Reverse Engineering HW1:

Rebecca Shahar Itzkovich 316276450

Mori Levinzon 308328467

PART B:

The problems with partb.nopt:

- The first parameter of the scanf should be an integer and not char, so we changed the letter from %c to %d.
- Scanf did not receive any parameters for his call and the calling program should have pushed the string for the required type to read (LC2) and the variable pointer to store the input, so we added these lines
- No matter what we inserted as input the program ended successfully, which could be interpreted as one of two ways: Either we were right at each guess (Not likely due to the number of times we run the executable) or there is a wrong jump after the number comparison to the randomized number.

The problems with partb.opt:

- The second print should be "Guess the number" we changed the second print to print the correct text (LC1)
- We did not receive a feedback if out number was bigger than the number we guessed, and the program went right to the part indicating to insert another number. We changed the jump from L2 to L3.
- The randomized number was exactly the same in each run. A call for "srand" was missing, so we added it with the appropriate parameter pushed (the result from the time(NULL) call).

חלק ג

חקירת האתר

התחלנו את דרכנו בכניסה לאתר של קטאן וחקירת קוד המקור (F12) ומצאנו כפתור לא מאופשר:

```
▶ <div style="position: relative;">…</div>
▶ <div style="position: relative;">…</div>
  <div class="button" onclick="attempt_login()">Login</div>
  <div class="button disabled" onclick="challenge_me()">Passwords Recovery
  </div> == $0
  <div class="alert">Please log in to access this page.</div>
</form>
```

הפכנו אותו למאופשר וגילינו כפתור של Password Recovery שהביא אותנו אל עמוד שמספר על Challenge. אצנו רצנו לייצר את הקוד כפי שביקשו, ייצאנו אותו לקובץ הרצה והגשנו לאתר אך אבוי, נודע לנו כי יש הגבלה של 2kb על גודל קובץ ההרצה.

יצירת הPE

ישבנו וחשבנו איך לייצר קובץ רזה יותר שיעמוד בדרישות והחלטנו לבנות לו את הPE בעצמנו כמו שראינו בתרגול.
השתמשנו בדוגמאות של print_handmade ו exit_handmade כדי להבין איך צריך להיראות הקובץ שלנו מבחינת headers, data directories ו- section table והוספנו את הקוד הבינארי שלנו ב text section ב200 offset ולאחר מכן את הבינארי של find function שסופק לנו (כל הנ"ל שורשרו לבניית הPE).
את קוד האסמבלי שלנו ייצרנו בעזרת Godbolt ושינינו אותו מעט כדי לא להשתמש בdata section ולחסוך מקום. במהלך הקוד השתמשנו בפונקציות ספריה כמו scanf וprintf. בעזרת find_function, load_library, ו- get_proc_address מצאנו את הכתובות שלהם בkernel32 ושמרנו אותן ברגיסטרים, ובעזרתם קראנו לפונקציות הספריה מבלי להשתמש בdata section.
לאחר שסיימנו לכתוב את הקוד, העברנו אותו לhex בעזרת האתר Defuse.ca והכנסנו את הפלט לPE שיצרנו.
הtext section הסתיים ב402 offset ולכן שינינו את הraw size והvirtual size ל202 בsection table (בעזרת הCFF Explorer).

כשסוף סוף הצלחנו לגרום לPE שלנו לעבוד ולהיות בגודל פצפון של 1KB (!!!) העלנו אותו לאתר והגענו לעמוד של הrecovery וממנו הורדנו את crackme.exe.

ניתוח הcrackme:

לאחר שניסינו לקמפל אותו וראינו שחסרה ספרייה sqlite3.dll, הורדנו אותה מהאינטרנט וקימפלנו שוב עם השורה gcc -o crackme crackme.S sqlite3.dll והכנסנו את קובץ ההרצה שנוצר לida.

Level 1

ראינו שיש בmain של התכנית קריאה לשגרה level1 שמקבלת כפרמטר את מספר הארגומנטים שמוקלד בהרצה של הקובץ (argc) ושהשגרה מדפיסה Level 1 Passed אם יש לפחות ארגומנט 1.

Level 2

בתחילת השגרה level2 יש השמה של 9 מספרים בגודל בית 1 כל אחד במערך במחסנית שנסמן את כתובת התחלתו arr. לאחר מכן השגרה מבצעת לולאה שרצה 9 פעמים ומגרילה 9 מספרים בגודל בית 1 כל אחד בעזרת rand_32 כתלות בseed שהוא הארגומנט הראשון שמוקלד בהרצה של הקובץ (מספרים אלה קבועים עבור כל seed). 9 המספרים המוגרלים נשמרים גם הם במערך במחסנית שנסמן את

כתובת התחלתו rand_arr. לאחר מכן נקלטים 2 מספרים מהמשתמש, נסמנם n1, n2. השלב הבא הוא קריאה לprint_array שמדפיסה תוכן רצוף מהמחסנית בין 2 כתובות שמחושבות בעזרת n1 ו-n2.
שמנו לב שאם נקלוט n1=0 נקבל את $\left\lceil \frac{n2}{4} \right\rceil$ האיברים הראשונים שבמערך rand_arr.
לאחר מכן, מתבצעת לולאה של קליטת $\left\lceil \frac{n2-n1}{4} \right\rceil$מספרים הקסדצימליים שעושה על כל אחד מהם xor עם המספרים מrand_arr ומאחסנת אותם במערך rand_arr במקום המספרים המוגרלים.
לאחר מכן מתבצעת השוואה של 9 איברי rand_arr (לאחר השינוי) עם 9 איברי arr לפי האינדקסים ומודפס Level 2 Passed אם כל האיברים שווים.
מכאן הסקנו כי הקלט ההקסדצימלי ללולאה שעושה xor צריך להיות מספר שכאשר עושים xor בינו ובין המספר המוגרל בrand_arr ייתן לנו את המספר המתאים בarr ולכן תכננו את הקלט שלנו בהתאם.

<u>Level 3</u>

מצאנו את ההדפסה של level3 בתוך השגרה dummy שאליה לא מצאנו קריאה בקוד, ולכן הסקנו שנצטרך לעשות buffer overflow על מנת לדרוס את כתובת החזרה מאחת הפונקציות ולשנות אותה לכתובת ההתחלה של dummy. מאחר וגילינו שבשלב 2 אנחנו יכולים להדפיס את תוכן המחסנית למשתמש בעזרת הינדוס הקלט של n1 ו-n2 (נתנו טווח גדול ביניהם, n1=0, n2=72), חישבנו בעזרת הקוד את הoffset של כתובת החזרה מmain ואת הכתובת של dummy (שנשמרה במחסנית בשלב מוקדם יותר בmain), ולאחר הדפסת תוכן המחסנית יכלנו לראות איזה קלט צריך לתת בשגרה של שלב 2 כדי שxor בינו ובין כתובת החזרה מmain ייתן את הכתובת של dummy ובנוסף יתאים לקלט שצריך כדי לעבור את שלב 2.
לאחר סיום השגרה level2, ריצת הקוד חזרה לmain ונעשו כל מיני פעולות חישוב עד שהגיעה לסוף ובret קפצה לdummy והדפיסה את Level 3 Passed.

<u>Level 4</u>

שמנו לב שבתחילת הקוד הוגדרה שגרת handler באמצעות פונקציית signal עבור סיגנל מספר 8- Floating Point Exception ובhandler נמצאת ההדפסה של Level 4 Passed. הסקנו מכך שעל מנת לעבור את שלב 4 יש צורך ליזום חריגה מסוג fpe ומה יותר מוכר מחלוקה ב0?
אז חיפשנו פעולת div בקוד ומצאנו אותה בdummy אחרי ההדפסה של שלב 3 (כמה נוח!!!) וראינו שמתבצעת חלוקה במספר ששמור בdivider. על מנת שתהיה חלוקה ב0 רצינו שdivider יכיל 0 וראינו שהתוכן שנקבע לו תלוי בחישוב בmain שתלוי בseed של rand_32 הוא הארגומנט הראשון המוקלד בהרצת הקובץ. אז עשינו את הדבר היעיל והרצנו את הקוד עם הרבה דוגמאות של ארגומנט ראשון עד שקרתה fpe והודפס Level 4 Passed. לאחר שמצענו את הארגומנט הזה (מצאנו כל מיני, אחד מהם היה 44) התאמנו מחדש את הקלט בlevel2 כדי שגם הוא יעבוד (כי המספרים הרנדומליים שם גם תלויים באותו seed) וככה הסתיים לו שלב 4.

<u>מציאת רשימת המשתמשים והסיסמאות</u>

לאחר הfpe משלב קודם, קוד הhandler שינה את תוכן הdivider לערך השונה מ0 וקפץ חזרה לנקודה שבה מכניסים את divider לרגיסטר לפני שמחלקים בו ולכן בפעם השנייה לא התבצעה חלוקה ב0 והקוד של dummy המשיך כרגיל עד לקריאה לdc_access.
ראינו שבdb_access יש מחרוזת של שאילתה לא מושלמת והסקנו שאנחנו צריכים להשלים אותה בעזרת הקלט שלנו והבחנו שבקוד משרשרים את arg_ למחרוזת של השאילתה וכן שבmain מכניסים לarg_ את הארגומנט השני שלנו מהרצת הקובץ ולכן בדקנו קצת בגוגל על sql injection 101 וגילינו שאפשר לגרום להדפסה של שמות המשתמש והסיסמאות של כל המשתמשים על ידי הצבת username = '' OR '1'='1'

בסוף השאילתה. אז הצבנו את זה ואבוי! גילינו שזה לא עובד! חפרנו עוד קצת בקוד וראינו שהוא מוסיף לבד גרש משמאל וגרש מימין ולכן סידרנו את זה מחדש ואז הודפסה לנו הטבלה הנכספת.