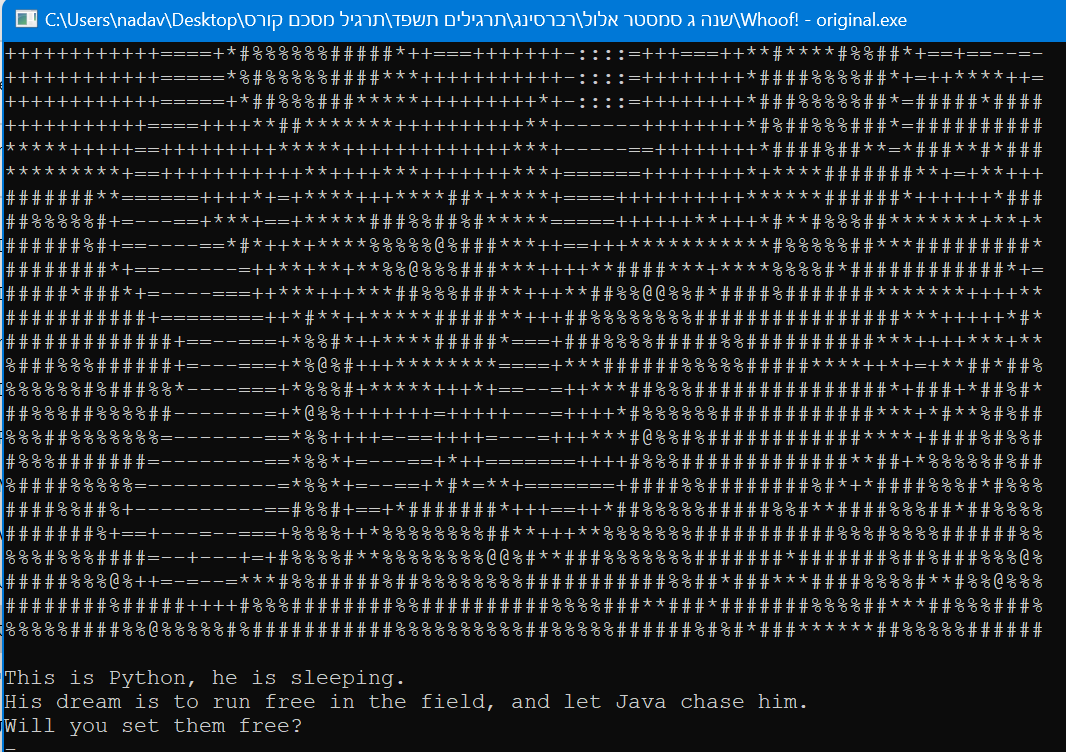
# Writeup – תרגיל מסכם רברסינג סמסטר א' התשפ"ד – נדב צימרמן

## שלב א "פייתון"

הקוד של התרגיל נמצא בשתי מקומות: בTLS ובMain.

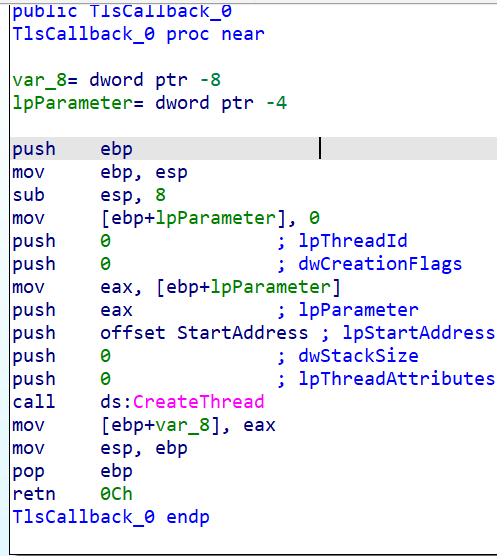
בTLS: נוצר פעם אחר פעם תהליכון שמתחיל כל פעם להריץ מחדש את הפונקציה שמתחילה בתווית "startAdress" שכותבת את התווים הבאים (תמונה 1):



התהליכונים הרבים שרצים מקשים על הדיבוג משום שפעמים רבות צריך להחזיק ראש ולעקוב אחרי ההתפתחות של התוכנית אשר מבצעת פעילויות מסונכרנות של התהליכים שמתרחשים בו זמנית, שמתבטאת בפועל בקפיצה של התוכנית לשורות שונות באותו הקוד ומקשה על הדיבוג ופיענוח של המתרחש בקוד.

תמונה 1 המסך שמתקבל בהרצת whoof

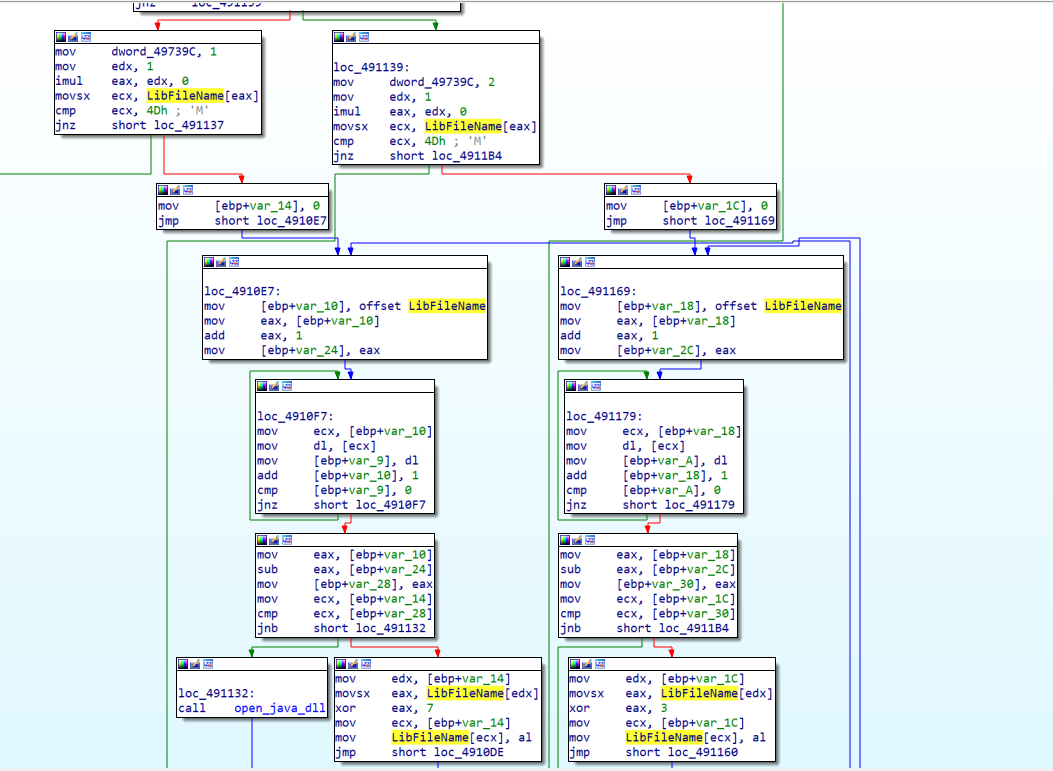
בפונקציה שמתחילה ב"startAdress"(תמונה 2), קיימת הסתעפות שמגיעה לפונקציה שיוצרת את הקובץ java.dll" " עליו נרחיב בחלק הבא.

העובדה שהקוד מתרחש לפני הmain מקשה על הבנת התוכנית, אנו רגילים לתוכניות בעלות מבנה סדור שמנוהל על ידי פונקציה ראשית main, מרגע שהmain רק מתחבר לתוכנית באמצע הרצתה זה מקשה קונספטואלית, ובנוסף פרקטית משום שצריך ללמוד מחדש את מבנה התוכנית. כמו כן קשה לעקוב אחר ערכי המשתנים שמשפיעים על התקיימות התנאים השונים – שינוי לפני הmain יכול להשפיע עליו ולכאורה לא נראה לכך כל סימן מהmain והלאה.

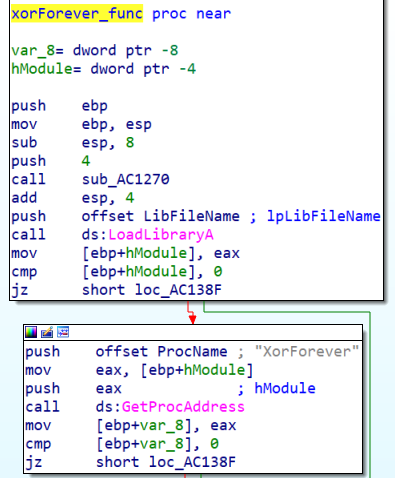
תמונה 2 יצירת התהליכונים בTLS שיתחילו ב"startAdress"

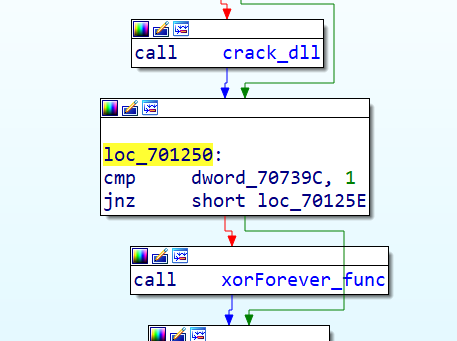
****נוסף על כך בפונקציה שמתחילה ב" startAdress" מתבצע עדכון של המשתנה LibFileName שבתחילה מכיל 4D אולם עובר בפונקציה זו מניפולציה מסויימת בסופה משתנה להכיל את הערך java.dll כפי שניתן לראות בתמונה הבאה: (בהמשך נסביר כי הדבר משמש לטעינת קובץ הdll ע"י הפונקציה loadLabararyA). דבר זה מקשה כמובן כי לא ניתן לזהות באנליזה סטטית את שם הdll שייטען אלא רק באופן דינאמי תוך ביצוע המניפולציה על המשתנה

תמונה 3 יצירת התווית java.dll לטעינת הdll בהמשך

****

**בmain** מתרחשת הדפסה של התו המסתובב שנראה במהלך הרצת התוכנית, בתוך הפונקציה שקראתי לה: "print\_python\_fun" (הדפסה שתלווה אותנו בכל עת הרצת התוכנית), תוך ריצת התהלכונים במקביל לפונקציה הראשית.

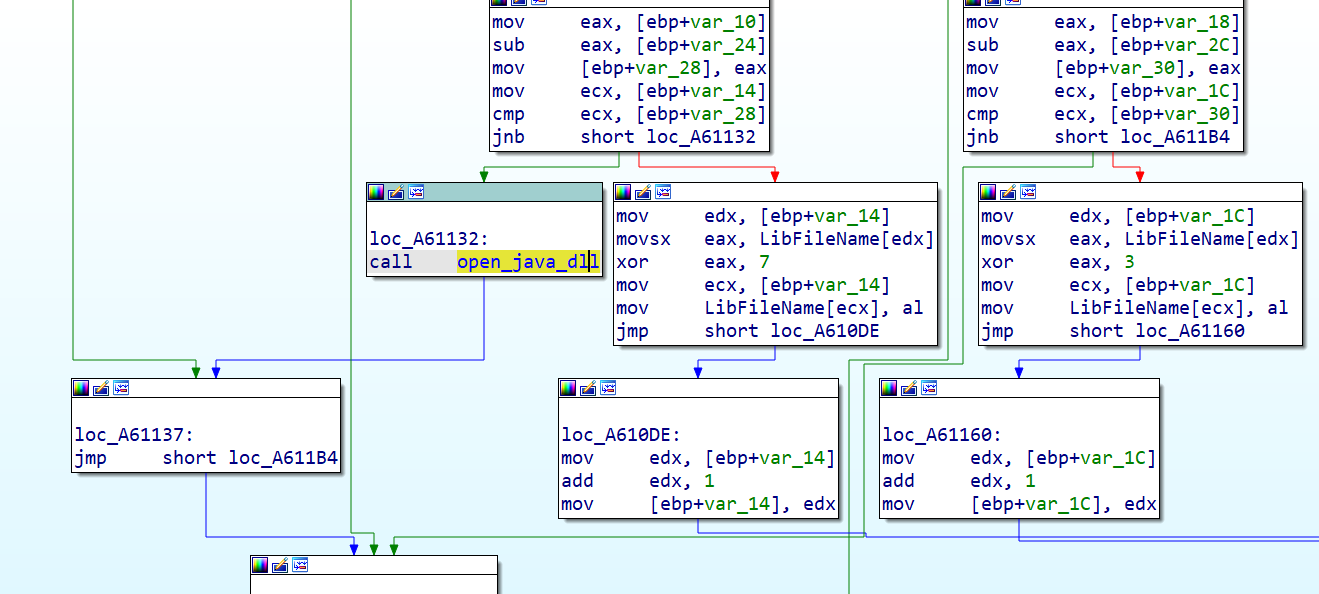
כרגע כל התוכנית תקועה ללא כל התפתחות, אולם כפי שניתן לראות בתמונה 4, קיימת בהמשך הmain- באחת קריאה לפונקציה LoadLibraryA שטוענת את java.ומייבאת מתוכו את הפונקציה XorForever (ניתן לדעת זאת על בסיס שם התוכנית אותו מנסה לייבא הmain בתמונה 5). אז הפונק' XorForever נקראת ולאחר מכן, נגמרת התכנית. לכן גם בלי להכיר את תוכנו של java.dll ניתן לדעת שקיימת בו פונקציה XorForever בעזרתה נצא מן המבוי הסתום אליו נקלענו.



תמונה 5 ייבוא java.dll ומתוכו הפונק' XorForever

תמונה 4 מבנה main, השימוש בפונקציה XorForever

## שלב ב' "ג'אווה"

תמונה שמכילה טקסט, צילום מסך, תצוגה, תוכנה

התיאור נוצר באופן אוטומטיעל מנת למצוא את java.dll צריך לנווט אל הפונקציה שכותבת קובץ זה בתוך startAdress: לצורך כך חיפשתי בIDA אחר המשתנה java וראיתי שנעשה כתיבה לתוך הקובץ java.dll (תמונה 7) בפונקציה מסויימת. קראתי לה open\_java\_dll, וניווטתי את הקוד על מנת שייכנס אליה (תמונה 6).

תמונה 7 כתיבת java.dll בתוך הפונק' open\_java\_dll

תמונה 6 הקריאה לפונקציה שתכתוב את java.dll

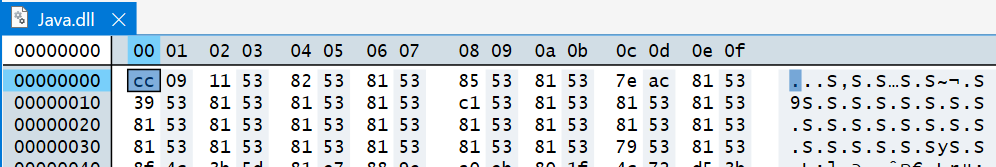
תמונה שמכילה טקסט, צילום מסך, תוכנה, מספר

התיאור נוצר באופן אוטומטי

פתחתי את הקובץ java.dll שנוצר בתוכנה cff כפי שלמדנו בהרצאה (תמונה 8), וראיתי כי מדובר בפורמט לא ידוע – קובץ לא קריא.

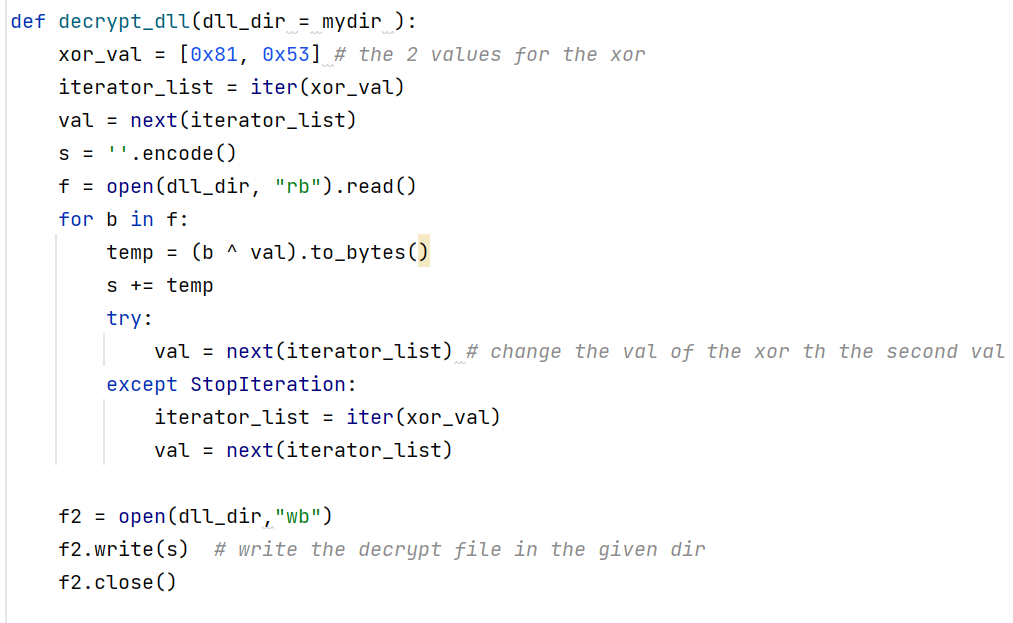
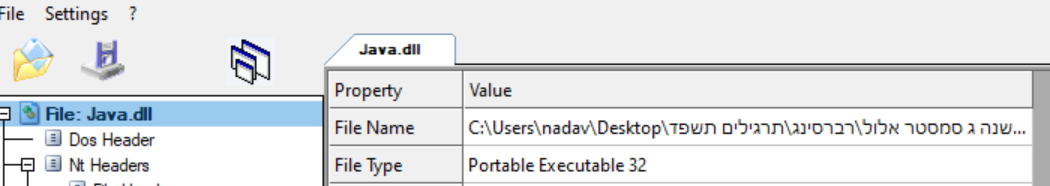
תמונה 8 פתיחת הקובץ java.dll בcff

פתחתי את הקובץ בעורך בינארי וראיתי שכל ערכי הקובץ מעורבלים בצורה כלשהיא (תמונה 9).



למדנו בשיעור שפורמט PE (הפורמט של קבצי exe ו-dll) מתחיל בתווים "MZ" לכן ידעתי ששני התווים הראשונים בקובץ: cc,09 אמורים היו להיות 4d, 5a ("MZ" באסקי). ביצעתי xor של cc עם 4d וקיבלתי 81 ומxor של 09 עם 5a קיבלתי 53 (הכל בהקסה כמובן). לכן כתבתי בפייתון תוכנית שמבצעת xor של שני מספרים אלו על כל תווי ה-dll כדלהלן (תמונה 10):

תמונה 9 הקובץ java.dll בעורך הקסה

שמרתי את הקובץ המפוענח בתיקייה של whoof.exe בשם java.dll כך שבפונקציה בmain שתוארה בחלק א' מתבצעת טעינה של java.dll וקריאה לפונקציה XorForever מתוכה. אולם ברגע שהתכנית whoof.exe ניסתה לטעון את הdll התוכנית כולה קרסה.

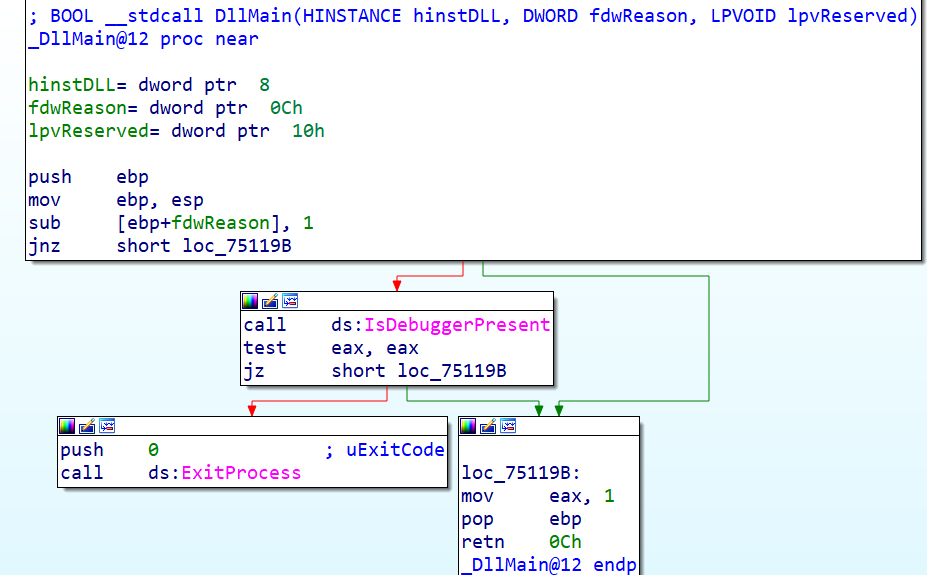
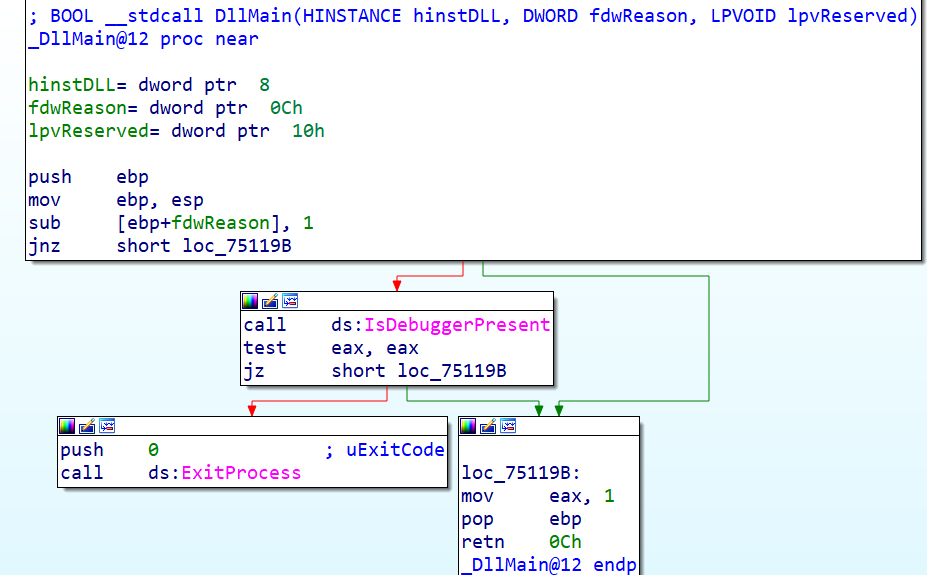
תמונה 11 java.dll בתוכנת CFF לאחר הפענוח

לאחר מכן פתחתי את java.dll המפוענח בcff וראיתי שאכן התקבל קובץ PE תקין:

תמונה 10 הקוד בפייתון שכתבתי לצורך פענוח java.dll

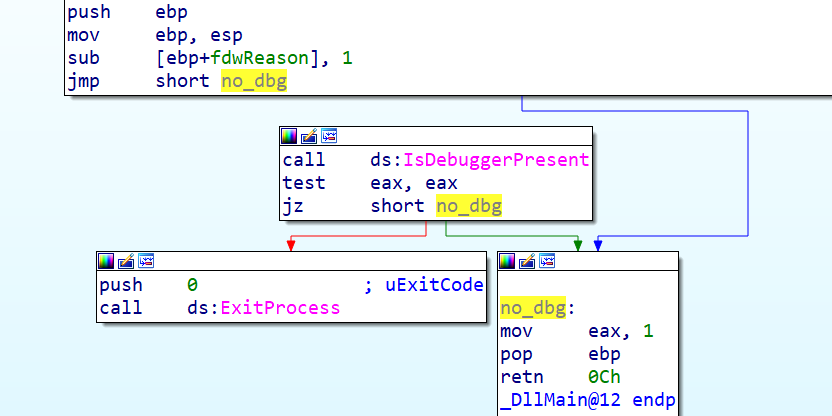
לצורך דיבוג הרצתי את הקובץ dll המפוענח בIDA תוך הוספת ברייקפוינט שיתרחש כאשר הקובץ whoof.exe יטען את הdll כפי שנלמד בהרצאה.

גיליתי כי Java.dll השתמש בפונקציה IsDebuggerPresent לצורך בדיקה שהקובץ איננו מדובג. במקרה בו התכנית מדובגת הבדיקה תכשיל את קיצת התוכנית וזה מה שהוביל לקריסת התוכנית whoof.exe אותה חוויתי קודם לכן.



לצורך מעבר האנטידיבאג התוכנית הוספתי patch (תמונה 13) אשר ידלג על הבדיקה – IsDebuggerPresent מה שיגרום לטעינת ה-dll בהצלחה לmain. ניתן לראות שהpatch מבצע jump ללא ביצוע בדיקת האנטידיבאג כלל.

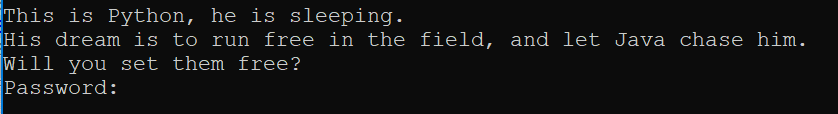
תמונה 12 אנטי דיבאג תוך שימוש בפונק' IsDebuggerPresent

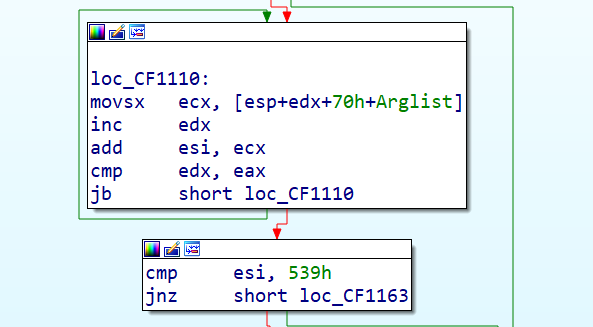


תמונה 13 ה-patch הדרוש כנגד ה"אנטידיבאג"

כפי שתיארתי לעיל היו מספר מנגנונים שהקשו עליי בטעינת java.dll לMain:

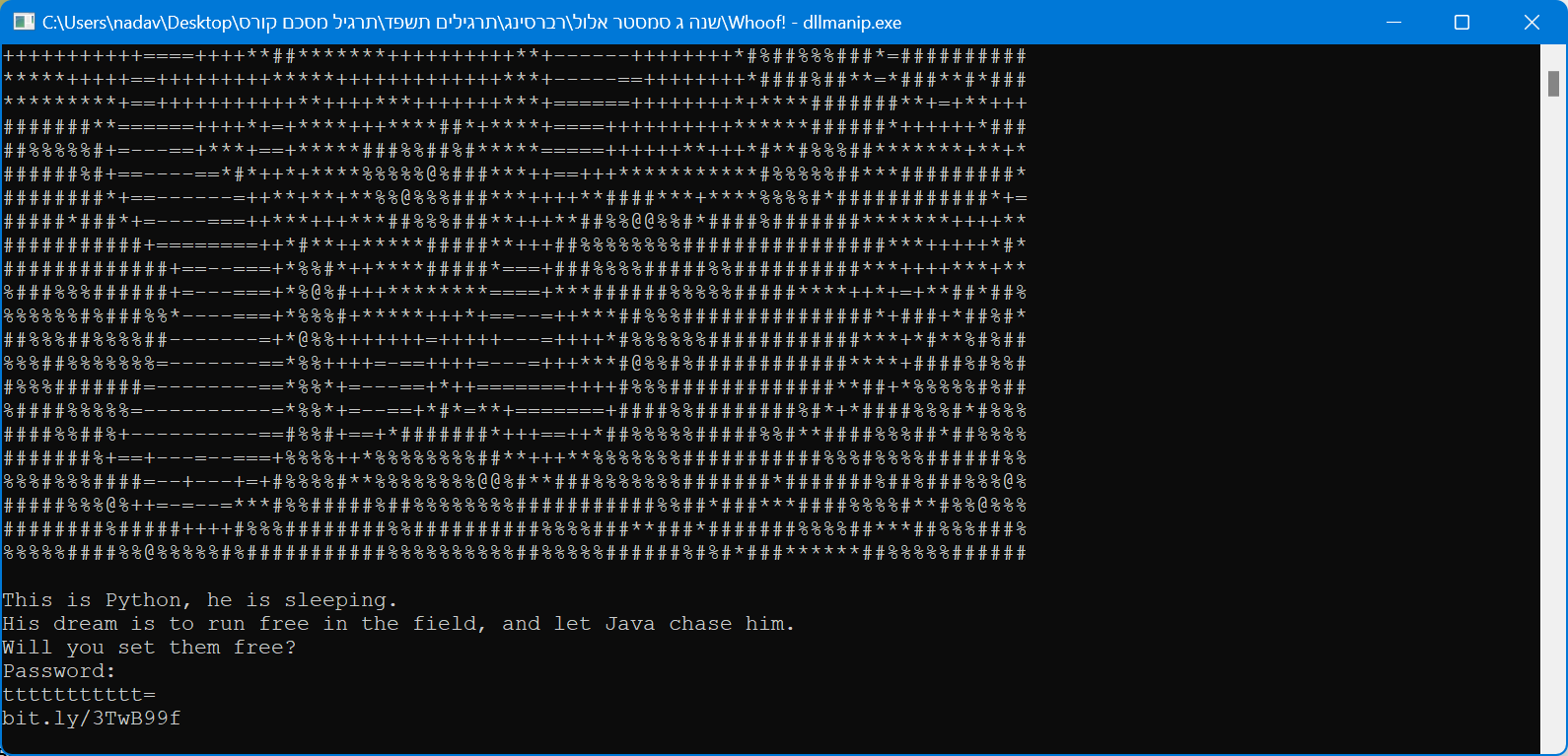
1. במקור לא קיים קובץ java.dll. קיימת התסעפות בפונקציה שנקראת ע"י הTLS שכותבת את הקובץ.
2. הקובץ לא היה בפורמט PE ובתחילה לא היה ברור מה עליי לעשות על מנת לתקן את הקובץ. גם כשהבנתי שככל הנראה הקובץ מוצפן לא היה ברור האם קיים מפתח כלשהוא לצופן , האם יש מקום בwhoof שמפענח את הקובץ או שמוטל עליי לעשות זאת עד שלבסוף חשבתי על דרך הפתרון שתוארה לעיל.
3. התוכניתwhoof קרסה מיד כשניסיתי לטעון את הdll, בתחילה לא היה ברור מה הוביל לקריסה – האם בעיה במציאת הקובץ, האם לא שלחתי באופן תקין את הפרמטרים לפונקציה LoadLibraryA, וכו'. בסופו של דבר הבנתי שייתכן מנגנון אנטידיבאג בתוך הdll ופעלתי כפי שתיארתי קודם לכן על ידי דיבוג הdll.

לאחר הpatch הרצית פעם נוספת את whoof.exe וחלה התקדמות! הופיע פלט להכניס את הסיסמא הרצויה:

חזרתי לדבג את java.dll וראיתי כי בתוך הפונקציה XorForever הפונקציה סוכמת את הערך האסקי של כל התווים שהתקבלו בסיסמא ומשווה אותו לערך ההקסדצימלי 5390x. (easter egg: ראה בהמשך התייחסות למספר 539 בהקסה).

תמונה 14 הקוד סוכם את האסקי של הסיסמא ומשווה ל539 בהקסה

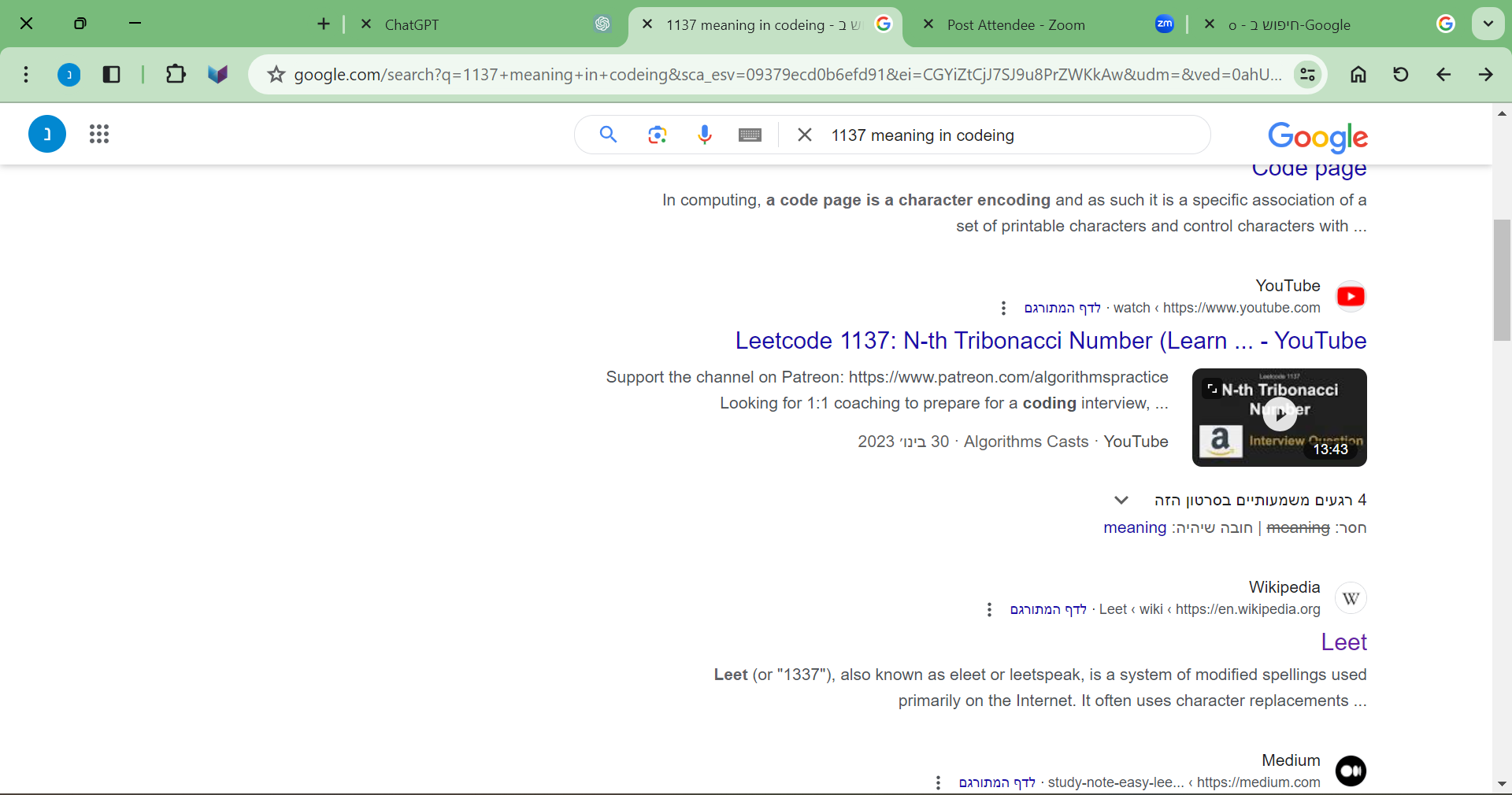
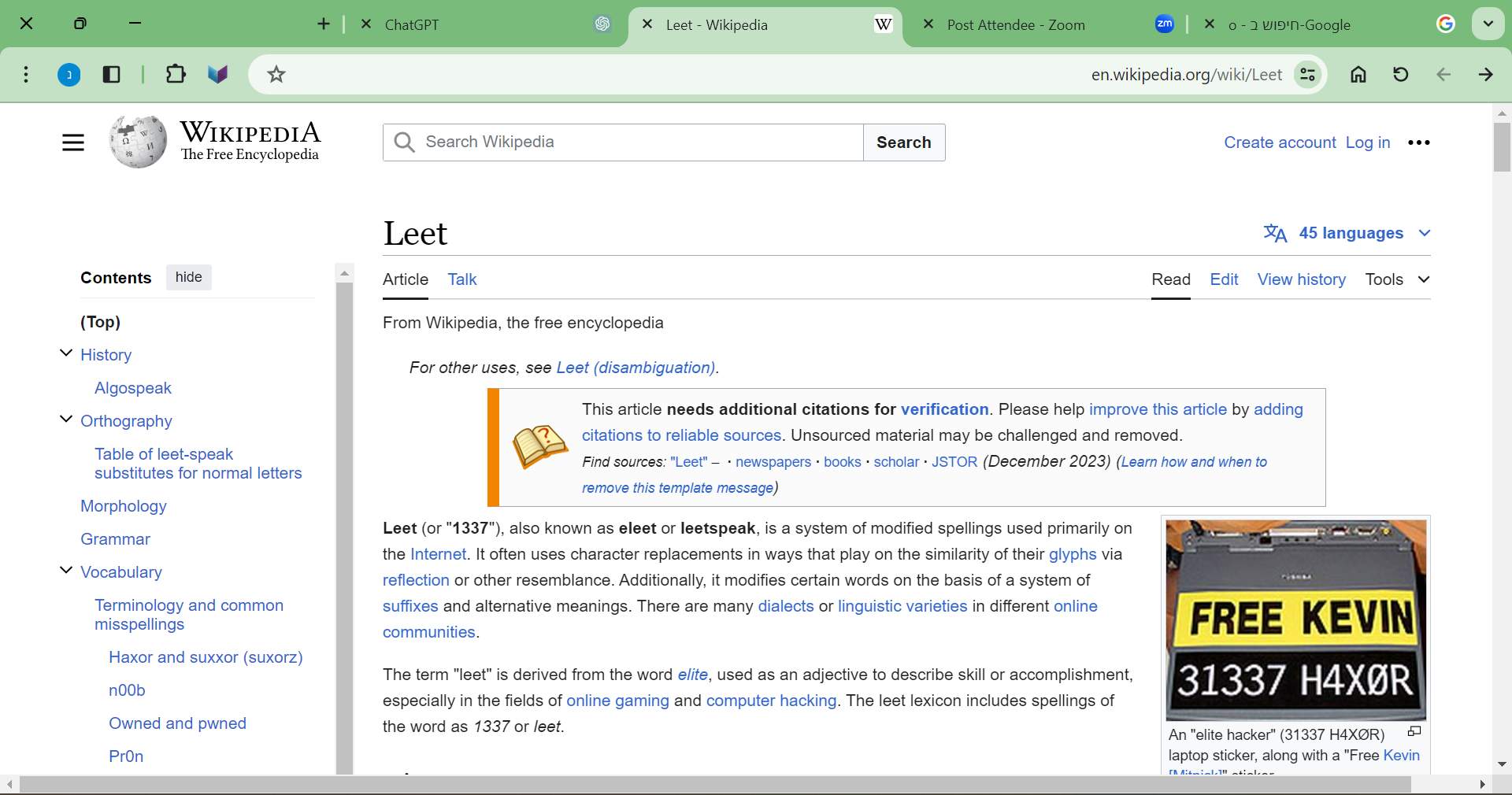
ניסיתי לקבל 539 באסקי. לצורך כך חישבתי כי הערך של t הוא 0x74 והערך של '=' הוא 0x3d ו*.*

*הכנסתי את הסיסמא שחישבתי: ttttttttttt= ואכן התקבל הקישור לשיר הפרידה The End של ה* Beatles

*אם לא פירטתי באופן בהיר דיו את ההמנגנונים שהקשו בחלק של "פייתון":*

1. *הקוד מתנהל בחלקו בTLS שמתחיל לרוץ לפני הmain, חריגה זו מהתפיסה של הסדר המובנה של התוכנית לה אנו מצפים מקשה עלינו להבין את התוכנית*
2. *הdll שאנו ננסה לטעון במהלך התוכנית נשמר בתחילה במשתנה שמכיל ערך אחר, ותוך כדי ההרצה מקבל את ערכו האמיתי: "java.dll" מה שמקשה על האנליזה הסטטית של התוכנית, ודורש לעקוב אחר השינויים שעובר המשתנה בזמן הריצה.*
3. *התהלוכונים רצים במקביל ומקשים על הדיבוג (אמנם זהו דבר שאינו מוכרח להיחשב כ'מנגנון שמקשה על פיענוח התכנית, אולם בסופו של דבר התוכנית קופצת ממקום למקום מה שמקשה על הדיבוג)*

## easter egg:

*הסכימה של כל תווי הסיסמא והשוואת הסכום ל539 עורר בי חשד לכן חקרתי על כך. ראיתי כי המספר 539 בהקסה הוא 1137 בבסיס העשרוני. מחיפוש אחר מספר זה בעולם התכנות ראיתי כי זה שם נרדף ל"Leet". 1337 או leet הוא שימוש בהחלפת אותיות בתווים מספריים או מיוחדים ליצירת מילים וביטויים.*

תמונה 16: leet או 1337 בויקיפדיה

תמונה 15: חיפוש אחר 1337 בהקשרי תכנות בגוגל

*נ"ב ללא קשר למטלה המסכמת – רציתי להגיד תודה על ההשקעה הבלתי נגמרת שלך בסטודנטים. בתור בוגר של רשתות מחשבים בגישה מחקרית וכעת גם של רברסינג, אני יכול להגיד שהקורסים שלך שונים בנוף. מלאים בלמידה מתוך העשייה, מלמדים סביבות עבודה, תהליכי עבודה, התמודדות מול סימני שאלה, הכרח לחפש את התשובות בעצמך ולהתמודד עם האתגר עד לפיצוח.*