Technologic Papers

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.

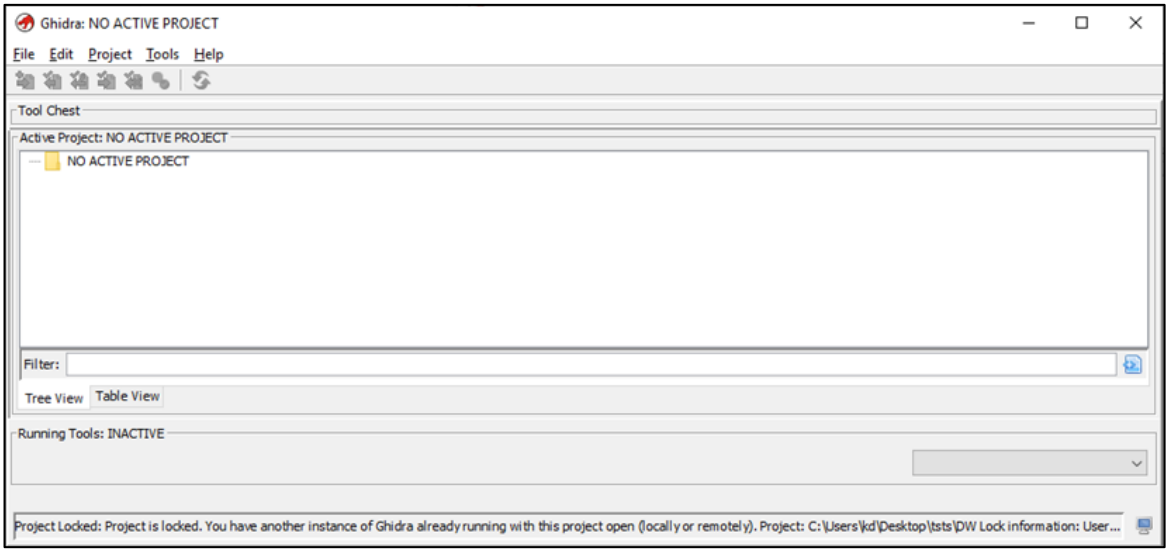
הכלי תומך במגוון רחב של Instruction Sets ומאפשר למשתמשים להוסיף Processors חדשים בקלות, דבר שכבר קרה, מספר פעמים, מהר מאוד עם שחרור הכלי. הכלי מאפשר סקריפטינג בשפות Java ו-Python וכמו כן במצב Headless CLI. עצם היותו כתוב ב-Java, שהיא שפה "ניידת", נתמך הכלי בכל מערכת הפעלה המאפשרת להריץ JVM.

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

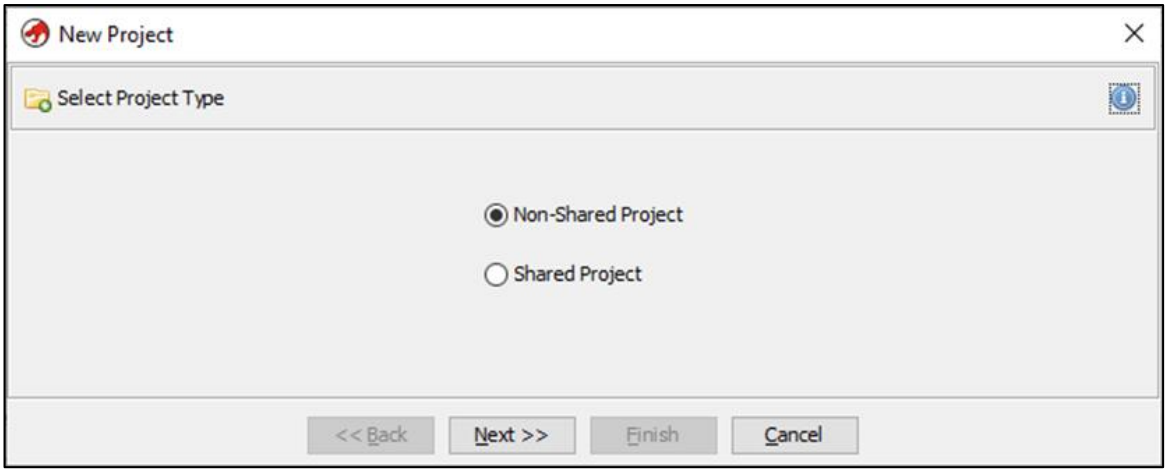
במאמר זה לא נעסוק ב-Reversing באופן כללי אלא נסקור את התכונות העיקריות וניתן טיפים כלליים לשימוש בכלי, הבה נתחיל.

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.

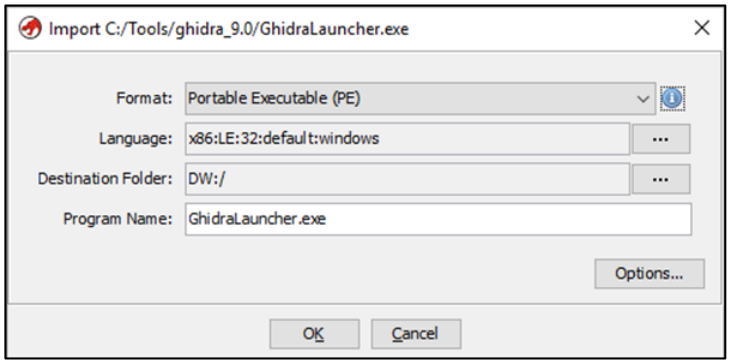
טעינת קבצים חדשים לפרויקט

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

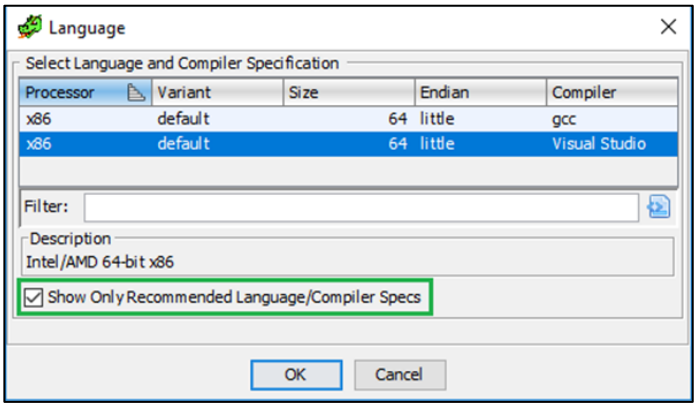
הקובץ ישירות לתוך החלון. נכון לרגע כתיבת המאמר, הכלי תומך בפורמטים הבאים Out-of-the-box (ללא פלאגינים חיצוניים):

- 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:

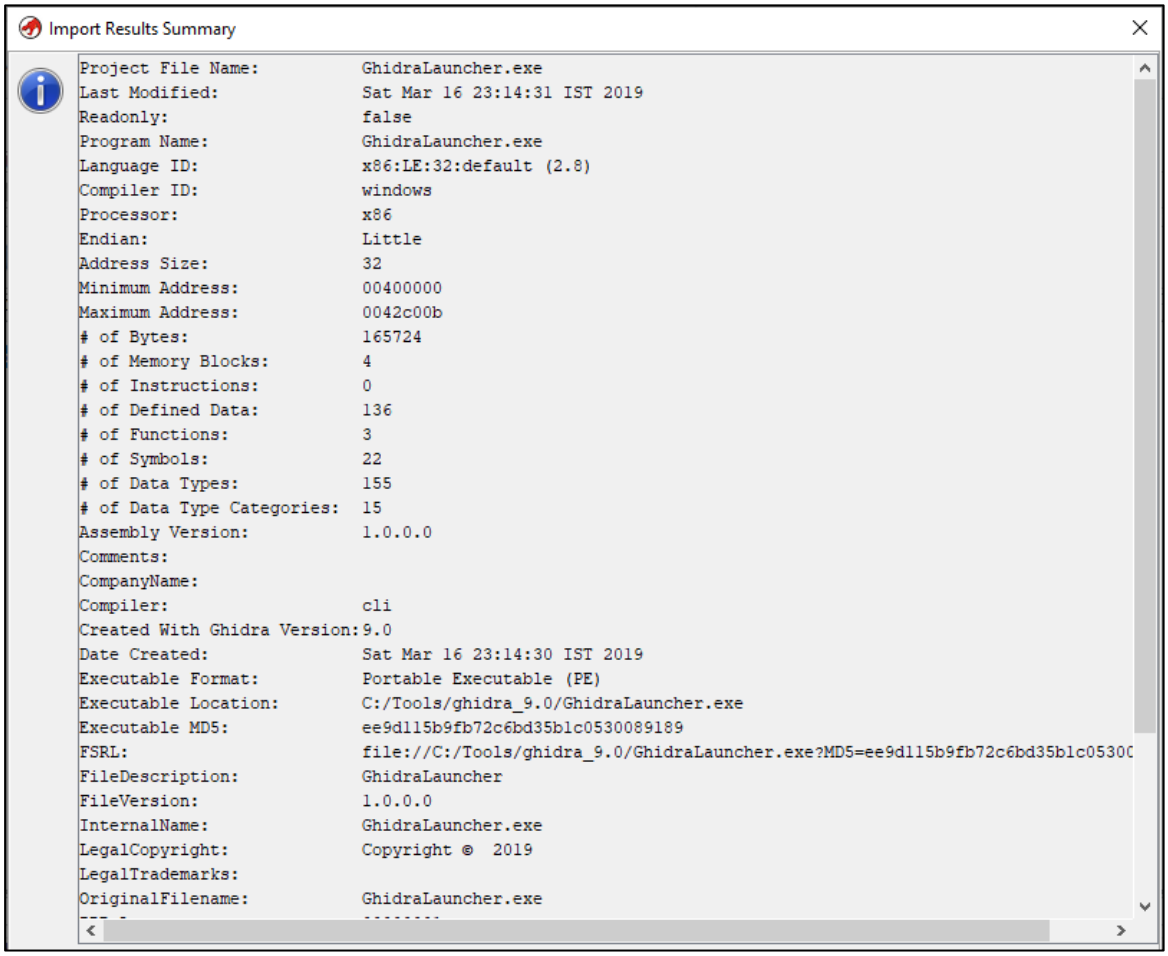


ניתן לראות ש-Ghidra כבר הבינה לבד באיזה קובץ מדובר (במידה וניתן לזיהוי) ואילו הגדרות להחיל עליו, אך במידה והנכם טוענים קובץ Raw מסוג כלשהו, למשל Firmware, או שצריך איזשהם כוונונים נוספים ניתן לעשות זאת על ידי לחיצה על כפתור שלוש נקודות ליד שדה ה-Language:

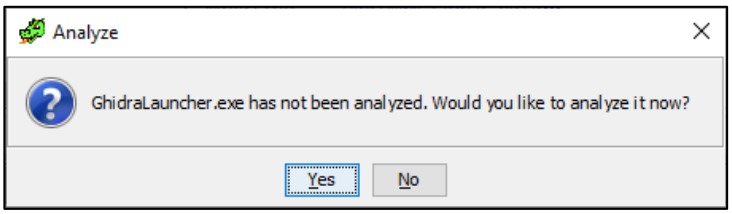


כאן ניתן לכוונן את בחירותיה של Ghidra לגבי הקובץ, כגון ארכיטקטורה (ביטול הסימון בעיגול הירוק יציג ארכיטקטורות נוספות אך לרוב לא תצטרכו לגעת בזה כאשר תתעסקו עם קבצים בעלי פורמט מוכר), סוג קומפיילר, סדר בתים (Endianness) וכדומה.

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

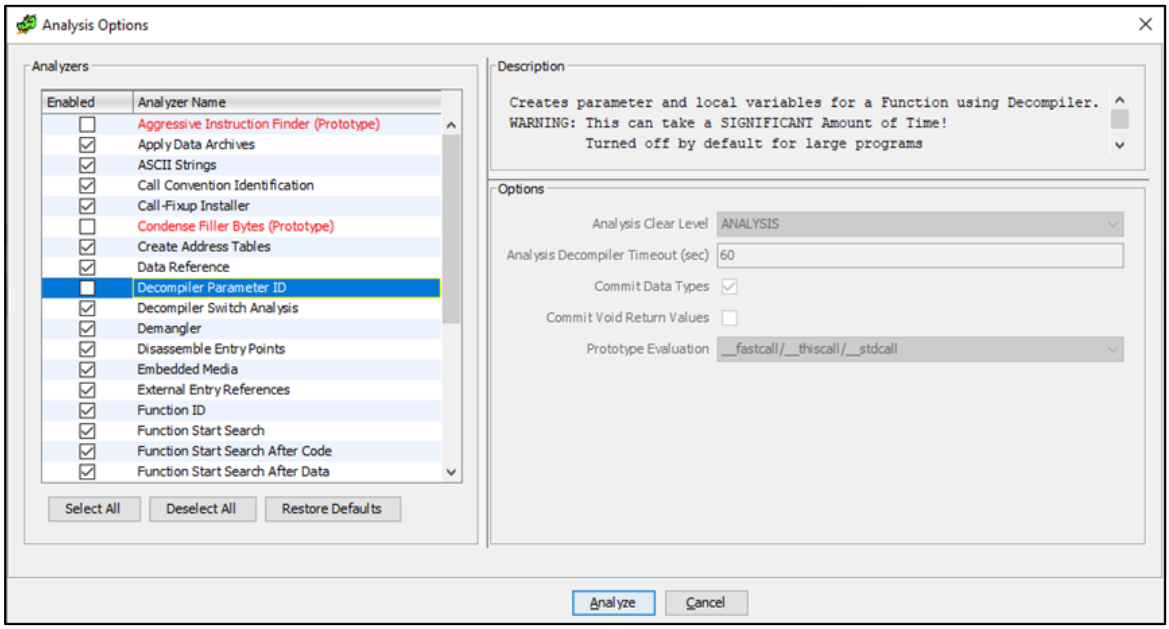


לחיצה כפולה על הקובץ שנטען או על הדרקון הירוק תפתח את מסך ה-CodeBrowser, מסך זה הינו המסך העיקרי איתו תעבדו בתוכנה ולמעשה משמש מין מסך ראשי לשאר המסכים. לאחר האנימציה המגניבה כמובן, נשאל אם ברצוננו להפעיל את ה-Auto Analysis כרגע:



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.

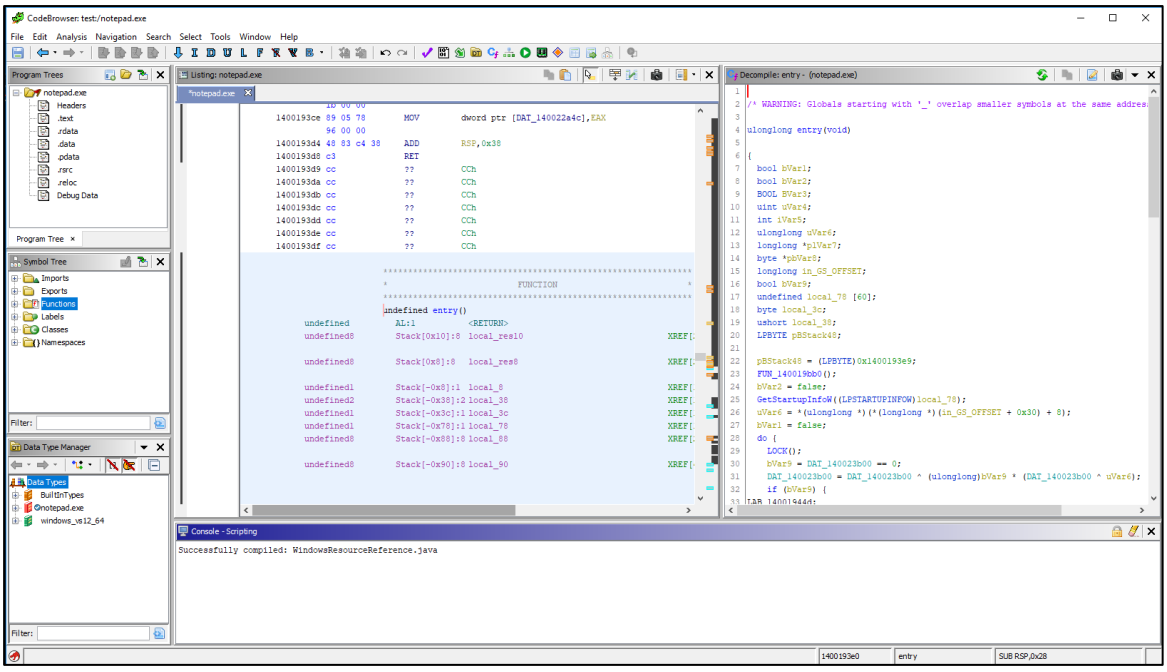
ניתוח בסיסי:

- 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

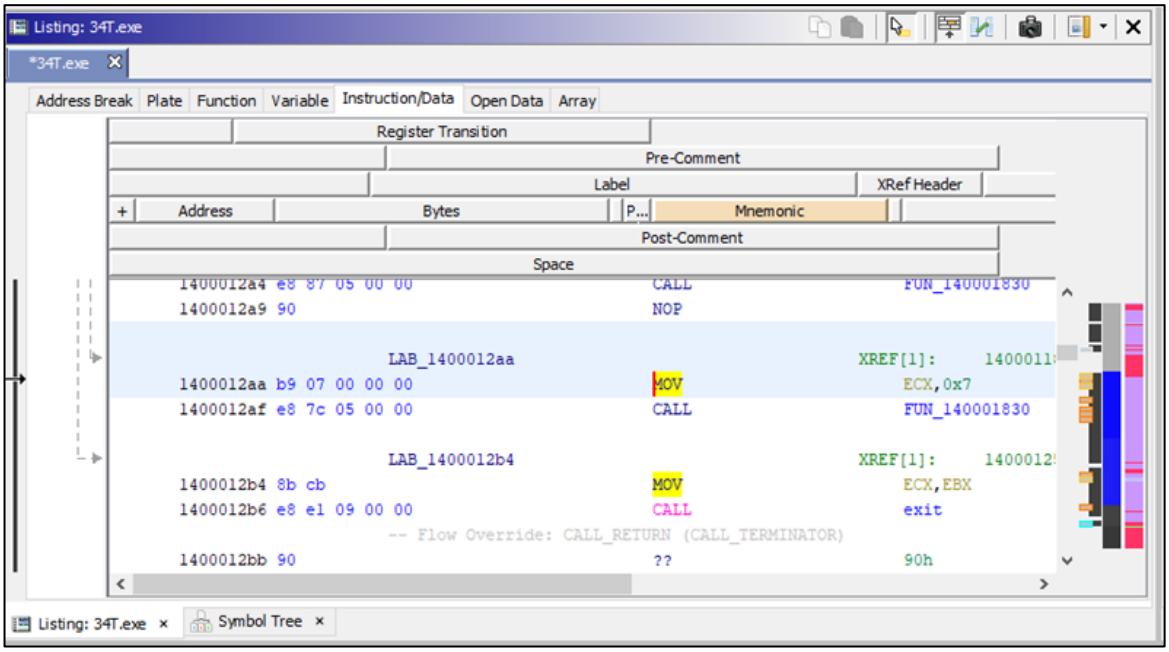
סביבת העבודה



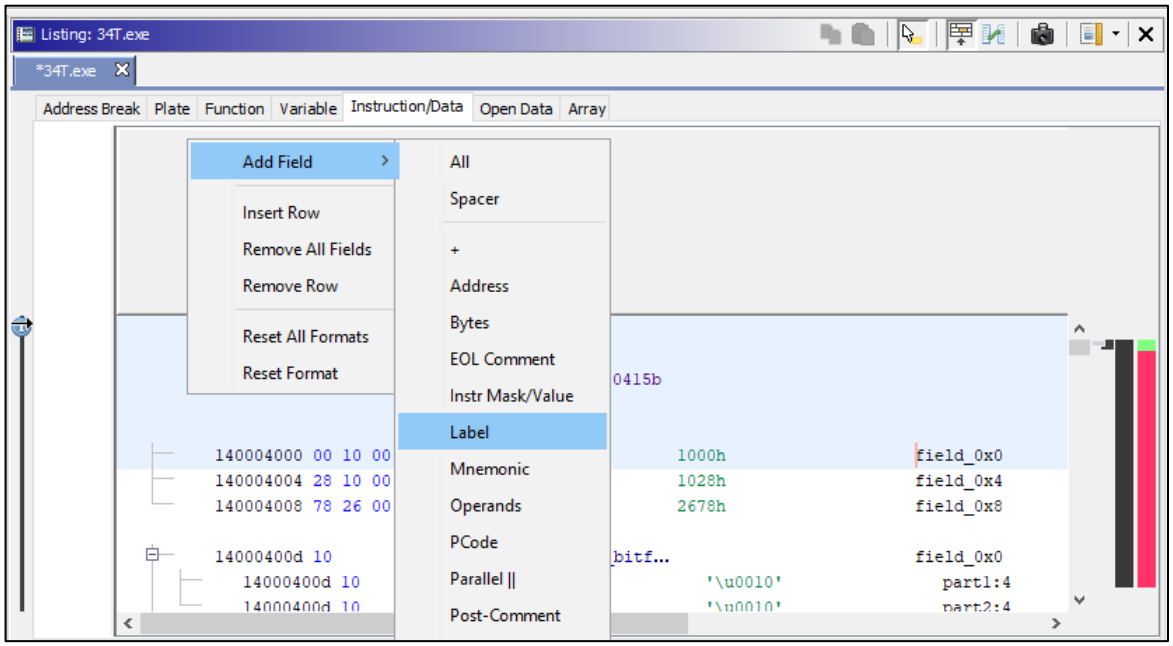
כך נראה החלון הראשי של Ghidra, שמו של חלון זה נקרא CodeBrowser. רוב הממשקים הויזואלים קיימים בתוך חלון זה, למשל: התפריטים הראשיים, חלונות ה-Listing, Decompiler, Graph, מנהל הטיפוסים, חיפוש מחרוזות, עורך Hex ועוד. רוב עבודת הניתוח תתבצע תחת חלון זה. הממשק מחולק לחלונות שאותם שניתן להזיז ולשנות. במהלך 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

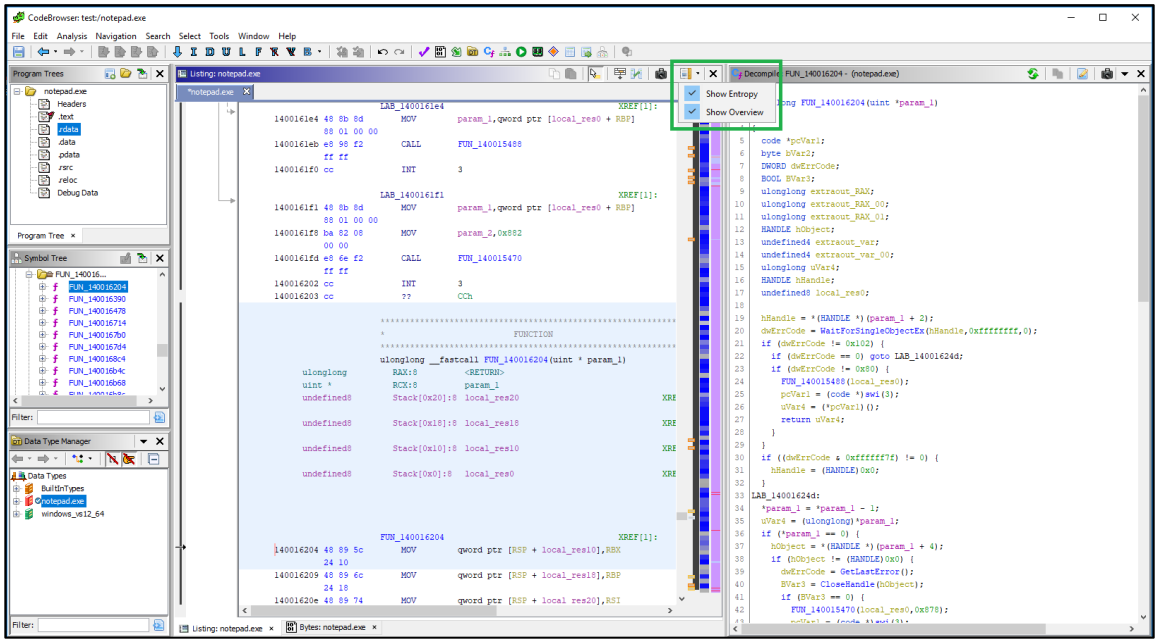
חלון זה הוא למעשה המקביל ל-IDA View-A במצב Flat ובדומה לו הוא מכיל ניתוח Disassembly, Data ואפילו דברים נוספים (בין היתר, אף הצגת מדיה). לחיצה על כפתור ה-  Edit תאפשר התאמה אישית של השדות בחלון:



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



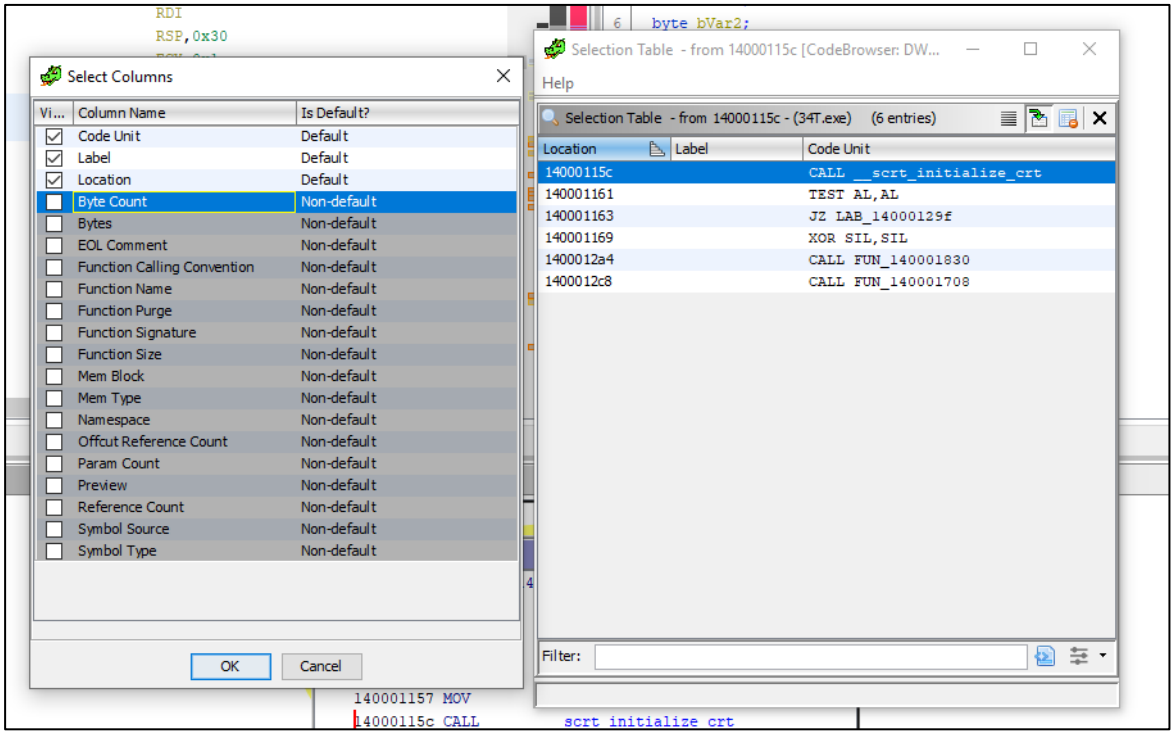
חלון ה-Listing מאפשר לקבל תצוגה ויזואלית של האנטרופיה בקובץ הנוכחי ו-Overview כללי של מפת הקובץ בדומה לפס העליון ב-IDA:



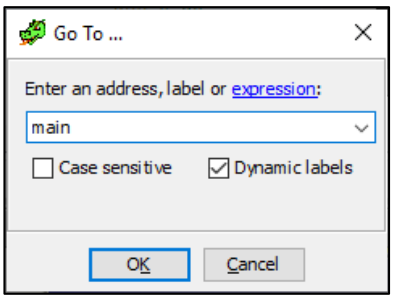
הכלי מאפשר לייצר טבלאות כמעט מכל אלמנט בתוכנה, על מנת להפעיל את הפיצ'ר סמנו את האלמנטים הרצויים ואז לחצו על:

Select -> Create Table From Selection

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



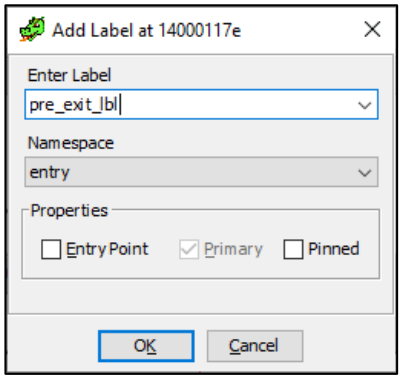
לאחר טעינת המידע הרצוי לטבלה ניתן לבצע עליה חיתוכים מתקדמים. על מנת "לקפוץ" אל Labels או כתובות ניתן ללחוץ על המקש 'G' בדומה ל-IDA:



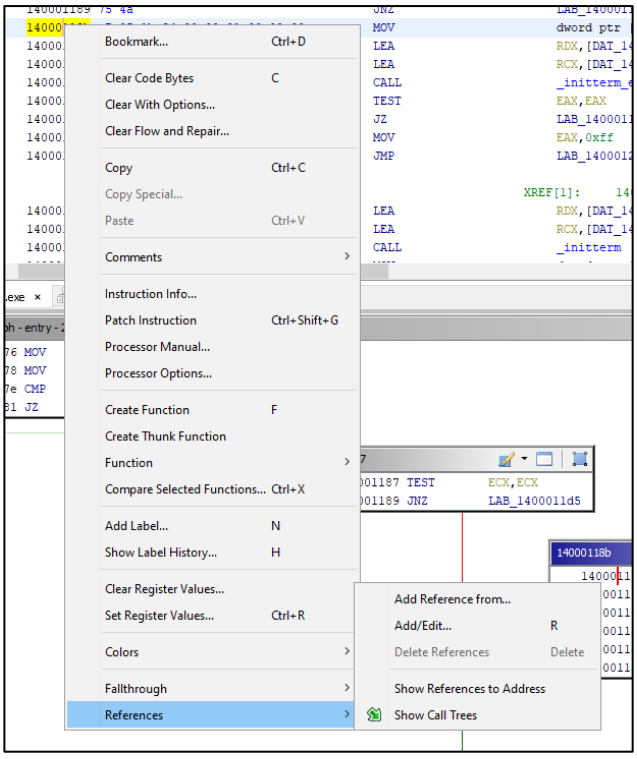
את כל קיצורי המקשים ניתן לשנות בנתיב ההגדרות הבא:

Edit -> Tool Options -> Key Bindings

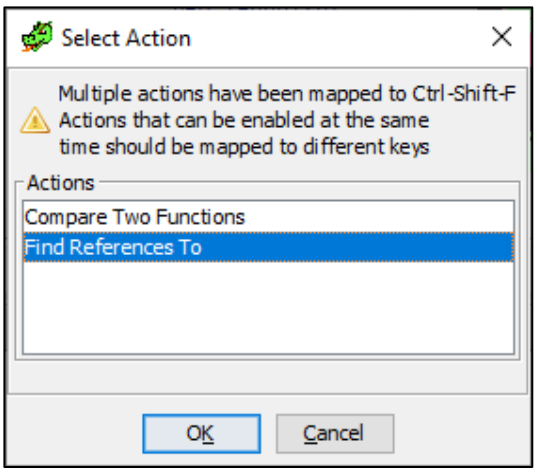
כדי לתת שם לפונקציה, Label, משתנה, או כל דבר אחר שאפשר לתת לו שם, לחצו על המקש 'L':



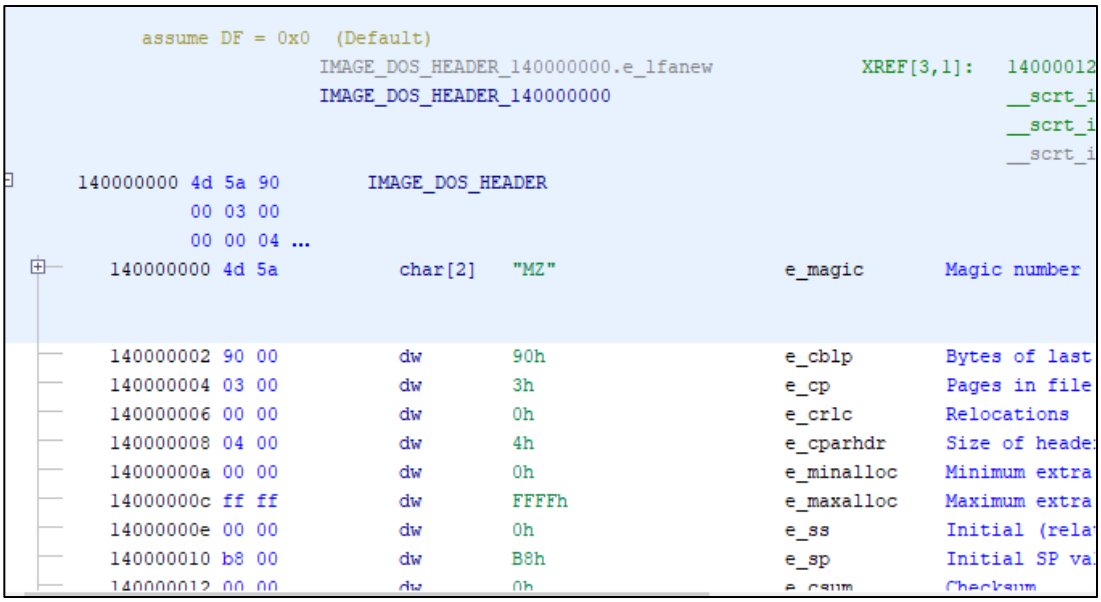
על מנת לחפש Cross References ניתן להשתמש בעכבר על ידי לחיצה ימנית על האובייקט הרצוי:



או על ידי קיצור במקלדת CTRL + SHIFT + F:

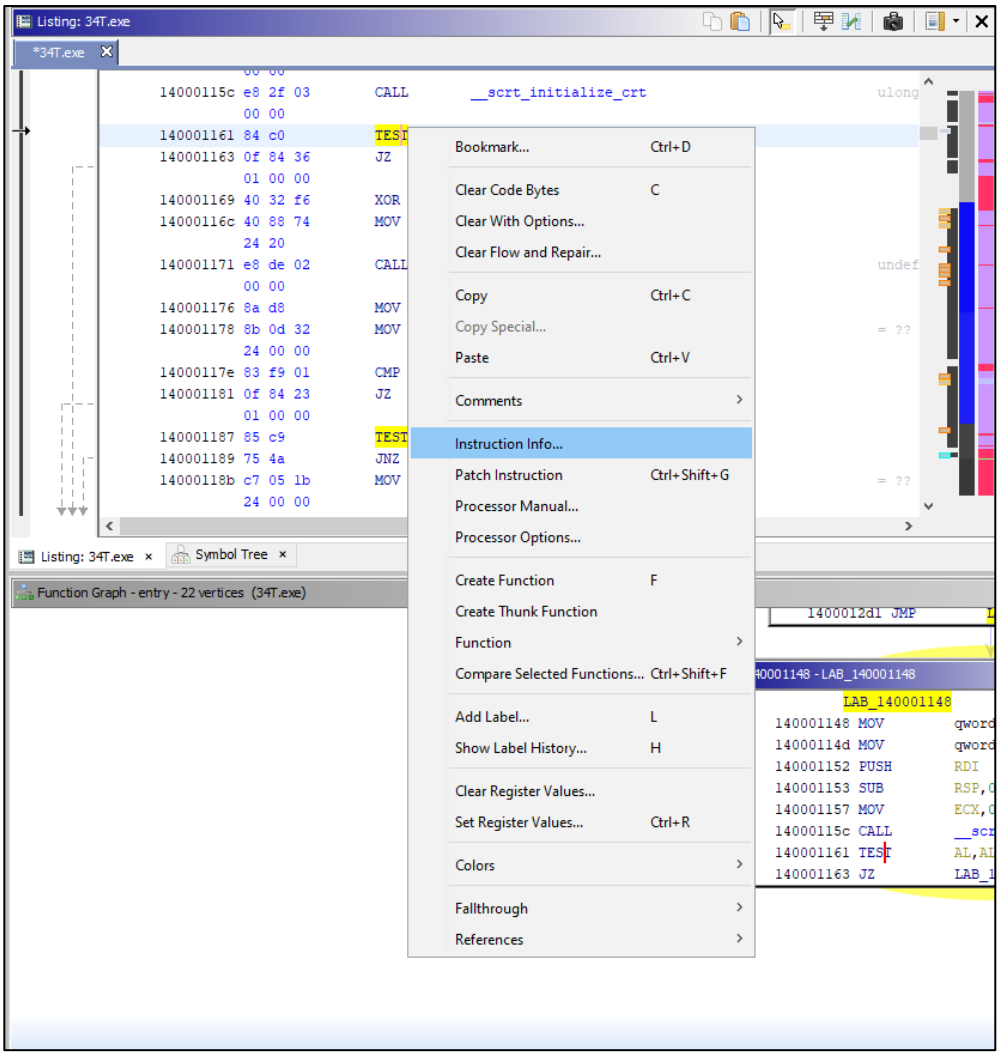


דבר נוסף ששווה להזכיר בהקשר של חלון ה-Listing הוא שהתוכנה מפרסרת באופן אוטומטי את הפורמט של הקובץ שנטען ומציגה את ה-Headers בחלון ה-Listing:

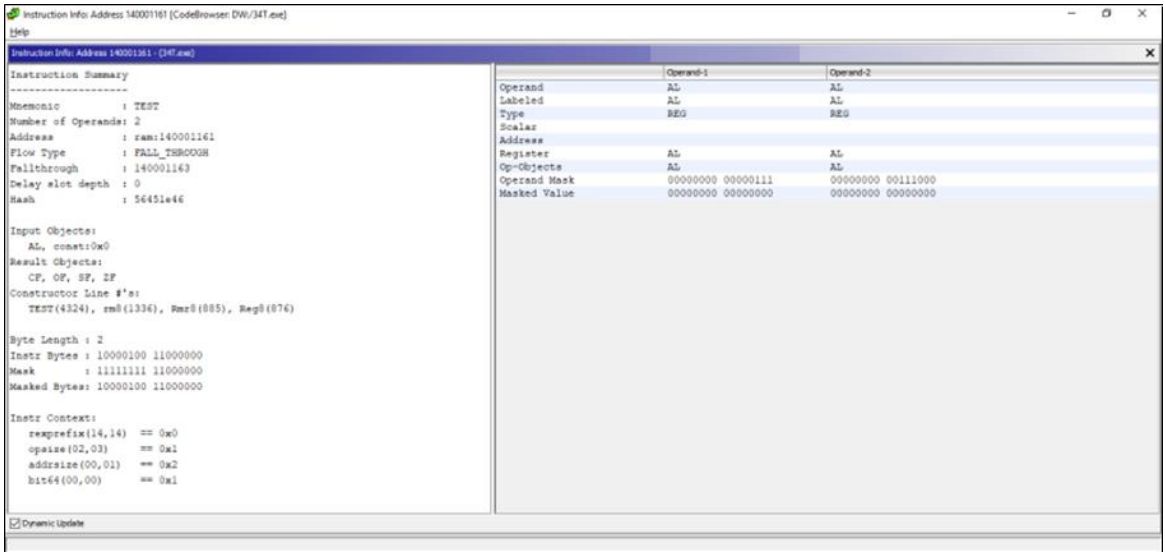


דבר שבהחלט יכול להיות שימושי.

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



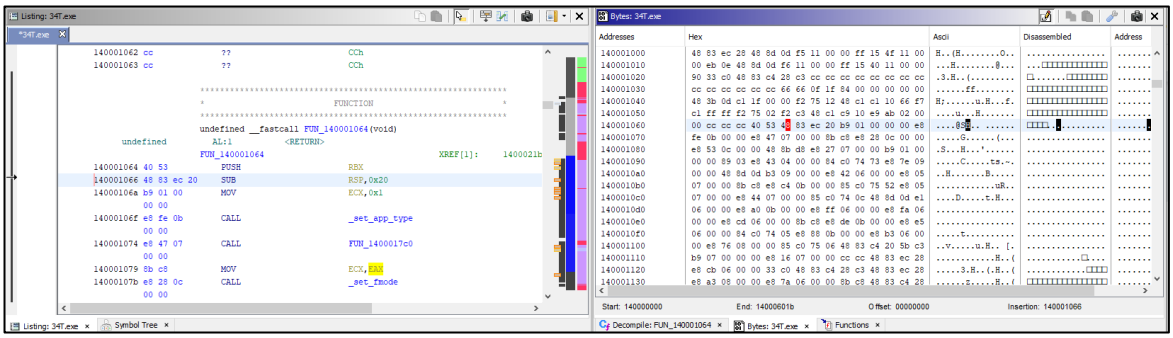
מיד ייפתח החלון החביב הבא אשר יציג פירוט שימושי מאוד אודות הפקודה:



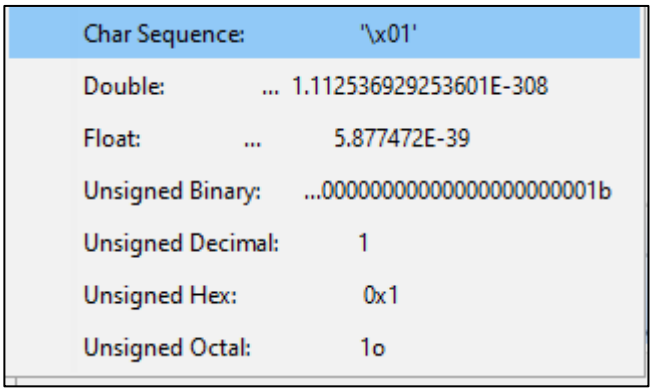
ניתן לערוך פקודות מעבד גם בחלון ה-Listing ובחלונות נוספים על ידי לחיצה ימנית בעכבר על הפקודה הרצויה ובחירה ב-Patch Instruction (או בקיצור המקשים CTRL + SHIFT + G):

```
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: 

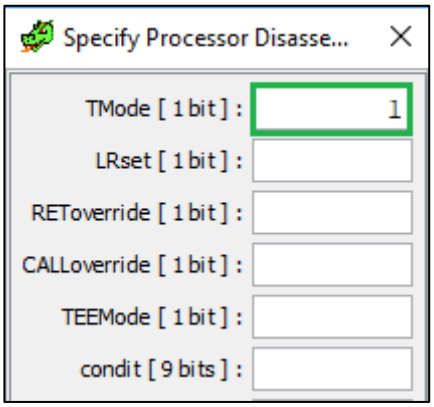


סימון ערך כלשהו בחלון ה-Listing ולחיצה על Convert בתפריט הלחצן הימני תציג בפניכם את החלון הבא:



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

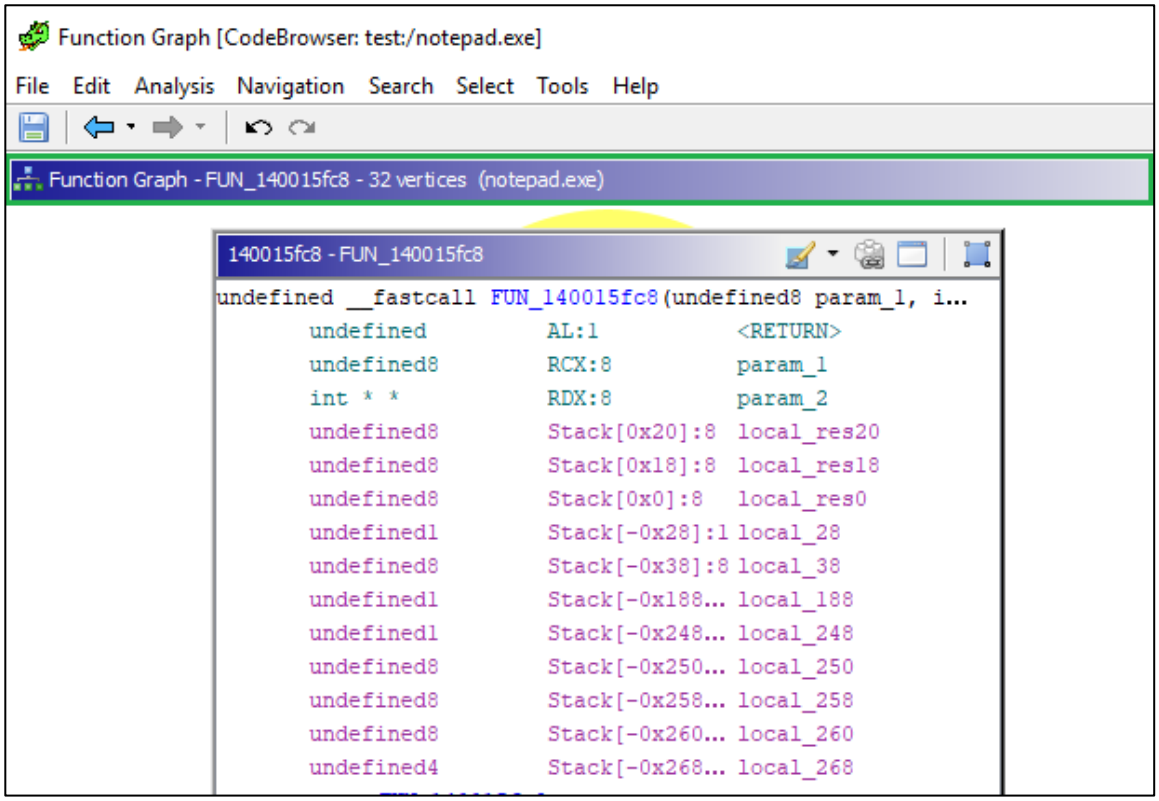
קיימים דברים פחות טריוויאלים בתוכנה כמו למשל איך לאכוף Disassembly במצב Thumb כאשר מתעסקים עם קבצי ARM. על מנת לבצע פעולה זו יש לבחור ב-"Processor Disassembly Options" בתפריט הלחצן הימני ואז להחיל את השינוי הבא:



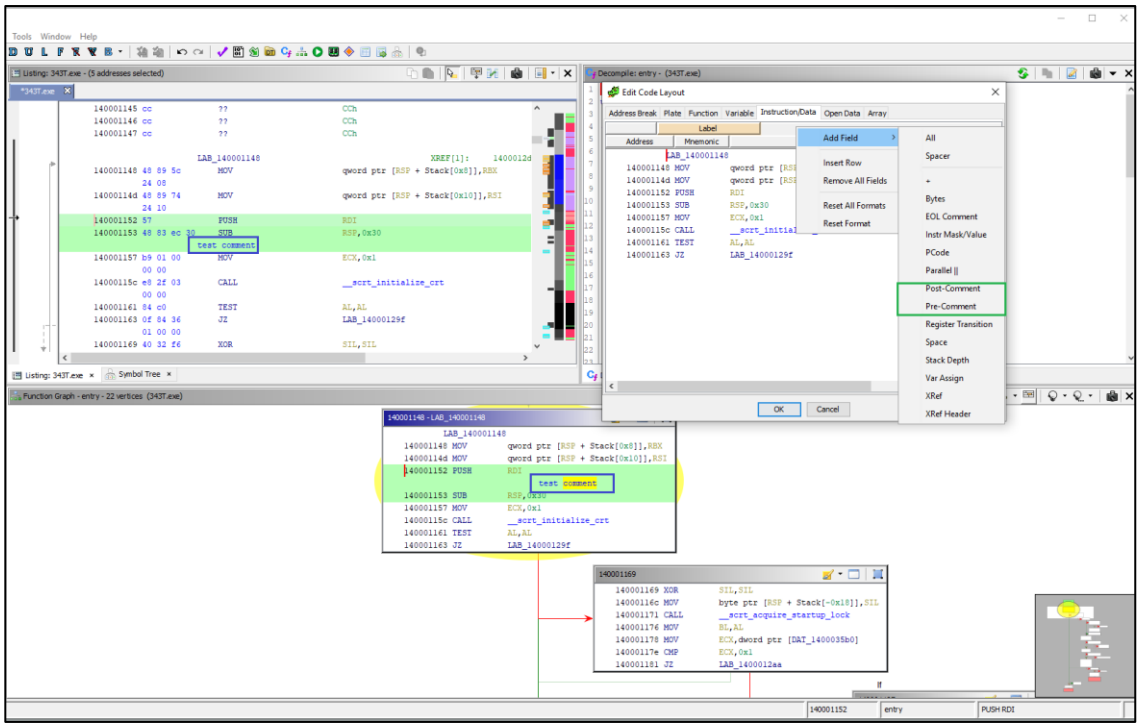
כפי שוודאי הבחנתם, תפריט הלחצן הימני מלא בפיצ'רים שימושיים. ככלל ב-Ghidra, רצוי להשתמש בלחצן הימני כאשר אתם מחפשים משהו, התשובה עשויה להיות שם.

חלון ה-Graph

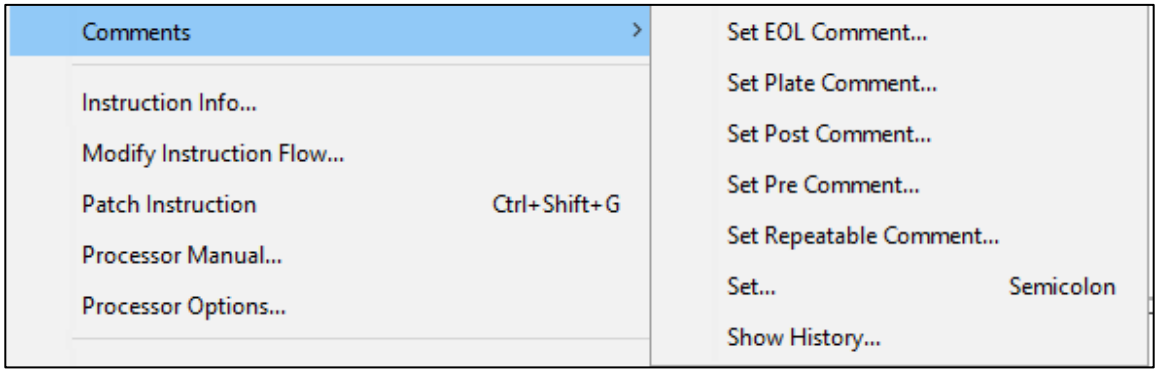
החלון אינו פעיל כברירת מחדל, ניתן לפתוח אותו על ידי לחיצה על  או בתפריט Window ואז לחיצה על Function Graph. החלון יפתח בנפרד, אך ניתן להצמיד אותו לחלון הראשי. יש לגרור את החלון הפנימי (עם הפס הכחול) ולא את המעטפת:



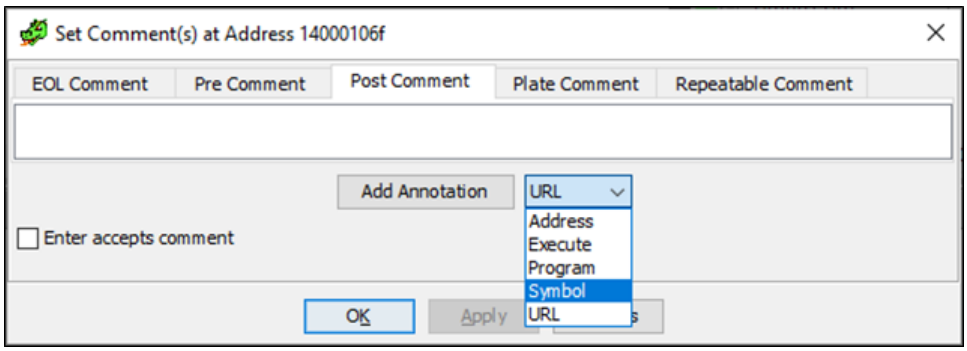
דבר נוסף שלא קיים כברירת מחדל הינו הצגת הערות בחלון ה-Graph. כפי שהוסבר לעיל, ניתן לערוך גם את חלון הגרף באמצעות הכפתור  , נלחץ על הכפתור ונבצע את הפעולות הבאות:



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

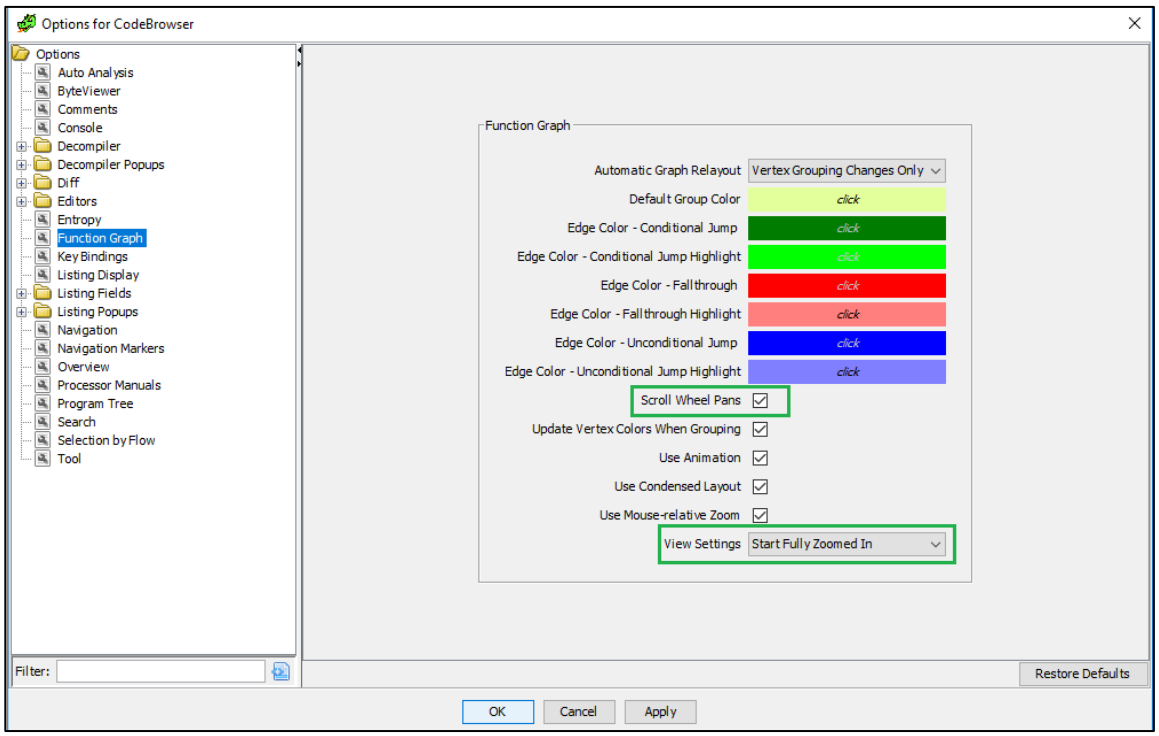


פיצ'ר חשוב נוסף בעניין זה הוא שניתן להוסיף Symbol להערות על ידי:



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

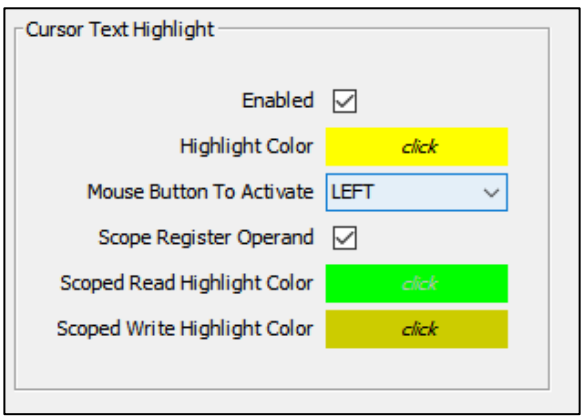
Edit -> Tool Options -> Function Graph



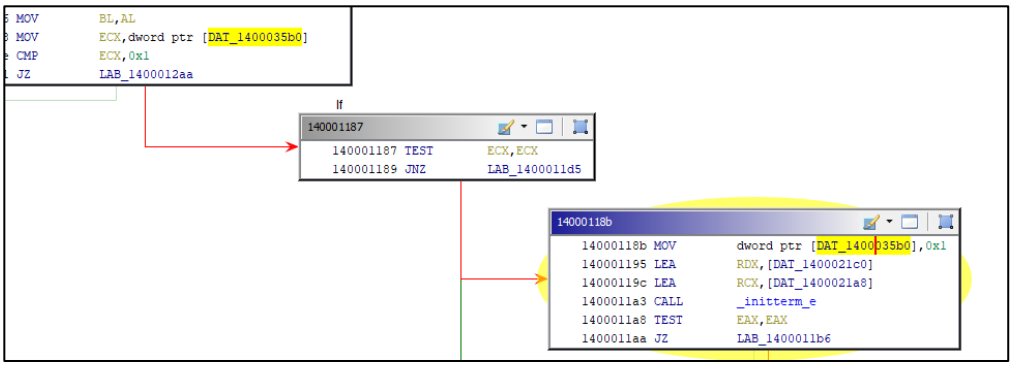
כפי שניתן להסיק לבד, שינוי ההגדרות הנ"ל יאפשר גלילה בגרף באמצעות הגלגלת בעכבר והגרף ייפתח במצב Zoomed in כברירת מחדל.

כמו כן, על מנת לבצע Highlight למשתנה, פקודה, קבוע או אוגר 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:

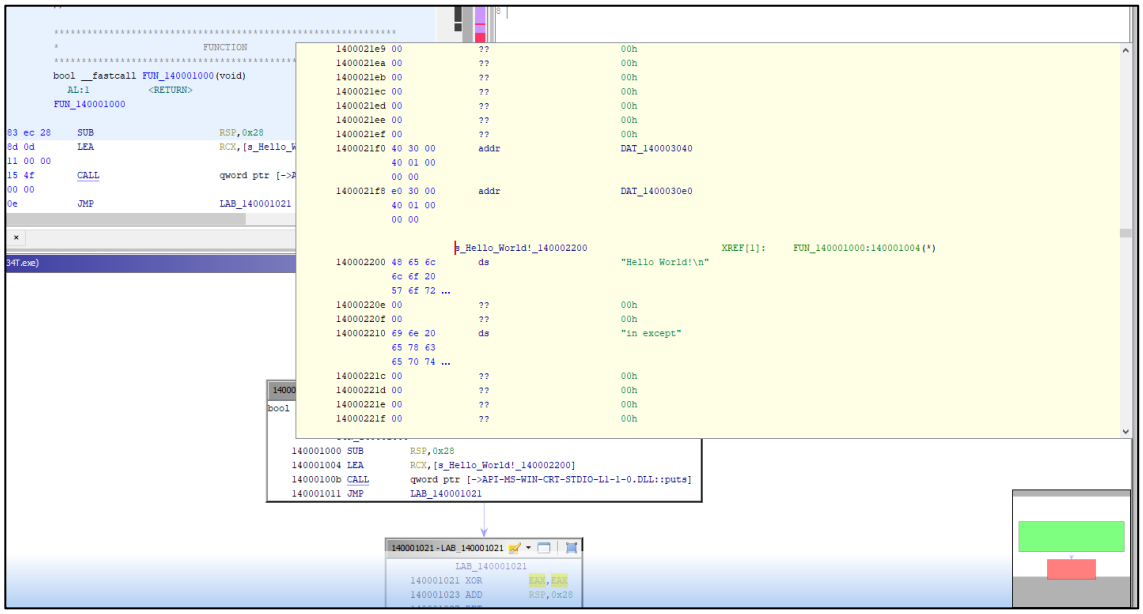
Edit -> Tool Options -> Listing Fields -> Cursor Text Highlight



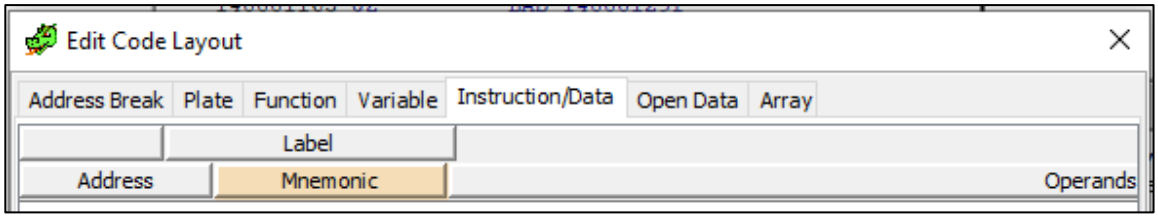
כעת התוכנה תמרקר לנו מופעים נוספים של האובייקט המסומן:



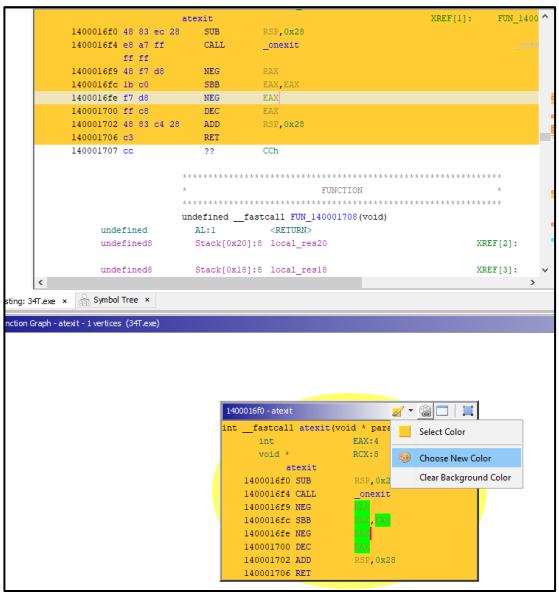
בדומה ל-IDA, כאשר תעמדו עם העכבר על אובייקט שמוביל למיקום אחר בגרף או ב-Listing תוצג בפניכם תצוגה מקדימה שנראת כך:



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:



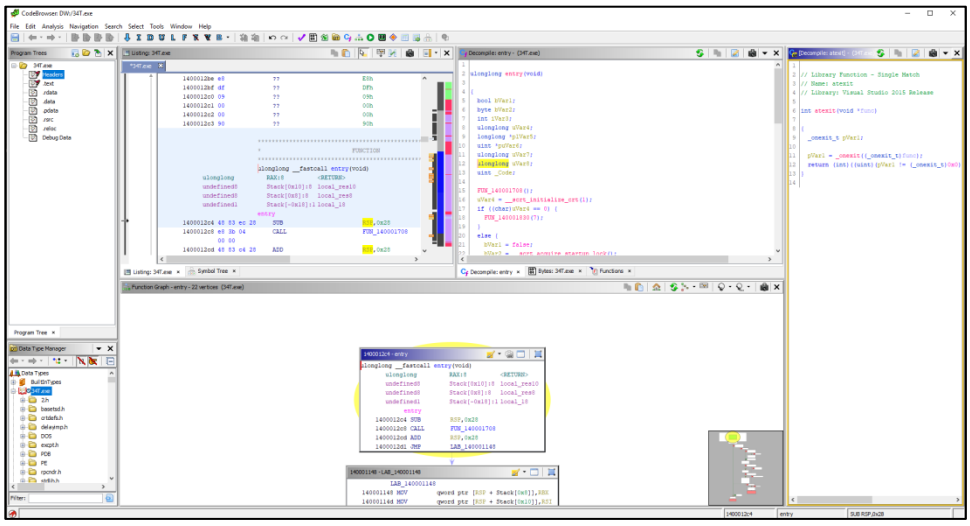
עוד פיצ'ר שימושי בגרף הוא צביעת Code blocks מסוימים בצבעים, ניתן לעשות זאת על ידי לחיצה על הכפתור  :



כפי שניתן לראות, הגרף מסתנכרן 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. 

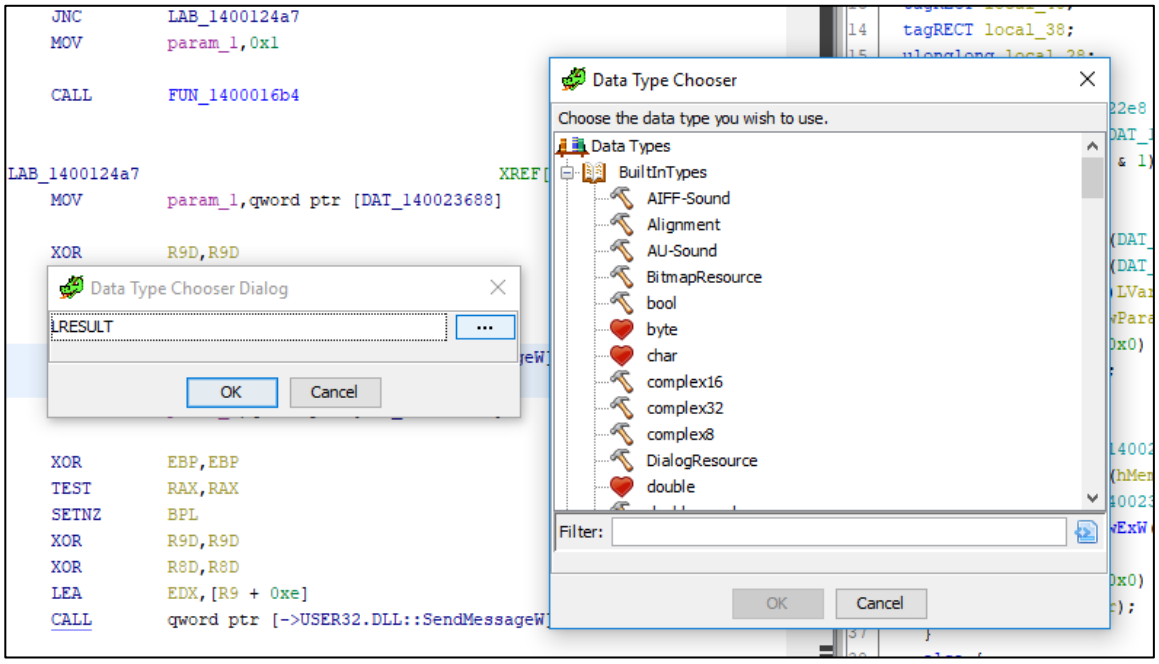
כשתיצרו חלון העתק (Snapshot) חדש הוא בעצם יהיה מנותק מהחלונות האחרים מבחינת הסנכרון שהתוכנה מבצעת:



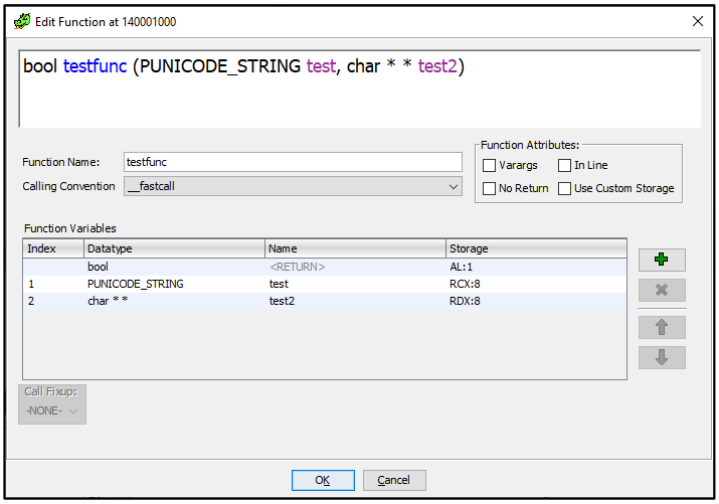
חלון ה-Decompiler

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:



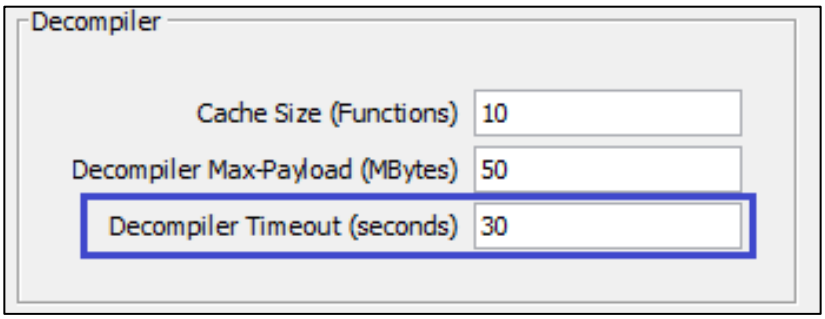
נסתכל על התפריט העליון בחלון ה-Decompiler:

1. כדי לבקש מה-Decompiler לבצע מחדש את הניתוח, נלחץ על הכפתור המוקף בצהוב.
2. כדי לייצא את הקוד לקובץ נלחץ על הכפתור המוקף בכחול.
3. כדי לבטל את הסנכרון בין החלונות בעותק חדש, נלחץ על הכפתור 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 }
```

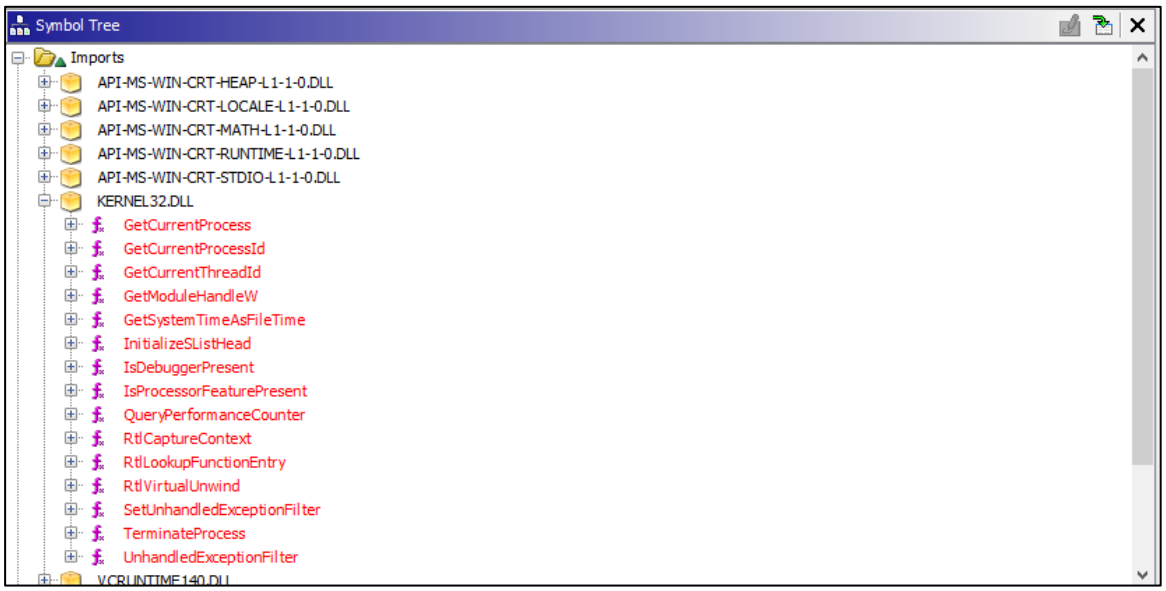
לאחר שינוי ההרשאות:

```
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 }
```

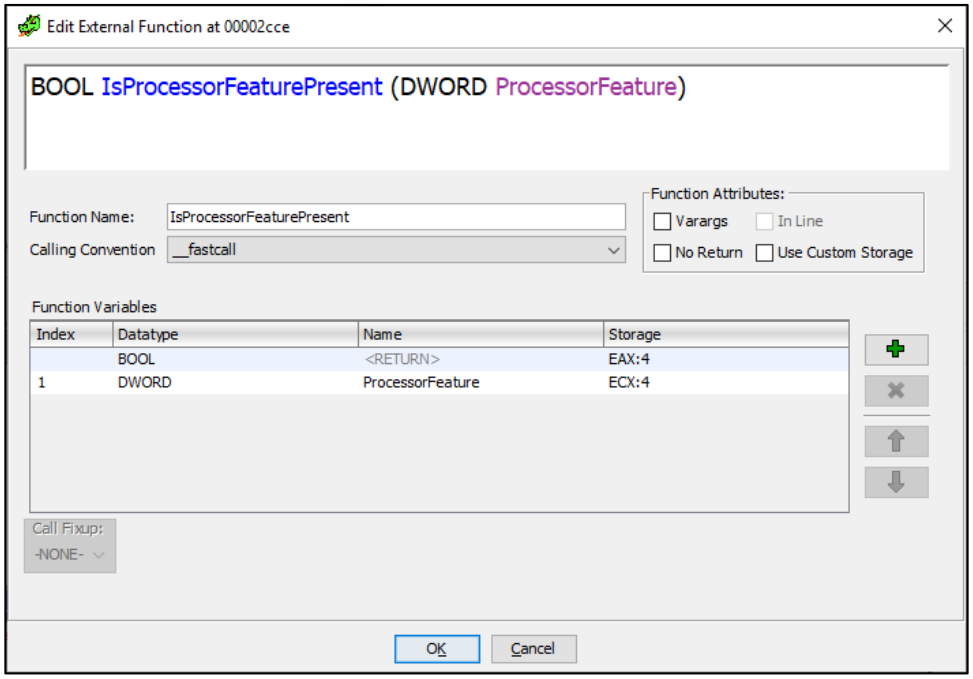
חלון ה-Symbols Tree

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.

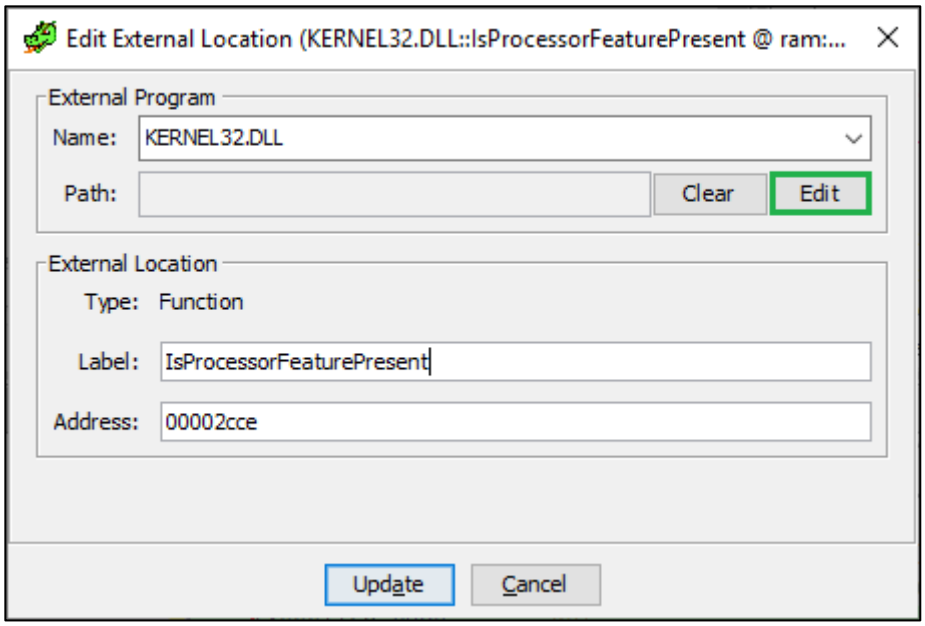
בתמונה הבאה ניתן לראות את הפונקציות שהתוכנית מיבאת מ-kernel32.dll:



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:

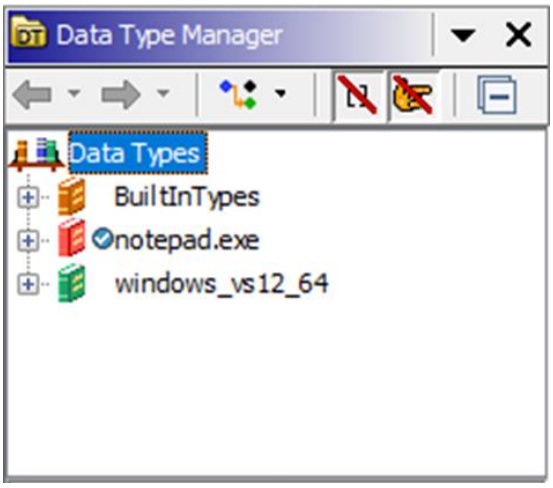


פיצ'ר מעניין נוסף הוא היכולת לחבר פריט מתוך ה-Symbols Tree ל-Imported File (כפי שכבר צוין, ניתן לטעון יותר מקובץ אחד לתוך ה-Code Browser) כך שיהיה חיבור בין הכתובות, לדוגמה, אנו יודעים שהפונקציה IsProcessorFeaturePresent מגיעה מ-KERNEL32.DLL, נוכל לטעון את הקובץ ואז להכנס לתפריט הלחצן הימני של הפונקציה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



חלון ה-Data Type Manager

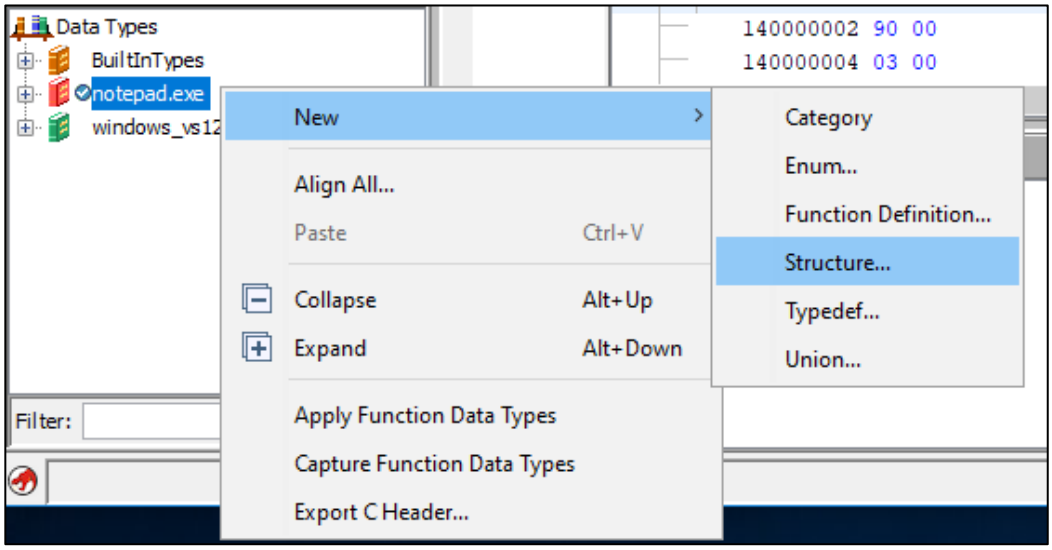
החלון מאפשר ניהול של טיפוסים והגדרות פונקציות, כברירת מחדל ממוקם החלון בצד השמאלי-התחתון של המסך, כך הוא נראה:



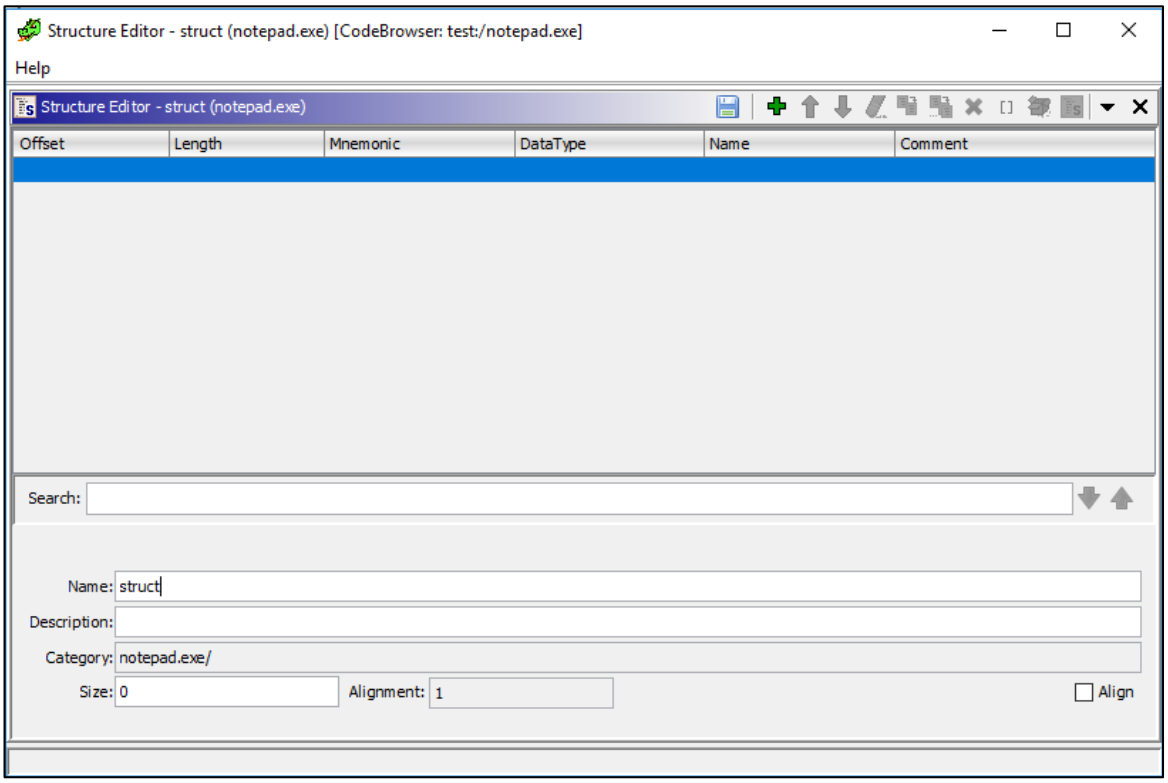
בחלון זה ישנם כמה תיקיות:

- 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.

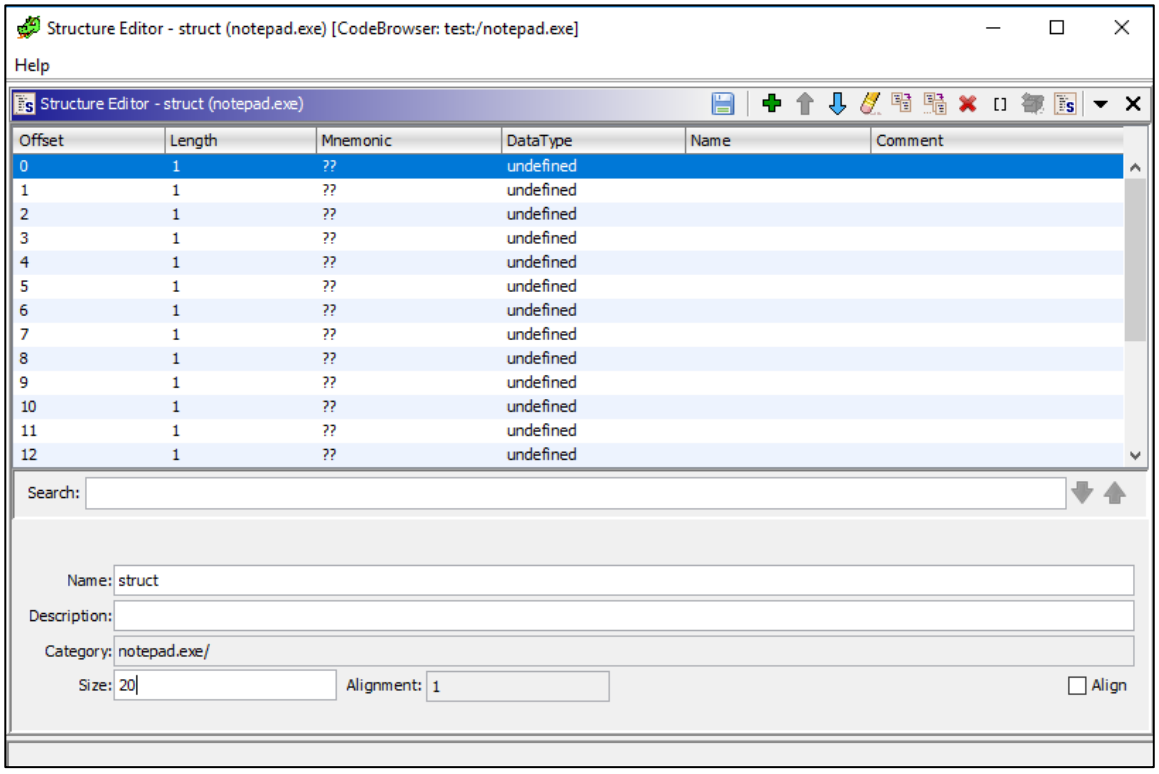
על מנת ליצור טיפוס חדש, נלחץ על תיקיית ה-Module שלנו, נבחר ב-New ולאחר מכן נבחר בטיפוס רצוי. לדוגמה, כך יוצרים מבנה חדש מסוג Structure:



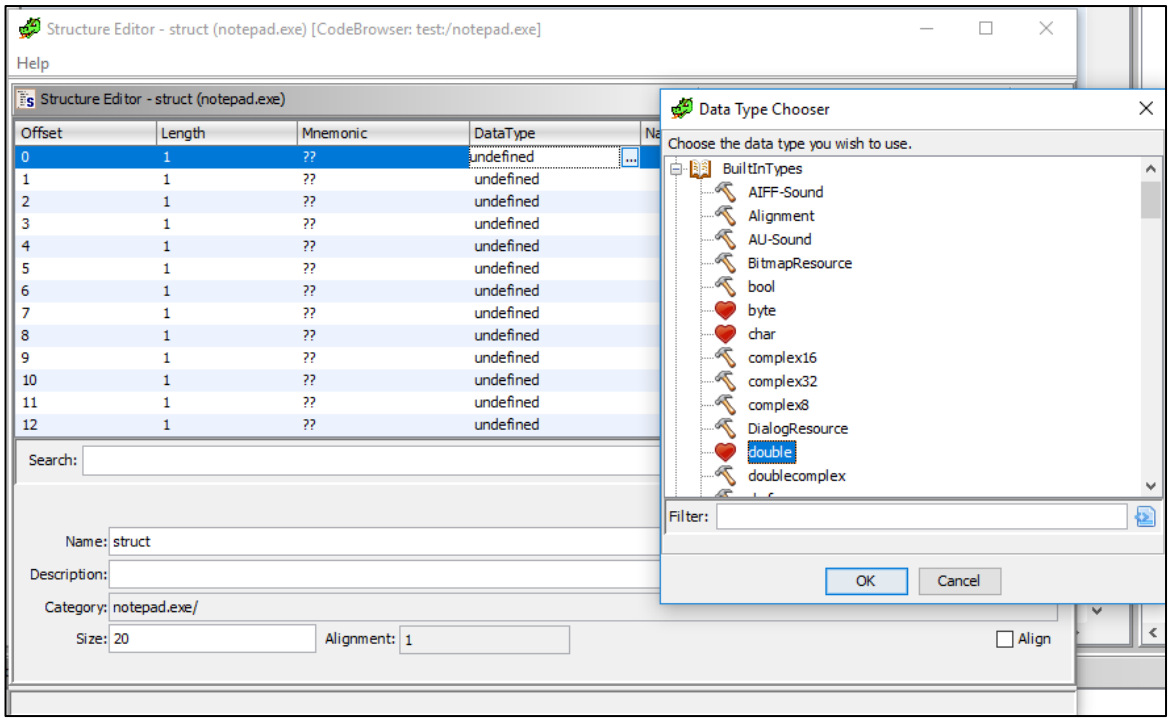
לאחר מכן יפתח לנו החלון הבא:



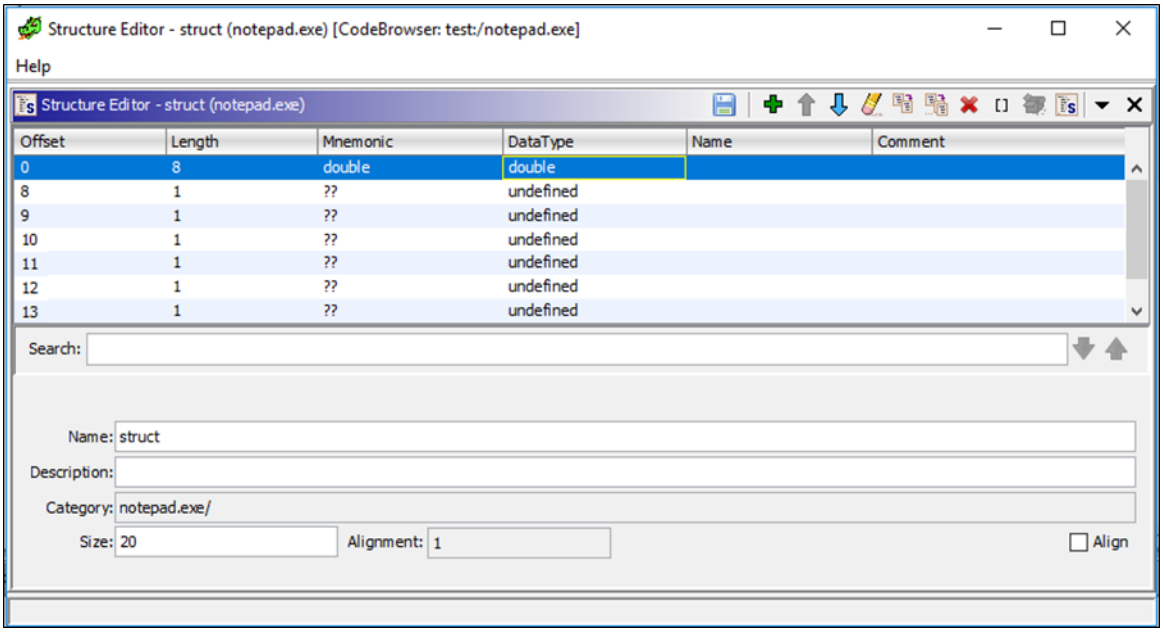
עלינו לבחור שם למבנה החדש, באפשרותינו לתת לו גם תיאור. בנוסף, אם אנו יודעים את גודל המבנה, נוכל להגדיר גם אותו, אחרת, נצטרך ללחוץ על סימן הפלוס למעלה כדי להוסיף שדה חדש. כאשר נגדיר גודל כללי למבנה, העורך יצור עבורנו אוטומטית ערכים ע"פ הגודל שסופק:



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:

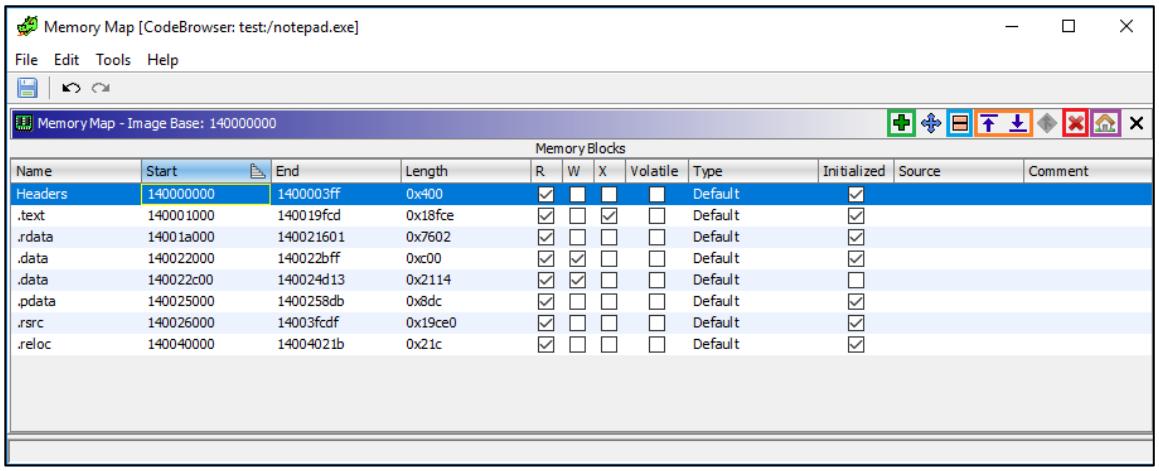
File -> Parse C Source

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



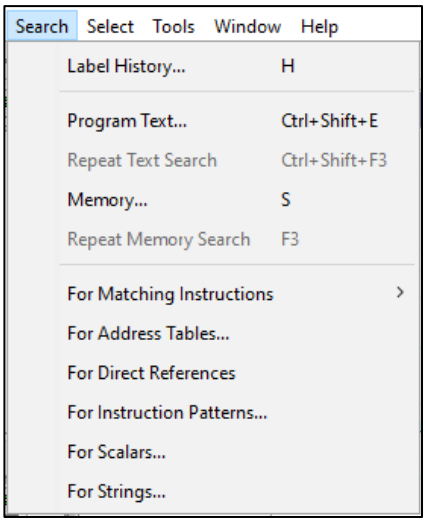
Name	Start	End	Length	R	W	X	Volatile	Type	Initialized	Source	Comment
Headers	140000000	1400003ff	0x400	☑	☐	☐	☐	Default	☑		
.text	140001000	140019fcd	0x18fce	☑	☐	☑	☐	Default	☑		
.rdata	14001a000	140021601	0x7602	☑	☐	☐	☐	Default	☑		
.data	140022000	140022bff	0xc00	☑	☑	☐	☐	Default	☑		
.data	140022c00	140024d13	0x2114	☑	☑	☐	☐	Default	☐		
.pdata	140025000	1400258db	0x8dc	☑	☐	☐	☐	Default	☑		
.rsrc	140026000	14003fcdf	0x19ce0	☑	☐	☐	☐	Default	☑		
.reloc	140040000	14004021b	0x21c	☑	☐	☐	☐	Default	☑		

- 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:

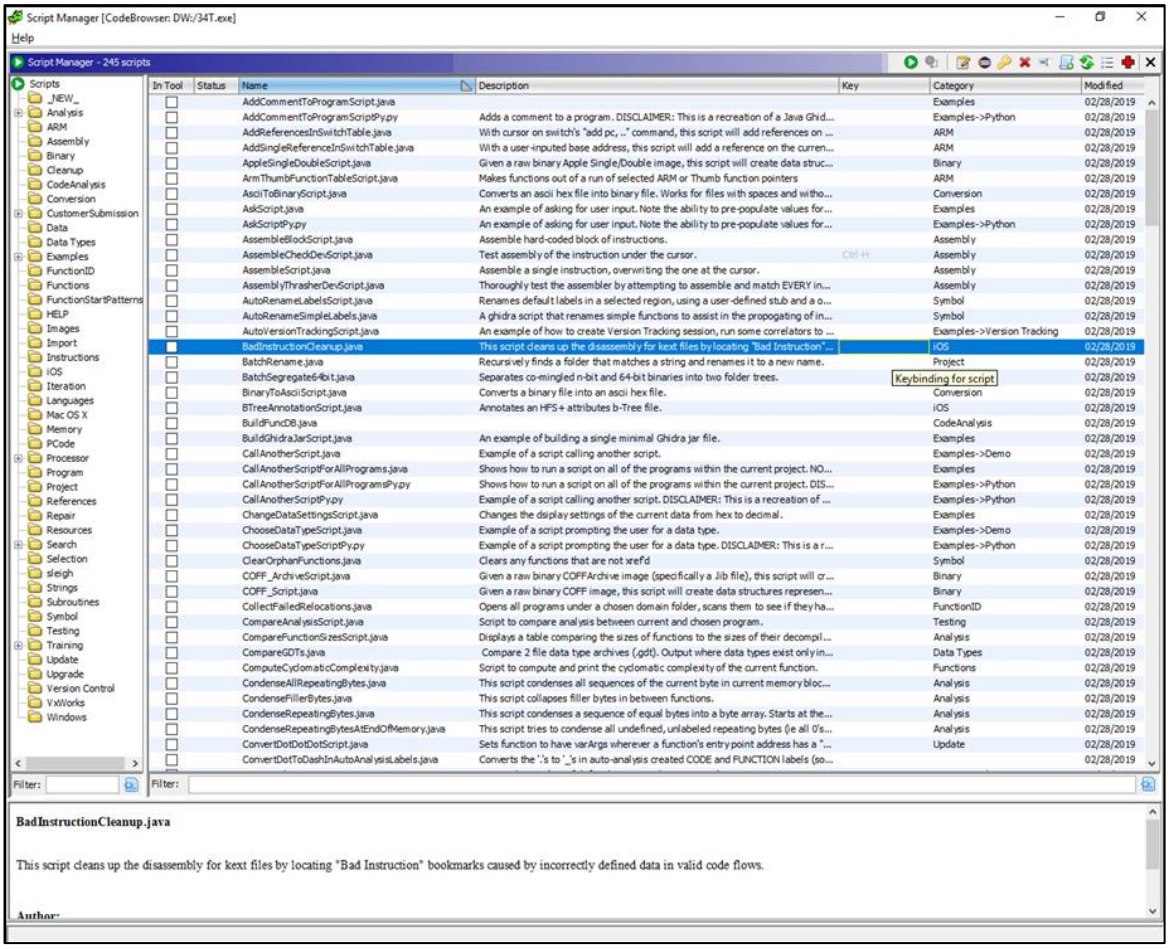


- חיפוש Program Text - מאפשר חיפוש של כל טקסט בחלונות שנוסף על ידי Ghidra או על ידי המשתמש, כולל: שדות, הערות, labels, רפרנסים, קוד ועוד.
- ניתן להפעיל את החיפוש על כלל התוכנית או על חלקים שנבחרו. בחירה ב-Search All Instances יוצרת טבלה עם כל תוצאות החיפוש.
- חיפוש Memory - מאפשר למשתמש לחפש סט של בתים 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".

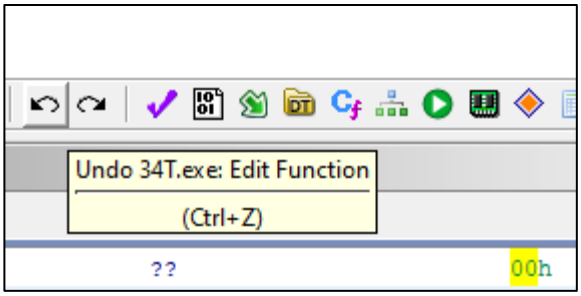
מספר אובייקטים שימושיים:

- 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.

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.

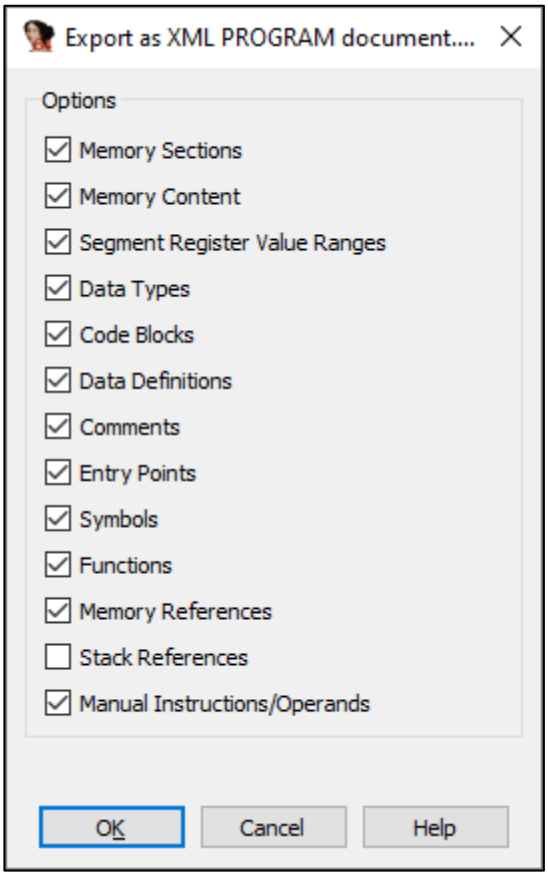
קובץ זה וקבצים נוספים, קיימים גם בעותק המקומי של התקנת התוכנה.

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