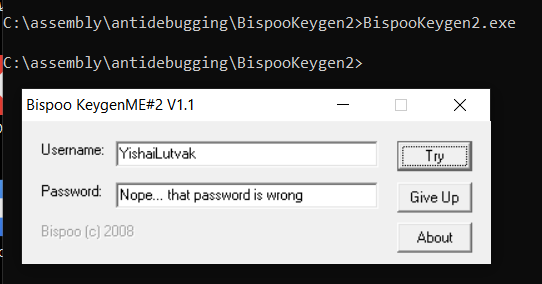
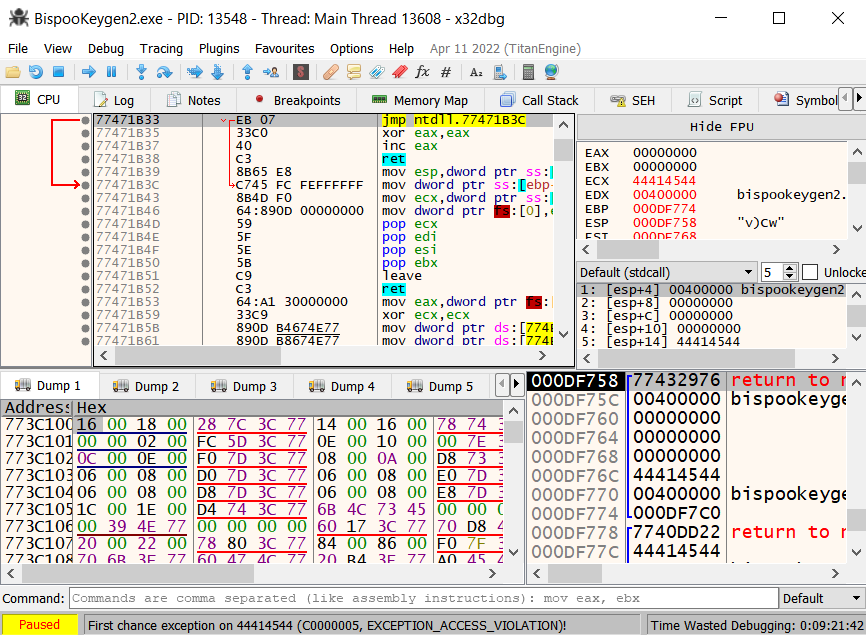
בס"ד

**BispooKeygen2 - הדרך לפיתרון**

נריץ את הקובץ דרך הcmd:

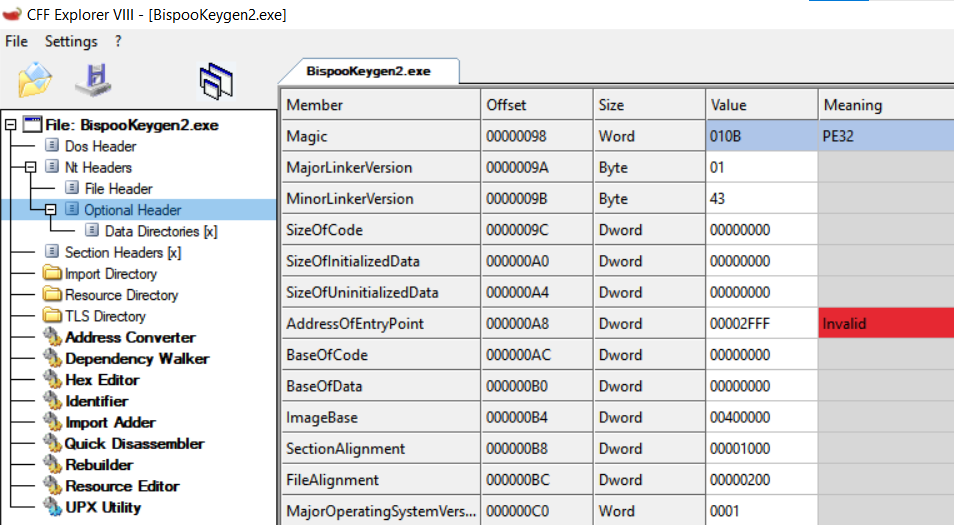


נפתח את הקובץ בexe32dbg וננסה להריץ:



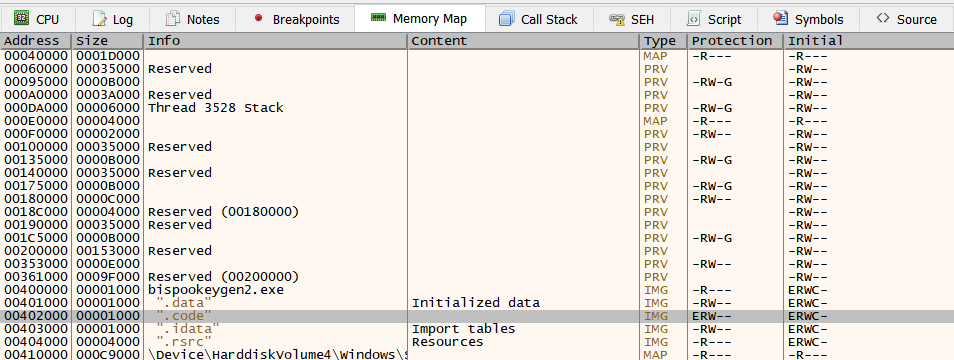
ניתן לראות בשורה האחרונה למטה שאירעה שגיאה.

נפתח את הקובץ בCffExplorer:



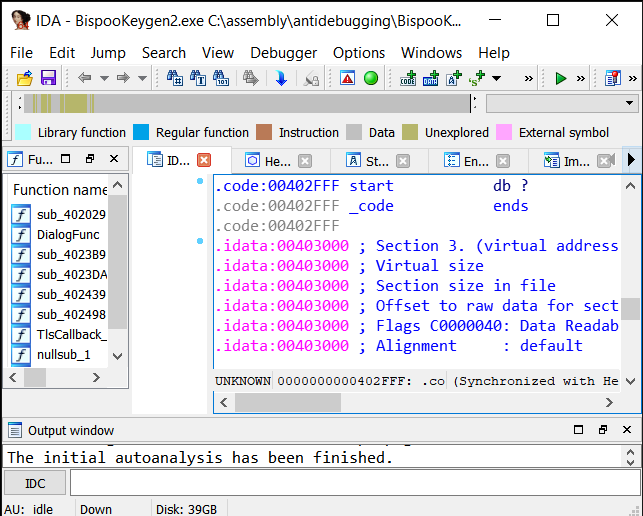
ניתן לראות שהEntryPoint לא תקין (הערך האמיתי של נקודת ההתחלה היא:

ImageBase + AddressOfEntryPoint כלומר 0x402FFF)

נסתכל בex32dbg בחלון של Memory Map: 

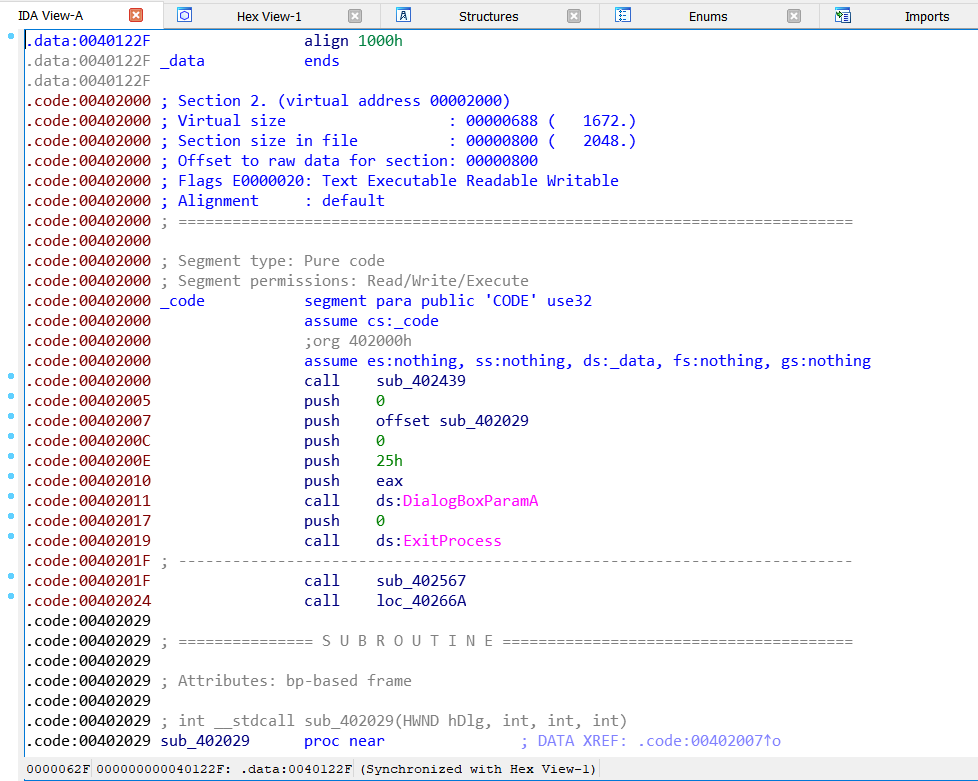
ניתן לראות שאזור הקוד של Bispookeygen2 מתחיל ב0x402000 ונגמר ב0x4021000.

נפתח את הקובץ בida:

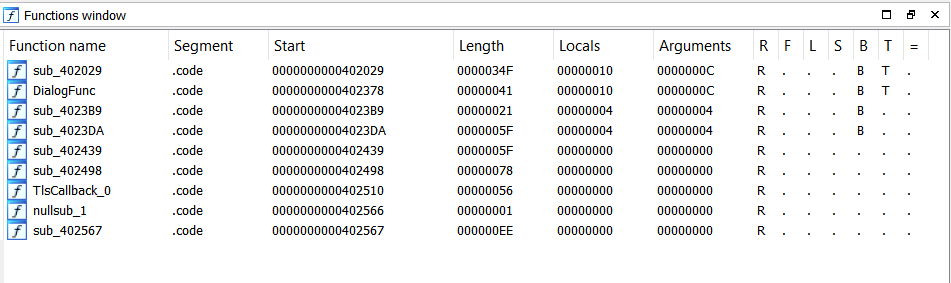


ניתן לראות שמשהו לא הגיוני פה. אזור הקוד נפתח ונסגר מיד בלי כלום.

אם נגלול למעלה נראה את אזור הקוד האמתי:

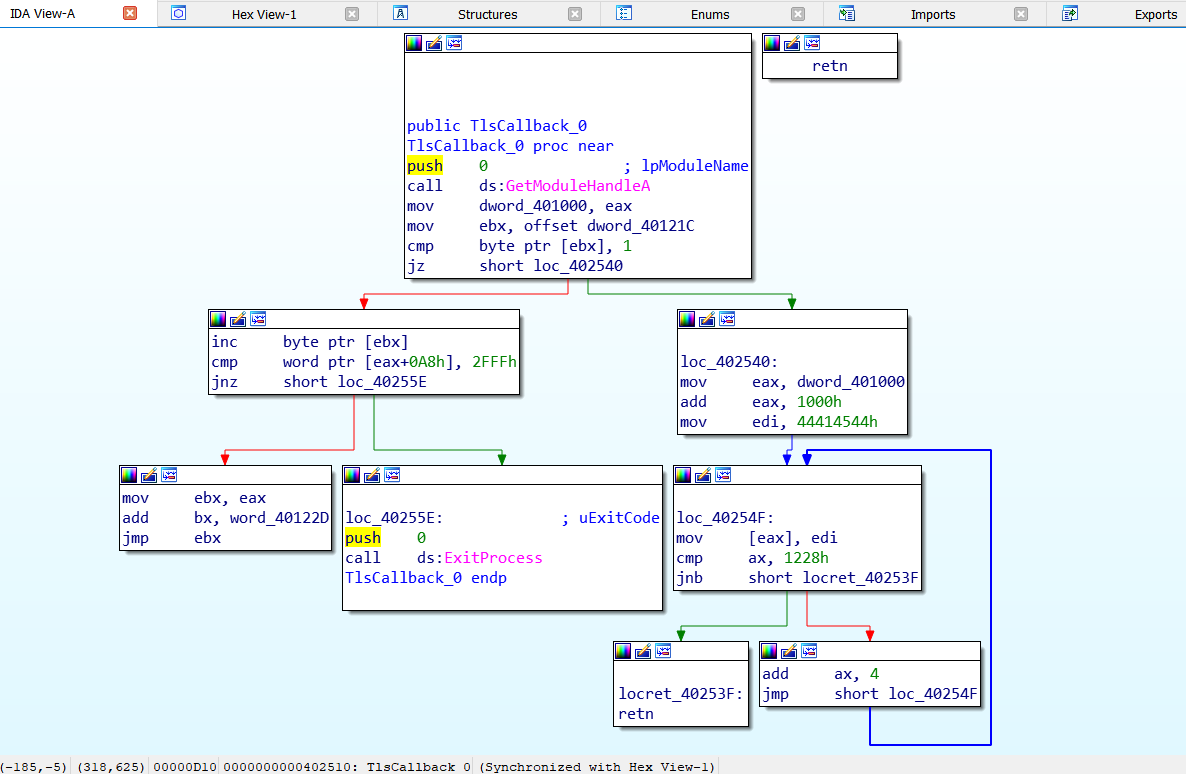


נשים לב לחלון משמאל שמביא לנו את שמות הפונקציות:



ניתן לראות בין הפונקציות את הפונקציה TlsCallback\_0.

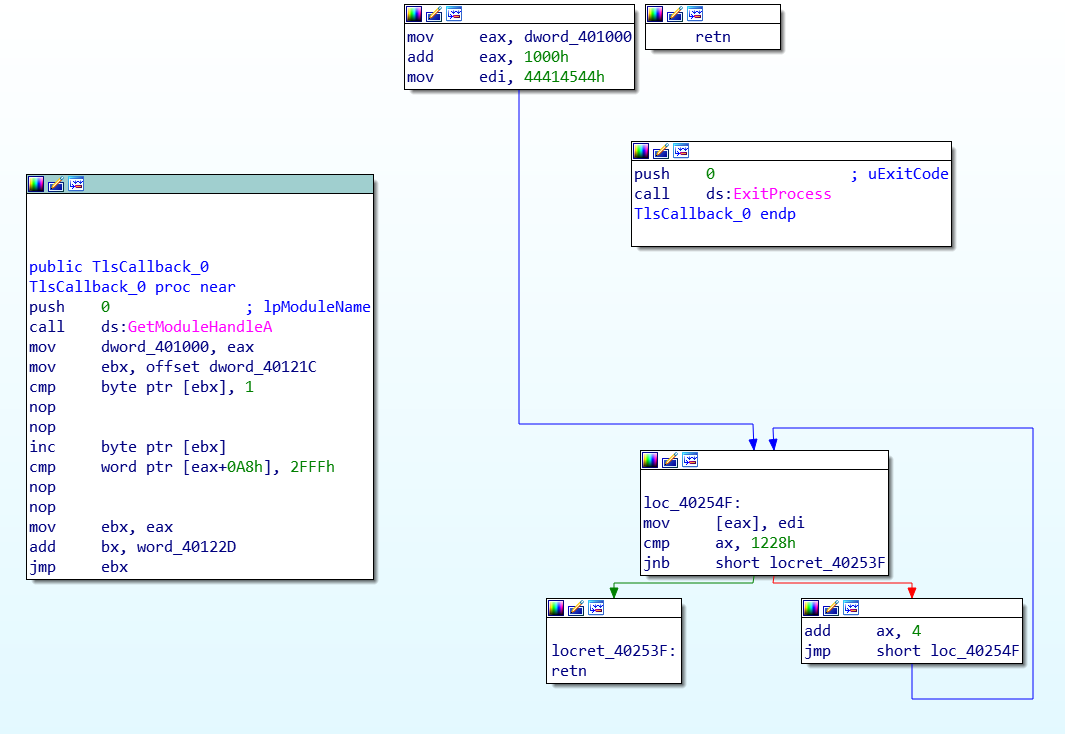
נכנס אליה ונראה מה היא עושה:



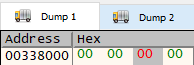
בענף השמאלי ביותר יש עדכון של הEntryPoint לערך הנכון (401000+1000=402000).

בענף הימני ביותר יש דריסה של אזור הdata. בענף האמצעי יש קריאה לפונקציה ExitProcess.

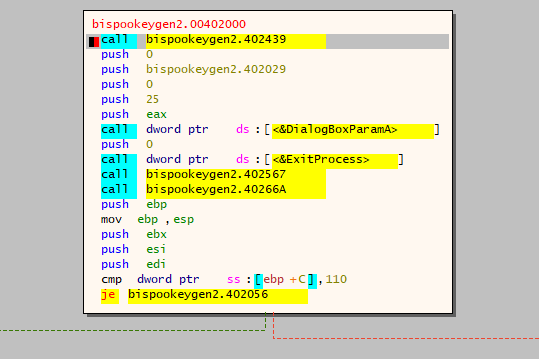
נפצפץ את הקוד באופן המתואר בתמונה הבאה, כך שתמיד נעבור דרך הענף השמאלי ביותר:



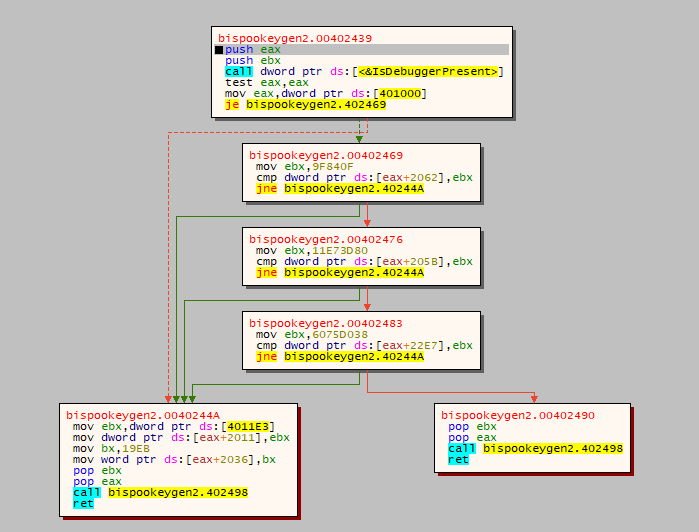
נפתח את הקובץ בexe32dbg, ודבר ראשון נשנה את הpeb ליתר ביטחון:



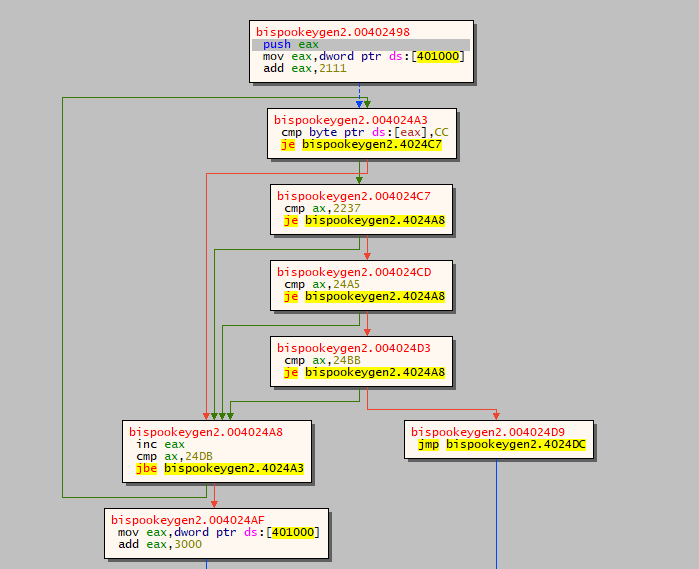
נשים bp בכתובת 0x402000 ונריץ:



נכנס לתוך הפונקציה בשורה הראשונה:



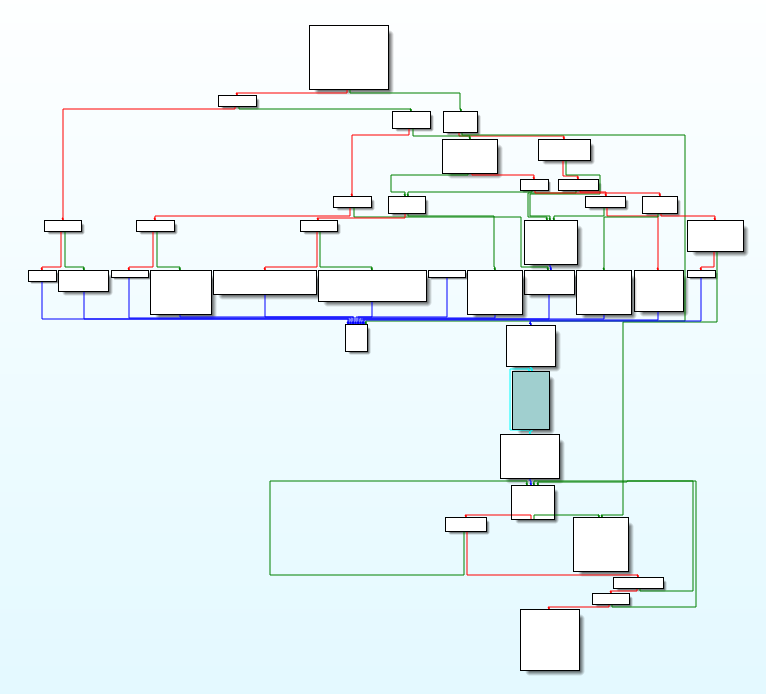
בשורה השלישית ניתן לראות שקוראים לפונקציה IsDebuggerPresent, כלומר זוהי פונקציה שבודקת אם אנו מדבגים את הקוד. בהמשך ניתן לראות בדיקות אם קוד בכתובות מסוימות נשאר כפי שהוא או פוצפץ. אם ניכנס לפונקציה בענף הימני ביותר למטה נראה בדיקות נוספות (CC) שקשורות לantidebug:



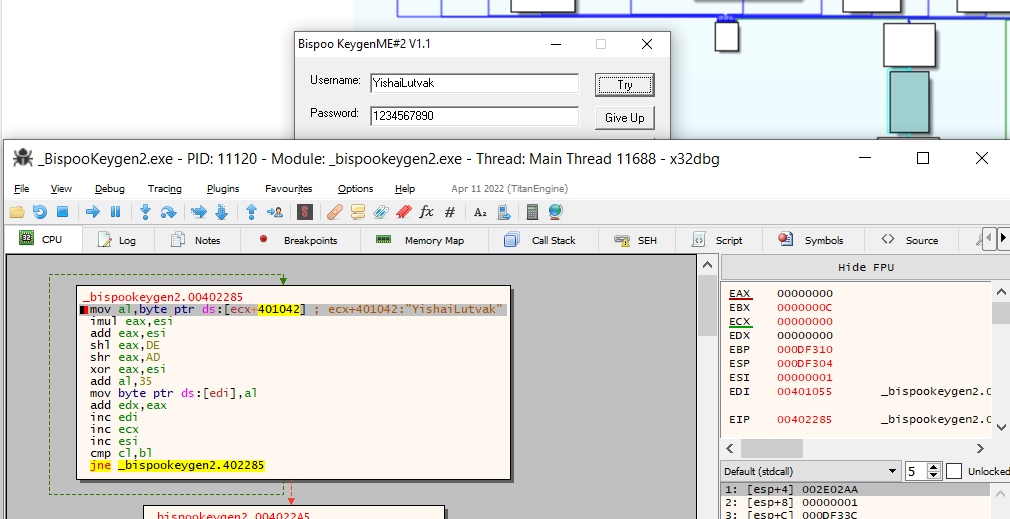
נפצפץ את הקוד על ידי שינוי הפקודה הראשונה בפונקציה לפקודה ret:

|  |  |
| --- | --- |
| לפני | אחרי |
|  |  |

נפתח את הפונקציה שנשלחת לDialogBoxParamA בתור פרמטר (הפונקציה בכתובת 0x402029) באמצעות ida, כדי לראות איפה כדאי לשים בה bp לפני שנריץ את הקובץ המפוצפץ שלנו ב.x32dbg



ממבנה הקוד וסקירה קצרה נראה שכדאי לשים bp בתחילת המלבן שסימנו בכחול. שם יש לולאה שאולי תהיה מעניינת, בנוסף נמלא את שם המשתמש והסיסמא כמתואר בתמונה הבאה:



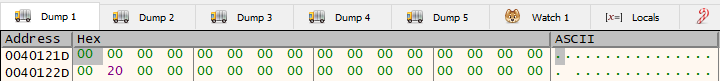
נשים לב ששם המשתמש נכנס לזיכרון בכתובת 0x401042 (ניתן לראות זאת בשורה הראשונה במלבן הראשון בתמונה להלן), והסיסמא נכנסה לזיכרון בכתובת 0x401004 (ניתן לראות זאת בשורה הראשונה במלבן השלישי בתמונה להלן).



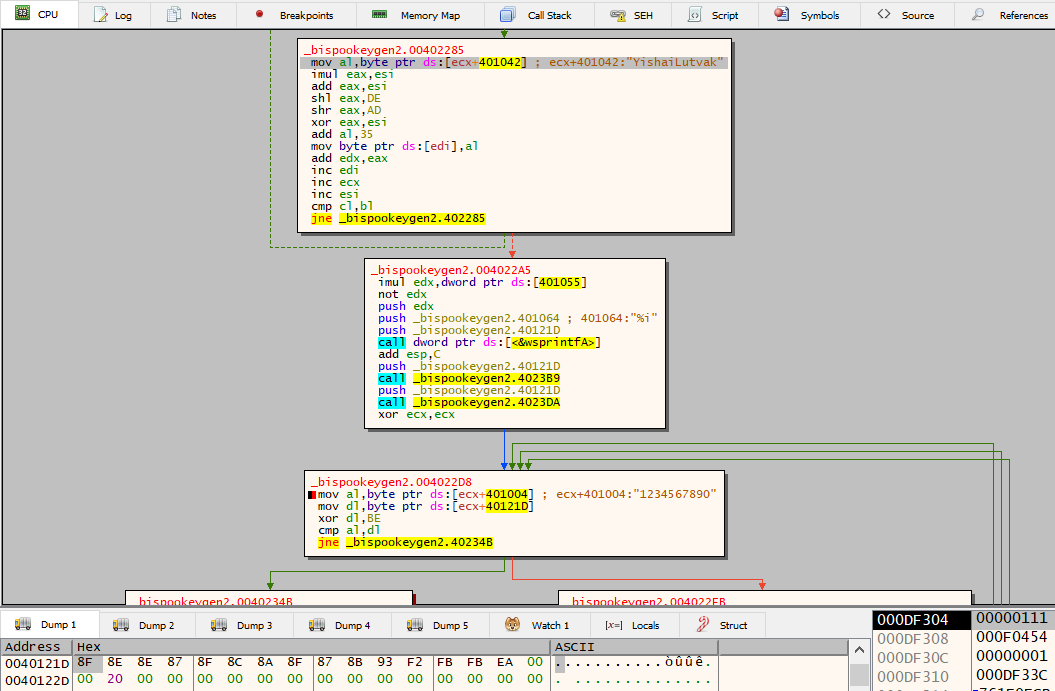
אם נסתכל נסקור את שאר הקוד באזור נוכל להבין שהקטע הקריטי ביותר נמצא במלבן השלישי לעיל. שם נעשית בדיקה האם הסיסמא מתאימה למחרוזת שנמצאת בזיכרון בכתובת 0x40121D **לאחר שאנו מבצעים על כל תו בה פעולת xor עם הערך BE** (ecx עולה בכל איטרציה ב1 כדי לרוץ במקביל על שתי המחרוזות עד הסוף).

מן הסתם המחרוזת שנמצאת בכתובת 0x40121D מושפעת משם המשתמש שהזנו, כך שלכל שם משתמש תהיה סיסמא שונה.

נסתכל מה יש כרגע בכתובת 0x40121D:

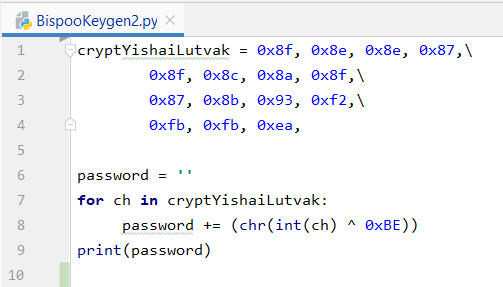


נסיר את הbp ששמנו בשורה הראשונה במלבן **הראשון** (שהופיע בתמונה לעיל), ונשים bp חדש בשורה הראשונה במלבן **השלישי** (שהופיע בתמונה לעיל) ונריץ:

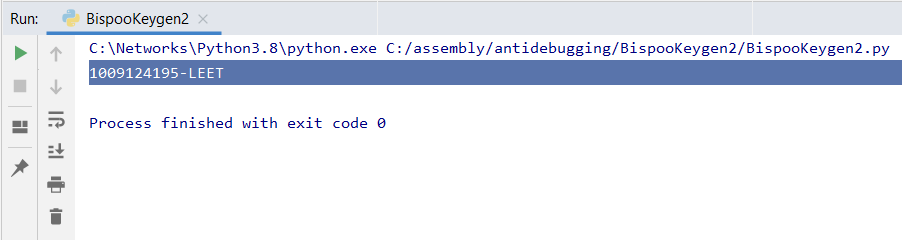


ניתן לראות בתמונה כי הזיכרון בכתובת 0x40121D התמלא ואורכו הוא 15 בתים.

נכתוב סקריפט בפייתון שיבצע xor על הערכים האלו עם BE ויחזיר לנו את המחרוזת שיוצאת:



נקבל את הסיסמא הבאה:

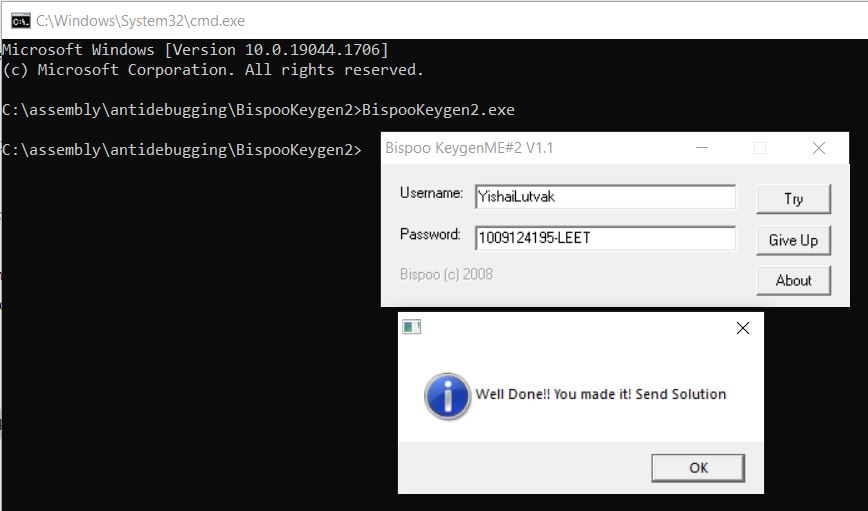


נריץ את הקובץ המקורי (לפני הפצפוץ) בcmd

כאשר נזין את שם המשתמש: "YishaiLutvak"

ואת הסיסמא: "1009124195-LEET"

ונבדוק מה קורה:

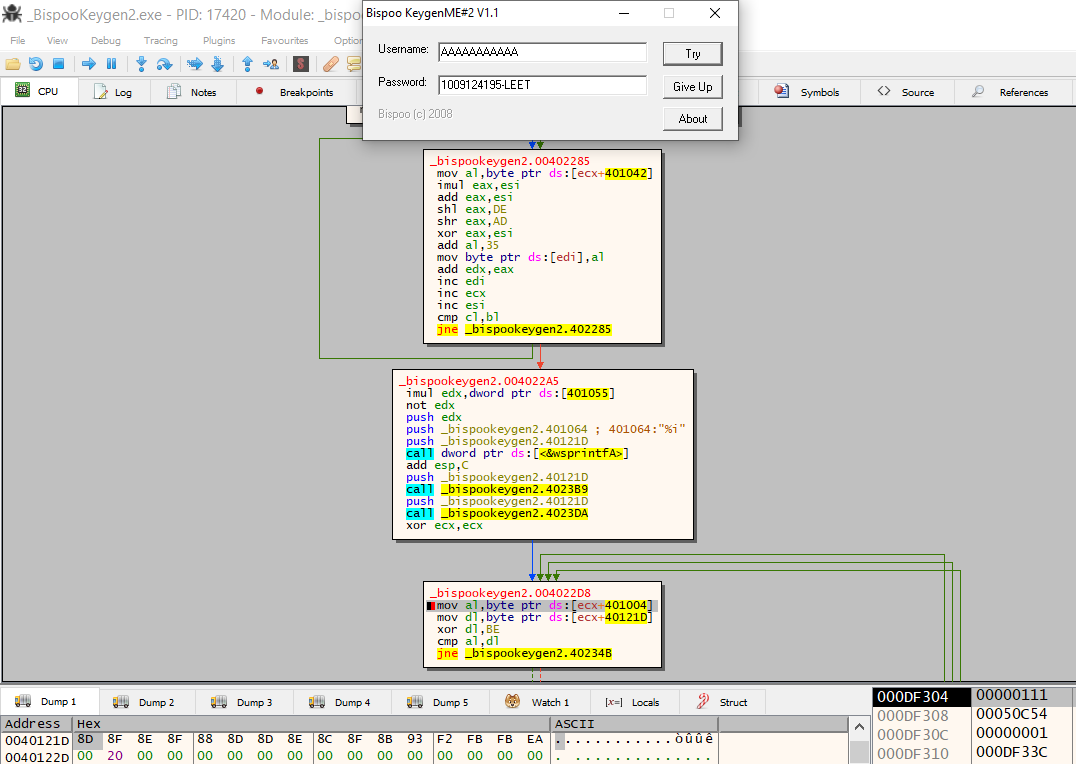


זה נראה שהצלחנו!!

נוודא רק שההנחה שתוכן הזיכרון בכתובת 0x40121D מושפע משם המשתמש נכונה:

נריץ שוב את הקובץ המפוצפץ בx32dbg, כאשר אנו משאירים את הbp ששמנו בכתובת 0x4022D8

ומזינים שם משתמש שונה:



כבר מהאורך של המחרוזת ניתן להבחין שהיא שונה ממה שהיה לנו קודם.

מכאן נראה בסבירות גבוהה כי ההנחה שלנו שתוכן הזיכרון בכתובת 0x40121D מושפע משם המשתמש אכן נכונה.