1 הנדסה לאחור 236496 – תרגיל בית

מגישים: שליו ריסין, אלון פליסקוב

חלק ב – היכרות ראשונית עם קובץ C מהודר

. partb.opt.s ו-partb.nopt.s

: partb.nopt.s בקובץ

- 1. בלייבל LC2, מופיעה המחרוזת "0\0"% במקום המחרוזת "0\%". בגלל טעות זו, בפונקציה ascii, כאשר המשתמש מכניס מספר לקלט, אז המספר נקרא בתור מחרוזת של ספרות ascii, כאשר המשתמש מכניס מספר לקלט, אז המספר נקרא בתור מספר (int). בנוסף, תיקרא רק ספרה אחת מהקלט במקרה הזה, כי מחרוזת הפורמט 5% מתאימה לקליטת תו יחיד.
- לפני הקריאה ל-scanf, לא נדחפים הפרמטרים למחסנית גם הכתובת של המשתנה בו יישמר קלט המשתמש, וגם המחרוזת המתאימה ל-scanf, שבמקרה זה היא "%d".
 המחרוזת שמועברת ל-scanf היא אותה המחרוזת שהועברה ל-printf, שהיא scanf. מכיוון שלא מופיעים במחרוזת אחוזים כגון %d %c לא ייקלט קלט "enter your guess."
 מהמשתמש. על מנת לתקן את הטעויות, הוספנו שתי שורות לקוד, המסומנות בהערות.

```
mov DWORD PTR [esp], OFFSET FLAT:LC1
call _printf
lea eax, [ebp-8]
mov [esp+4], eax # first addition
mov DWORD PTR [esp], OFFSET FLAT:LC2 # second addition
call _scanf
```

ללייבל L3 אמורים להגיע אחרי ניחוש לא מוצלח, וממנו אמורים לקפוץ חזרה לנסיון הניחוש – L2, אך פקודת ה-jmp בסוף L3 לא קיימת, לכן כל נסיון ניחוש של המשתמש היה מצליח.
 כדי לתקן את טעות זו, הוספנו פקודה "jmp L6" תחת הלייבל L3.
 כעת, אחרי שהוספנו את השורה הזו, יש להוסיף גם את השורה "jmp L8" לפני הלייבל L3, כדי שבעת ניחוש מוצלח (L4), נקפוץ לסיום התכנית. ולא חזרה ל-L6.

: partb.opt.s בקובץ.

נשים לב, כי נקראה הפונקציה time ונקראה הפונקציה rand, אך לא נקראה הפונקציה time.
 לא שימוש ב-srand לפניה, מובילה לכך שה-seed קבוע ושווה ל-1, כלומר שבכל הרצה של התכנית, המספר ה"רנדומלי" יהיה זהה. תיקנו זאת על ידי הוספת שתי שורות:

```
mov DWORD PTR [esp], 0

call _time

mov [esp], eax # first addition

call _srand # second addition

call _rand
```

כעת תיקרא srand עם הפלט של time כפרמטר, מה שיוביל לריצה המצופה של התכנית.

2. בשורה 42 (לאחר הוספת שתי השורות לעיל), מתבצעת קריאה ל-printf עם המחרוזת puts .puts אחרי שכבר הדפסנו את המחרוזת הזו עם "Guess a number between 1 and 100\0" החלפנו את הלייבל של המחרוזת בשורה 42 ללייבל המתאים כך שתודפס המחרוזת המתאימה: "Enter your guess:\0"

```
mov DWORD PTR [esp], OFFSET FLAT:LC1 # was LC0
```

3. בשורה 47 (לאחר הוספת שתי השורות), רשומה הפקודה "jge L2". הפקודה תגרום לכך שלא אפשר יהיה להגיע ל-L3/L5, כלומר לא ניתן לסיים את ריצת התכנית. בנוסף, זה לא מתאים ללוגיקה של התכנית. לכן, החלפנו את הפקודה ב-"jge L3" – ששם בודקים האם הניחוש נכון ואפשר לסיים את התכנית, או שהמספר קטן מדי, ויש לנסות שנית.

```
47 jge L3 # Mistake, was L2
```

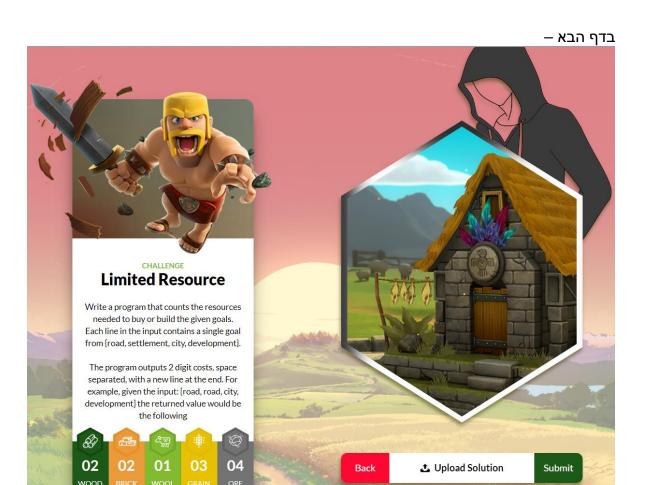
חלק ג – היריבים של קטאן

– אשית, נכנסנו לאתר, פתחנו את קוד ה-HTML של האתר, וזיהינו כפתור חבוי

```
<div class="button disabled" onclick="challenge_me()">
Passwords Recovery</div>
```

לאחר מחיקת המילה "disabled", הכפתור נגלה ולחיצה עליו הובילה לאתר ה- disabled", הרכפעור נגלה ולחיצה עליו הובילה לאתר ה- recovery.

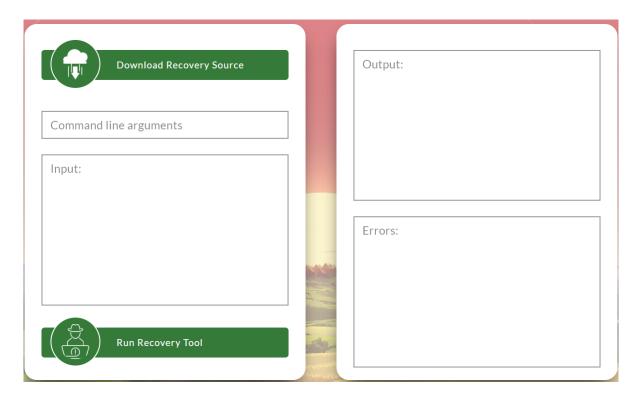




נתבקשנו לכתוב תכנית מסוימת ולכווצה ל-2KB ומטה:

כתבנו תכנית פשוטה ב-C שעושה את הנדרש. העברנו אותה ב-compiler explorer באינטרנט, כדי לקבל קוד אסמבלי. לאחר מכן, שינינו את כל הקריאות ל-printf \ scanf, וטענו אותן דינמית כפי שהראו בהדגמה בתרגול 2 (LoadLibraryA וכן הלאה). בקוד האסמבלי, המחרוזות היו נתונות תחת לייבלים, ושינינו זאת לדחיפה ידנית על המחסנית תוך כדי ריצה – כפי שראינו בהדגמה.

קמפלנו את קובץ האסמבלי לקובץ בינארי וחיברנו אותו למימוש של FindFunction, ואחר כך ל-exe מתאים, תוך התאמת גודל ה-text section. העלינו את קובץ ה-exe לאתר, ועברנו לשלב הבא:



הורדנו את ה-recovery source, והתחלנו לקרוא אותו.

<u>:level 1</u>

תחילה, הסתכלנו על "main", ושמנו לב שלפני ההדפסה של "Level 1 passed!", משווים את הארגומנט הראשון שהפונקציה מקבלת, עם 1. גילינו שזו השוואה שבודקת האם argc גדול מ-1, ואם כן אז עברנו את השלב הראשון. מכיוון שתמיד $c \geq argc$, כי תמיד שם התכנית הוא הארגומנט הראשון, אז השלב הראשון יעבור אם הכנסנו לפחות ארגומנט אחד נוסף.

:level 2

לפני הקריאה ל-level1, מתבצעת קריאה ל-atoi, על האיבר השני במערך argv – שהוא בעצם הקריאה ל-level2 הראשון. את ערך החזרה (integer) מעבירים כארגומנט ל-level2. command line argument של מנת לקבל start offset שאותם מוסיפים לכתובת של התכנית קוראת ל-scanf על מנת לקבל alignment לגודל של 4 בתים.

לאחר מכן, נקראת הפונקציה printArray במידה והקלט שהכנסנו תקין – שהמספר הראשון הוא אי-שלילי. הפונקציה הזו מדפיסה למסך את תוכן הזכרון בין שתי הכתובות הללו. השתמשנו בפלט הפונקציה כדי לראות מה בדיוק קיים בזכרון, והשתמשנו במידע הזה כדי לעבור את רמה 2

ראינו כי על מנת שיודפס "Level 2 passed!", עלינו לעבור בהצלחה סדרה של 9 השוואות בין בתים שונים בזכרון. כברירת מחדל, ההשוואות לא עוברות – הבתים אינם זהים.

סדרת הבתים הראשונה: ראינו כי עם seed מקובע (0), אנחנו מקבלים באופן עקבי את 12 הבתים הראשונים) הבאים, על ידי קריאה ל-printArray בריצת התכנית:

042C5444 5FAC5A1F 04AF1079

סדרת הבתים השנייה:

המספרים שנקבעו מראש בתכנית, והם:

```
403 | mov DWORD PTR [ebp-46], 0x4FC3698C
404 | mov DWORD PTR [ebp-42], 0x5444BA19
405 | mov BYTE PTR [ebp-38], 0x2C
```

ראשית, שמים לב כי הסדרה הראשונה באורך 12 והשנייה באורך 9, לכן 3 בתים מהסדרה הראשונים ברצף – הראשונה הם מיותרים, ולא נערכת איתם השוואה. אלה הם שלושה הבתים הראשונים ברצף – התייחסנו אליהם כ-don't-care.

נתבונן כעת בחלק הזה של התכנית, שרץ במסגרת level2:

```
L38:
   lea eax, [ebp-52] # address of hexa variable
   mov DWORD PTR [esp+4], eax # push hexa variable address
   mov DWORD PTR [esp], OFFSET FLAT:LC16 # string "%X"
   call
           scanf
   mov eax, DWORD PTR [ebp-8] # eax = start address
   mov eax, DWORD PTR [eax] # eax = *eax
   mov edx, eax
                               # edx = array[start]
   mov eax, DWORD PTR [ebp-52] # eax = hexa (4 bytes)
   xor eax, edx
                               # eax = eax xor edx
                               \# edx = eax
   mov edx, eax
   mov eax, DWORD PTR [ebp-8] # eax = start address
   mov DWORD PTR [eax], edx
   add DWORD PTR [ebp-8], 4
                             # address += 4
```

החלק הזה מתבצע בתכנית מספר פעמים כמספר הdouble words שמודפסות למסך בארק הצימלי. הקלט (4 בתים) עובר scanf. הפונקציה צימלי את כדי לקרוא קלט הקסה דצימלי. הקלט (4 בתים) עובר מעולת xor יחד עם 4 בתים מתאימים במערך, והתוצאה דורסת את תוכן המערך.

למשל:

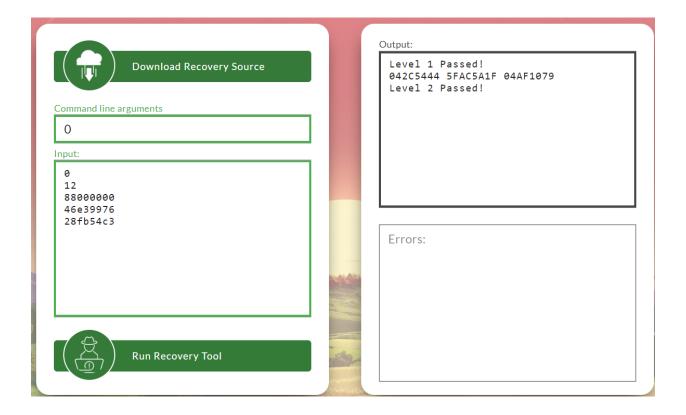
הקלט הראשון שהכנסנו הוא 0x88000000.

מתבצע :0x88000000 XOR 042C5444 . התוצאה של ה-xor. כעת, דורסים את מתבצע :0x88000000 . התוצאה של ה-8C2C5444 . מתבצע

.C XOR B = A אז גם A XOR B = C מקיימת שאם xor-ידענו כי פעולת ה

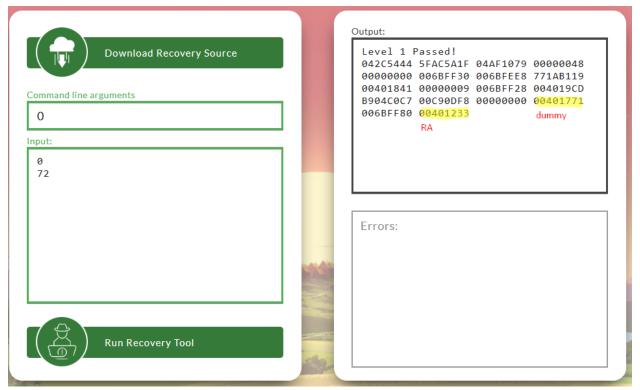
אז הסקנו כי אם נבצע XOR בין סדרת המספרים הראשונה לסדרת המספרים השנייה, תוך כדי התעלמות משלושת הבתים הראשונים בסדרה הראשונה – אז נקבל סדרת קלט מתאימה. ואז כשנכניס את הקלט הזה לתכנית בחלק של L38, תוכן המערך בזכרון(סדרת הבתים הראשונה) יידרס עם סדרת הבתים השנייה – מה שיגרום לשתי סדרות הבתים להיות זהות, ובסוף לסיום השלב.

הוכחה לסיום השלב השני:



:Level 3

לאחר התבוננות נוספת בקוד ראינו שצריך לקרוא לפונקציה ___dummy__ על מנת לעבור את שלב 3. לצערנו הרב (⊗) הפונקציה אינה נקראת באף שורה בקוד. לכן הרעיון היה לשנות את כתובת החזרה מmain כדי שתהיה הכתובת של dummy (חשבנו גם על לשנות את כתובת החזרה של level 2 אבל כבר ראינו שנצטרך לשלב הבא את המשך main ולכן הלכנו עם הגישה הראשונה) בשביל לכתוב לאכתוב למmmy השתמשנו בחלק משלב 2 אשר כותב על המחסנית. כעת היינו צריכים לגלות מה הכתובת של dummy ומה הכתובת חזרה מmain (נזכור שהכתיבה בשלב 2 איא מיו אולא פשוט החלפת התוכן לכן צריך לדעת את שניהם). לפי הקוד, בתחילת main הכתובת נשמרת על המחסנית בדיוק 8 בתים מתחת לreturn address מלא מול השלב 1 ניתן את המספרים 0 ו-print array 72 תכתוב מספיק בתים מהמחסנית ונוכל שאם בשלב 2 ניתן את המספרים 0 ו-print array 72 תכתוב מספיק בתים מהמחסנית ונוכל dummy.



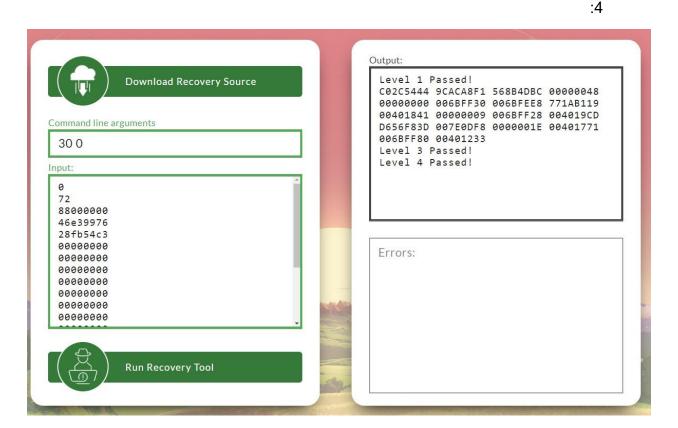
שקוראת maina לחזרה בין level2 בנוסף, ישנה שורה בין 00401771 xor 00401233 = 542. segfault מהרת מקבלים command line argument השני ולכן הוספנו ארגומנט נוסף אחרת מקבלים command line argument. לכן עבור הלט:

Output: Level 1 Passed! 042C5444 5FAC5A1F 04AF1079 00000048 00000000 006BFF30 006BFEE8 771AB119 00401841 00000009 006BFF28 004019CD B904C0C7 00720DF8 00000000 00401771 006BFF80 00401233 Level 2 Passed! Level 3 Passed!

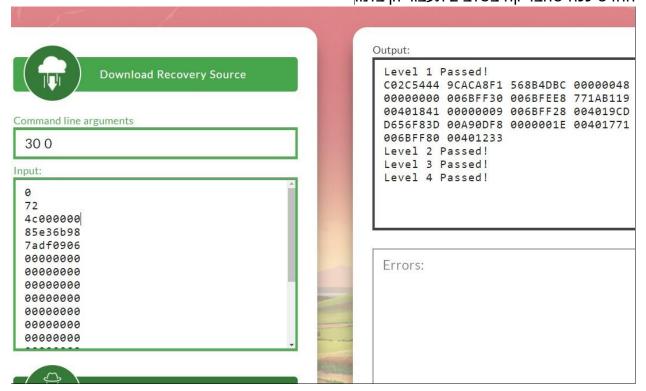
<u>:Level 4</u>

בתחילת הקוד קוראים לפונקציה signal עם הפונקציה handler ומספר הסיגנל 8. הקריאה מגדירה כי כעת שגרת הטיפול לסיגנל מספר 8 (floating point exception) שמתרחש בחלוקה ב 0. היא handler.

החלוקה היחידה בתכנית הינה בפונקציה dummy שבה מחלקים בdivider, אשר ערכו מתקבל על ידי שימוש בארגומנט הראשון בתור seed להגרלת מספר רנדומלי. לאחר מכן יש מניפולציות אלגבריות על התוצאה כשלקראת הסוף עושים AND עם המספר 31. כלומר, modulo 32. לכן ניחשנו מספרים עבור הארגומנט הראשון עד שהמספר שנכנס לdivider היה 0 ועברנו את שלב



אך כעת, מכיוון ששינינו את הארגומנט הראשון גם הseed בשלב 2 השתנה ולכן המידע בstack array אך כעת, מכיוון ששינינו את הארגומנט הראשון גם השמוגרל באקראי השתנה ואנחנו לא עוברים את הבדיקה של השלב. לכן התאמנו את input למידע החדש ככה שהבדיקה בשלב 2 תעבור וקיבלנו:



:db_access

ראינו כי בקוד של dummy נקראת הפונקציה db_access. חשדנו שבעזרתה נוכל לחלץ את פרטי ההתחברות של המשתמשים.

ראינו שלפני db access, מופיעה השאילתא הבאה:

```
124 v LC7:
125 ascii "select username, password from users where username='\0"
```

ותוך כדי ריצת db access, מתבצע:

- .database- חיבור ל
- .database- שאילתא ל-
- .3 סגירת החיבור ל-database.

השאילתא ששואלים הינה:

select username, password from users where username=' || argv[2] || '

כאשר ראינו את השאילתא שנוצרת, עלה לנו הרעיון של SQL Injection, כלומר – אם נוכל להכניס argv[2], כלומר – אז יחזרו כל ה- ארגומנט מתאים ב-argv[2], כך שהתנאי של ה-where תמיד יתקיים (טאוטולוגיה) אז יחזרו כל ה- usernames

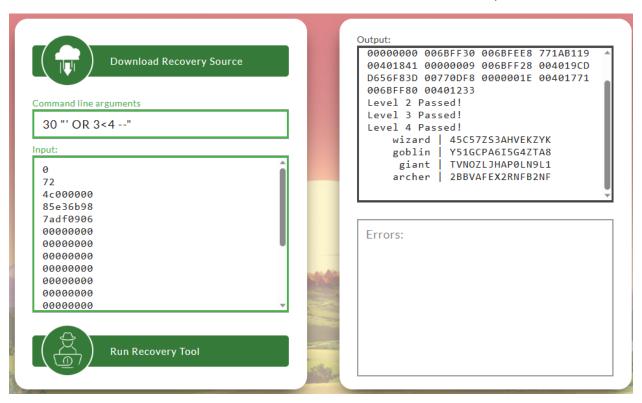
בנוסף, לפי הפונקציה, הוגבלנו ב-11 תווים (לפי שורה 159) –

```
mov eax, DWORD PTR _arg
mov DWORD PTR [esp+8], 11
mov DWORD PTR [esp+4], eax
lea eax, [ebp-276]
mov DWORD PTR [esp], eax
call _strncat
```

בשביל לבצע SQL Injection, המחרוזת צריכה להכיל:

- 1. גרש (התו ') שסוגר את הגרש שכבר קיים במחרוזת ואז מקבלים empty string.
 - 2. תנאי נוסף ל-where כך שיהיה טאוטולוגיה.
 - 3. הערת שורה (שני מקפים --) לביטול הגרש שמתווסף אחר כך.

הגענו למחרוזת הבאה, כפי שרואים בתמונה:



קיבלנו פרטים של ארבעה משתמשים, והצלחנו להתחבר בעזרתם למערכת.