**חלק ב':**

* **partb.opt.s:** בקובץ זה נמצאו מספר טעויות:
  + בעת הגרלת המספר יש שימוש בפונקציה rand מ-libc שמגרילה מספר רנדומלי אל רגיסטר, שזו אמנם המטרה, אך היא מגרילה בכל פעם את אותו המספר (במקרה שלנו יצא 42). לכן, על מנת לפתור בעיה זו השתמשנו ב-srand. הפונקציה srand יוצרת seed לרנדום לפי הערך בראש המחסנית, ולכן קודם כל דחפנו את ערך החזרה של time הנמצא ב-eax לראש המחסנית כך שהרנדום יתבסס על זמן המערכת. לאחר מכן, ישנה הקריאה המקורית ל-rand שכעת תגריל בכל פעם מספר אחר.
  + בעת ריצת התוכנית, מודפסת פעמיים ההודעה “Guess a number between 1 and 100” כאשר בפעם השנייה אמורה להופיע ההודעה “Enter your guess:”. לכן, על מנת לתקן זאת, החלפנו בשורה 39 בקובץ המקורי את LC0 ל-LC1 כך שתודפס ההודעה הרצויה.
  + לאחר קליטת הניחוש מהמשתמש, המספר שבחר מופיע במחסנית והמספר שהוגרל מופיע ברגיסטר ebx. בעת השוואת המספרים, בשורה 44 בקובץ המקורי, ניגשים אל המספר שהזין המשתמש ע"י ebp-12. אם המספר שניחש המשתמש קטן מהמספר שהוגרל, ישנה קפיצה אל L2 שגורמת לתחילת התהליך מהתחלה, זאת במקום להודיע על השגיאה בבחירת המספר והודעת בחירה מחדש. לכן, על מנת לתקן זאת, החלפנו בשורה 45 את jge L2 ל-jge L3 שתגרום להדפסת הודעה האומרת שהמספר שהמשתמש בחר קטן מדי ויש לבחור מחדש.
* **partb.nopt.s:** בקובץ זה נמצאו מספר טעויות:
  + בתחילת התכנית, בשורה 24, מקטינים את esp ב-16 כך שמשאירים מקום ל-4 משתנים/ ארגומנטים, למרות שבפועל משתמשים רק ב-3. לכן, הקטנו את esp ב-12 במקום.
  + המחרוזת הנמצאת ב-LC2 בשורה 11 מכילה %c ולא %d, ועל כן הקלט אינו מתבפרש כראוי (מתפרש כתו במקום כמספר שלם). לכן, שינינו זאת בהתאם.
  + בשורות 58 ו-65 (בקובץ המקורי), אליהן מגיעים כאשר המספר אותו הזין המשתמש הוא גדול מדי או קטן מדי, מודפסת הודעת שגיאה ולאחרי קופצים אל L3 שם מסתיימת ריצת התוכנית. על מנת לסדר זאת, יש להחליף את הקפיצה הנ"ל ולקפוץ אל L6 במקום, שם תתבצע קליטה מחדש של מספר ע"י המשתמש.

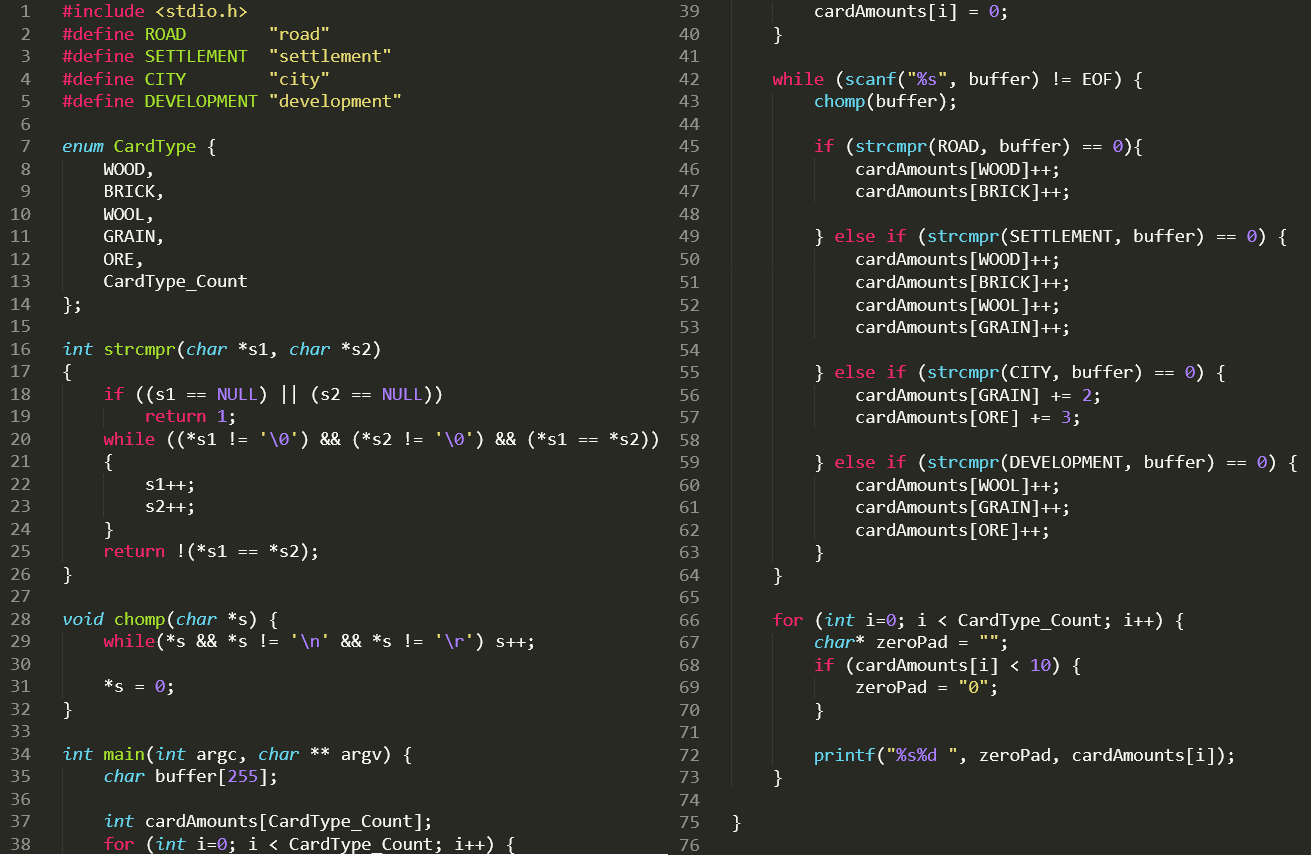
**חלק ג':**

להלן שלבי הפתרון בחלק זה:

* ראשית כל, נכנסנו לקוד המקור בכתובת המצורפת. לאחר סקירה מהירה ובחינת העצמים הנמצאים בקוד, הגענו לחלק שבו מופיע כפתור מוסתר שלחיצה עליו תביא לאתגר, זאת הסקנו ע"י הקוד:

לכן, בעריכת קוד המקור מחקנו את “disabled” על מנת לאפשר את הכפתור וקיבלנו בתצוגה כפתור “Passwords Recovery”. לחיצה על הכפתור חשפה תיאור של ה-challenge שעלינו לבצע עם אפשרות להעלאת קובץ exe.

* כתבנו קוד C הפותר את ה-challenge המבוקש. להלן הקוד המקורי:



לאחר קומפילציה, קיבלנו קובץ executable במשקל 103KB. עם נסיון העלאתו, קיבלנו הודעת שגיאה כי יש להעלות קובץ שגודלו לא עולה על 2KB.

* המרנו את הקוד הנ"ל לאסמבלי ע"י godbolt.org. נעזרנו בקוד שראינו בתרגול לביצוע הדפסה ע"י מציאת הפונקציות ויישמנו זאת עבור printf ו-scanf. התהליך כולו כלל את טעינת msvcrt.dll בצורה דינאמית ע"י LoadLibraryA, חיפוש כל אחת מהפונקציות ב-dll ע"י GetProcAddress שאותה מצאנו באמצעות הפונקציה search\_kernel32. את הפונקציה search\_kernel32, לה קראנו בקוד FindFunction, קיבלנו באתר הקורס כבר כ-hexa ולכן רק השארנו לה label תואם בסוף האסמבלי. עבור כל אחת מהפונקציות הנ"ל, יצרנו שגרות GetPrintF ו-GetScanF המוצאות את הכתובת לפונקציה ומחזירות אותה על גבי הרגיסטר eax.
* שתלנו בקוד האמסבלי שלנו את הפונקציות שיצרנו והחלפנו כל קריאה סטנדרטית ל-scanf ול-printf בקריאה לפונקציות היעודיות שיצרנו. בנוסף, הסרנו כל שימוש במחרוזות ובמקום דחפנו את הקידוד שלהן על המחסנית.
* המרנו את האסמבלי שקיבלנו ל-hexa ע"י שימוש באסמבלר של defuse ועברנו ליצירת קובץ PE. ראשית, נשים לב כי לא השתמשנו בפונקציות חיצוניות בצורה סטטית ולכן אין טבלת import, ואיננו מייצאים פונקציות ועל כן אין טבלת export. בנוסף, לא השתמשנו במחרוזות ודחפנו במקום על המחסנית את קידודן ועל כן גם אין לנו data section. כלומר, כל שיופיע הוא ה-header וה-text section שמכיל את הקוד, בדומה למה שראינו בתרגול. לכן, השתמשנו ב-header שהופיע בתרגול היות ואין צורך לשנות בו כלום, ואילו רק בטבלת ה-sections עלינו לשנות את המידע אודות ה-text section שלנו המכיל את הקוד. לכן, רצינו להגיע לתחילת ה-section. שמו הוא “.text”, שזה ב-hexa מתורגם כ-“2E74657874” ולכן מצאנו ערך זה וקפצנו ישירות אליו (ב-PE הקיים מהדוגמה שראינו). הערכים הבאים מוגדרים כ-DWORD, כלומר מילה כפולה – 32 ביט, ולכן נעבור עליהם לפי כפולות של 8 ערכי hexa. בפועל, הערכים היחידים שעלינו לשנות הם הערכים המתארים את גודל ה-section, כלומר גודל הקוד, שהם VirtualSize ו-SizeOfRawData, שהם הערכים הראשון והשלישי בהתאמה לאחר שם ה-section. גודל הקוד שלנו הוא 853B וגודל הפונקציה search\_kernel32 הוא 146B, כלומר ביחד הם 999B. בייצוג hexa של שני בתים זה מתפרש כ-0x03e7 וכמו שראינו בתרגול הבית העליון הוא הימני והתחתון הוא השמאלי ולכן הפכנו את הבתים כך שהגודל הוא e703. שינינו את שני הגדלים ב-header לגודל שהתקבל, והשתמשנו ב-hexed.it שבו שתלנו את ה-header החדש, אחריו את הקוד שלנו ואחריו את הפונקציה search\_kernel. ייצאנו לקובץ challenge.exe שגודלו 2KB והעלינו אותו לאתר.