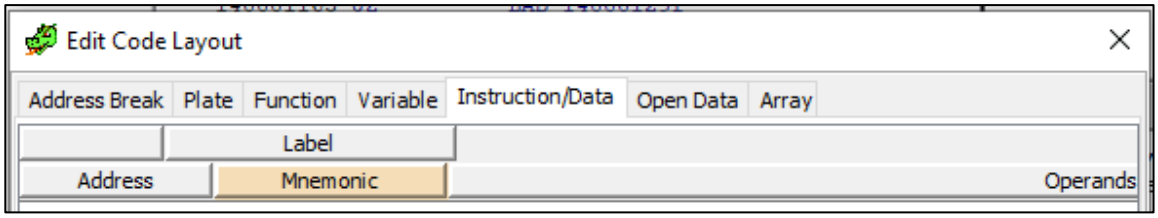
כעת התוכנה תמרקר לנו מופעים נוספים של האובייקט המסומן:



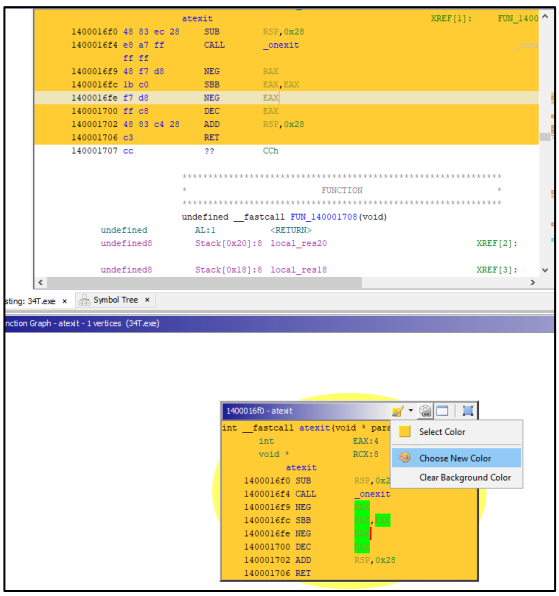
בדומה ל-IDA, כאשר תעמדו עם העכבר על אובייקט שמוביל למיקום אחר בגרף או ב-Listing תוצג בפניכם תצוגה מקדימה שנראת כך:



כברירת מחדל, במצב הגרף יש הגבלה לאורך הקוד המוצג, ניתן להגדיל אותו באופן הבא: לחיצה על העורך  ולאחר מכן לגרור את קצוות השדה לאורך הרצוי, כפי שמופיע בתמונה:



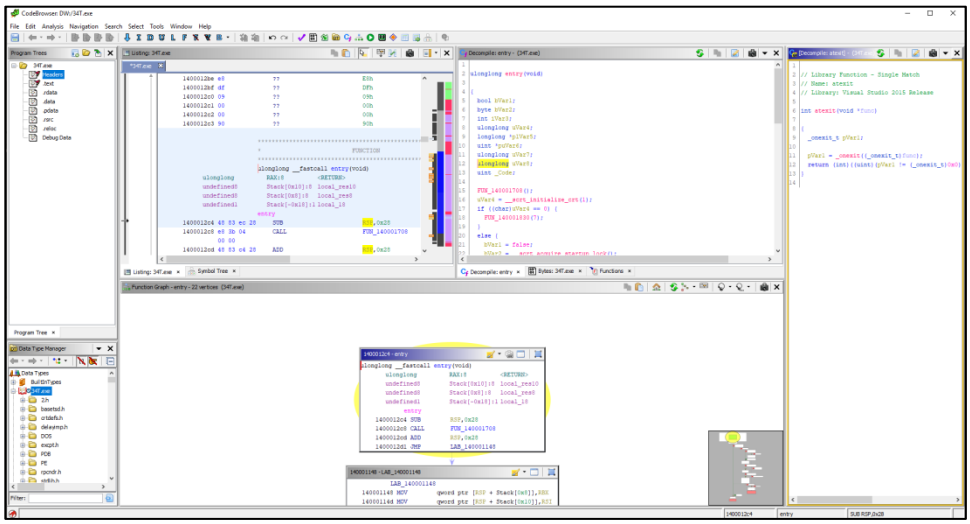
עוד פיצ'ר שימושי בגרף הוא צביעת Code blocks מסוימים בצבעים, ניתן לעשות זאת על ידי לחיצה על הכפתור  :



כפי שניתן לראות, הגרף מסתנכן עם חלון ה-Listing והחלקים הרלוונטים בקוד נצבעים גם שם. באופן כללי, התוכנה שומרת על סנכרון של הסימון הנוכחי בין שלושת החלונות העיקריים: Listing, Decompiler וכמובן ה-Graph. ניתן להרחיב זאת לחלונות נוספים כפי שנראה בהמשך.

כפתור ה-Snapshot שנראה כך:  מאפשר לכם ליצור עותק של החלון הנוכחי על מנת למנוע "טיולים" מיותרים בניווט בין מיקומים בקובץ, פיצ'ר זה הינו רלוונטי לכלל חלונות התוכנה שמציגים קוד שנראה בהמשך. במידה ותרצו לנווט בהיסטוריית המיקומים, תוכלו לעשות זאת על ידי שימוש בכפתורים הללו בצד השמאלי-עליון של המסך או על ידי קיצורי המקשים ALT + RIGHT/LEFT ARROW. 

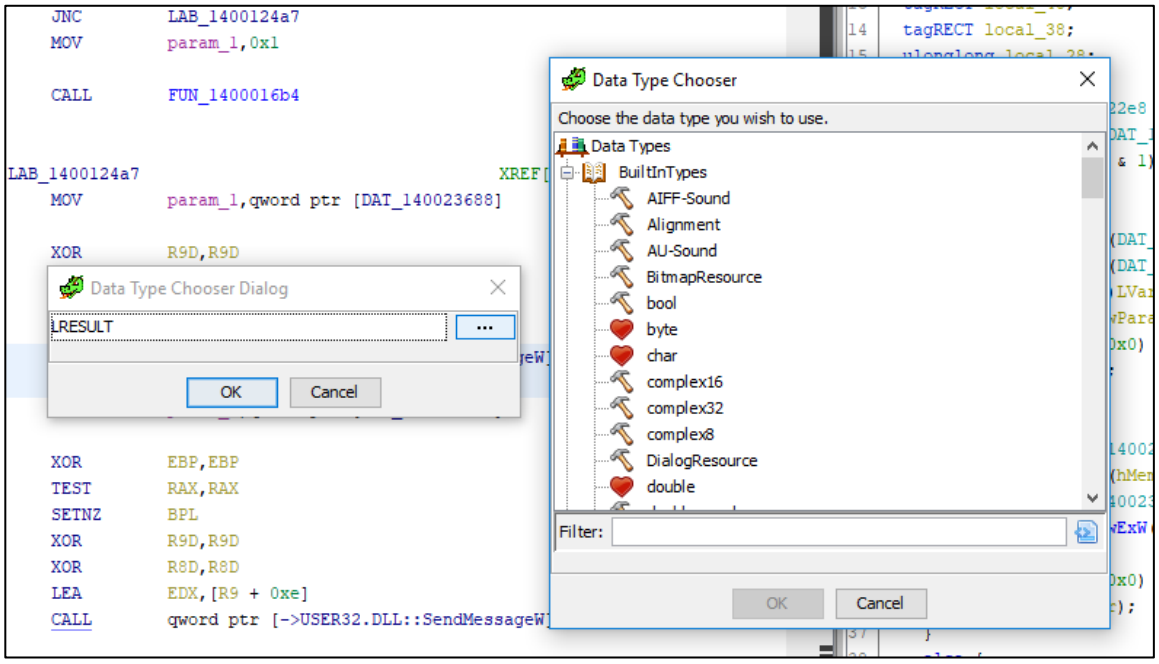
כשתיצרו חלון העתק (Snapshot) חדש הוא בעצם יהיה מנותק מהחלונות האחרים מבחינת הסנכרון שהתוכנה מבצעת:



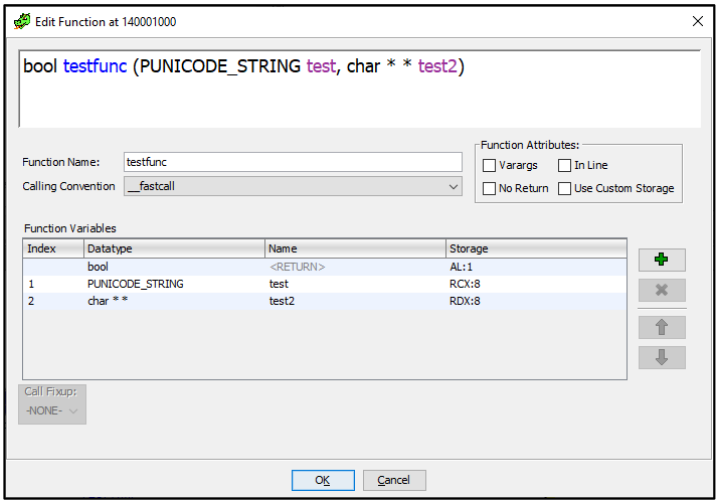
חלון ה-Decompiler

תהליך דיקומפיליצה של קוד הוא תהליך קשה ומסורבל. אך כאשר הוא עובד כראוי, תועלתו מאפשרת להאיץ את עבודת הניתוח ולאפשר מיקוד בדברים אחרים. כיום הדיקומפיילר של Ghidra יודע להמיר בצורה טובה קוד אסמבלי ל-C ונותן פייט רציני מול HexRays של IDA.

כפי שצוין, הסנכרון בין החלונות השונים, הוא מעולה ורוב קיצורי הדרך הקיימים, תקפים גם בחלון זה. חשוב לציין שבשלב זה יכולת זיהוי הטיפוסים לוקה בחסר, לכן, על מנת לשנות את טיפוס המשתנה נלחץ על הלחצן הימני ואז Retype Variable או לחלופין בעזרת קיצור המקשים CTRL + L. לאחר מכן ייפתח לנו החלון הבא ושם נוכל לבחור את הטיפוס הרצוי:



ככל שנדייק יותר בבחירת טיפוסי המשתנים וחתימת הפונקציות, כך נוכל לקבל קוד ברור, נכון ומובן. כך נראה חלון עריכת חתימת הפונקציה:



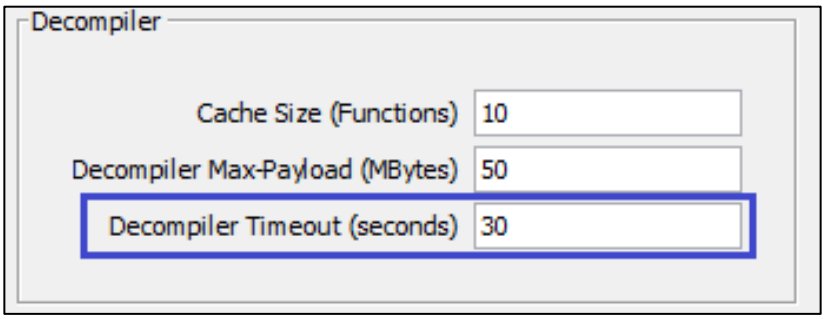
נסתכל על התפריט העליון בחלון ה-Decompiler:

1. כדי לבקש מה-Decompiler לבצע מחדש את הניתוח, נלחץ על הכפתור המוקף בצהוב.
2. כדי לייצא את הקוד לקובץ נלחץ על הכתפור המוקף בכחול.
3. כדי לבטל את הסנכרון בין החלונות בעותק חדש, נלחץ על הכתפור המוקף בצבע סגול, כפי שהוסבר כבר בחלון ה-Graph.

עוד פיצ'ר מעניין ושימושי מאוד הינו זיהוי מבנים באופן אוטומטי. במידה ואנו משערים שאחד מהמשתנים בקוד הוא מבנה, נוכל לבקש מה-Decompiler לייצר מבנה חדש באופן אוטומטי. כאשר נפעיל אותו על אובייקט מסוים, התוכנה תנסה לאבחן את ה-Struct Members על פי הגישות אליו בקוד ותייצר מבנה מתאים. ניתן להשתמש באפשרות זו על ידי לחיצה ימנית בעכבר על המשתנה הרצוי ואז ב- Auto Create Structure או דרך קיצור הדרך "CTRL +]".

כאשר אנו מתמודדים עם פונקציות גדולות, לעיתים תהליך ה-Initial Auto Analysis עלול לקחת זמן רב, באפשרותינו להגדיל את ה-Time Out שהוגדר, ובכך לאפשר את התקדמות הניתוח.

Edit -> Tool Options -> Decompiler



טיפ נוסף שמשפר פלאים את פלט ה-Decompiler: כאשר מנתחים קבצי Firmware: סמנו את הסגמנטים של ה-Flash memory כ-Read only, כך תדע התוכנה שמדובר במידע קבוע והקוד ייראה טוב וקריא יותר, על מנת לשנות את ההרשאות בסגמנטים יש להכנס ל:

Window -> Memory Map

או פשוט ללחוץ על הקיצור דרך  בחלון ה-CodeBrowser.

לדוגמה, כך זה נראה לפני שינוי ההרשאות:

```
2  void FUN_00000400(void)
3
4  {
5    int iVar1;
6    undefined4 *puVar2;
7    int iVar3;
8    undefined4 *puVar4;
9    undefined4 *puVar5;
10   int iVar6;
11   undefined4 *puVar7;
12
13   iVar3 = DAT_00000434;
14   puVar2 = DAT_00000430;
15   iVar1 = DAT_0000042c;
16   iVar6 = 0;
17   while (puVar4 = DAT_0000043c, puVar5 = (undefined4 *)(iVar6 + iVar1), p
18         puVar5 < puVar2) {
19     puVar4 = (undefined4 *)(iVar3 + iVar6);
20     iVar6 = iVar6 + 4;
21     *puVar5 = *puVar4;
22   }
23   while (puVar7 < puVar4) {
24     *puVar7 = 0;
25     puVar7 = puVar7 + 1;
26   }
27   FUN_00000aa8();
28   return;
29 }
```

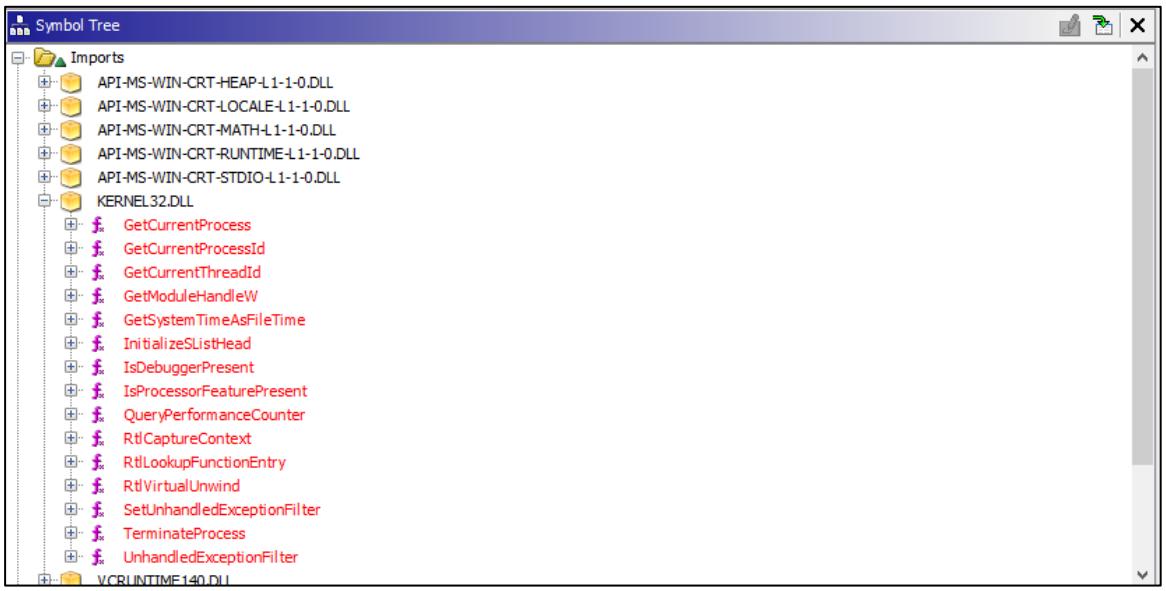
לאחר שינוי ההרשאות:

```
2  void FUN_00000400(void)
3
4  {
5    undefined4 *puVar1;
6    int iVar2;
7    undefined4 *puVar3;
8
9    iVar2 = 0;
10   while (puVar3 = (undefined4 *)(&DAT_20000008 + iVar2), puVar3 < &DAT_20
11     puVar1 = (undefined4 *)(&DAT_08002f80 + iVar2);
12     iVar2 = iVar2 + 4;
13     *puVar3 = *puVar1;
14   }
15   puVar3 = (undefined4 *)&DAT_20000008;
16   while (puVar3 < (undefined4 *)0x200009ec) {
17     *puVar3 = 0;
18     puVar3 = puVar3 + 1;
19   }
20   FUN_00000aa8();
21   return;
22 }
```

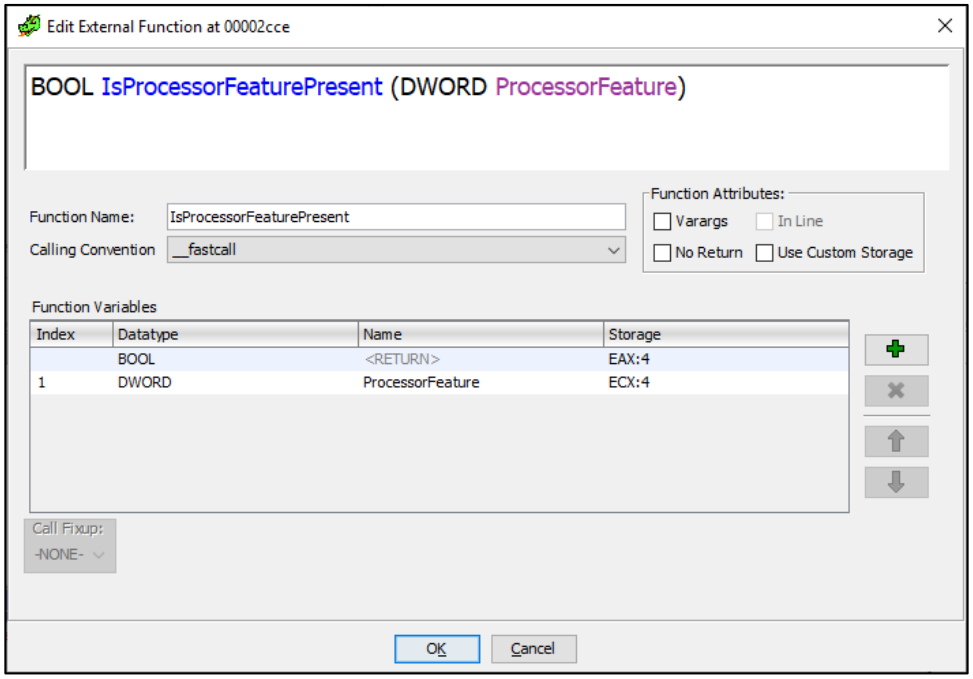
חלון ה-Symbols Tree

חלון זה מאפשר לנו לבחון את המידע שהתוכנה זיהתה לאחר תהליך האנליזה, לרוב מידע זה כולל מחלקות, לייבלים, פונקציות, Imports ו-Exports. כמו כן, גם ניתן לבצע שינויים על אותם סוגי מידע מתוך חלון זה.

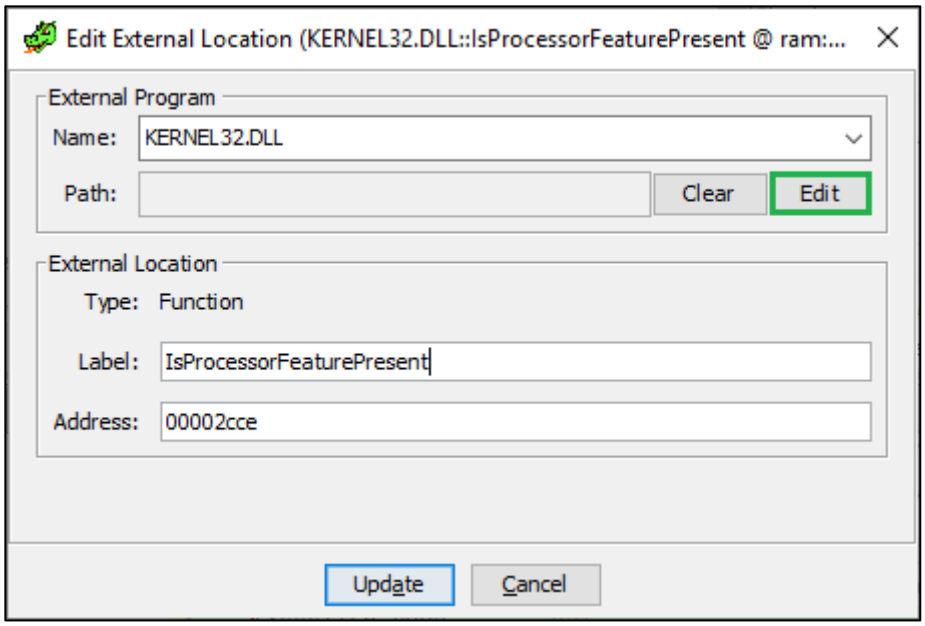
בתמונה הבאה ניתן לראות את הפונקציות שהתוכנית מיבאת מ-kernel32.dll:



לחיצה על פריט בתוך החלון תנווט אתכם בשאר החלונות לאותה פונקציה. אם אינכם מרוצים מחתימת פונקציה מסוימת שיובאה תוכלו לערוך אותה על ידי לחיצה ימנית ו-Edit Function Signature, בדומה למה שעשינו קודם:

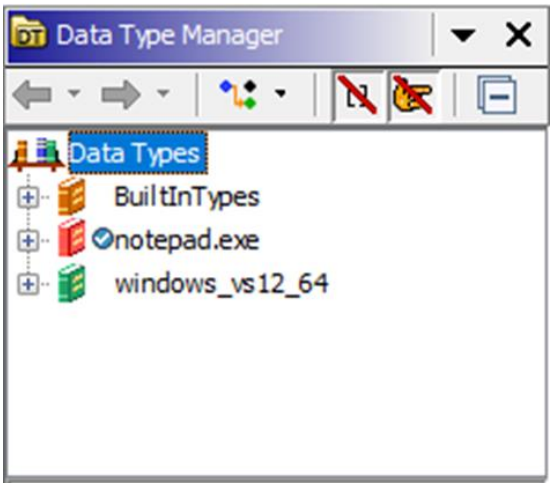


פיצ'ר מעניין נוסף הוא היכולת לחבר פריט מתוך ה-Symbols Tree ל-Imported File (כפי שכבר צוין, ניתן לטעון יותר מקובץ אחד לתוך ה-Code Browser) כך שיהיה חיבור בין הכתובות, לדוגמה, אנו יודעים שהפונקציה IsProcessorFeaturePresent מגיעה מ-KERNEL32.DLL, נוכל לטעון את הקובץ ואז להכנס לתפריט הלחצן הימני של הפונקציה בחלון ה-Symbols Tree וללחוץ Edit External Location ולאחר מכן ללחוץ Edit Path:



חלון ה-Data Type Manager

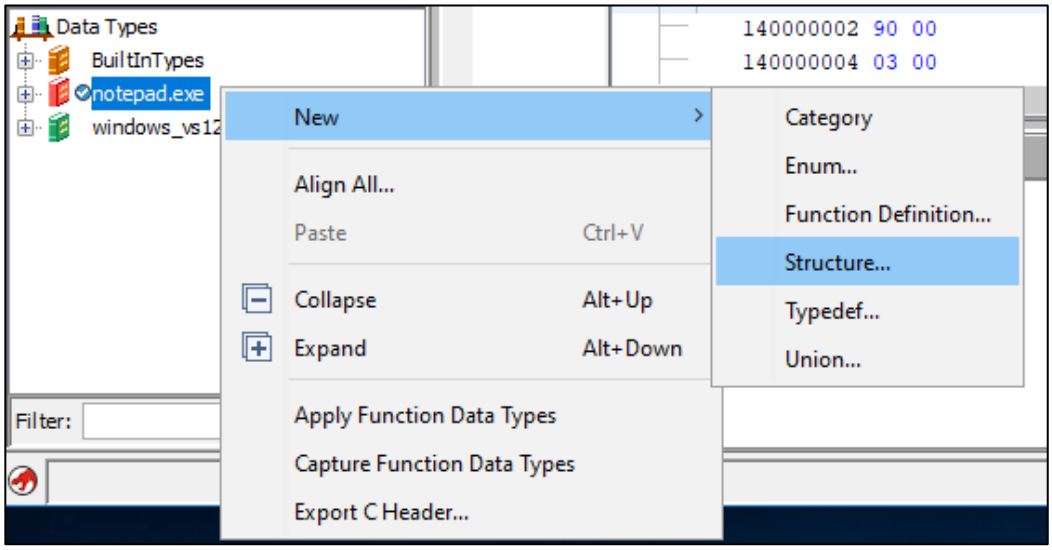
החלון מאפשר ניהול של טיפוסים והגדרות פונקציות, כברירת מחדל ממוקם החלון בצד השמאלי-התחתון של המסך, כך הוא נראה:



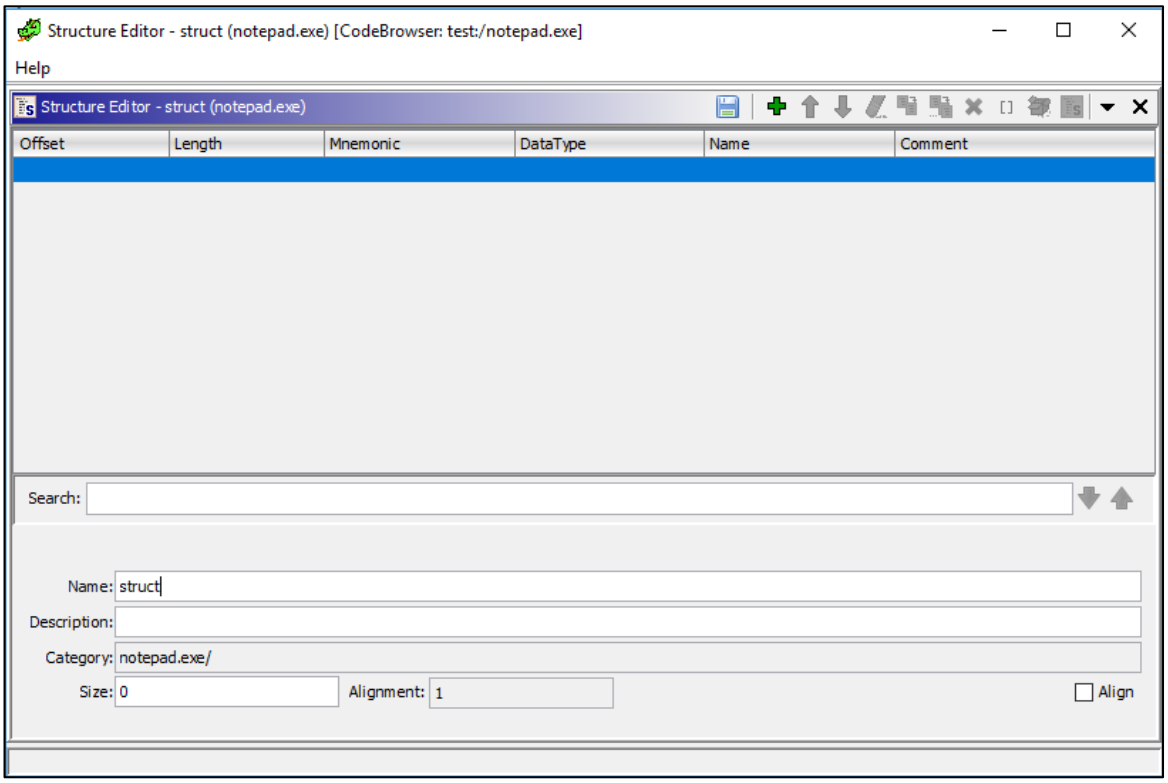
בחלון זה ישנם כמה סוגי תיקיות:

- תיקיית ה-BuiltInTypes אשר מכילה את טיפוסי ברירת מחדל שמפתחי Ghidra הוסיפו
- תיקיית ה-Module שאנו מנתחים, תכיל טיפוסים והגדרות פונקציות שכרגע בשימוש או שהוספנו
- תיקיית הטיפוסים והגדרות פונקציות שהתוכנה טענה אוטומטית מקובץ בפורמט GDT, שאותו היא בחרה לפי סוג הקובץ שאנחנו מנתחים. פורמט זה נקרא Ghidra Data Type והינו המקביל ל-TIL של IDA. ככל הנראה, על בסיס השערות, ה-NSA עדיין לא פרסמו את כל קבצי ה-GDT כך שיהיה אפשר לבצע בעזרתם ניתוח נוח.

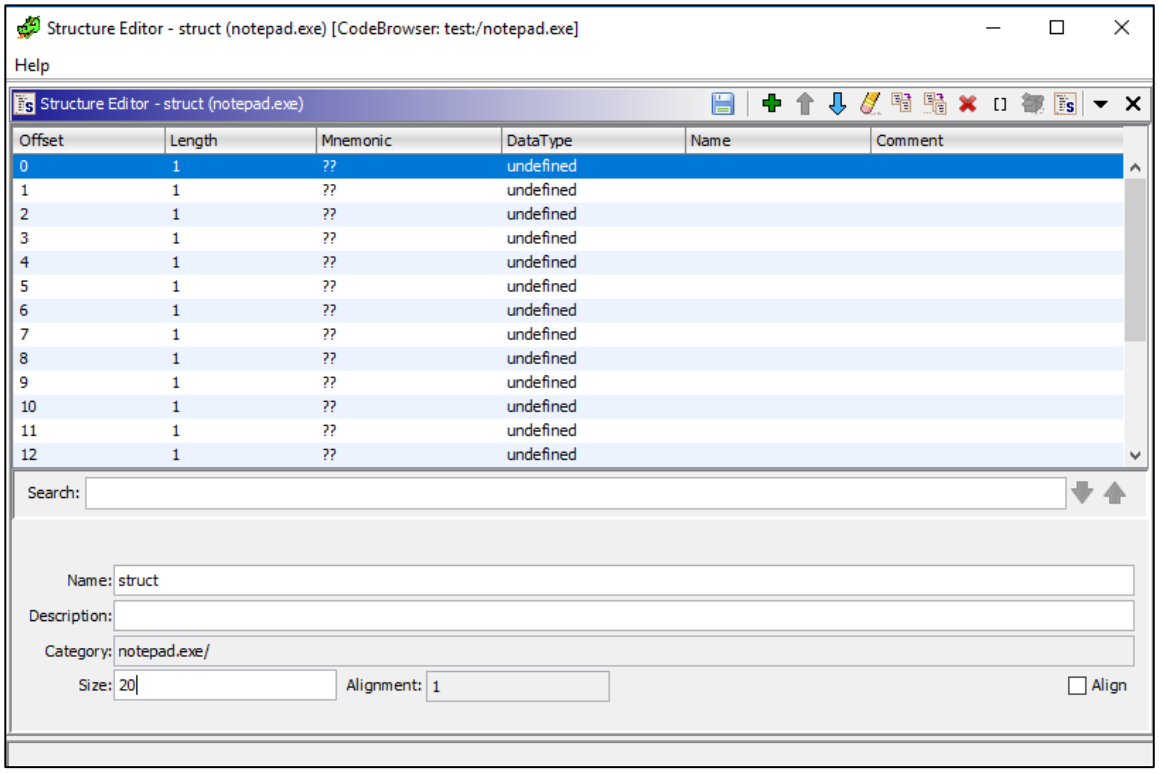
על מנת ליצור טיפוס חדש, נלחץ על תיקיית ה-Module שלנו, נבחר ב-New ולאחר מכן נבחר בטיפוס רצוי. לדוגמה, כך יוצרים מבנה חדש מסוג Structure:



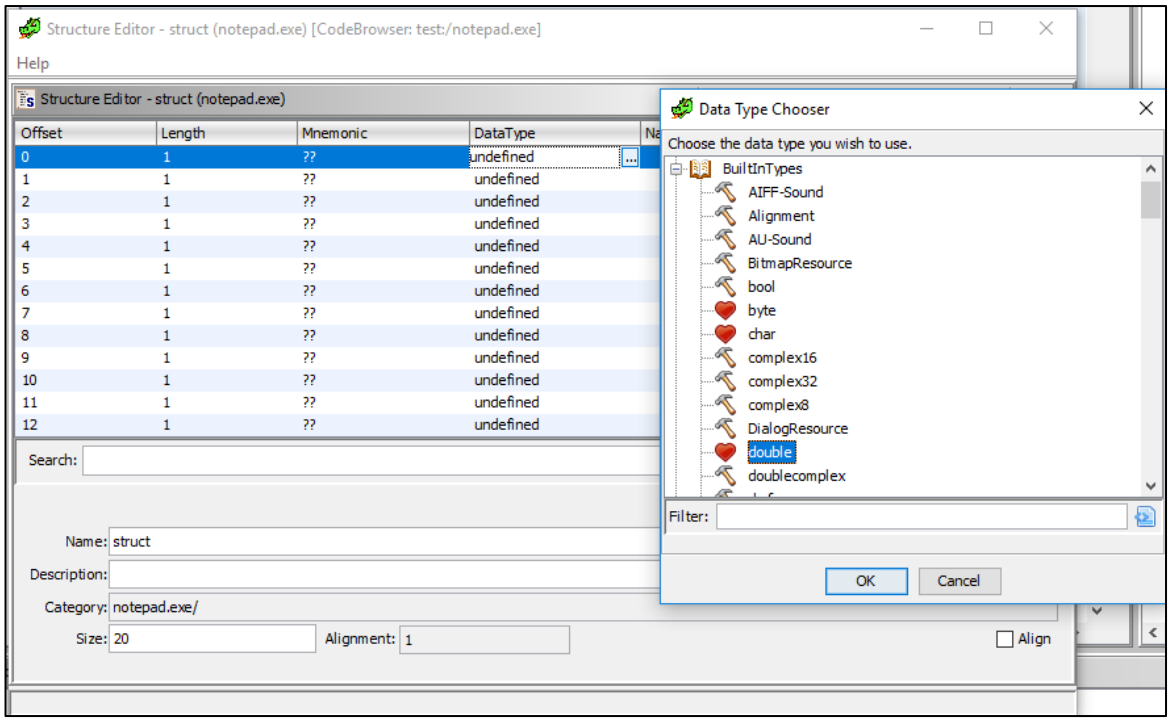
לאחר מכן יפתח לנו החלון הבא:



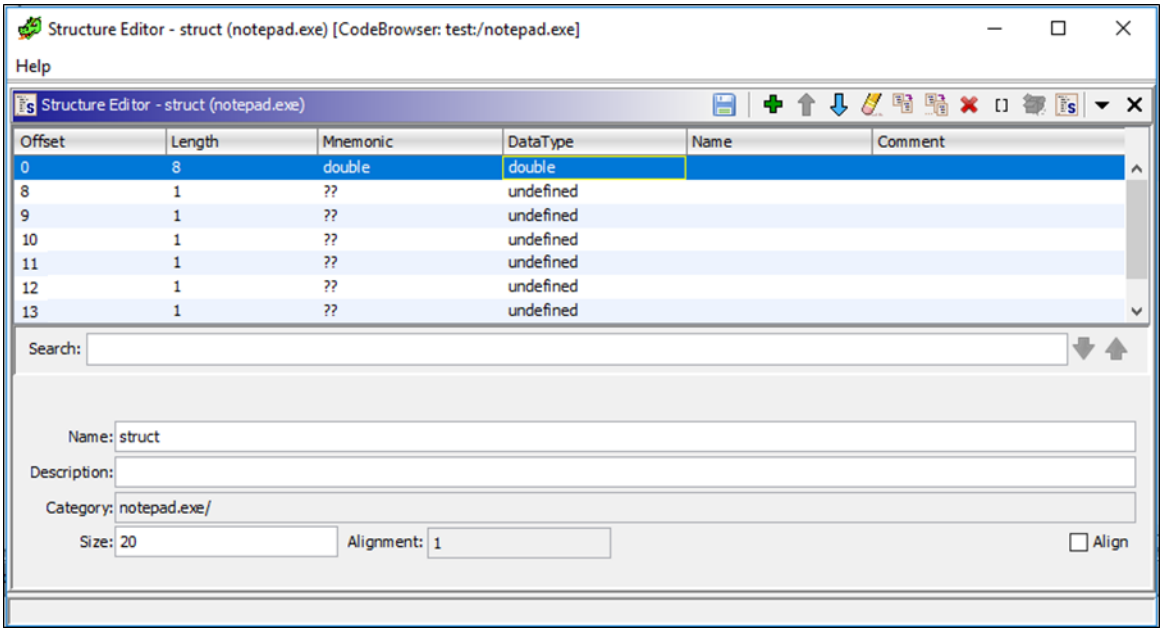
עלינו לבחור שם למבנה החדש, באפשרותינו לתת לו גם תיאור. בנוסף, אם אנו יודעים את הגודל המבנה, נוכל להגדיר גם אותו, אחרת, נצטרך ללחוץ על סימן הפלוס למעלה כדי להוסיף שדה חדש. כאשר נגדיר גודל כללי למבנה, העורך יצור עבורנו אוטומטית ערכים ע"פ הגודל שסופק:



כאשר נגדיר את סוג הטיפוס על ידי הגדרת ה-DataType, העורך אוטומטית יתאים את ה-Offests לפי הגודל:



כך זה נראה לאחר בחירת ה-DataType (שימו לב ל-Offsets):



הכלי מציע גם פיצ'רים נוספים כגון ייצוא וייבוא של קבצי C Headers, כדי לייבא קובץ Header ניגש ל:

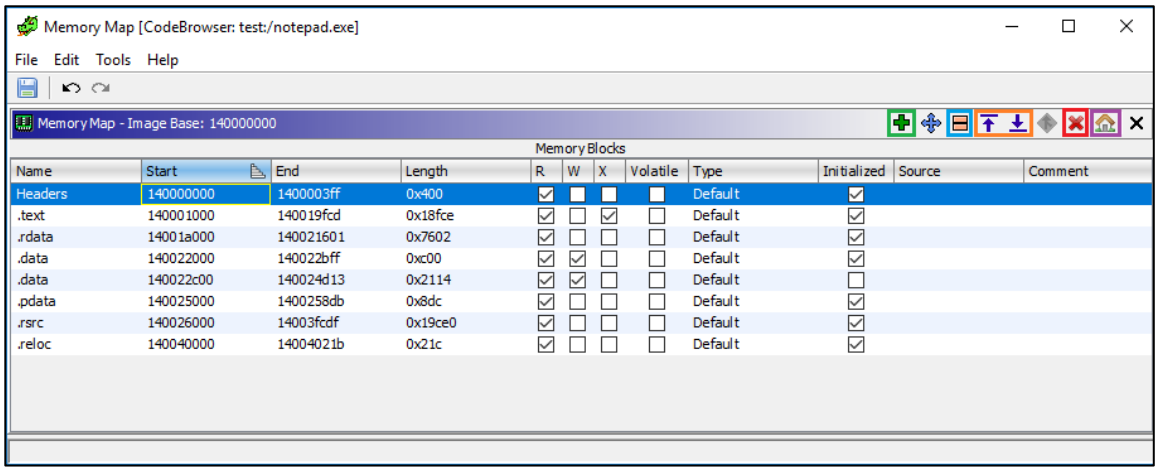
File -> Parse C Source

יצוא חתימות וטיפוסים מתאפשר על ידי לחיצה ימנית על ה-Module שלנו, ולאחר מכן Export C Header.

חלון ה-Memory Map

כחלק מתהליך המחקר, חשוב לדעת כיצד ממופה הזיכרון בתוכנית, מהן ההרשאות ב-Sections, גודלן ועוד. נוכל לעשות זאת בחלון זה:

Window -> Memory Map

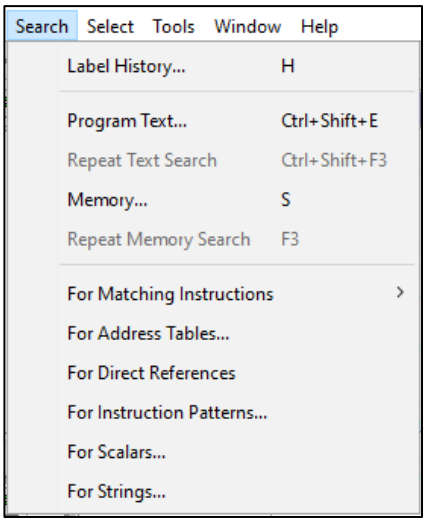


- כדי לבצע Rebase לתוכנית, נלחץ על הכפתור המוקף בסגול (בית), שם נוכל לציין את הכתובת החדשה.
- כדי להוסיף בלוק זיכרון חדש, נלחץ על הכפתור המוקף בירוק (+) ונוכל לציין את:
  - שם הבלוק
  - כתובת התחלתית
  - גודל
  - הרשאות
  - ועוד
- כדי לפצל בלוק זיכרון קיים, נלחץ על הכפתור המסומן בכחול.
- כדי לשנות כתובת התחלה או סוף של בלוק קיים, נלחץ על הכפתורים המוקפים בכתום
- וכדי למחוק בלוק זכרון נלחץ על הכפתור המוקף באדום (X).

תכונות נוספות

חיפוש

ל-Ghidra מנוע חיפוש מפואר, כעת נציג את תכונותיו. ראשית כדי לגשת לחיפוש נלחץ על Search בתפריט העליון ונבחר באופציה הרצויה. להלן סוגי החיפוש הקיימים וחלק מקיצורי הדרך הקשורים בהם:

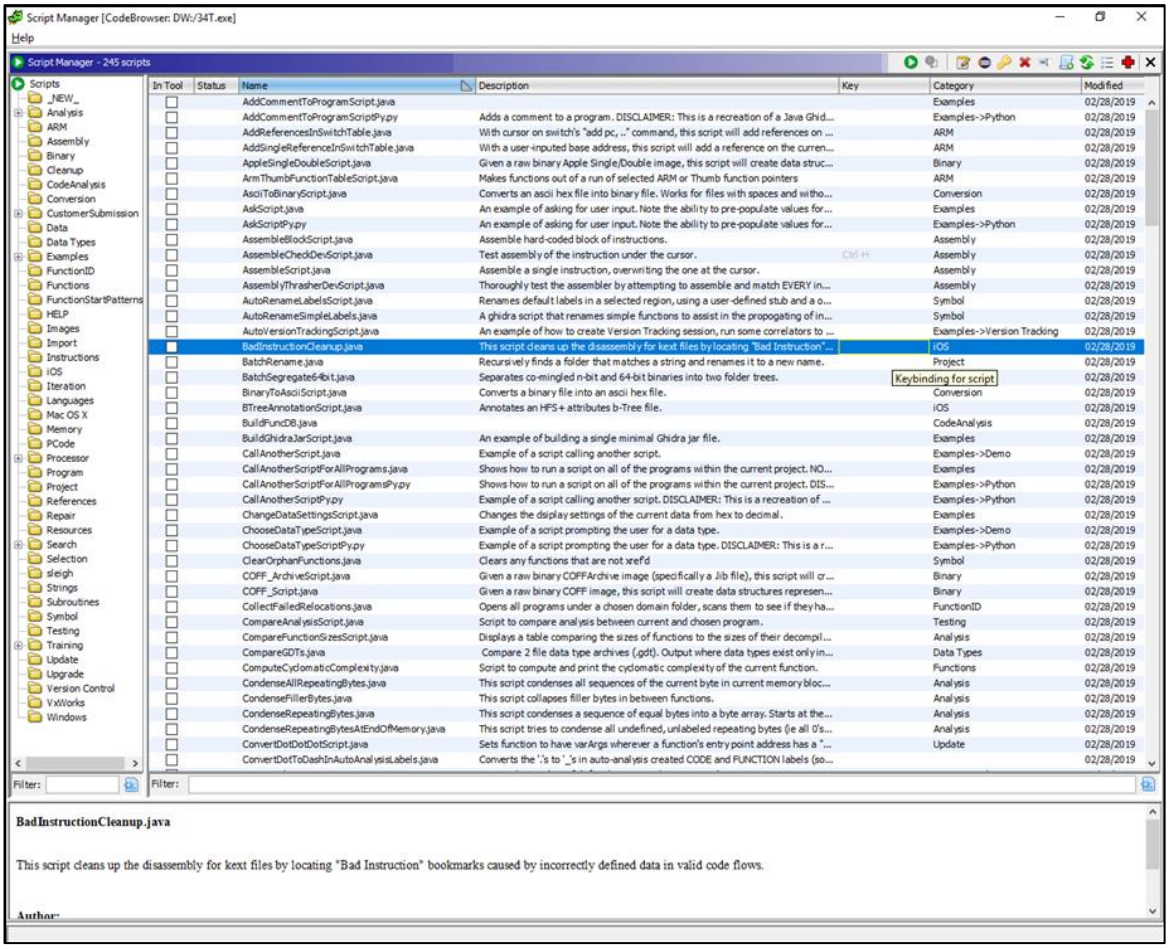


- חיפוש Program Text - מאפשר חיפוש של כל טקסט בחלונות שנוסף על ידי Ghidra או על ידי המשתמש, כולל: שדות, הערות, labels, רפרנסים, קוד ועוד.
- ניתן להפעיל את החיפוש על כלל התוכנית או על חלקים שנבחרו. בחירה ב-Search All Instances יוצרת טבלה עם כל תוצאות החיפוש.
- חיפוש Memory - מאפשר למשתמש לחפש סט של בתים בכלל התוכנית או בחלקים שנבחרו, ניתן לחפש בייצוג Hex, ASCII, Unicode ועוד. ניתן גם להשתמש ב-RegEx. גם בסוג חיפוש זה ניתן לייצר טבלה מתוצאות החיפוש.
- חיפוש Strings - מאפשר חיפוש של מחרוזות בכל התוכנית או בחלקים נבחרים מסוג ASCII, Unicode ו-Pascal. בסוג חיפוש זה התוצאות מגיעות כברירת מחדל בטבלה, בה ניתן לחפש ולייצר Labels מתוך חלון התוצאות.
- חיפוש Direct References - מאפשר למשתמש לחפש מקומות בתוכנית בהם בתים מהווים רפרנס (למשל Call instruction) למיקום הנוכחי בקוד או למספר כתובות שנבחרו.
- חיפוש Address Tables - פיצ'ר מעניין, מאפשר לחפש טבלאות של כתובות, ניתן לציין גודל מינימלי של הממצאים, alignment וכמה בתים לדלג בין כל כתובת.
- חיפוש Instructions - מאפשר למשתמשים לחפש instructions לפי Pattern מסוים, ניתן לציין אם לכלול או לא אופרנדים. האופציה הזו למעשה ממירה את בקשת החיפוש שלנו לבתים ומבצעת חיפוש memory רגיל.
- חיפוש Scalars - מאפשר למשתמש לחפש ערכים סקלרים. ניתן לציין טווח ערכים או ערך ספציפי.

סקריפטינג

בדומה ל-IDA, גם Ghidra מציעה ממשק Scripting עוצמתי ביותר ומתועד היטב. ממשק מנהל הסקריפטים (Script Manager) למעשה עוטף את כל הפיצ'רים הקשורים ב-Scripting. ניתן לגשת אליו באמצעות:

Window -> Script Manager



כך נראה הממשק, כפי שניתן לראות, התוכנה מגיעה עם קרוב ל-250 סקריפטים הכתובים רובם ב-Java וחלקם ב-Python.

מכיוון שרוב התוכנה כתובה ב-Java, זוהי השפה הטבעית לכתיבת פלאגינים וסקריפטים, אך ניתן גם לכתוב סקריפטים ב-Python.

בממשק זה ניתן להגדיר נתיבים חדשים לחיפוש סקריפטים באמצעות הכפתור , על מנת להריץ סקריפט אפשר ללחוץ עליו פעמיים או ללחוץ על כפתור ה-Play, ממנו יש את כפתור ה-Play Again שמאפשר להריץ את הסקריפט האחרון שוב. כפתור המפתח מאפשר למשתמש להגדיר קיצור מקשים לסקריפט המסומן.

ניתן לערוך סקריפטים באמצעות עורך הטקסט הפשוט של הממשק או באמצעות Eclipse. על מנת שיהיה ניתן לכתוב סקריפטים ב-Eclipse יש להתקין את התוסף GhidraDEV שנמצא בנתיב:

```
ghidra_9.0\Extensions\Eclipse\GhidraDev
```

לחיצה על הכפתור עם הסמל  , מאפשרת אוטומציה של התהליך הנ"ל ותתקין לכם לבד את GhidraDEV, במידה וכבר לא הותקן בעבר ובהינתן ו-Eclipse מותקנת על המכונה.

לאחר ש-GhidraDEV מותקנת ב-Eclipse ממשק העבודה הרבה יותר נוח, ניתן לדבג את הסקריפטים שלנו ואת Ghidra עצמה, וגם IntelliSense (בגרסתו ב-Eclipse) עבור ה-API של Ghidra יחל לפעול.

כפי שהוזכר, קיימת דוקומנטציה מפורטת מאוד לגבי כל ה-API של Ghidra שנמצאת בתיקיית התקנה שלכם.

כמו כן העלנו גרסה מקוונת לדוקומנטציה שניתן למצוא כאן, אותה אנו מתחזקים ומתקנים.

אם ברצונכם להחיל את התיקונים בגרסה שלכם תוכלו להחיל את התלאי הבא וכמו כן לעקוב על מנת לקבל תיקונים נוספים:

https://github.com/NationalSecurityAgency/ghidra/issues/129

בנוסף, אם מסיבה כלשהי אתם נזקקים להציץ בקוד המקור, תוכלו למצוא אותו בכל תיקיית Features בקובץ Zip בשם feature-src.zip:

```
kd@DESKTOP-L5CEVEP: /mnt/c/Tools/ghidra_9.0
kd@DESKTOP-L5CEVEP:/mnt/c/Tools/ghidra_9.0$ find ./Ghidra/Features/ -iname *src.zip
./Ghidra/Features/Base/lib/Base-src.zip
./Ghidra/Features/BytePatterns/lib/BytePatterns-src.zip
./Ghidra/Features/ByteViewer/lib/ByteViewer-src.zip
./Ghidra/Features/DebugUtils/lib/DebugUtils-src.zip
./Ghidra/Features/Decompiler/lib/Decompiler-src.zip
./Ghidra/Features/DecompilerDependent/lib/DecompilerDependent-src.zip
./Ghidra/Features/FileFormats/lib/FileFormats-src.zip
./Ghidra/Features/FunctionGraph/lib/FunctionGraph-src.zip
./Ghidra/Features/FunctionGraphDecompilerExtension/lib/FunctionGraphDecompilerExtension-src.zip
./Ghidra/Features/FunctionID/lib/FunctionID-src.zip
./Ghidra/Features/GhidraServer/lib/GhidraServer-src.zip
./Ghidra/Features/GnuDemangler/lib/GnuDemangler-src.zip
./Ghidra/Features/GraphFunctionCalls/lib/GraphFunctionCalls-src.zip
./Ghidra/Features/MicrosoftCodeAnalyzer/lib/MicrosoftCodeAnalyzer-src.zip
./Ghidra/Features/MicrosoftDemangler/lib/MicrosoftDemangler-src.zip
./Ghidra/Features/MicrosoftDmang/lib/MicrosoftDmang-src.zip
./Ghidra/Features/PDB/lib/PDB-src.zip
./Ghidra/Features/ProgramDiff/lib/ProgramDiff-src.zip
./Ghidra/Features/Python/lib/Python-src.zip
./Ghidra/Features/Recognizers/lib/Recognizers-src.zip
./Ghidra/Features/SourceCodeLookup/lib/SourceCodeLookup-src.zip
./Ghidra/Features/VersionTracking/lib/VersionTracking-src.zip
kd@DESKTOP-L5CEVEP:/mnt/c/Tools/ghidra_9.0$
```

למעשה, כל התוכנה מורכבת מפלאגינים, כל פלאגין מספק פונקציונליות מסוימת. פלאגינים מתקשרים בינם ובין עצמם בתוך אותו Tool שבתוכו הם מעוגנים. Tool הינו אוסף של פלאגינים וההגדרות שלהם,

התוכנה מגיעה עם מספר Tools מוגדרים מראש שניתן להתאימם לדרישות המשתמש. ניתן להוסיף ולהסיר פלאגינים מבלי לפתוח מחדש את התוכנה וכמובן ניתן לכתוב פלאגינים חדשים.

בשונה מפלאגינים, שהינם גלוים וקבועים ב-CodeBrowser, סקריפטים נועדו יותר לפעולות מסוג One-Shot. הודעות מסקריפטים יוצגו בחלון ה-Console שפתוח באופן כברירת מחדל בתחתית המסך, אך סקריפטים יכולים גם להשתמש במחלקות GUI ולתקשר עם המשתמש. במידה וה Console מזהה איזשהי כתובת או Label הוא יהפוך אותם ללחיצים על מנת להקל עליכם.

כך נראה Template של סקריפט ב-Ghidra:

```
1  //Writes "Hello World" to console.
2  //@category     Examples
3  //@menupath     Help.Examples.Hello World
4  //@keybinding   ctrl shift COMMA
5  //@toolbar     world.png
6
7  import ghidra.app.script.GhidraScript;
8
9  public class HelloWorldScript extends GhidraScript {
10
11     @Override
12     public void run() throws Exception {
13         println("Hello World");
14     }
15 }
```

הסקריפט צריך לרשת מ-GhidraScript, ה-"entrypoint" שלנו בסקריפט נמצא בפונקציה "run".

מספר אובייקטים שימושיים:

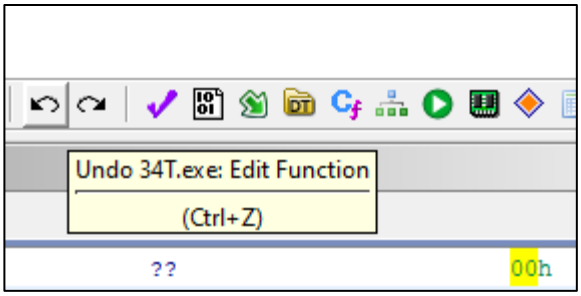
- currentProgram - מופע שמתאר את התוכנית הנוכחית
- currentAddress - הכתובת של המיקום הנוכחי בתוכנית
- currentSelection - מופע שמייצג את החלקים שנבחרו בעת הרצת הסקריפט
- currentHighlight - מייצג את האלמנט שמסומן כרגע

חשוב לציין ש-Ghidra יכולה לפעול במצב Headless Analyzer, במצב זה לא מוצג GUI כלל וכל Analysis וכל Tool שלא תלוי ב-GUI יכול לפעול במצב זה.

מדיניות ביטול

אחד הפיצ'רים החזקים ביותר של בתוכנה הינו תמיכה ב-Undo/Redo בקיצורי מקשים CTRL + Z לביטול הפעולה ובקיצור CTRL + SHIFT + Z להחזרת הפעולה. תסמכו עלינו, זה יחסוך לכם הרבה סבל.

בתוך ה-Code Browser וחלונות נוספים דומים, כל פעולה, לא משנה כמה קטנה או גדולה - ניתנת לביטול או חזרה (Undo/Redo) על ידי החצים  העבירו את העכבר מעל החצים כדי לראות לאיזו פעולה הנכם מבצעים ביטול או החזרה:



קיימים עמודי Help מקיפים מאוד אשר מסבירים ומפרטים כל פעולה בתוכנה, סימון אובייקט רצוי ולחיצה על F1 לרוב תפתח את חלון העזרה הרלוונטי, ולעיתים תדרוש קצת חיפוש ידני. קיצורים שימושיים נוספים ניתן למצוא בעותק האונליין ל-Cheat Sheet של התוכנה.

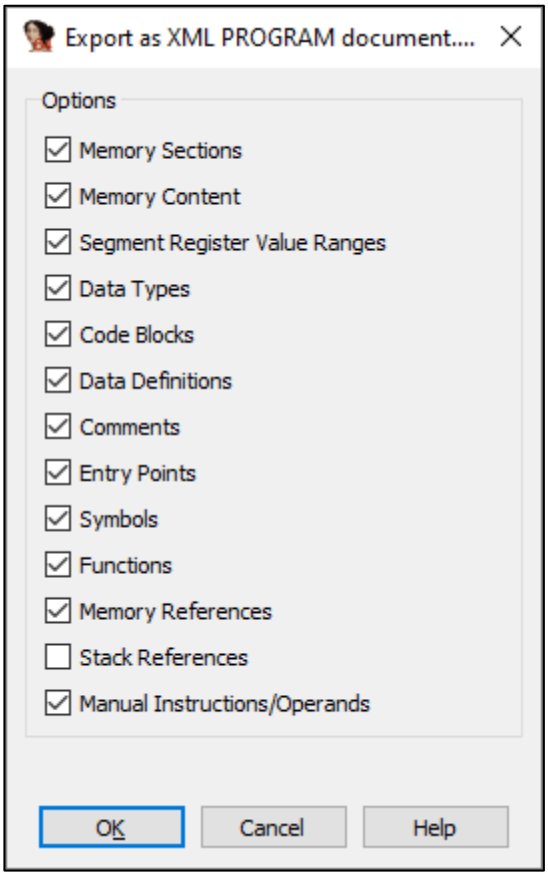
קובץ זה וקבצים נוספים, קיימים גם בעותק המקומי של התקנת התוכנה.

מעבר חלק

מפתחי התוכנה דאגו לנו עם כלים על-מנת לאפשר מעבר מ-IDA ל-Ghidra וכתבו פלאגין שמאפשר לייצא IDB קיים מתוך IDA ולטעון אותו ל-Ghidra. כמו כן קיימת גם הגרסה ההפוכה - טעינת Progress קיים מ-Ghidra אל IDA. הפלאגין נמצא בנתיב:

```
\ghidra_9.0\Extensions\IDAPro
```

על מנת לבצע ייצוא מתוך IDA תצטרכו להשתמש בפלאגין xml_exporter.py, לאחר שתטענו את הפלאגין ל-IDA ותפעילו אותו יופיע בפניכם החלון הבא:



כאן תוכלו לבחור איזה מידע בדיוק ברצונכם לייצא, לאחר מכן תוכלו לטעון את קובץ ה-xml לתוך התוכנה.

סיכום

במאמר זה נגענו רק בקצה הקרחון. קיימים עוד שלל פיצ'רים שבהם לא עסקנו, אך אפשר להגדיר את ההיכרות עם Ghidra כהתאהבות מיידית, כלי כה רב עוצמה - בחינם, קשה להישאר אדישים.

אפשר לומר בביטחון ש-Ghidra תוכל לעמוד במשימות RE רבות בכבוד רב אך עדיין יש לאן לשאוף ואיפה להשתפר. למשל, לא קיימת כרגע תמיכה ב-Debugging (או לפחות עדיין לא שוחררה לציבור). בשביל זה אנחנו פה, הקהילה, על מנת שהפרויקט "יישאר באוויר" ולא ייזנח, נצטרך להשקיע זמן ומאמץ על מנת לתחזק אותו, לפתור באגים, לכתוב פיצ'רים ופלאגינים חדשים ועוד.

מיד עם שחרורה עשתה Ghidra רעש גדול, ובצדק. אחד הסימנים הכי גדולים לכך הינו שחרור ה-Debugger בגרסה החינמית של IDA, וכמובן התוכנית למוסדות לימודיים (אך עם מגבלות רבות). הנושא שנוי במחלוקת, האם שני הצעדים האלו נגרמו עקב שחרור Ghidra אך אנו חושבים שכן ☺. נכון לרגע כתיבת המאמר כבר נפתחו קרוב ל-250 Issues ב-Github של הפרויקט וחלקם כבר תוקנו על ידי הקהילה עצמה, ללא עזרה מ-NSA, הקהילה בהחלט נכנסה לזה חזק. בנוסף, יצאה גרסה 9.0.1 הכוללת שלל תיקונים ושיפורים.

בנוסף לכך אנו רואים כל יום מדריכים חדשים, סרטונים ביוטיוב ופלאגינים שמתפרסמים על ידי הקהילה ההולכת ומתגבשת, אתם מוזמנים לבקר בעמוד הטוויטר שלנו: @GHIDRA_RE, שם נדאג לפרסם עדכונים, פיצ'רים, טיפים ופלאגינים חדשים.

הוקמה פלטפורמה פעילה לשיח סביב הפרויקט בטלגרם וב-Matrix והנכם מוזמנים להצטרף בלינק הבא. כמו כן, אנו משתדלים לרכז את החומרים הקשורים לפרויקט באתר הבא כך שיהיו ניגשים ברשת ומכל מקום בנוחות.

על הכותבים

- כסיף דקל - עובד כחוקר אבטחת מידע עצמאי, לכל פניה או שאלה ניתן לפנות בטוויטר.
- רונן שוסטין - עובד כחוקר אבטחת מידע בחברת Check Point, לכל פניה או שאלה ניתן לפנות בטוויטר.

ביבליוגרפיה

[1] http://ghidra.re/ghidra_docs/api/

[2] https://ghidra.re/online-courses/

[3] https://twitter.com/GHIDRA_RE

[4] https://twitter.com/ghidraninja/status/1103637622465986560

[5] Credit to **Shaked Reiner** for the undo sticker design

יצירת ערוץ תקשורת חשאי בין שני קונטיינרים

מאת יהודה כורסיה

הקדמה

בשנים האחרונות אנו שומעים עוד ועוד על מקרי פריצה לרשתות והדלפת מאגרי מידע עצומים בגודלם (2.2 מיליארד רשומות בינואר השנה, עוד כמעט 700 מיליון רשומות בפברואר, ועוד אינסוף רשומות במהלך השנים האחרונות), במקרים כאלה, נראה שלמצוא SQL Injection בשדה מסוים זה החלק הקל, בעוד שהמשימה של להדליף מאגר ששוקל מעל 800GB מרשת מאובטחת מבלי שירגישו - נשמעת כמו משימה בלתי אפשרית.

"בתחום אבטחת המחשבים, Covert Channel היא סוג של מתקפה אשר מאפשרת לתוקף להעביר מידע בין שני תהליכים אשר אינם אמורים להיות מסוגלים לתקשר ביניהם על פי מדיניות האבטחה של המערכת" (תרגום חופשי מויקיפדיה).

מציאת ערוצים סמויים, או - ערוצי דלף - אם תרצו, ושימוש בהם כחלק ממתקפה כוללת עשוי להקשות מאוד על איתור התוקף מאחר וערוצים אלו יהיו בדרך כלל מחוץ ליכולות הניטור של מערך ההגנה של המערכת/רשת.

במאמר זה אציג בפניכם את התוצאות של מחקר שביצעתי על Docker - ובמסגרתו, מצאתי דרך להעביר מידע בין שני קונטיינרים למרות שהוגדרו כך שלא יוכלו לתקשר ביניהם.

אך לפני שנמשיך, אני ממליץ מאוד לקרוא את המאמר של יגאל אלפנט ותומר זית העוסק בנדבך אבטחה נוסף ב-Docker בעיקר את ההקדמה המעולה שלהם שמסבירה בצורה טובה מה זה Docker וכיצד הוא בנוי.

הסבר קצר על Docker

לא ארחיב פה על הנושא, מפני שאני מניח שאתם מכירים את התחום או שכבר קראתם את ההקדמה למאמר של יגאל אלפנט ותומר זית (ובעיקר כי הם כבר כתבו עליו בצורה מעולה), אך ממש על רגל אחת: Docker מאפשר לנו להריץ תהליכים רגילים של מערכת ההפעלה בתוך קונטיינרים - בצורה מבודלת, כלומר באופן שבו התהליך יכיר אך רק את הסביבה הקשורה אליו. לדוגמא רק את המערכת קבצים שלו, רק את התהליכים שהוא יצר וכדומה וכן לאפשר הגבלה של משאבים לאותו תהליך, כגון הגבלה של RAM הגבלה של כמות CPU וכדומה.

Docker עושה זאת באמצעות שימוש במודולים הקיימים כבר ב-Linux כדוגמת Namespace, UnionFS, Cgroups, ו-LXC. בנוסף, מנגנון זה מאפשר להריץ סביבות באופן סימולטני ונפרד על גבי Host אחד.

למי שרוצה באמת להבין את הנושא לעומק אני ממליץ בחום רב לראות גם את ההרצאה המעולה של אחד מהאנשים שכתבו את Docker.

הנקודה שחשוב לי להעביר לכם בחלק זה היא שאנשים סומכים על Docker בכך שהוא יודע להפריד בין תהליכים בצורה הדוקה. ואם הרצנו שני תהליכים בקונטיינרים אשר קונפגו להיות נפרדים וחסרי יכולת לתקשר אחד עם השני - אז אין שום סיבה או דרך שהם יוכלו לעשות זאת.

הסבר על המתקפה

המתקפה שנספר עליה כעת, מאפשרת לתוקף בעל היכולת להריץ קוד בשני קונטיינרים הרצים על אותו Host, להצליח להעביר ביניהם מידע בצורה חופשית גם אם הם קונפגו כך שלא תהיה ביניהם שום תקשורת.

הקונטיינרים קונפגו כך שהם לא יוכלו לדבר ביניהם באופן ישיר, ולכן על מנת לעשות זאת - נאלץ לחשוב על שיטה עקיפה לעשות זאת.

לצורך ההדגמה נניח והקונטיינרים הוגדרו עם ה-Flag:

--network none

שבעצם משאיר את הקונטיינר רק עם כרטיס רשת לביצוע Loopback.

את המתקפה גיליתי בעקבות שיחה עם חבר שאמר לי שקיימת לצוות שלו בעיה בעבודה עם Docker. הם מנסים לעבוד עם Docker עבור הרצה של אפליקציה מבוססת Java ומשום מה הקונטיינר שלהם קורס כל הזמן.

הוא אמר לי שהם בדקו ושזה כנראה קשור לעובדה המעניינת: כאשר קונטיינר בודק כמה **RAM** פנוי יש לו, הוא לא מקבל תשובה לגבי כמה **RAM** פנוי יש ל-**instance** שלו אלא כמה **RAM** פנוי יש ל-**Host** שעליו הוא יושב.

בהתחלה העניין היה נשמע לי ממש מוזר, ואמרתי לו שאני אחקור על הנושא. חקרתי ומסתבר שהוא צדק ובאמת כאשר קונטיינר עולה אז Docker Engine מבצע mount לקובץ /proc/meminfo של ה-Host בלינוקס.

התיקיה /proc הינה "פסודו" מערכת קבצים (היא לא באמת קיימת על הדיסק עצמו), והמערכת הפעלה דואגת לייצר אותה ולשמור אותה בזיכרון. היא נועדה לספק מידע על המערכת (בדגש על processes במערכת ומכאן נובע השם שלה). הקובץ /proc/meminfo הוא "קובץ מיוחד" שמטרתו להציג סטטיסטיקה על הזיכרון של המערכת.

זאת ההגדרה שלו לפי man page:

```
This file reports statistics about memory usage on the system.
It is used by free(1) to report the amount of free and used
memory (both physical and swap) on the system as well as the
shared memory and buffers used by the kernel.
```

למתעניינים: ניתן לקרוא עליו בהרחבה כאן.

נחזור לסיפור שלנו: בצוות של חבר שלי היו מרימים קונטיינרים עם הגבלת זיכרון כלשהי. ה-JVM לא היה מודע להגבלה שביצעו על הקונטיינר ולכן הוא לא היה קורא ל-garbage collector. הבעיה הייתה שמבחינת Docker engine הוא כן היה מוגבל בכמות הזיכרון ולכן הוא היה מקריס אותו!

לקריאה מורחבת על הנושא כולל פתרון של הבעיה מוזמנים לקרוא כאן:
https://developers.redhat.com/blog/2017/03/14/java-inside-docker/

הנקודה החשובה ביותר שעלתה לי ברגע שהבנתי את הסיפור היא שבעצם קונטיינר שרץ על **Host** כלשהו יכול לדעת מידע שלכאורה לא הייתי רוצה שהוא ידע, כלומר כל המידע לגבי מצב הזיכרון של ה-**Host**.

ויותר חמור מזה הוא גם יכול להשפיע על המידע הזה בצורה מאוד קלה על ידי הקצאה פשוטה של זכרון. ובמקביל קונטיינרים אחרים שגם רצים על אותו **Host** גם יכולים לראות את המידע שהוא ישנה שם.

ואז נפל לי בעצם האסימון שקיים פה פוטנציאל להתקפה מסוג Covert Channel attack שתאפשר לנו ליצור ערוץ תקשורת סמוי בין שני קונטיינרים.

הרי אם קונטיינר A ירצה להעביר מידע כלשהו לקונטיינר B הוא יכול פשוט להפוך את המידע לצורה בינארית, ואז באמצעות הקצאה או אי הקצאה של כמות קבועה מראש של זיכרון הוא יוכל להעביר "0" ו-"1" שמייצגים את הבינארי ל-B.

הסיפור לא נגמר כאן! תוך כדי מחקר על הבעייתיות בשיתוף המידע הקיים באמצעות הקובץ meminfo נתקלתי במאמר המעולה הזה שמזכיר את הנקודה החשובה שקיים עוד מקום שבו קונטיינר יכול לראות מידע הניתן לשינוי ב-Host, המקום הזה הוא ה-Syscall sysinfo זה בעצם קריאה למערכת הפעלה ובקשה לקבל ממנה מידע לגבי משאבי המערכת.

הקטע המעניין במאמר:

> ## /proc is not the only issue: `sysinfo`
>
> Even if we could find a solution to containerize `/proc/meminfo` with which everyone is happy, it **would not be enough**.
>
> Linux also provides the `sysinfo(2)` syscall, which returns information about system resources (e.g. memory). As with `/proc/meminfo`, it is not containerized: it always returns metrics for the box as a whole.

מה שבעצם אומר שקיים עוד מרחב שבו אפשר להעביר מידע, הסתכלתי קצת על ה-Syscall sysinfo:

```
Since Linux 2.3.23 (i386) and Linux 2.3.48 (all architectures) the
structure is:

    struct sysinfo {
        long uptime;             /* Seconds since boot */
        unsigned long loads[3];  /* 1, 5, and 15 minute load averages */
        unsigned long totalram;  /* Total usable main memory size */
        unsigned long freeram;   /* Available memory size */
        unsigned long sharedram; /* Amount of shared memory */
        unsigned long bufferram; /* Memory used by buffers */
        unsigned long totalswap; /* Total swap space size */
        unsigned long freeswap;  /* Swap space still available */
        unsigned short procs;    /* Number of current processes */
        unsigned long totalhigh; /* Total high memory size */
        unsigned long freehigh;  /* Available high memory size */
        unsigned int mem_unit;   /* Memory unit size in bytes */
        char _f[20-2*sizeof(long)-sizeof(int)];
                                 /* Padding to 64 bytes */
    };
```

וראיתי שאחד השדות שהוא מחזיר זה מידע על כמות התהליכים שרצים כרגע במחשב. מיהרתי לבדוק את הנושא על שני קונטיינרים ומסתבר שבאמת קונטיינר אחד יכול לדעת מה סך כל התהליכים הרצים כרגע על ה-**Host** כולל אלו הרצים בתוך קונטיינרים אחרים! ולכן אפשר להכין גם עוד ערוץ תקשורת שיהיה מבוסס על שינויים בכמות התהליכים הרצים בכל רגע נתון, בדיוק כמו הדוגמא המצורפת של שינויים בכמות הזיכרון שבשימוש.

אני מעריך שאם נשקיע עוד כמה שעות מחקר בשימושים הנוספים האפשריים בכל השדות של Sysinfo וכן בכל השדות שב-Meminfo נוכל למצוא בוודאי עוד דרכים להעביר מידע אך הדוגמאות פה מספיקות להעברת הרעיון שקיימים דרכים להעביר מידע בצורת Covert channel בין קונטיינרים (כמובן שאתם מוזמנים לספר לי על רעיונות נוספים).

מימוש

“Talk is cheap. Show me the code.” - Linus Torvalds

הכנתי הדגמה יחסית פשוטה שמראה את הרעיון המתואר בפרק השני, בהדגמה ההנחה היא שקיימת כבר אחיזה של התוקף בשני קונטיינרים שרצים על אותו Host והדבר היחיד שחסר לתוקף זה היכולת להעביר ביניהם מידע מכיוון שהם מקונפגים בצורה כזאת שאין ביניהם תקשורת.

ההדגמה מנסה להמחיש את היכולת של קונטיינר אחד לשלוח פקודה לקונטיינר השני על מנת שהוא יבצע אותה (C&C).

ההדגמה מכילה שני קבצים:

- קובץ אחד נקרא sender.py זה הקובץ שהקונטיינר שרוצה לשלוח מידע כלשהו מריץ.
- וקובץ נוסף שנקרא receiver.py שזה הקובץ שהקונטיינר שרוצה לקבל את המידע מריץ.

חשוב לי לציין: ההדגמה לא מתייחסת בצורה רצינית לעובדה שייתכן ויש עוד הרבה דברים שרצים במקביל אשר משפיעים גם על מצב השימוש ב-RAM, כלי תקיפה אמיתי אשר ירצה להשתמש בשיטת תקשורת זו יאלץ להרכיב על גביה פרוטוקול שיחה שלם, ובו יכולת תיקון שגיאות, מנגנוני תזמון, סנכרון גודל של Buffer-ים להקצאה, זמנים בהם ה-Host יחסית יציב וכו'.

את הקוד המלא אפשר למצוא כאן.

החלק המרכזי ביותר של sender.py הוא:

```
string_to_send = raw_input("Enter your command:\n")

# Append the first zero since we're sending ascii values
bin_to_send = "0" + bin(int(binascii.hexlify(string_to_send), 16))[2:]

raw_input("Press Enter to start sending...\n")

logger.info("Synchronize bit window time")
while datetime.datetime.now().second % 20 != 0:
    time.sleep(0.1)

for curr_binary in bin_to_send:
    if curr_binary == "1":
        # save in memory
        temp = "a" * BYTES_IN_RAM_TO_USE
        time.sleep(time_between_send_one)
        del temp
        logger.info("{} : Send : 1".format(datetime.datetime.now()))
    else:
        time.sleep(TIME_BETWEEN_SEND_ZERO)
```

הסבר:

1. מקבלים מהמשתמש פקודה שאותה נרצה לבקש מהקונטיינר השני לבצע.
2. אנחנו מעבירים אותה לפורמט בינארי.
3. אנחנו מחכים עד לזמן שהגדרנו מראש בשני הצדדים, כרגע זה סתם המתנה של עד הרגע שבו השניות מתחלקות ב-20 בלי שארית. הסיבה להמתנה היא שאנו רוצים לוודא ששני הקונטיינרים מסונכרנים בדיוק על השנייה שבה הם מתחילים להעביר ביניהם את המידע.
4. אנחנו עוברים על הערך הבינארי שאנחנו רוצים להעביר, לדוגמה אנחנו רוצים להעביר את הערך '1' אז נקצה בזיכרון משתנה בגודל קבוע מראש, נחכה גם כן זמן שקבוע מראש עם הקונטיינר הנוסף. ואם נרצה להעביר את הבינארי '0' אז נחכה רק את הזמן אשר קבוע בין שני הקונטיינרים.

במקביל, הקטע קוד המרכזי ב-receiver.py הינו:

```
raw_input("Press Enter to start receiving...\n")
logger.info("Synchronize bit window time")
while datetime.datetime.now().second % 20 != 0:
    time.sleep(0.1)

need_get_more_binary = True
while need_get_more_binary:
    curr_binary_string = ""
    for i in range(SIZE_BYTE_IN_BINARY):
        curr_binary_string += check_if_transform_data()

    if curr_binary_string == NULL_ASCII_BIN_STR:
        logger.info("get null, so finish transfer.")
        need_get_more_binary = False
    else:
        n = int(curr_binary_string, 2)
        full_receive_string += binascii.unhexlify('%x' % n)

logger.info("Receiving : {}\n".format(full_receive_string))
logger.info("Running it as a command...")
os.system(full_receive_string)
```

1. מסנכרנים את השעה בדיוק כמו ב-sender
2. נקבל עוד ועוד ערכים בינארים עד שנקבל null (שמונה אפסים) שמסמן עצירה.
3. כל פעם משרשרים ל-string שנוצר.
4. בסוף מריצים את זה בתור פקודה.

אני ממליץ לעבור על הקוד המלא ב-Github ומי שרוצה מוזמן גם להרחיב אותו או לכתוב שם המלצות לשיפור (יש המון) וגם לפרגן ב-Star לא יזיק ☺. מי שמעוניין לבדוק את זה אצלו מוזמן להשתמש ב-Images המוכנים כלומר פשוט להריץ את הפקודות הבאות בשני terminals שונים:

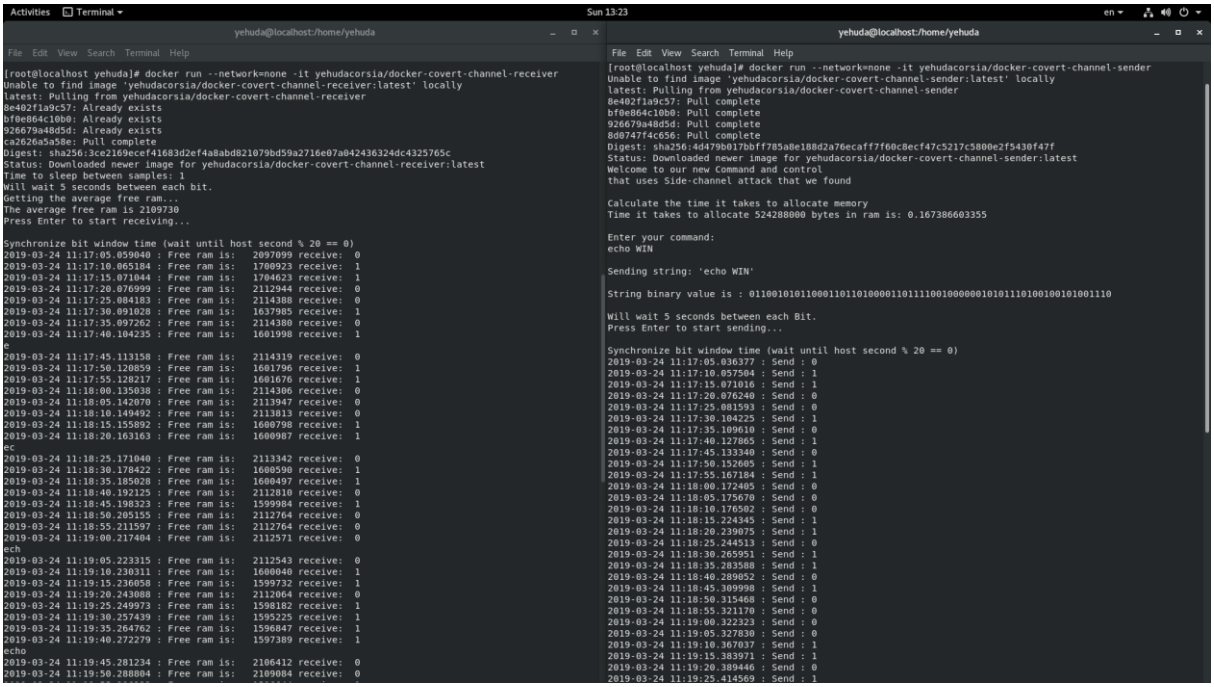
```
docker run --network=none -it yehudacorsia/docker-covert-channel-sender
```

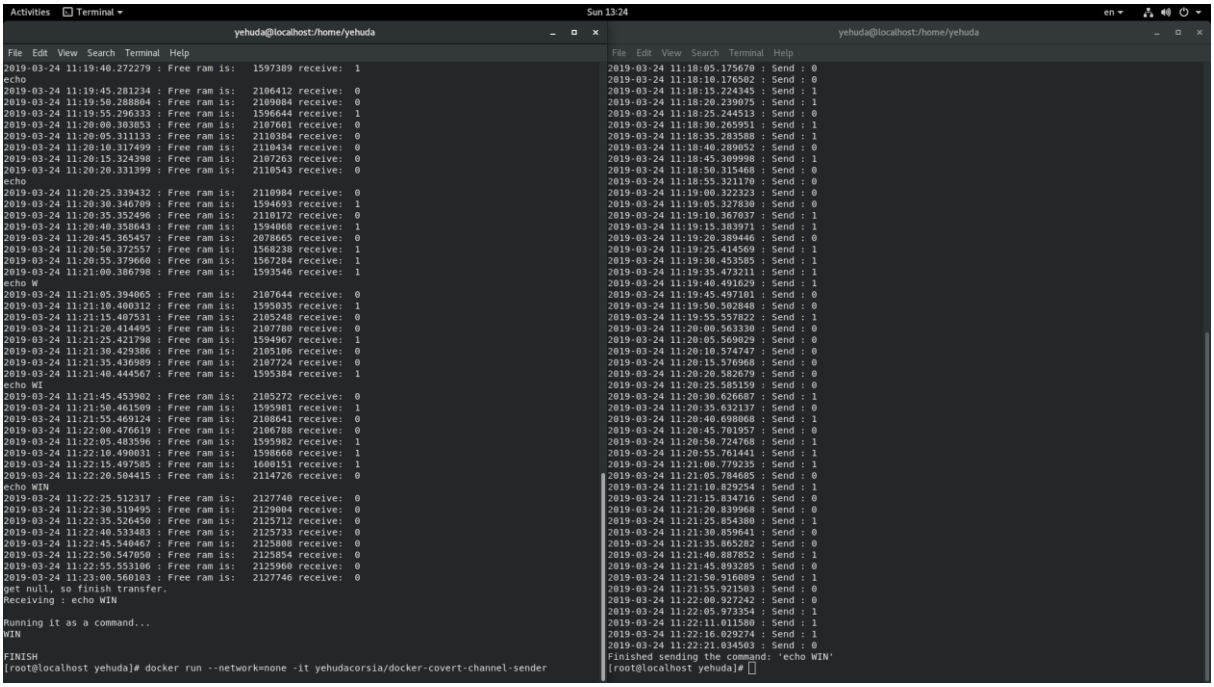
```
docker run --network=none -it yehudacorsia/docker-covert-channel-receiver
```

ולראות את הקסם קורה.

בנוסף, מי שרוצה מוזמן לבנות את ה-Images בעצמו מה-Dockerfiles המופיעים ב-Github.

בתמונה הבאה ניתן לראות הדגמה של הריצה של שני הקונטיינרים במקביל. מצד ימין רואים את Sender ומצד שמאל את receiver. בתמונה הראשונה ניתן לראות את התחלת השליחה של המידע, התמונה השנייה מראה את המצב בגמר השליחה וביצוע הפקודה על ידי ה-receiver:





הכנתי גם סרטון שמראה את ההדגמה ואפשר לראות אותו כאן.

תגובת Docker

שלחתי כמובן את המידע לגבי החולשה לחברת Docker בצירוף הקוד, וכל השיחה איתם מופיעה ב-Github של הפרויקט, זה חלק מהתגובה שלהם:

```
Thanks for this report (and neat PoC), unfortunately we don't agree that
this is a vulnerability -- for a few reasons.

1. In order to exploit this you already have two containers that you
   wish to communicate on a target system -- which means that you
   already have code-execution within containers on the system and they
   are mutually co-operative. In other words, there is no "victim"
   process. I'm sure this would be a neat trick if you manage to get
   "blind container execution" to exfiltrate information between
   containers, but given that all the processes are co-operative I don't
   see this as an issue.

2. This issue is actually a kernel issue (/proc/meminfo and the other
   memory-info APIs aren't cgroup-aware, leading to information about
   global system resources being scoped inside the container). As you
   said, this issue has been known about for a really long time -- but
   kernel developers have categorically stated they aren't interested in
   /proc/meminfo becoming cgroup-aware because of how complicated the
   infrastructure would be to make it work. So, we don't really have the
   ability to fix the fundamental issue.

   There is a project called LXCFS which allows you to mask
   /proc/meminfo in containers, and you could use this to prevent some
   of the problem -- but there are other memory-info APIs that can't be
   fixed so easily. You could trick them with the new seccomp
   RET_USER_NOTIF API, but that would not help older kernels and so on
   (and ptrace would kill performance in containers by a fair amount,
   and break gdb).

3. Isolating machines to avoid side-channels like this is almost an
   intractable problem -- because at the end of the day you are running
   on the same hardware and there will always be hardware tells (latency
   in HDD access or even CPU load -- both can be limited with cgroups
   but you'd likely be able to tell). Even VMs are vulnerable to
   side-channels like this, because shared-tenant systems fundamentally
   have to share resources that aren't designed to be side-channel
   resistant.
```

אעיר ואומר שבנוגע לנקודה הראשונה שלהם אני מסכים עם הנקודה שזה דורש משהו לא טריוויאלי כלומר זה דורש באמת הרצת קוד על שני קונטיינרים אבל זה עדיין לא סותר את העובדה שזה כן מאפשר לנו יכולת שמעניינת תוקפים והיא היכולת לעבור בין רשתות ולהעביר מידע בין שני תהליכים שצריכים להיות נפרדים וכולי.

בנוגע לנקודה השנייה שלהם, היא כמובן נכונה והיא בעיקר מביאה לנקודה שסקופ הבעיה כאן הוא הרבה יותר גדול וההשפעה שציינתי ב-**Docker** היא רק היבט אחד שלה. בנוגע לנקודה השלישית, אמנם זה נכון שייתכנו מתקפות דומות גם על VM, אך הם יהיו הרבה יותר מסובכים עד כדי לא אפשריים לעומת החולשות המתוארות פה (side channel attack in VM's here).

מה עוד?

כמו שאתם יכולים להבין, בהינתן ממשק משותף וחשאי בין ה-Host לבין כלל הקונטיינרים, ניתן לעשות עוד מספר רב של דברים, ויצירת ערוץ דלף הוא רק יישום אפשרי אחד. להלן מספר דוגמאות נוספות שניתן לממש:

- **Side channel attack** - ואסביר קצת על הנושא: ניתן לחלק את סוגי המתקפות בעולם המחשבים לשני סוגי מתקפות עיקריות:
  - סוג המתקפה הראשון הוא התקפה ישירה, אלו הן ההתקפות הסטנדרטיות שאנו מכירים. בהן התוקף מבצע מתקפה ישירה על המטרה. ולאחר מכן מצליח להשיג הרשאות אשר לא ניתנו לו מראש.
  - סוג המתקפה השני, והמעט פחות מוכר, הינו התקפה עקיפה (או באנגלית Side channel attack). על הנושא נכתב בהרחבה מאמר מאת יובל סיני ופורסם בגיליון ה-79 של המגזין: התקפות מסוג Side channel attack יהיו התקפות עקיפות על מטרתו של התוקף. בהן במקום לבחון את המידע שאותו נרצה לתקוף באופן ישיר - נחפש נגזרות שלו באופן עקיף ואותן ננסה לנתח. הדוגמאות המוכרות הן בדיקת הזמן שלוקח לפעולת חישוב להתבצע, צריכת חשמל, רעש או טמפרטורה שמעגל מסויים צורך או מפיק, ועל פי הנתונים הללו להסיק מסקנות על התהליך ובכך לחשוף נתונים על המטרה שלנו באופן עקיף.

בעוד שהמטרה של סוג המתקפה הראשון יהיה להשיג יכולות כגון הרצת קוד על המטרה, שינוי הרשאות או עריכה של מידע, המטרה של המתקפה השניה תהיה הרבה פחות מוגדרת ושלמה ובהרבה מקרים תהיה רק חלק משרשרת חולשות שישרתו בסופו של דבר מטרה גדולה יותר. דוגמאות לתוצאות של מתקפות צדדיות יכולות להיות: הבנה האם תהליך פענוח הצליח (ללא ידיעת התוכן שפוענח), הבנה האם נעשה שימוש באלגוריתם דחיסה ספציפי וכדומה. הרעיון הוא שברוב המקרים, לא נקבל את התוצאה הישירה שנרצה לקבל, אלא איזשהו נתון עקיף שבעזרתו נוכל להסיק מסקנות על מטרת התקיפה שלנו.

למרות שעל פי שתי הפסקאות האחרונות ניתן לחשוב בטעות כי סוג המתקפות השני הוא סוג מתקפות "חלש" יותר, אין לזלזל בו - בעזרת סוג מתקפות זה הצליחו בלא מעט פעמים לרסק אלגוריתמי הצפנה כבדים מאוד, וחולשות - מהמפורסמות ביותר - אשר מתבססות על קונספטים מסוג זה (Meltdown ו-Spectre לדוגמא, או BREACH ו-CRIME על TLS).

ובמקרה שלנו, כאשר אנו יכולים לדעת מה כמות ה-RAM בשימוש וכן מה הם מספר התהליכים הרצים כרגע ב-Host ועם מחקר מספיק מקיף נוכל להצליח לבצע המון מתקפות מסוג Side channel attack לדוגמא להצליח לשבור הצפנה של קונטיינר שמצפין מידע ונמצא גם על אותו Host, זיהוי של סוג השרת הנמצא בקונטיינר לידינו גם אם הוא מנסה לבלבל אותנו ומסתיר את סוג השרת (מוזמנים לקרוא עוד על הרעיון במאמר המצוין של אפיק קסטיאל, בכללי לא ארחיב כעת רק אומר שהתחלנו

מחקר בנושא ויש לבינתיים תוצאות מאוד מעניינות ואולי בקרוב יתפרסם מאמר המשך עם מימוש של העניין ☺) ועוד אינספור דברים שכבר בוצעו בעבר בנושא ודברים נוספים בסגנון שלהם.

- שיפור רוחב הערוץ - כרגע ההדגמה היא יחסית מאוד פשוטה אך ניתן לשפר אותה באמצעות אלגוריתמי דחיסה ואף באמצעות שימוש בתאים נוספים שניתן לשלוט עליהם ב-meminfo וכדומה

- **Hide and Seek with AVs** - ערוץ שאינו מפוקח ע"י תוכנת ה-Antivirus: ע"י הרצת ה-sender.py וה-receiver.py על אותה המכונה, ניתן ליצור ערוץ תקשורת חשאי בין שני תהליכים שונים על אותו הקונטיינר ובכך להתחמק מתוכנות כגון Antivirus אשר מחפשות קישורים ישירים כגון יצירת קובץ, ובדיקה איזה תהליכים אחרים ניגשים אליו, או כתיבה ישירה לזיכרון של תהליך אחר. באמצעות יצירת פרוטוקול תקשורת המתבסס על הקצאת ושחרור זיכרון ניתן להעביר מידע בין תהליכים באופן עקיף.

- בחרתי להראות את הדוגמא ספציפית על Docker אבל כמו שכבר צוין היקף הבעיה הוא הרבה יותר גדול, ואפשר באמצעות החולשה הזאת להצליח להעביר מידע בין שני תהליכים גם על רכיב המריץ Linux בצורה רגילה ולהרוויח את היתרונות המוזכרים לעיל.

סיכום

הנקודה המרכזית שרציתי להראות במאמר זה שכאשר חושבים על כל הנושא של הפרדה רשתית חשוב לשים לב שאם ההפרדה באמת חשובה לנו גם מבחינה אבטחתית, אנו צריכים תמיד לשקול את האפשרות להשתמש בהפרדה חזקה יותר מאשר ההפרדה הפשוטה שמקבלים באמצעות קונטיינרים, בדרך כלל נרצה להשתמש בהפרדה של vm-ים שהיא כידוע הרבה יותר חזקה, אך כמו שגם Docker בעצמם ציינו גם בהפרדה באמצעות vm-ים ייתכנו מתקפות מסוג side channel attack או covert channel וכן ייתכנו התקפות של בריחה מ-VM אשר ראוי לקחת בחשבון ולכן לפעמים נרצה ממש הפרדת Air gap חשוב שוב לציין שאמנם ההתקפות מהסוג הנ"ל ייתכנו גם ב-VM אבל הם יהיו משמעותית הרבה יותר קשות עד בלתי אפשריות.

כמו שראיתם, חברת Docker איננה מתייחסת לעניין זה כאל נושא שצריך לתקן אותו, ולכן הדרך הטובה ביותר להתגונן מפניו הוא - פשוט להכיר אותו, לדעת שהוא שם ולהתייחס אליו כאשר מאפיינים את אבטחת המערכת עוד בשלב התכנון שלה.

כאשר מדברים על התגוננות מפני מתקפות אלו, חשוב להבין ש-Docker אינו מתיימר להיות פתרון אבטחתי. ולכן אין להסתמך עליו כאל פתרון זה. במידה ונרצה לספק הפרדה אמיתית בין רשתות / מערכות עלינו לבצע זאת באופן פיזי. ניתן לקרוא וליישם את ההמלצות של NERC, לבצע הפרדה פיזית בין הרשתות הארגוניות לבין הרשתות המבצעיות.

תוך כדי כתיבת המאמר, אפיק קסטיאל הפנה את תשומת ליבי למחקר מרתק שנעשה בתחום.

מדובר על קבוצת חוקרים שיצרו כלי אוטומטי לזיהוי חולשות מסוג Side channel attack באנדרואיד המבוססות על פגיעויות הקשורות ב-proc/ ממליץ בחום רב לקרוא את המחקר שלהם.

על המחבר

יהודה כורסיה הוא חוקר ומפתח יכולות הגנה בדגש על טכנולוגיות ענן.

- אימייל: yehudacorsia@gmail.com
- גיטהאב: https://github.com/YehudaCorsia
- טוויטר: https://twitter.com/YehudaCorsia

רישום CVE-ים - מדריך לחוקר המתחיל

מאת אייל איטקין

הקדמה

אז התחלת מחקר, מצאת חולשה מגניבה במוצר מסחרי / ספריית קוד פתוח ואפילו הצלחת להרים הדגמה מגניבה, סחתיין ☺ אבל זה לא הסוף, עכשיו צריך להתחיל בתהליך הבירוקרטי של "responsible disclosure" והוצאת CVE רשמי לחולשה שמצאת.

מדריך זה נכתב על מנת שתהליך רישום ה-CVE ילך לכם בצורה החלקה ביותר, מקווה שתמצאו אותו מועיל.

הערת שוליים:

- מדריך זה (כמעט) ולא יעסוק בתהליך ה-"responsible disclosure". עם זאת, המדריך יכלול המלצות בנוגע לסנכרון הזמנים בין שני התהליכים הנ"ל.
- מדריך זה נכתב מנסיוני האישי כחוקר עצמאי וכחוקר כיום בחברת Check Point. עם זאת, המדריך נכתב מנקודת מבטי האישית כחוקר, וביוזמתי האישית, ללא קשר למעסיק שלי.

למרות שמדי פעם ישנם שינויים בבירוקרטיית רישום ה-CVE-ים, המדריך נכון למועד כתיבתו וכולי תקווה שיהיה מועיל ככל הניתן גם בעתיד.

מבוא

קצת מונחים

לפני שנתחיל, בואו נסביר את המונחים השונים:

- **C**ommon **V**ulnerability and **E**xposure - CVE: מזהה ייחודי (חד-חד-ערכי) לחולשה
- **C**VE **N**umbering **A**uthority - CNA: גוף או חברה אשר מורשה להנפיק מזהי חולשות
- **C**ommon **W**eakness **E**numeration - CWE: מזהה חד-חד-ערכי לסוג טכני של חולשה
- MITRE (לינק): הגוף הראשי האחראי על רישום והפצת מזהי החולשות
- **N**ational **V**ulnerability **D**atabase - NVD (לינק): מאגר פומבי נוח של מזהי החולשות שהתפרסמו לאורך ההיסטוריה. המאגר מנוהל ומתוחזק על ידי ממשלת ארה"ב.

עד לשנים האחרונות, מזהה החולשה היה מהצורה הבאה: CVE-2016-1234. כלומר, בדוגמה זו:

- החולשה **נרשמה** בשנת 2016 - במידה והתיקון מתעכב, החולשה לפעמים תתפרסם בשנה העוקבת (2017 למשל). על כן החלק הראשון במזהה מציין מתי החולשה התחילה את תהליך הרישום.
- 1234 - מזהה חד-חד-ערכי (חח"ע) של החולשה בשנה בה היא נרשמה.

למה קיימים CVE-ים?

בעוד שחוקרים רבים משתמשים ב-CVE-ים כדי לנופף בכמה חולשות הם מצאו, לא זו המטרה המקורית לשמה הוחלט לתת מזהים לחולשות. מזהה חולשה תוכנן כך שיהיה חח"ע, בכדי לעמוד במספר יעדים:

1. תיאום וסינכרון בין צוות המחקר ובין הגוף או הגופים הפגיעים לאותה החולשה, ונמצאים בתהליך התיקון שלה:
   a. דוגמא טובה לכך היא CVE-2014-0160 הידוע גם בכינוי (heartbleed)
   b. במקרה הזה מספר רב של גופים נדרשו לתקן את אותה החולשה שנבעה מספריית קוד פתוח מאוד פופולארית, ששולבה במספר רב של מוצרים ושרתים
   c. השם עצמו ניתן לחולשה רק לאחר הפרסום שלה כאמצעי מיתוג שנוי במחלוקת (פרקטיקה שנהוגה גם כיום, אם כי בעצימות נמוכה יותר)
2. עדכון הציבור בנוגע לתיקון חולשה:
   a. (בעולם תקין) מזהה החולשה יופיע ב-Git Commit שמתקן את החולשה (קוד פתוח)
   b. (בעולם תקין) מזהה החולשה יופיע ב-Release Notes של הגרסה המתוקנת של המוצר
3. במקרה הפחות טוב, עדכון פומבי של הציבור בנוגע לחולשה שטרם תוקנה:
   a. עבור חולשות שנמצאו כשהן מנוצלות בפומבי ("in the wild") ללא דיווח של חוקר ליצרן
   b. במקרים בהם המוצר לא תוקן בחלון הזמנים שהחוקר הגדיר, או שהיצרן בחר שלא לתקן

או בקיצור, CVE הוא השם המדעי של החולשה, והוא תוכנן כך שיהיה ייחודי וברור.

מאחורי הקלעים

בשנים האחרונות דווחה כמות גבוהה יחסית של חולשות, ובכל שנה מחדש נגמר מאגר המזהים בני 4 הספרות של אותה השנה... כדי להתמודד עם הבעיה הוחלט להאריך את מזהי החולשות, ולכן כיום חלק מהמזהים כוללים 5 ספרות ויותר. לדוגמא: המזהה CVE-2018-20174 מתאר חולשה שדווחה בסוף שנת 2018 ולכן קיבלה מזהה של 5 ספרות במקום 4.

מכיוון והמזהים מחולקים לשנים, מאחורי הקלעים מתבצע תהליך הרישום הבא:

1. בתחילת השנה (לרוב) ה-CNA-ים השונים יבקשו מ-MITRE בלוקים של מזהי חולשות, כדי שיוכלו להשתמש בהם במהלך השנה
2. במידה ובלוק המזהים נגמר, CNA יכול לבקש מ-MITRE בלוק נוסף של מזהים לאותה השנה
3. MITRE גם היא מתפקדת בתור CNA, ולכן היא שומרת לעצמה בלוק מכובד של מזהי חולשות
4. במידת הצורך, עם כל הבלוקים "נגמרו", MITRE פשוט תקצה בלוקים חדשים עם מזהים גדולים יותר (ולכן אפשר להגיע גם למזהים של יותר מ-4 ספרות בשנה מסויימת)

לדוגמא, נגיד שבתחילת שנת 2019 חברת Google ביקשה מ-MITRE בלוק של 100 מזהים. MITRE תקצה לה בלוק של מזהים והיא תקבל לאחריותה את כל המזהים בתחום. למשל: CVE-2019-2200 ועד CVE-2019-2300. מרגע שההקצאה התבצעה, האתר של MITRE יראה כי כל אחד מהמזהים בתחום הוא

"שמור" (reserved). לעומת זאת, באתר של NVD לא נמצא אזכור לאף אחד מהמזהים הנ"ל, בגלל שהם עדיין לא הוקצו רשמית לחולשות ספציפיות.

שימו לב: קיומו של מזהה בעל 5 ספרות לא יעיד על כך שהיו 10,000 חולשות שדווחו באותה השנה. בסיכוי סביר להרבה מהחברות יש בלוקים של מזהים לא מנוצלים, משום שהקצאה בבלוקים היא מטבעה לא יעילה.

אז מי בכלל CNA ולמה זה חשוב?

כשבאים לדווח לחברה / פרויקט קוד פתוח על חולשה, נרצה לפנות ל-CNA המתאים ביותר, זה שבאחריותו לרשום את החולשה שמצאנו ולהקצות עבורנו את ה-CVE שניתן לה. לשם כך חשוב להבין מי הם ה-CNA השונים:

1. חברות מסחריות (כדוגמאת Adobe) ישמשו בד"כ כ-CNA ויקבלו בלוק של CVE-ים שישמש אותן כדי לרשום את החולשות שנמצאו במוצרים של החברה
2. חברות פיתוח ומחקר ישמשו בד"כ כ-CNA משולב (כדוגמאת Google)
   a. רישום של חולשות במוצרי החברה
   b. רישום של חולשות שנמצאו על ידי חוקרי החברה, במקרים בהם לא נמצא גוף מתאים יותר בכדי לרשום את החולשה בעצמו
3. פרויקטי גוף פתוח גדולים מספיק (קהילת מפתחי הקרנל של Linux למשל)
4. MITRE (ג'וקר)
   a. במידת הצורך אפשר לפנות ל-MITRE ישירות להנפקת CVE (יפורט בהרחבה בהמשך)
5. כל השאר - Distributed Weakness Filing Project (DWF)
   a. לכאן ידווחו כל החולשות שאין למי לדווח, לרוב חולשות בפרויקטי קוד פתוח קטנים, ושנמצאו על ידי חוקרים עצמאיים / חוקרים מחברות שאינן CNA
   b. הוספת ה-CNA הזה היא שינוי של השנים האחרונות, שנועד להפחית עומס מ-MITRE

את הרשימה המלאה של כל גופי ה-CNA אפשר למצוא בלינק הבא לאתר של MITRE.

פנייה ישירה ל-MITRE

באתר של MITRE יש טופס נוח למילוי כל הפרטים הנדרשים לרישום CVE - לינק (בתפריט יש לבחור באופציה "Request a CVE ID"). ישנם מספר מקרי קצה בהם נרשה לעצמנו לפנות ישירות אליהם, אבל העיקרי שבהם הוא שאתם מוגבלים בזמן והתהליך דרך DWF איטי מדי.

בכל פנייה ישירה ל-MITRE אני נוהג לכתוב בהערות למה פניתי ישירות אליהם, ועד עכשיו התהליך היה חלק.

הנפקת CVE

אני צריך לעשות משהו בעצמי?

פעמים רבות, החולשה מדווחת לחברה / פרויקט קוד פתוח שמנהל בעצמו את הקצאת ה-CVE-ים, לרוב מכיוון שהגוף האחראי על המוצר הינו CNA. דוגמאות:

1. באגים בקוד של PHP שמתוייגים כאבטחתיים, יקבלו CVE שיוקצה ישירות על ידי המפתחים של PHP (זמן טוב בד"כ גם להגיש את מה שמצאתם ל-bug bounty שלהם ב-hackerOne).
2. באגים בקוד של מוצרי Google, Microsoft או רוב החברות הגדולות יטופלו על ידי החברה עצמה, והיא תעדכן אתכם מה ה-CVE שהוקצה לה

בעוד שבמקרים אלו נחסך מכם הצורך לרשום בעצמכם את ה-CVE, יש לכך גם מספר חסרונות:

1. תיאור החולשה יכתב על ידי ה-CNA ועלול להכיל טעויות או שבקושי יכלול פרטים טכניים שיועילו לחוקרים אחרים ("נמצאה חולשה במודול X שעלולה לגרום Y" וזהו)
2. חומרת החולשה תדורג על ידי ה-CNA, ובמקרה שהוא היצרן הוא מטבעו ינסה להפחית מחשיבות הנזק שנמצא במוצר שהוא פיתח

או בקיצור, מי שכותב את הפרטים שולט בהיסטוריה.

טיפ: במהלך תהליך דיווח החולשה למפתחים ואחרי שאישרו את נכונות החולשה (ולא ישר במייל הראשון שנשלח), רצוי לשאול אותם אם אפשר "לסייע להם" בתהליך הקצאת ה-CVE. זאת כדי להבין האם אתם צריכים לעשות זאת, או שהם יקחו את האחריות לכך.

מלכוד: לא מעט פרויקטי קוד פתוח יגידו לכם כחלק מתהליך התיקון שאם לא הקצתם עדיין CVE לחולשה, אז הם יעדיפו שכבר לא יוקצה לה מזהה כלל. במקרה כזה תרצו להחליט האם להנפיק בעצמכם מזהה בכל זאת או לוותר על זה. את ההחלטה רצוי לעשות כתלות בחומרת החולשה שמצאתם.

באחריותי להנפיק CVE וגם מצאתי CNA מתאים, מה עכשיו?

חלק זה יתפצל בין שני תרחישים:

1. חוקר עצמאי שאינו חלק מ-CNA - כנראה שזה התרחיש הנפוץ
2. חוקר שעובד בחברה שהיא CNA

אני חוקר עצמאי

יש 3 דרכים נפוצות לפנות אל CNA:

1. טופס אינטרנט של MITRE (הוסבר קודם) - מענה ראשוני יתקבל תוך 24-48 שעות (די מהיר)
2. Google Form של DWF (ראו לינק קודם) - תהליך יחסית איטי אך פשוט
3. שליחת מייל ישירות ל-CNA

שליחת מייל ישירות ל-CNA תתבצע במספר מועט של מקרים, משום שלרוב יהיה מדובר על אותה החברה שדיווחתם לה על החולשה, ואז היא כבר תנפיק את ה-CVE בעצמה. עם זאת, במידה ונאלצתם לשלוח מייל, ההמלצה שלי היא לפתוח במקביל את הטופס האינטרנטי של MITRE, ולרשום במייל את כל הפרטים הטכניים, סעיף-אחר-סעיף כמו בטופס. בצורה זו תהיו בטוחים שלא פספסתם שום פרט.

אני חלק מקבוצת מחקר, והמעסיק הוא CNA

באופן מפתיע, התהליך במקרה הזה יהיה **מורכב** יותר, Go Figure.

היות והחברה שאתה עובד בה היא כבר CNA, זה אומר ש(בתקווה) יש לה בלוק מוכן של CVE-ים. גש אל האחראי על ניהול הבלוק בחברה שלכם, ותבקש ממנו שיקצה לך מזהה. באחריותו לדאוג לרישום כך שלא יהיו כפילויות פנימיות אצלכם בחברה.

במידה ולחברה שלכם עוד אין בלוק מוכן, או שהוא התמלא, גשו אל הטופס האינטרנט של MITRE, ובחרו הפעם באופציה: "Request a block of IDs (For CNAs Only)".

אז מתי MITRE ידעו מה פרטי החולשה?

בטופס של MITRE ישנה אופציה לדווח להם על חולשה שהונפקה על ידי ה-CNA ("Request an update to an existing CVE Entry"). אופציה זו מתפקדת בשתי צורות:

1. חולשה שהונפקה על ידי MITRE או שכבר התפרסמה - פרטי החולשה יעודכנו
2. חולשה שהוקצתה לבלוק של CNA וטרם פורסמה - החולשה תרשם על ידי MITRE והם לא ידעו מה לעשות איתה...

כן, קראתם נכון. מסיבה שעוד לא הבנתי, במקרה הזה, למרות שמילאתם את כל הפרטים בטופס האינטרנטי, אתם תקבלו מ-MITRE מייל שמבקש מכם להגיש שוב את הפרטים המלאים של החולשה באחת מהצורות הנתמכות (ראו רשימה כאן). אני לרוב מגיש את החולשות כ-"flat file" ושולח אותן בגוף המייל.

מלכוד #1: אם בפרטים שתשלחו ל-MITRE לא יהיו הפניות חיצוניות (בלוג שמסביר על החולשה, לינק להודעה של המפתח / יצרן בנוגע לחולשה או לתיקון שלה), הם **יסרבו** לרשום את החולשה, כי אין לה ראיות חיצוניות. וזה מתקשר למלכוד הבא.

מלכוד #2: במידה ו-MITRE קיבלו את הפרטים ולא היו בעיות, הם ככל הנראה **יפרסמו** את הפרטים תוך שבוע-שבועיים ממועד קבלתם. כן, גם אם כתבתם במפורש שלא יפרסמו כי אתם מסנכרנים הרבה חולשות בין מספר גופי פיתוח, לצורך פרסום מחקר גדול.

מסקנה: בשורה התחתונה, MITRE מתייחסים לדיווח שלכם כאל אישור לפרסום, ולכן רצוי לדווח להם על ההקצאה שביצעתם מהבלוק שלכם רק לקראת הפרסום המתוכנן.

אז מה הפרטים שמופיעים ב-CVE שנמלא?

לצורך ההסבר אני אעבור על הטופס הפומבי של MITRE (שלדעתי הוא הנוח ביותר), ואנסה להסביר כל סעיף שרלבנטי לחולשה עצמה (ולא שם היצרן, גרסא או האם היצרן אישר):

1. **סוג חולשה (Vulnerability Type)**
   a. זהו הסיווג הטכני של ה**חולשה** (ולא של המשמעות שלה כלפי המוצר הפגיע)
   b. MITRE מציעים ממש מעט אופציות, וזה די עצוב. אני לרוב בוחר ב-"Other or unknown" ורושם את הסוג המתאים בשדה החדש שמתווסף לטופס.
   c. בפורמט "flat file" השדה יקרא "Problem Type" ואז לרוב אחפש CWE מתאים שיתאר את החולשה (ראו רשימה מלאה כאן)
2. **משמעות החולשה (Impact)**
   a. מה תוקף יוכל להשיג בהינתן החולשה?
3. **רכיבים מושפעים / פגיעים (Affected Components)**
   a. מה הרכיב הפגיע במוצר?
   b. איזה מנגנון במוצר הוא המנגנון שאתם תוקפים?
4. **וקטור תקיפה (Attack vectors)**
   a. דרך איזה ממשק אתם תוקפים את המוצר? (רשת, קריאה מקובץ, וכו')
5. **המלצה לתיאור טכני (Suggested description)**
   a. במידה והתיאור שלכם מנוסח נכון, הוא ישאר כמעט כמו שהוא
   b. לפעמים MITRE יעשו איזה שינוי קוסמטי (למשל: הם שינו לי long packet field ל-long string, אפילו שהשדה המדובר בהודעה לא היה מחרוזת...)
   c. תנסו להסתכל על מזהים מפורטים של חולשות ולנסח כמו שהם ניסחו. לדוגמא:

> "FreeRDP prior to version 2.0.0-rc4 contains an Integer Truncation that leads to a Heap-Based Buffer Overflow in function update_read_bitmap_update() and results in a memory corruption and probably even a remote code execution".

   d. אחרי ניסוח אחד מוצלח, פשוט תחזרו אליו בכל פעם ותנסחו שוב כמוהו
   e. **חשוב:** אל תתהדרו במשמעות שלא הצלחתם להדגים בפועל. רצוי לכתוב "הרצת קוד" ללא מילות סייג, רק כאשר הצלחתם להדגים אחת. אחרת, תחליטו מה לדעתכם הסיכוי להשמשה מוצלחת, ותסייגו בהתאם:
      i. "probably" - אפשר ולא הדגמתי
      ii. "possibly" - יהיה קשה להדגים
6. **קרדיט (Discoverers)**
   a. זה השדה היחידי ש**לא** יופיע בפרסום הסופי - באסה, נכון?
7. **הפניות חיצוניות (references)**

a. לינקים לכל דבר שימושי (אפשר להשאיר ריק בנתיים):
   i. Release notes של היצרן שמודיע על התיקון
   ii. Git commit שמתאר את התיקון
   iii. Blog post שלכם שמתאר את המחקר שעשיתם

בסיכוי סביר, בזמן מילוי הטופס לא יהיו לכם הפניות חיצוניות. הפניות אלה יתווספו כאשר תרצו לפרסם את החולשה ולהודיע עליה לעולם. כרגע, החולשה תופיע כ"שמורה" (reserved) ולא תהיה פומבית לציבור.

פרסום החולשה - Public Disclosure

אז תהליך התיקונים הסתיים סוף סוף, והגיע הזמן לפרסם את המחקר שלכם. לשם כך יש 3 אופציות:

1. החברה רשמה את ה-CVE וכנראה תפרסם אותו בעצמה או בתיאום איתכם
2. אתם עובדים ב-CNA - תשלחו ל-MITRE את הפרטים כשבוע - שבועיים לפני הפרסום הרצוי
3. אחרת, נמלא את הטופס האחרון של MITRE מיד אחרי הפרסום

הטופס האחרון של MITRE הוא כאשר בוחרים באופציה ("Notify CVE about a publication") ובו נוכל לציין את רשימת כל ההפניות שנרצה להוסיף על אלה שרשמנו קודם.

חשוב: אין אפשרות לפרסם ללא הפניה למקור הפרסום: blog post שלכם או הודעה של היצרן.

טיפ: מכיוון ואין שדה "קרדיט" לאחר שה-CVE מתפרסם, הקפידו להוסיף הפנייה חיצונית לבלוג שלכם בו תכתבו על החולשה שמצאתם. בצורה זו החולשה תשוייך אלייכם, ותופתעו לגלות שלא מעט אנשים יכנסו לקרוא את הבלוג שכתבתם בזכות ההפניות הללו.

עדכון פרטים / סגירת קצוות

במידת הצורך, ניתן לחזור אל MITRE ולמלא "עדכון פרטים" ל-CVE שפורסם. בשביל העדכון כנראה שתצטרכו להוכיח זיקה לפרסום המקורי של ה-CVE, או למחקר שמצא את החולשה שדווחה. עדכון לרוב יתבצע במקרים הבאים:

1. הוספת הפניות חיצוניות - כקבוצת מחקר, את הפרטים שלחתם ל-MITRE לפרסום לפני שיצא Blog post, זה הזמן להוסיף הפנייה אליו
2. תיקון שגיאות - קורה מדי פעם, יחסית נדיר

התוצר הסופי

סיימתם את כל התהליך, ועכשיו יש לכם CVE משלכם, הוא מפנה למחקר פומבי שלכם (אחרת לא תקבלו קרדיט לחולשה) והוא פומבי כך שכל העולם יכול לראות אותו, מזל טוב ☺

אז איך יראה הפרסום, ואיזה פרטים הוא כולל? הפרסום הרחב ביותר יהיה ב-NVD, והוא יכלול את הפרטים הבאים:

1. תיאור החולשה - שדה הטקסט שכתבתם
2. הפניות חיצוניות - רשימת הלינקים שלכם ועוד מיליון לינקים לא מועילים שמתווספים עם הזמן ומתאמים בין יותר מדי אתרים שכל אחד מפרסם העתק משלו של מאגר החולשות הפומבי
3. מפרט טכני של החולשה, לרוב במבנה CVSS 3.0

CVSS מה?

CVSS 3.0 הוא התקן העדכני (בטח ישתנה שוב בזמן הקרוב) לפירוט הפרטים הטכניים של החולשה. מדובר על אוסף שאלות שמגדירות יחד את הסיכון שגלום בחולשה, ובהתאם ינתן לה ניקוד (רציונאלי, אך לאו דווקא שלם) בין 0 ל-10. התקשורת לפעמים מתלהבת כשמדווחות חולשות עם ניקוד מושלם של 10, למרות שהנסיון שלי עד כה מראה שהסיווג שמתבצע לחולשות הוא ברמה די ירודה, ולא תואם בפועל למורכבות הניצול / רמת הסיכון שהחולשה משקפת למערכת בסיטואציה אמיתית.

מחשבון מועיל לחישוב הניקוד של חולשה ניתן למצוא כאן, המחשבון גם מוסיף הסבר בנוגע לכל אחד מהקריטריונים, כולל דוגמאות די טובות לכל סעיף.

אז מה הקריטריונים וממה הם מורכבים?

1. **וקטור תקיפה:** רשתי (אינטרנט), קרבה (רשת מקומית), מקומי (לרוב קריאה מקובץ), גישה פיסית.
2. **מורכבות התקיפה:** פשוטה / מורכבת. תקיפה תהיה מורכבת במידה והיא הסתברותית ו/או תלויה במרכיבים שלא בשליטתו של התוקף ושעלולים להקשות על הניצול. תקיפה יכולה להחשב "פשוטה" גם אם היא דורשת מחוקר להיות מנוסה מאוד בעולם השמשת החולשות.
3. **הרשאות נדרשות:** כלום, מעט (הרשאות חלשות ובסיסיות), חזקות (הרשאות admin למשל)
4. **אינטראקצית משתמש:** נדרשת / לא נדרשת. מדובר בביט בודד של החלטה, אין כאן סקאלה.
5. **הקשר:** השתנה / לא השתנה. האם תקיפה מוצלחת משפיעה גם על רכיבים מלבד זה שנתקף?
6. **פגיעה בסודיות:** אין, מועטה, גבוהה.
7. **פגיעה בשלמות:** אין, מועטה, גבוהה.
8. **פגיעה בזמינות:** אין, מועטה, גבוהה.

קריטריונים אלה לרוב מחושבים על ידי MITRE בעצמה, אך לעיתים נקבעים על ידי ה-CNA שמנפיק את החולשה (דבר שמאפשר לו להפחית מעט בנזק כדי להמנע מרעש תקשורתי פוטנציאלי).

סיכום

תהליך הנפקת מזהה חולשה (CVE-ID) הוא טכני בעיקרו, ומשתנה בהתאם לגוף / מוצר בו נמצאה החולשה. רוב שרשרת הטיפול ברישום מורכבת מפקידים, ולכן אין סיבה לפחד מהתהליך. אני מקווה שהמדריך העביר בצורה טובה את הפרטים ואני מקווה שתמצאו אותו מועיל.

הערת המחבר

בשנים האחרונות מונפקים המון CVE-ים, ורובם המוחלט **לא** מועיל כאשר באים לתקוף מערכת (לצרכי מחקר כמובן). אנא הקפידו כי החולשות שאתם מדווחים עליהן יהיו חולשות אמיתיות ומועילות, והמנעו מהגזמה בחשיבות / תיאור החולשה. כלל האצבע שלי הוא שאשתדל ש**לא לדווח על חולשה שלא אוכל להשתמש בה בפועל**.

על המחבר

אייל איטקין: חוקר אבטחת מידע בקבוצת המחקר של חברת Check Point. עוסק בעיקר בתחומי ה-Embedded והתקשורת, בד"כ שבירת פרוטוקולי רשת ששמם מורכב רק מ-3 אותיות (FAX, I2P, PTP, RDP וכו'). מפעם לפעם עוסק בציד חולשות בפרויקטי Bug bounty, ברשימה הכוללת את: Microsoft, PHP, Perl (C)Python, (C)Ruby, MRuby ועוד.

- **בלוג אבטחה:** https://eyalitkin.wordpress.com
- **אימייל:** eyal.itkin@hotmail.com
- **Twitter:** @EyalItkin

זיוף שידור של ה-Ring Doorbell

מאת אור צינגיסר

הקדמה

מטעם עבודתי כחוקר אבטחה בדוג'ו בולגארד את מרבית יומי אני מבלה בחיפוש אחר חולשות IoT. ב-27 לפברואר הדגמנו על הבמה ב-Mobile World Congress Barcelona 2019 את הפריצה שלנו לשידור האינטרקום החכם של Amazon, אשר זכתה לסיקור תקשורתי רחב היקף. לצד ההדגמה החיה המוצלחת פרסמנו כתבה בבלוג החברה בה פירטנו אודות המתקפה. מאמר זה הינו ברובו תרגום הפוסט וכן מרחיב בנושא השמשת ההתקפה, אשר מאפשרת האזנת סתר לשידור וידאו ושמע מהאינטרקום וכן הזרקה של פקטות אלו כרצוננו. דרישת הקדם היחידה לביצוע המתקפה היא להיות עם רגל באותה הרשת כמו הטלפון בו האפליקציה מותקנת.

נתחיל בהקדמה על המוצר המדובר: Ring היא חברת בת של אמזון ועוסקת במוצרי IoT כמו מצלמות פנים, אינטרקומים, תאורה חכמה ועוד. ה-Ring Doorbell מקובע מחוץ לבית ומחליף את הפעמון הקלאסי. המשתמש מתקין את אפלקציית Ring ומקבל התראה כאשר מישהו מזמזם בפעמון. המשתמש יכול לראות ולשמוע את השידור מה-Doorbell ואם יבחר לענות, יוכל לדבר עם מי שמחוץ לדלת דרך הרמקול המובנה. בהנחה שמותקנת גם מערכת מנעול חכם כדוגמת Amazon Key, ניתן לפתוח את הדלת באמצעות האפליקציה הרלוונטית, למשל בשביל לפתוח למנקה או לשליח של אמזון.

כפי שראוי לעשות, התחלנו את המחקר על ידי סריקת ידיעות ופרסומים קודמים אודות המוצר. אכן, ב-2015 נמצאה חולשה שמאפשרת על ידי לחיצה על כפתור ההתקנה ב-Doorbell גניבה של פרטי ה-WiFi של הבית. הבעיה הייתה כפי שקורה רבות בעולם ה-IoT, בשאריות מה-SDK שמציעה חברת הצ'יפים

המייצרת את ה-SoC שבשימוש. דרך הרשת WiFi שה-Doorbell פותח במצב התקנה אפשר לגשת לעמודי אדמיניסטרציה אשר כוללים את פרטי ה-WiFi. שווה להזכיר ש-Ring מהרה לבצע עידכון תוכנה על כלל המכשירים בתוך החלון Disclosure שהוקצה.

מבוא לתעבורת זמן אמת

כדי להבין כיצד ממומשת המצלמה נתחיל בהסבר קצר על איך עובדת תשדורת זמן אמת: כאשר יש טריגר, בדרך כלל פיזי על ידי חיוג או זמזום, נשלחת הודעת הקמת שיחה בפרוטוקול הנקרא SIP (Session Initiation Protocol). ההודעה, שעוברת מעל UDP, מעבירה מספר פרטים חשובים - מזהה המחייג והמחויג, מזהה השיחה, סוג קידוד ה-stream, פורט פנוי שישמש להעברת התוכן ועוד. לאחר הודעה זו שנקראת INVITE, המחויג שולח הודעת SIP TRYING להבהיר שהוא בתהליך עיבוד הבקשה, ולבסוף SIP OK שמסמן שהערוץ מוכן, בו כתוב הפורט אליו יש לגשת. על מנת לפתוח את פורט ה-Data בסביבת NAT השירות מוציא פקטת STUN (Session Traversal Utilities for NAT) אשר מפעילה הפניה של פורט מסוים. לאחר מכן נשלח ACK סופי בדומה ל-TCP וניתן להתחיל לשלוח הודעות Data על גבי הפורטים שנפתחו. נציין ש-SIP הוא פרוטוקול ורסטילי שתומך במספר משתמשים מאחורי מרכזיה אחת (ריבוב מעל IP יחיד) וכן בניתוב דרך פרוקסים בדרך. בשלב זה משתמשים ב-RTP (קיצור של Real Time Protocol Transport) בשביל להעביר את ה-Codec (שיטת קידוד השמע / וידאו) שנבחר, פרוטוקול התומך ב-Sequencing וריבוב מקורות שונים.

ניתוח פעילות רשת

לאחר הבנת ייסודות התקשורת זמן אמת, נתמקד כעת על תעבורת הרשת שמתרחשת ב-Doorbell וכיצד השידור מגיע לאפליקציה. הארכיטקטורת תקשורת ש-Ring בחרה היא להשתמש ב-AWS (Amazon Web Services) כשרתי תמסורת. האפליקציה והמכשיר משדרים לענן ומקבלים ממנו את השידור של המקביל להם. תחילה הפעמון נלחץ ונשלחת בקשת REST לענן, אשר שולח notification לאפליקציה. האפליקציה וה-Doorbell שולחים במקביל SIP INVITE לשרת, כל אחד עם פורטים מוקצים משלו. ה-Doorbell עושה זאת ב-SIP עם headers מיוחדים ש-Ring הוסיפו:

```
> Ethernet II, Src: Tp-LinkT_15:0e:f7 (14:cc:20:15:0e:f7), Dst: D-Link_6a:3d:76 (5c:d9:98:6a:3d:76)
> Internet Protocol Version 4, Src: 192.168.36.129, Dst: 18.197.187.54
> User Datagram Protocol, Src Port: 15063, Dst Port: 15063
v Session Initiation Protocol (INVITE)
  > Request-Line: INVITE sip:5nlmmc40em6gm-2037kgh65bhbni@18.197.187.54 SIP/2.0
  v Message Header
    > Via: SIP/2.0/UDP 192.168.36.129:15063;rport;branch=z9hG4bK705780083
    > From: <sip:f4844c55c577@ring.com>;tag=TUF0000000020030F38
    > To: <sip:5nlmmc40em6gm-2037kgh65bhbni@18.197.187.54>
      Call-ID: KIYQRJDLGONFQMWPNXQJXZUDTJNYFINPK0000000020030F38
    > CSeq: 1 INVITE
    > Contact: sip:f4844c55c577@192.168.36.129
      Content-Type: application/sdp
      Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, MESSAGE
      Max-Forwards: 70
      User-Agent: Device/dpdv3/1.21.63
      Subject: Ding
    v X-SSRC-A: 0247647989
      > [Expert Info (Note/Undecoded): Unrecognised SIP header (x-ssrc-a)]
    v X-SSRC-V: 0704495836
      > [Expert Info (Note/Undecoded): Unrecognised SIP header (x-ssrc-v)]
    v X-Session-Hash:i9N27wjQZo3IKeToDvDQfXiQwwaPhLDg9mMRPRm/g6g=,2576
      > [Expert Info (Note/Undecoded): Unrecognised SIP header (x-session-hash)]
      Content-Length:  391
  > Message Body
```

[הודעת SIP מה-Doorbell לענן]

בתמונה רואים שימוש ב-X-SSRC-A, X-SSRC-V ו-X-SESSION-HASH, שדות שמרחיבים את הפרוטוקול אשר Ring הוסיפה. ככל הנראה מדובר בחתימה על הפרמטרים של השיחה (פורט מקור ויעד, IP מקור ויעד, מזהה שיחה) כדי שלא יהיה ניתן לשנותם, וכן מפתח ההצפנה שמשמש בשביל העברת ה-RTP שבאה לאחר מכן. נציין רק שהשיטה הסטנדרטית להעביר SIP מאובטח היא מעל TLS, ול-RTP יש וריאנט בשם SRTP בשביל שידור מאובטח.

לאחר הקמת השיחה ניתן לראות ש-Wireshark אינו יודע לפענח את תעבורת ה-RTP שעוברת:

ip.addr == 192.168.36.129

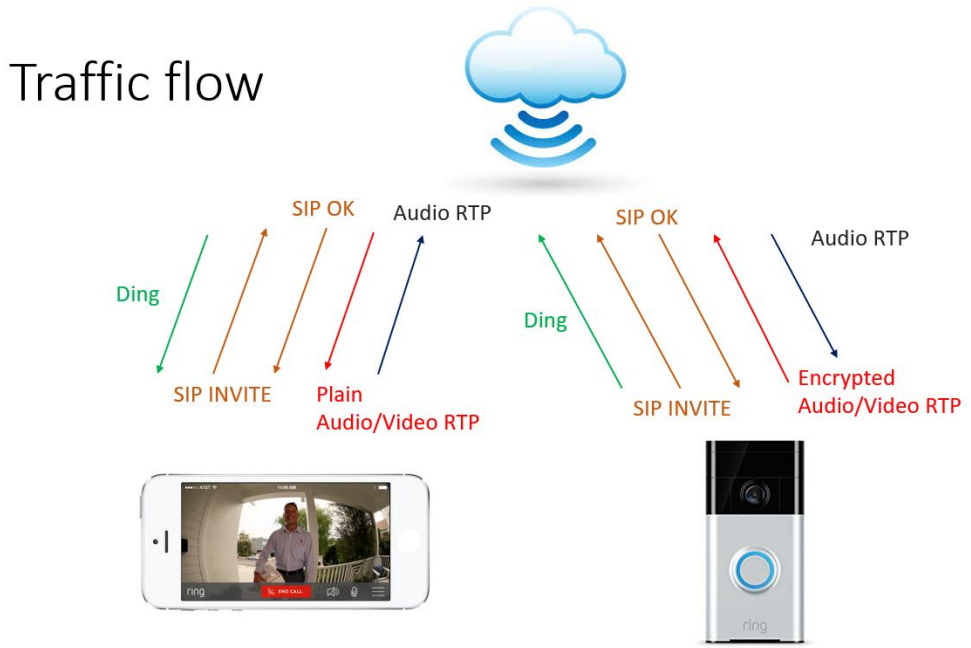
No.	Time	Source	Destination	Protocol	Length	Sequence number	Info
143	5.406392567	192.168.36.129	18.197.187.54	H264	600	35	PT=H264, SSRC=0x29FDC0DC, Seq=35, Time=30000, Mark PPS
144	5.420818303	192.168.36.129	18.197.187.54	H264	1331	36	PT=H264, SSRC=0x29FDC0DC, Seq=36, Time=36000 SEI
145	5.429096494	192.168.36.129	18.197.187.54	H264	1336	37	PT=H264, SSRC=0x29FDC0DC, Seq=37, Time=36000 Prefix
146	5.434674554	192.168.36.129	18.197.187.54	H264	1336	38	PT=H264, SSRC=0x29FDC0DC, Seq=38, Time=36000 IDR-Slice
147	5.443508968	192.168.36.129	18.197.187.54	H264	584	39	PT=H264, SSRC=0x29FDC0DC, Seq=39, Time=36000, Mark PPS
148	5.444573354	192.168.36.129	18.197.187.54	H264	1331	40	PT=H264, SSRC=0x29FDC0DC, Seq=40, Time=42000 EXT Unknown Subtype (16)
149	5.452881888	192.168.36.129	18.197.187.54	H264	1336	41	PT=H264, SSRC=0x29FDC0DC, Seq=41, Time=42000 End-of-Seq
150	5.463684223	192.168.36.129	18.197.187.54	H264	888	42	PT=H264, SSRC=0x29FDC0DC, Seq=42, Time=42000, Mark Reserved
151	5.488399791	192.168.36.129	18.197.187.54	H264	1331	43	PT=H264, SSRC=0x29FDC0DC, Seq=43, Time=48000 SPS
152	5.488453112	192.168.36.129	18.197.187.54	H264	728	44	PT=H264, SSRC=0x29FDC0DC, Seq=44, Time=48000, Mark End-of-Stream
153	5.491474228	192.168.36.129	18.197.187.54	H264	1331	45	PT=H264, SSRC=0x29FDC0DC, Seq=45, Time=54000 Slice-C
154	5.491516012	192.168.36.129	18.197.187.54	H264	888	46	PT=H264, SSRC=0x29FDC0DC, Seq=46, Time=54000, Mark EXT Unknown Subtype (29)
155	5.491539933	192.168.36.129	18.197.187.54	H264	1219	47	PT=H264, SSRC=0x29FDC0DC, Seq=47, Time=60000, Mark Reserved
156	5.494382503	192.168.36.129	18.197.187.54	H264	1331	48	PT=H264, SSRC=0x29FDC0DC, Seq=48, Time=66000 MTAP16 [Bad NAL Length]
157	5.506305783	192.168.36.129	18.197.187.54	H264	1320	49	PT=H264, SSRC=0x29FDC0DC, Seq=49, Time=66000, Mark IDR-Slice
158	5.508303475	192.168.36.129	18.197.187.54	H264	1267	50	PT=H264, SSRC=0x29FDC0DC, Seq=50, Time=72000, Mark Slice-C
159	5.515659724	192.168.36.129	18.197.187.54	H264	1331	51	PT=H264, SSRC=0x29FDC0DC, Seq=51, Time=78000 STAP-A [Bad NAL Length]
160	5.516410252	192.168.36.129	18.197.187.54	H264	1240	52	PT=H264, SSRC=0x29FDC0DC, Seq=52, Time=78000, Mark Reserved
161	5.528463180	192.168.36.129	18.197.187.54	H264	1331	53	PT=H264, SSRC=0x29FDC0DC, Seq=53, Time=84000 FU-A Start:SPS-Ext
162	5.539767242	192.168.36.129	18.197.187.54	H264	136	54	PT=H264, SSRC=0x29FDC0DC, Seq=54, Time=84000, Mark Reserved
163	5.539815603	192.168.36.129	18.197.187.54	H264	66	55	PT=H264, SSRC=0x29FDC0DC, Seq=55, Time=90000, Mark FU-A Start:MTAP16 [Bad NAL Length]

התוכנה חכמה מספיק להשליך את הצימודי קידוד-פורט שהוצהרו בשלב ה-SIP ל-dissectors העתידיים, אבל אפשר לראות שהתעבורה לא נראית הגיונית. הפירוק בעזרת codec של H264 (בשימוש ב-MP4) זורק שגיאות של Unknown Subtype, Bad NAL Length, Reserved ועוד מכיוון שהשכבה השביעית מוצפנת. לא הצלחנו לפענח את ההצפנה בעזרת הפרמטרים המיוחדים שעברו ב-SIP ולא הייתה לנו גישה ל-Firmware על מנת להבין כיצד היא מתבצעת.

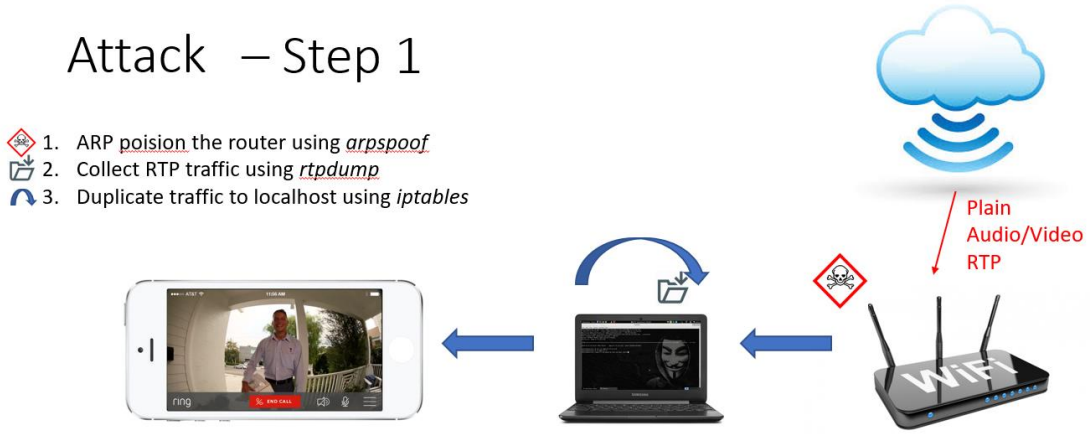
בשלב זה עברנו להסניף את האפליקציה. לצערנו ראינו שהתעבורת התראות, עדכונים, נתונים ו-SIP מוצפנת לשרת של Ring, כך שלא ניתן לראות את הקמת השיחה ישירות (אפשר בעזרת שיטות הסנפה יותר מתקדמות שמעבר ל-scope כאן). למרות זאת, אחרי שמצלצלים ורואים את השידור של המצלמה אפשר לראות את התוכן RTP ב-plain:

1310	11.878801752	18.197.187.115	192.168.36.124	H264	328	514 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=514, Time=660000, Mark FU-A End
1318	11.922528715	18.197.187.115	192.168.36.124	H264	1331	515 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=515, Time=666000 FU-A Start:non-IDR-Slice
1320	11.934711849	18.197.187.115	192.168.36.124	H264	1336	516 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=516, Time=666000 FU-A
1322	11.936379847	18.197.187.115	192.168.36.124	H264	1336	517 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=517, Time=666000 FU-A
1323	11.938401968	18.197.187.115	192.168.36.124	H264	344	518 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=518, Time=666000, Mark FU-A End
1356	12.003577768	18.197.187.115	192.168.36.124	H264	1331	519 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=519, Time=672000 FU-A Start:non-IDR-Slice
1357	12.003811649	18.197.187.115	192.168.36.124	H264	1336	520 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=520, Time=672000 FU-A
1359	12.005795296	18.197.187.115	192.168.36.124	H264	1336	521 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=521, Time=672000 FU-A
1362	12.007092537	18.197.187.115	192.168.36.124	H264	360	522 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=522, Time=672000, Mark FU-A End
1378	12.074726072	18.197.187.115	192.168.36.124	H264	1331	523 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=523, Time=678000 FU-A Start:non-IDR-Slice
1383	12.095755588	18.197.187.115	192.168.36.124	H264	1336	524 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=524, Time=678000 FU-A
1384	12.095763412	18.197.187.115	192.168.36.124	H264	1336	525 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=525, Time=678000 FU-A
1385	12.095767842	18.197.187.115	192.168.36.124	H264	280	526 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=526, Time=678000, Mark FU-A End
1401	12.145133318	18.197.187.115	192.168.36.124	H264	1331	527 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=527, Time=684000 FU-A Start:non-IDR-Slice
1416	12.152027842	18.197.187.115	192.168.36.124	H264	1336	528 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=528, Time=684000 FU-A
1431	12.216097001	18.197.187.115	192.168.36.124	H264	1336	529 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=529, Time=684000 FU-A
1432	12.216104244	18.197.187.115	192.168.36.124	H264	264	530 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=530, Time=684000, Mark FU-A End
1434	12.222939150	18.197.187.115	192.168.36.124	H264	1331	531 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=531, Time=690000 FU-A Start:non-IDR-Slice
1436	12.224437516	18.197.187.115	192.168.36.124	H264	1336	532 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=532, Time=690000 FU-A
1437	12.224441613	18.197.187.115	192.168.36.124	H264	1336	533 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=533, Time=690000 FU-A
1439	12.230072848	18.197.187.115	192.168.36.124	H264	408	534 PT=DynamicRTP-Type-96, SSRC=0x51F63391, Seq=534, Time=690000, Mark FU-A End

אלו פקטות ש-Wireshark מבין כהגיוניות בפורמט H264 ולכן רצינו לראות אם אפשר לשחזר מהן את הסרטון. השתמשנו בכלי בשם videosnarf, שמחלץ סרטונים / שיחות מתוך קובץ pcap, והצלחנו לראות את הסרטון מהמצלמה (H264) ולשמוע מהמיקרופון (G711)! המשמעות היא שכל מי שיכול לשים את ידו על הפקטות של האפליקציה יכול לראות את השידור. התמונה התבהרה ונראית כך:



אז איך נוכל לצפות בשידור? בתור התחלה יש להיות ברשת משותפת עם הטלפון. אפשר להיות ברשת Wi-Fi ציבורית משותפת כמו ברכבת או בבית קפה, להרים AP זדוני שהטלפון ייצטרף אליו או להשתמש במכשיר ביתי פרוץ בשביל pivoting. לאחר מכן, מבצעים מתקפת ARP בין הנתב והטלפון ומתחזים להם בשכבה שתיים. כעת נקבל את התעבורה המקורית ונוכל להעביר אותה הלאה גם לטלפון. התרחיש תקיפה ייראה כך:



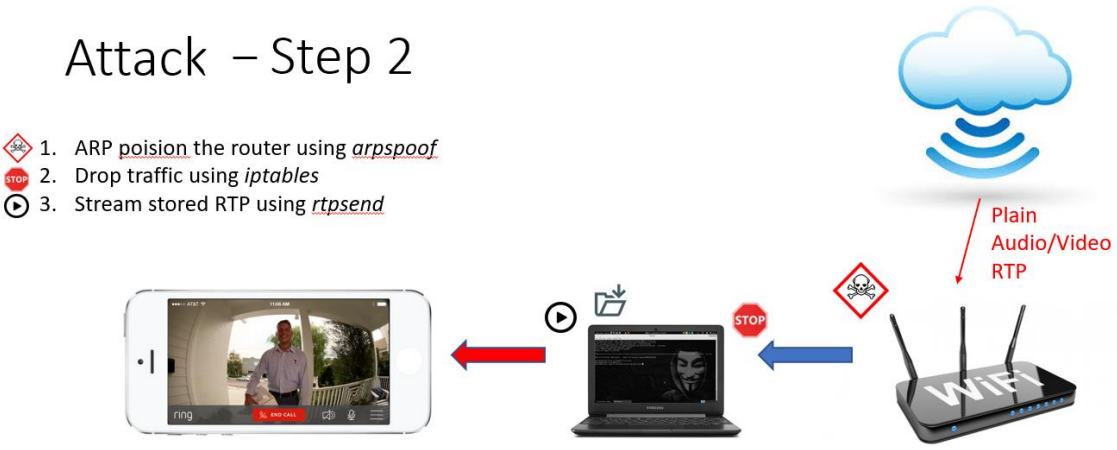
מרימים Kali ומריצים:

1. Arpspoof - מרעילים את הrouter ואת הטלפון כפי שהוסבר
2. Rtpdump - כלי לכתיבת תעבורת RTP מפורט לקובץ, אותו ניתן לקרוא
3. Iptables - שימוש ב-target בשם TEE לשכפל את התעבורה ליעד 127.0.0.1

כמובן שצריך להפוך את המחשב שלנו לנתב כדי להעביר את הפקטות ליעדם האמיתי על ידי:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

בעזרת setup זה יש לנו יכולת מרשימה של האזנת סתר לדלת כניסה של אנשים, בעזרתה ניתן לעשות הנדסה חברתית ולאסוף פרטים מעניינים רבים אודות ההרגלים בבית הקורבן. השלב הבא המתבקש הוא לזייף תעבורה:



שני שינויים על מנת לעבור למצב התקפה:

1. Iptables - הופכים את חוקי ה-TEE ממקודם לחוקי DROP ובכך מפילים תעבורת שמע ווידאו
2. Rtpsend - משדרים את קבצי ה-RTP ששמרנו בשלב הקודם לעבר הקורבן

בכך אנו משלימים התקפת replay בסיסית. בעזרתה, עשויים לשכנע את הקורבן לפתוח את הדלת מרחוק למי שהוא חושב שמכיר ושהוא סומך עליו. ההשלכות יכולות לנוע בסקלה שבין מתיחה מצחיקה לבין אירועים נוראיים כמו חטיפת ילדים שההורה שלהם פתח מרחוק למי שנראתה כמו הבייביסיטר.

דיווח וסגירת החולשה

החולשה דווחה לחברת Ring ובגרסה 3.4.7 באנדרויד הבעיה תוקנה.

סיכום

התקיפה שהוצגה במאמר זה מראה שגם בחברות הענק ובמוצרי דגל ניתן למצוא חולשות לוגיות חמורות שדרוש מעט מאוד ידע ומיומנות לנצלן. ניכר שנעשה מאמץ לאבטח את השידור בעזרת שכבת ה-TLS באפליקציה והשדות המיוחדים ב-SIP מהמכשיר כמו גם ה-RTP המוצפן. למרות זאת, מערך ההגנה חזק רק כחוזק החולייה החלשה שלו, שבמקרה זה הייתה התעבורת RTP לטלפון. אירוני שדווקא מוצר אבטחה שאמור לחזק את תחושת הביטחון פותח הזדמנויות תקיפה חדשות שלא היו קודם, וזהו מוטיב שחוזר על עצמו שוב ושוב בעולם האבטחת מידע.

מקורות

- תמונת Ring Doorbell:

https://target.scene7.com/is/image/Target/GUEST_6ae6702b-c0b5-427b-8b74-cf2761aaca3b?wid=488&hei=488&fmt=pjpeg

מבוא למערכות בקרה בעולם ה-OT

מאת גלעד זינגר

הקדמה

אנו רגילים לשמוע בכל מקום על אבטחת מידע וסייבר במערכות מידע, IT, אזורים פיננסיים, הגנה על אתרי אינטרנט, בסיסי נתונים הגנה על פרטיות וכד'. בנוסף, המילים: נתב, חומת אש ואפילו סוגי התקפות על מערכות כגון מניעת שרות, רוגלות ורושעות למיניהן, הפכו להיות שגרת השיח היומיומית בתקשורת.

אבטחת מידע "קלאסית" מושתתת לרוב על מודל ה-CIA - Confidentiality Integrity Availability ומתוך כך מרבית הנכסים עליהם נרצה להגן יהיו נכסי מידע (מוחשיים ולא מוחשיים) אשר הפגיעה בהם תוכל לייצר פגיעה בסודיות המידע, מהימנותו וזמינותו למשתמש.

הידעתם שיש יקום מקביל ל-IT בשם OT?

OT - Operation Technology - הינו התחום בו עולם הנכסים טיפה משתנה. מדובר ביקום בו חיים בקרים מתוכנתים, עמדות מהנדס ותפעול, שרתי סקאדה והמון הפתעות אשר למעשה משפיעות על כל מהלך בחיי היום יום שלנו לא פחות מעולם ה-IT.

ביקום זה עולם ה-**CIA** טיפה משתנה ונהפך ל-**AIC** שכן מדובר בתהליכים ופחות במידע. תהליכים בהם נרצה ראשית זמינות (לסגור את המשאבה מיד), מהימנות (מה הטמפרטורה האמיתית של מיכל אמוניה?) ולבסוף סודיות שכן לא נרצה לחשוף את תהליך העבודה של הבקר.

מאמר זה הינו ראשון מתוך סדרת מאמרים בה אפרט על הקווים לדמות עולם ה-OT, פרוט אשר יאפשר לכם הצצה קלה לעולם חדש-ישן זה. במאמרים הבאים נצלול לתחומי משנה בעולם ה-OT כגון עולם הבנייה (בנייה חכמה), מפעלים (Industry 4.0), עולם הספנות וכמובן נחבר את נושא ההגנה בסייבר בכל אחד מהתחומים.

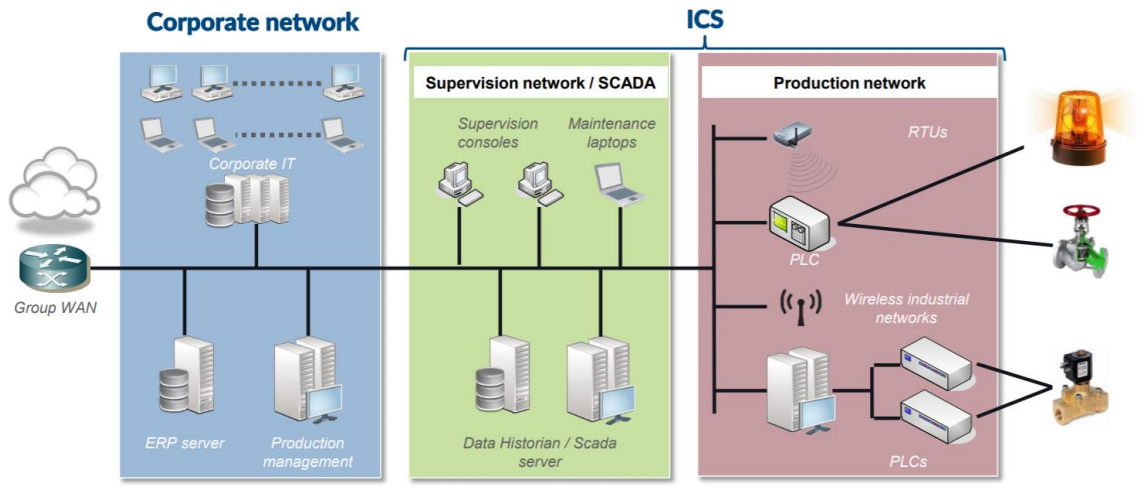
קריאה מהנה!

מבוא ל-ICS

נתחיל בקצת ראשי תיבות:

- Operation Technology - **OT**
- Industrial Control Systems - **ICS**
- Supervisory Control and Data Acquisition - **Scada**
- Distributed Control System - **DCS**

כיום אנשי מקצוע בתחום מתייחסים למערכות הללו לרוב בשם מערכות סקאדה. וכך היא נראת בכלליות:



מרכיבי המערכת (ארכז את המרכיבים המרכזיים בתחום):

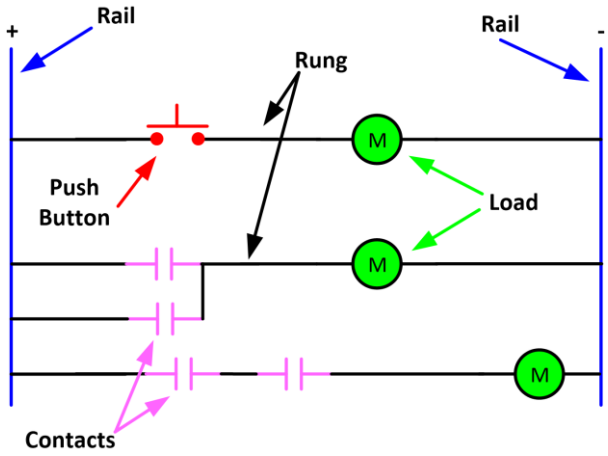
- **חיישנים/סנסורים/מפעילים:**

מאפשרים את החיבור בין עולם הבקרה לעולם הפיסי. חיישן יכול להיות לדוג' מדיד טמפרטורה אנלוגי או דיגיטלי אשר דוגם בזמן אמת טמפ' בחדר ומעבירה לרכיב הבא בשרשרת (בקר). מפעיל יכול להיות מנוע חשמלי או משאבה אשר מבצעת פעולה בהינתן פקודה (מהבקר...נכון). החיישן לרוב יעביר מידע לבקר והמפעיל בדיוק להיפך, יקבל מידע (1/0) אשר יפעיל או יפסיק את פעולתו.

- **בקר - Programmable Logic Controller - PLC:**

מחשב זעיר בעל כניסות ויציאות (Inputs/Outputs) של מידע בינארי (לרוב). קיימים כרטיסי הרחבה לבקרים כולל עבודה רשתית ללא כניסה ויציאות ומגוון רחב של אפשרויות עליהם לא נדון בשלב זה. הבקר יתוכנת לרוב בלוגיקה קבועה מראש, אשר תאפשר פעילות ללא התערבות מפעיל בשגרה. הבקר יבנה לרוב מחומר עמיד וקשיח שכן קיימים מקרים בהם תידרש עמידות לטמפרטורות לא שגרתיות ולתנאים סביבתיים קשים.

לבקר תוכנה וחומרה אשר ניתנים לעדכון ושינוי. אחת משפות התכנות של הבקר הינה "דיאגרמת סולם", שפה המדמה את פעולת הבקר כפי שנוכל לראות בתרשים הבא:



דוגמא ללוגיקה שתיצרב בבקר: הפעלת מסוע במפעל בהינתן הגעת חבילה לאזור מסוים בו יעצור המסוע והחבילה תסומן ותיסגר [מה היה לנו כאן? מסוע שמקבל (INPUT), סנסור שמצביע על כך שהחבילה הגיע למקומה (OUTPUT), עצירת המסוע (INPUT)].

- **Engineering Station /Human Machine Interface - HMI**

עמדת שליטה ובקרה על הבקר, כתלות בהרשאות וסוג העמדה, היא תאפשר צפייה ב-FLOW העסקי של המערכת (מה מחובר למה תהליכית), צפייה בסנסור (לדוג' מהי הטמפרטורה, מה מפלס המים וכד'). בעמדות המוגדרות עמדות מהנדס או פיקוח, יהיה ניתן לבצע שינויים בבקר עצמו ואף לשנות את לוגיקת ההפעלה מרחוק או לבצע הפעלה או הפסקה ידנית של תהליכים.

לדוגמא, מקרה בו קיימת תקלה בבקר ועל המהנדס לבצע מעקף של פעילות קריטית בתהליך, תינתן האפשרות לבצע התערבות בתהליך בצורה ידנית ולכבות/להדליק רכיב פיזי מרחוק (למשל משאיבת מים). לרוב נוכל לראות עמודת HMI מותקנות עם מערכות הפעלה מסוג חלונות (גם כאלו שלא נתמכות כבר, עליהן נדבר בהמשך), כמו כן לעיתים ישלבו תחנות אלו אמצעי תקשורת נוספים מלבד רשת רגילה (חיבורי רדיו, חיבורי סריאליים וכד').

- **Data Historian - היסטוריין**

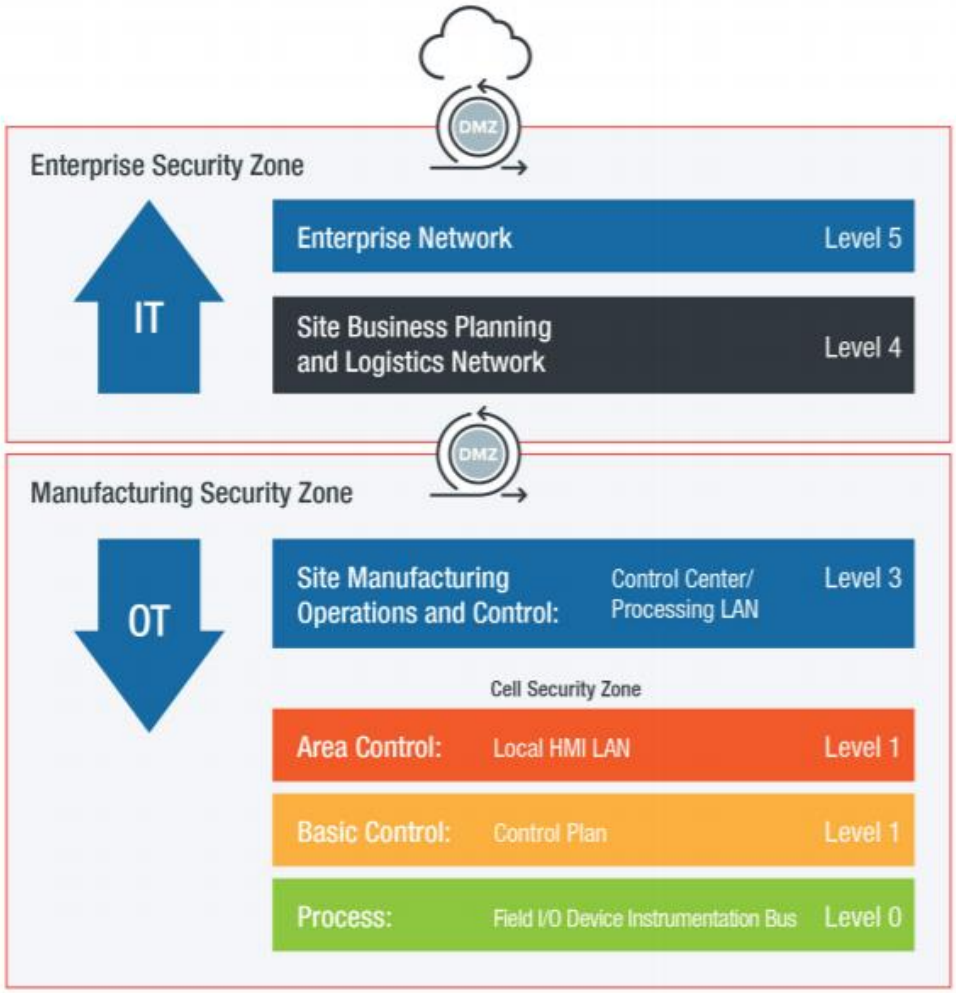
רכיב זה הינו למעשה בסיס נתונים האוגר כל פעילות המתרחשת ברשת ה-OT. הפעלת רכיב, קבלת חיווי מסנסור, טעינת קוד לבקר, עדכונים וכל פעולה אחרת, תנוטר ותועבר לשרת ההיסטוריין, משם תוכל להישלח כקובץ לוג למערכת SIEM או למרכז שליטה ובקרה (SOC) לטובת אגרגרציה וקורלציה עם נתונים אחרים על מנת לאתר בעיות תפעוליות, ניתוח אוחר וכמובן איתור אירועי אבטחת מידע ברשת.

אז כיצד כל היופי מתחבר לרשת?

פה נבצע הפרדה בין שני מושגים בתחום - רשת עסקית/מנהלתית ורשת תפעולית:

הרשת המנהלתית (IT) הינה הרשת בה מתנהל הארגון, בין אם זה מפעל מזון או תחנת כוח. ברשת זו נוכל למצוא את שרת הדוא"ל, שרת הקבצים, מערכות ERP, חיבור לדומיין מרכזי וכמובן לרוב חיבור לאינטרנט!

הרשת התפעולית (OT) כשמה כן היא, משמשת לתפעול התהליכים, בין אם מדובר בתהליכי ייצור תרופה בפס ייצור, תהליך הזרמת שפכים או העברת מים אליכם לברז בבית, בקצה כל פעילות כזו יושב בקר אשר מחליט - כן או לא.



כמו בעולם ה-IT גם כאן נעשה שימוש בפרוטוקולי תקשורת רבים ומגוונים.

חלקם ייעודיים למערכות או תעשיות ספציפיות, חלקם פרי פיתוח של יצרן וניתנים לשימוש רק במערכותיו וחלקם נפוצים במרבית המערכות הקיימות כיום.

שמות כגון **DNP**, **PROFIBUS**, **61850 IEC**, **BACNET** הינם רשימה חלקית ומצומצמת מאוד של פרוטוקולים בשימוש התעשייה.

אחד הפרוטוקולים הנפוצים ביותר הינו **MODBUS**, פרוטוקול סריאלי במקורו משנת 1979 אשר הוסב לעבודה בסביבת תקשורת IP/TCP.

מדובר בפרוטוקול בתצורת MASTER SLAVE, כאשר סוג הפעילות המועבר נקבע ע"ב מספר ידוע מראש (Function Code). הפרוטוקול מאפשר קריאה וכתיבה לבקר, התעבורה בו אינה מוצפנת ולא נדרשים כל אמצעי זיהוי ואימות.

מכיוון שעולם ה-OT וותיק, תחילה העבודה התקשורתית התבצעה ללא תצורת התקשורת המוכרת לכולנו היום אך עם ההתפתחות הטכנולוגית נולדו תוספות והרחבות לפרוטוקולים הקיימים על מנת שיוכלו לשלב ולהשתלב בתקשורות מודרניות, אחת מהן הינה ההרחבה לעולם ה-IP/TCP, הרחבה שאפשרה קישוריות לעולם החיצוני אבל לא תוכננה לעבודה מאובטחת במקור.

אז אם באבטחת סייבר עסקינן, כמה מילים על פערים באבטחת תשתיות OT.

מהם הפערים העיקריים באבטחת מערכות ICS/OT?

- חיבור בין רשתות - כאמור במערכות OT אנו צפויים לפגוש לפחות שתי רשתות שונות, רשת תפעולית ורשת מנהלתית. ברוב המקרים הרשת המנהלתית תחובר לאינטרנט (כמובן עם/בלי אבטחה ייעודית), אך מה קורה במקרים בהם קיים קישור ישיר בין הרשת התפעולית לרשת המנהלתית? תארו לכם מצב בו תוקף (ועל כך נרחיב בפרק הבא) "גולש" לבקר אשר אחראי על פעולת משאיבת מים של חברת אספקה, כמה נזק ניתן לייצר באותו רגע?
- ניהול עדכונים - קיים ויכוח אינסופי בתעשייה הבקרה, האם לעדכן או לא לעדכן את מערכת ההפעלה של הבקר, של רכיב ה- HMI או של כל רכיב אחר במערכת שכן עדכון או שינוי של מערכות אשר מחד בנות לעיתים 20 שנה ומאידך אחראיות על תהליכים רגישים של ייצור, עשוי לייצר פגיעה או השבתה העלולה לגרום לנזק כספי או אף פגיעה בחיי אדם. כתוצאה מכך נוכל לצפות למערכות לא עדכנית (Un Patched) ולעיתים קרובות לפגוש "דינוזאורים" בדמות חלונות XP ואף גרסאות ישנות יותר אשר אינן נתמכות ופגיעות ביותר.

- פרוטוקולים לא מאובטחים - הזכרתי את פרוטוקול MODBUS ככזה שאינו מוצפן בגרסתו הבסיסית והנפוצה, כמוהו קיימים פרוטוקולים רבים המעבירים את התעבורה בצורה גלויה וברורה, ללא מנגנוני הזדהות או אימות משתמשים וללא תמיכה בכל מערך אבטחה כזה או אחר המוכר לנו מעולם ה-IT.

- מודעות עובדים - איפה אתחיל? בעולם ה-IT מונהג בארגון המכבד את עצמו, תפקיד של CISO אשר אחראי על אבטחת המידע בארגון וחלק ממעגלי האבטחה הינה האדם עצמו (או העובד) לכן אנו רואים תהליכים רבים של העלאת מודעות העובדים לאבטחת מידע בארגונים רבים, גדולים כקטנים.

אך מה קורה בעולם ה-OT? כפי שהזכרתי בפתיח, לנכסי המידע קיימת חשיבות נמוכה יותר בעולם זה וברוב הארגונים לא תאתרו CISO אשר אמון על נכסי ה-OT מכאן שגם פחות נראה תהליכי מודעות עובדים או הכרות עם בעיות ההגנה בסייבר בעולם זה. עובדי תפעול אחראים שהתהליך בקצה יעבוד שכן כל עצירה של תהליך המפעל הינה אובדן כסף למפעל, האם במקרה כזה תמיד יבצעו חשיבה נוספת לפני שלמשל יחברו כרטיס סלולארי לבקר קריטי כדי לשפר את הקליטה שלו? לא חושב...

- ניהול מרוחק: מערכות הבקרה בעולם ה-OT מורכבות מאוד ולרוב נמכרות למפעל כמקשה אחת מהיצרן (או מיותר) כולל חבילת תמיכה מרחוק. תמיכה זו יכולה להתבטא בהשתלטות מרוחקת על בקר, עמדת מפעיל או כל רכיב אחר בתהליך המפעלי. בנוסף, קיימים ספקים (מחו"ל) אשר לא יאפשרו אחריות על המוצרים ללא פתיחת האפשרות לגישה מרחוקת לרכיבי המפעל בכל עת וללא אישור מבעוד מעוד. כמובן שפתיחת גישה מרחוק ללא הגבלה או שליטה, מאפשרת כר רחב לאפשרויות תקיפה של מערכת הבקרה דרך "שרשרת האספקה" (עליה יורחב במאמר הבא).
- בקרה- אנו מכירים היטב מעולם מערכות המידע תהליכי SOC/SIEM המאפשרים ניטור של רכיבי הרשת ומשתמשיה באופן כמעט הרמטי ומאפשרים קבלת התראות בזמן אמת אודות פרצות אפשריות, כניסות למערכת או תהליכים לא מורשים ולמעשה מסייעות ביצירת תהליך Incident Response ראוי במקרה של תקיפה או ניסיונות תקיפה והכלתה. בעולם ה-OT המצב קצת יותר מורכב בשל חוסר בשלות (קיימים ניצנים בתחום) הנובע בעיקרו מתפיסה שגויה של "אני מנותק מהעולם לכן אני מוגן". בשנים האחרונות אנו מתחילים לראות פתרונות ראויים בשוק אשר יסייעו לחבר את המפעל ותהליכיו למוקדי SOC (באתר או באתרים חיצוניים) תוך שמירה על אבטחת המידע (חזרנו למידע בדמות לוגי הרישום של המערכת).
- הכשרות - קיימות מעט מאוד הכשרות בתחום הגנה בסייבר של מערכות OT ואלו שקיימות מתקיימות בחול בלבד ועולות יותר משנה אקדמית (לקורס של חמישה ימים). כתוצאה מכך קשה לאתר גורמים בארץ הניתנים להגדרה כבעלי נסיון בתחום (אני בכוונה נמנע מהמילה "מומחה") וקיימים מקרים בהם ארגונים עשויים לקבל מענה אבטחתי המתאים לעולם ה-IT , לגורמי האיום והסיכונים הנלווים אליו ולא לאלו המתאימים לעולם ה-OT - וכן, הסיכונים שונים לחלוטין!
- תכנון ללא חשיבה אבטחתית - עולם הבקרה נולד שנים לפני עולם ה-WEB המוכר לכולנו. מדובר בתשתיות ורכיבים אשר לא תוכננו להתחבר לעולם החיצון, בעלי נגישות במערכות סגורות בלבד והחשיבה סביב תכנון לא כללה היבטי אבטחת מידע או הגנה בסייבר כלל שכן האיומים המוכרים לנו היום לא היו מוכרים או לחילופין לא היו רלוונטיים בעת תכנון מערכות הבקרה הוותיקות.

סיכום

מערכות ה-OT מלוות את חיינו לא פחות ממערכות הבנק או מערכות המידע אליהן אנו רגילים להתייחס בהיבטי אבטחת מידע. תהליכים תעשייתיים כוללים ייצור, בקרת תהליכים, ייצור חשמל, ייצור וזיקוק, הם עשויים להיות ציבוריים או פרטיים, והם כוללים טיפול במים והפצה, איסוף וטיפול בשפכים, צינורות נפט וגז, הולכה וחלוקה של חשמל המיוצר בתהליכים ושיטות שונות.

מדובר בשדות תעופה ואפילו אוניות וספנות, מעליות ומערכות מבנה, בהם נדרש לפקח ולשלוט על מערכות אקלים ומיזוג אוויר מערכות (HVAC) וצריכת אנרגיה. בתעשיות אלו נעשה שימוש יומיומי בתשתיות OT קריטיות אשר כל הפרעה לפעולתן עלולה לייצר תגובת שרשרת של פגיעה במוצר, פגיעה באספקה, שיבוש ועד פגיעה בנפש. כמעט בכולן יש בקר קטן בקצה המחליט האם המעלית תרד, האם חם מידי ויש לקרר, או האם פרצה שריפה ויש לקרוא באופן אוטומטי לכוחות הכיבוי.

אם הגעתם עד כאן, אתם יכולים כבר להבין כי המורכבות של התהליכים, נכסי ההגנה, האיומים והסיכונים בקצה, במרבית המקרים, שונים לחלוטין מעולם ה-IT וכל טעות או חדירה עלולה לייצר תגובה הרסנית.

אחד האתגרים המשמעותיים בתחום ה-OT, עליו אכתוב בהרחבה במאמר הבא, הינו אבטחת המערכות שכאמור לא תוכננו לכך מראש, עם "הפרעה" מינימלית לתהליכי יצור קריטיים ואפשור עבודה תקינה ורציפה לבעלי התפקידים הרלוונטיים.

אמינות מערכות SCADA בתשתית המודרנית שלנו עשויה להיות חיונית לביטחון ולבריאות הציבור. לפיכך, התקפות על מערכות אלו עשויות להיות קריטיות. תקיפה כזו כבר התרחשה, שבוצעה על מערכת בקרת שפכים של המועצה של מארוצ'י שייר בקווינס לנד, אוסטרליה. זמן קצר לאחר שהקבלן התקין מערכת SCADA בינואר 2000, רכיבי המערכת החלו לפעול באופן לא יציב. משאבות פעלו שלא צורך ומערכות ההתרעה לא פעלו באופן תקין. שפכים הציפו פארק סמוך וזיהמו תעלת ניקוז פתוחה של מי התהום. במערכת ה-SCADA כוונו שסתומי ביוב על פתיחה כאשר הפעולה המתוכננת הייתה אמורה לשמור אותם סגורים. בתחילה זה נראה כבאג במערכת אך מעקב אחר יומני המערכת גילה כי התקלות היו תוצאה של התקפות סייבר. החוקרים דיווחו על 46 מקרים נפרדים של התערבות חיצונית זדונית לפני שהאשם זוהה. ההתקפות בוצעו על ידי עובד לשעבר ממורמר של החברה אשר התקין את מערכת SCADA, תוך תקווה שישכרו את שרותיו כדי להגן על המערכת הפגועה.

חומר למחשבה בפעם הבאה שאתם פותחים את הברז והמים זורמים...

פרטים ליצירת קשר:

http://linkedin.com/in/gilad-zinger

http://twitter.com/GiladZinger

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-105 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 אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

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' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש אפריל 2019.

אפיק קסטיאל,

ניר אדר,

31.03.2019