Reverse HW 2

התחברנו לאתר בעזרת הcredentials מהתרגיל הקודם, במשתמשים goblin, giant, wizard היו קבצים פרטיים וגם קבצים פומביים משותפים לכולם.

Goblin

תחילה, כאשר ניגשנו לקובץ ניסינו להבין איפה מתחיל הmain. ניסינו להסתכל על המחרוזות בתכנית, אך לא הצלחנו באופן מובהק להבין מהן איזו פונקציה היא הmain, ולכן בסופו של דבר הסקנו איזה פונקציה היא הmain לפי קריאה של הstart. חיפשנו את הפונקציה שמקבלת את envp, argv, argc, ובעזרת מידע נוסף מהIDA הגענו לפונקציה הנכונה:

```
mov     eax, [eax]
mov     [esp+1Ch+envp], eax ; envp
mov     eax, ds:argv
mov     [esp+1Ch+Mode], eax ; argv
mov     eax, ds:argc
mov     [esp+1Ch+lpTopLevelExceptionFilter], eax ; argc
call    main
mov     ebx, eax
call    _cexit
mov     [esp+1Ch+lpTopLevelExceptionFilter], ebx ; uExitCode
call    ExitProcess
```

לאחר מכן, הסתכלנו על תחילת הריצה של main. תחילה, היא קוראת לפונקציה שלא ממש משפיעה על הלוגיקה של התכנית. לאחר מכן, main קוראת לפונקציה שבהמשך נתנו לה את השם PrepareMaze.

הפונקציה PrepareMaze: ניתן לראות כי הפונקציה כותבת לרצף של 64 בתים בזכרון.
אחרי בחינה מעמיקה, הגענו למסקנה שמה שהיא כותבת לזכרון נראה ככה:

```
.2.F.7..
XXXXXXXX
......4.
1.......
........
XXXXXXXX
.......F
........
```

תחילה חשבנו שהרצף הזה מייצג את המפתח שאנחנו מחפשים, ושבמהלך ריצת התכנית מתמלאים "הערכים הנכונים" . לקראת סוף הניתוח של הקובץ, הבנו שמדובר במבוך.

בנוסף, על ידי התבוננות בחלק הבא של הפונקציה:

```
var_C= dword ptr -0Ch
Val= dword ptr -8
Size= dword ptr -4
row= dword ptr  8
column= dword ptr  0Ch
sideways_direction= dword ptr  10h

push    ebp
mov     ebp, esp
sub     esp, 0Ch        ; prologue
mov     eax, [ebp+sideways_direction]
mov     dword ptr [eax], 4Ch ; 'L'
mov     eax, [ebp+column]
mov     dword ptr [eax], 7
mov     eax, [ebp+row]
mov     dword ptr [eax], 0
mov     ds:column_goal_0, 0
mov     ds:row_goal_7, 7
mov     ds:mem3, 0      ; the above section writes to non corelated-bytes
```

הגענו למסקנה שהפונקציה מקבלת כפרמטרים שני מצביעים ל-int ומצביע ל-char. בהמשך הבנו את מטרת הפרמטרים הללו – בעזרתם הפונקציה מאתחלת את המיקום ההתחלתי שלנו במבוך (שורה ועמודה) ואת הכיוון ההתחלתי שבו אנו זזים (בתמונה זהו התו 'L').

הפונקציה מאתחלת (בלי קשר לפרמטרים) את מיקום המטרה במבוך, ואת מונה הדגלים.

בכך סיימנו את ניתוח הפונקציה PrepareMaze.

המשך main:

בתחילת main, מאותחלים שני משתנים. הסקנו שתפקידיהם הם של counter ו-limit של לולאה, וכעת הגענו לניתוח תוכן הלולאה.

הלולאה רצה עד 7 פעמים. איטרציה של הלולאה נראית כך:

תחילה, קוראים לפונקציה שמאוחר יותר קראנו לה GetCommand. נפרט על ניתוחה כעת:

GetCommand:

הפונקציה מקבלת שני פרמטרים – *char, int.

ה-*char משמש לאחסון ה-command שניתן על ידי המשתמש. ה-*int משמש לאחסון מספר.

הפונקציה קולטת תו מהמשתמש ושומרת אותו בפרמטר ה-*char. אחרי כן, יש קריאה לפונקציה נוספת. בהתחלה לא כל כך הבנו מה הפונקציה הנוספת עושה, אך כן הבנו שהיא לא משפיעה על הלוגיקה של התכנית(מאוחר יותר גילינו שהיא מקודדת את המפתח בהתאם למהלכים במבוך).

לאחר קליטת תו מהמשתמש ב-GetCommand – ראינו כי אם התו אינו 'C' או 'D' או 'U' – ריצת התכנית מסתיימת.
אם התו הוא 'C', אז קוראים תו נומרי שמתורגם למספר בין 0(כולל) ל-9(כולל).

המשך הלולאה ב-main: יש שני מסלולים בלולאה, וההחלטה לאיזה מסלול לגשת נקבעת על פי התו שנקלט ב-GetCommand.

מסלול א' – אם התו שנקלט אינו 'C':

נקראת הפונקציה שמאוחר יותר קראנו לה MoveRow. נפרט עליה כעת.

הפונקציה MoveRow: מקבלת *int, int, char.

בעזרת הפרמטרים המספריים, הפונקציה מחשבת את המיקום הנוכחי במבוך ובודקת שאין במיקום זה את התו 'X' או '.'(נקודה). אחרת התכנית מסתיימת.
אחר כך, בהתאם להאם התו הוא 'U' או 'D', משנים את ערך הארגומנט הראשון לפי התו שנמצא במיקום הנוכחי.
בשלב זה של הניתוח, עדיין חשבנו שהמבוך הוא מפתח, אך התחלנו לפקפק בכך, כי הייתה לנו הרגשה ש-U ו-D מציינים Up או Down בהתאמה, לפי הצורה שבה משנים את הארגומנט הראשון (מספר שורה).

לאחר שינוי הארגומנט הראשון, קוראים איתו לפונקציה אחרת, שקראנו לה PathBlockCheck. פונקציה זו בודקת שהמיקום החדש הוא בתחומי הלוח ושהמיקום החדש אינו מיקום של תו 'X'. אחרת התכנית מסתיימת.

עברנו כעת למסלול האחר בלולאה ב-main. מסלול זה מתרחש אם התו שנקלט מהמשתמש הוא 'C'. במסלול זה נקראת פונקציה חדשה, שקראנו לה מאוחר יותר MoveInSidewayDirection.
הפונקציה MoveInSidewayDirection: מקבלת ארבעה פרמטרים – int*, int*, char*, int.

מאוחר יותר הבנו שהפרמטרים מהווים את מספר השורה, מספר העמודה, כיוון התזוזה, ומספר הצעדים.

הפונקציה קוראת לעצמה באופן רקורסיבי כמספר הצעדים. בכל צעד רקורסיבי, היא בודקת את כיוון התנועה ולפיו היא משנה את מספר העמודה. אם הגענו לקצה המבוך, אז משנים את מספר השורה ואת הכיוון. בנוסף, אם עוברים במשבצת שבה התו 'F', מקדמים את מונה הדגלים.

בשלב זה התמונה התבהרה לנו, והבנו כי מדובר במבוך ולא במפתח, וכי R\L\D\U הם כיווני תנועה. בנוסף, הבנו כי C והמספר מסמלים צעדים \ התקדמות.

מכאן, חזרנו אחורה, התחלנו לתת שמות למשתנים\לפונקציות, ולפענח את חוקי המשחק. בנוסף, סיימנו לפענח את הלולאה ב-main, שבודקת בסופה האם הגענו למשבצת הסיום, והאם אספנו שני דגלים.

חוקי המשחק שהבנו הם:

- אנחנו נמצאים בתוך מבוך 8x8, מתחילים במשבצת [0,7], וכיוון התנועה הוא שמאלה.
- יש במשחק עד שבעה מהלכים.
- בכל שלב, נותנים אחת משלוש פקודות :
  א. U
  ב. D
  ג. C + Number

אם הפקודה היא U או D : מסתכלים על התו במיקום הנוכחי במבוך. עולים או יורדים מספר שורות כמספר התו.

אם הפקודה היא C + Number, אז מתקדמים לפי כיוון התנועה הנוכחי, כ-Number צעדים. במידה ומתקדמים מעבר לגבולות הלוח, אז משנים את כיוון התנועה, ועולים שורה במקום להתקדם לצדדים. לאחר עליית השורה ממשיכים להתקדם את יתרת הצעדים. תוך כדי ההתקדמות, בודקים האם עברנו במשבצת שיש בה דגל 'F', ומקדמים את מונה הדגלים בהתאם.

- כאשר מגיעים למשבצת [7,0], בודקים האם אספנו שני דגלים ואם כן, מודפס מפתח ההצפנה הסופר סודי. 😊

בהתחלה, חשבנו שצריך לעצור בכל משבצת שיש בה F אם אנחנו רוצים לאסוף את הדגל, ומספר המהלכים לא הספיק לנו תחת המגבלות הללו. לכן שינינו את קובץ הריצה כך שיהיו לנו יותר מהלכים.

בהבסת המשחק (לאחר ששינינו את הקובץ) התקבל לנו המפתח הבא:

```
E:\technion\reverse\Reverse2023\HW2\goblin>goblin_safe.exe
C 4 C 2 U C 2 D C 6 U C 1 C 8
TjsQ⊕i↕
```

המפתח שקיבלנו היה נראה לנו משונה, ולכן חזרנו אחורה והבחנו שאפשר לאסוף דגלים במהלך תנועה. פתרנו את המבוך בשבעה צעדים, וקיבלנו את המפתח הבא:

```
E:\technion\reverse\Reverse2023\HW2\goblin>goblin_safe.exe
C 6 U C 2 D C 6 U C 9
The encryption super secret key is ~]{r.E\MEy:c!#;[
```

ההבדל בין המפתחות השונים שקיבלנו נובע מכך שהמפתח נוצר על בסיס הפקודות שאנחנו מכניסים.

Giant

תחילה, בדומה ל-Goblin, הסקנו איזו פונקציה היא ה-main. ראינו כי הmain מכילה 4 קריאות לפונקציות. הראשונה אינה משפיעה על הלוגיקה של התכנית ואחריה 3 קריאות לפונקציות שאינן קשורות אחת לשנייה. כל פונקציה קולטת קלט מסוים, אשר בהינתן הקלט הנכון היא מדפיסה חלק ממשפט התוצאה.

פונקציה 1:

פונקציה זו מקבל ארבעה קלטי char מהמשתמש, ומחברת אותם לint אחד בצורה הבאה:

$$char1|char3|char2|char4$$

בנוסף, היא מגדירה פונקציה נוספת בתור ה-UnhandledExceptionFilter.

לאחר מכן מבצעים חיסור בין המילה "NLUL" לקלט הנוצר וקוראים באמצעות printf מכתובת זו. לאחר מכן יש בדיקה על התוכן של dword_408020 ואם הוא לא 0 יש הדפסה של תחילת המשפט כתלות בקלט.

הפונקציה אשר מגדירים בתור ה-filter, כותבת nops לטקסט ושמה 1 בכתובת dword_408020. לכן היינו רוצים שהיא תיקרא ונקבל את תחילת המשפט.

לשם כך, ניתן את הקלט "NULL" מה שיגרום לחיסור להיות 0 ויגרור null pointer dereference, ה-filter יקרא והמשפט יודפס.

פונקציה 2:

הפונקציה מאתחלת מערך של int על ה-stack, מוסיפה פרמטרים וקוראת לפונקציה רקורסיבית. לאחר מכן, היא עוברת בלולאה ולפי ערכי המספרים בכתובת המערך מדפיסה את המשך משפט התוצאה. השאלה הייתה מה הפונקציה הרקורסיבית עושה ומה הקלט הדרוש בשלב זה?

שמנו לב שבתחילת הפונקציה הרקורסיבית בודקים האם הפרמטר השני קטן מהפרמטר השלישי, ואם כן מחשבים את האמצע ביניהם. חישוב זה הזכיר לנו אלגוריתמים כמו binary search או merge sort. לאחר מכן קוראים 5 בתים ב-Hexa מהמשתמש, משנים את הרשאות הגישה ל-text בכתובת 0x4019BF שיהיו Read, Write, Execute, וכותבים לשם את

הקלט. בהסתכלות לכתובת זו אנו רואים שיש 6 פקודות nop רצופות. בנוסף, שמנו לב שחישוב זה קורה רק בקריאה הראשונה לפונקציה ולא בקריאות רקורסיביות חוזרות.

לאחר מכן מכינים את הארגומנטים וקוראים לפונקציה הרקורסיבית שוב. מכינים שוב את הפרמטרים אך אין קריאה נוספת. הארגומנטים הוכנו באופן הבא:

Recursion(arg0, arg4, average of arg4 and arg8, argC+1)

ובפעם שבה אין קריאה, ויש פקודות nop במקום, הכנת הארגומנטים נראית כך:

(arg0, average of arg4 and arg8 + 1, arg8, argC+1)

בשלב זה הבנו שככל הנראה מדובר ב-merge sort ולכן חסרה הקריאה הרקורסיבית הנוספת.

חישבנו את המרחק הרלטיבי שיש לקפוץ מפקודת ה-nop השישית בעת קריאה רקורסיבית ל-MergeSort. שמנו לב כי קידוד של פקודת call מתחיל ב-E8 ואז 4 בתים של offset:

$$40188E - 4019C4 \ = -136 = \ FFFF\ FECA$$

ולכן הקלט לפונקציה על מנת לבצע את הקריאה היה: E8 CA FE FF FF

שמתורגם ל-call MergeSort בזמן ריצה.

לאחר השלמת הפונקציה MergeSort ריצת פונקציה 2 נותנת את המשך משפט הפלט.

<u>פונקציה 3:</u>

בפונקציה זו ישנם סה"כ ארבעה קלטים אשר לא קשורים זה לזה, וכל אחד מהם נבדק בנפרד. הקלטים הם מספר ב-HEXA ושלושה מספרים נוספים. נתאר את הבדיקות על כל קלט:

קלט 1: HEXA

הפונקציה מתחילה בלקבל 4 בתים ב-HEXA מהמשתמש. לאחר מכן היא בודקת האם המספר שלילי. אם כן, היא עושה לו neg (על ידי חישוב (0 – num)) ושומרת בזכרון. ניתן לפרש זאת כפעולת ערך מוחלט. אולם, בסוף הפונקציה אנו צריכים שהמספר יהיה שלילי. אנחנו פתרנו זאת בשתי דרכים שונות:

א. מציאת מספר x כך ש 0-x עדיין מקודד למספר שלילי. מספר זה הוא 0x80000000.
ב. מכיוון שמספר זה אינו משפיע על הקלט המודפס, גישה נוספת הייתה לשנות את קובץ ה-exe כך שהבדיקה האחרונה תבדוק שהמספר חיובי במקום שלילי, ולכן כל מספר אחר, לדוגמה 10, יעבור את הבדיקה.

קלט 2: int

כל הבדיקות על הקלט הזה הוא שה-byte הראשון שלו הינו EC. המספר 236 יעבוד במקרה זה. חשוב לציין שגם בהדפסת הפלט מסתכלים רק על הבייט הראשון של קלט זה ולכן המספר המדויק לא משנה כל עוד ה-byte הראשון יהיה כמו שרשום לעיל.

קלט 3: int

ההגבלות על מספר זה הן שהוא יהיה גם קטן מאפס וגם שלאחר שנוריד ממנו 1 ונעשה test עם עצמו הפקודה jle לא תקפוץ.

על מנת שהפקודה jle תקפוץ צריך ש ZF=1 או ש OF!=SF, בנוסף ידוע ש-Test מכבה את OF.

ולכן סה"כ אנחנו צריכים שהמספר שיתקבל יהיה חיובי. לכן נשתמש במספר השלילי הקטן ביותר, מה שיגרום ל-integer underflow ויהפוך את המספר לחיובי, כלומר המספר הינו 0x80000000.

קלט 4: int

היינו צריכים לתת קלט אשר יפתור את המשוואה $1 = -25x^2 - 3500x - 122499$, זוהי משוואה ריבועית פשוטה והקלט אשר פותר אותה הינו -70.

כלומר, לאחר הכנסת הקלט,
$80000000 \quad 236 \quad -2147483648 \quad -70$
(כאשר -2147483648_{10} = 0x80000000),
קיבלנו את המשך משפט התוצאה.

סיכום:

סה"כ היו לנו 3 פונקציות. כל אחת מקבלת קלט משלה, פולטת חלק ממשפט התוצאה ואינה קשורה לפונקציות האחרות. עבור הקלט

NULL E8 CA FE FF FF 80000000 236 -2147483648 -70

(שצבועים לפי הפונקציה הרלוונטית) קיבלנו את הפלט

```
E:\technion\reverse\Reverse2023\HW2\giant>safe.exe < input.txt
The encryption params are 8 (rounds) and 125549079 (delta)
```

וסיימנו עם *giant*.

Wizard

כמו בgoblin ו-giant, גם בwizard מצאנו את המain והתעלמנו מהפונקציה הראשונה שנקראת ממנה.

בשונה משני הקבצים הקודמים, שמנו לב שבהרצת הקובץ מודפס pattern של מגן דוד עם סימני שאלה אדומים בנקודת מפגש.

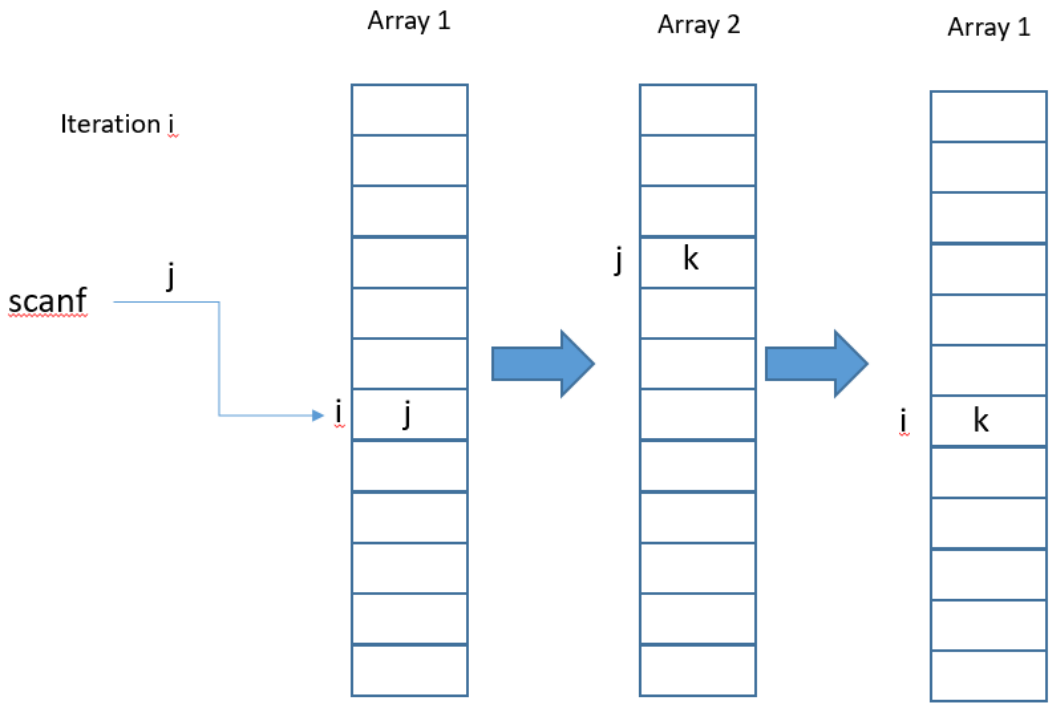
קוד זה מחולק לשתי פונקציות שאינן קשורות זו לזו:

StarOfDavidPuzzle:

בתחילת הפונקציה הראשונה מאתחלים 12 בתים על המחסנית במספרים 1-12 בסדר כלשהו. לצורך ההסבר, נקרא לו מערך 1. בנוסף מאתחלים באפסים 12 בתים נוספים במקום אחר על המחסנית. לצורך ההסבר, נקרא לו מערך 2.

החלק הבא בפונקציה הכיל לולאות רבות והדפסות רבות. ללא הסתכלות מעמיקה, הנחנו כי זהו קטע הקוד אשר מדפיס את מגן הדוד עם הסימנים האדומים.

בשביל להבין איפה אנחנו צריכים לעשות אינטרקציה עם הקוד, חיפשנו איפה scanf נקראת בפונקציה. לאחר חקירה, הבנו כי scanf נקראת בלולאה 12 פעמים, כאשר כל קריאה עובדת בצורה הבאה:



כאשר j הוא מספר גדול שווה מ-0 וקטן שווה מ-11, ויש בדיקה שמוודאת שלא השתמשנו באותו אינדקס פעמיים.

בשלב זה הבנו שבגלל שיש 12 סימני שאלה אדומים, לכן ככל הנראה כל איבר במערך מייצג סימן שאלה.

לאחר מכן ראינו בניתוח שסוכמים 6 רביעיות ושישיה אחת וכדי לעבור את השלב צריך שהסכום של כל רביעייה ושל השישייה יהיו 26. הדבר הראשון שעשינו היה לבדוק מיהם הרביעיות והשישיה. על ידי הכנסת פרמוטציה אקראית בתור קלט, יכולנו למפות את הקלטים לסימני השאלה וגילינו שהרביעיות הן הנקודות שנמצאות על צלעות המשולשים במגן דוד, ושהשישיה היא קצוות המגן.

כתבנו script בפייתון (search.py)אשר עובר על כל הפרמוטציות האפשריות עד שהוא מוצא פתרון שדואג שסכום הרביעיות והשישיה יהיה 26, ואז התאמנו את הקלט כדי שלכל אינדקס יכנס המספר שאנחנו רוצים. למשל, רצינו שהמספר העליון בכוכב יהיה 1, ולכן המספר הראשון שהכנסנו הוא 2, כי באינדקס 2 במערך 2 יש את הספרה 1, וכך הלאה.

לבסוף הבנו שהחלק השני של הפונקציה הוא הדפסת הפתרון.

<u>פונקציה 2:</u>

לפונקציה 2 יש שני חלקים. החלק הראשון הוא החלק בו מצפים לקבל קלט שהוא מחרוזת עם ירידת שורה.

לאחר מכן, מורידים מכל האיברים במחרוזת את הערך המספרי של האות 'A' לפי ASCII. עושים פעולה על המחרוזת ומוסיפים חזרה לכל האיברים 'A'. לאחר מכן משווים את המחרוזת הסופית למחרוזת נתונה, ובמידה והן שוות כותבים את כתובת של הפונקציה על המחסנית לאחר המחרוזת שהכנסנו. לבסוף מעתיקים את 29 הבתים הראשונים של מחרוזת הקלט למקום כלשהו בזיכרון, ומקדמים ב-1 את מה ששמור ב-dword_409024.

הפעולה שעושים על המחרוזת: לאחר חקירה של הפעולה, הכללה חקירה סטטית ודינמית, גילינו שעושים למחרוזת Xor ציקלי לכל שני איברים סמוכים, כאשר לאחרון עושים Xor עם הראשון. נשים לב שעבור האיבר האחרון ה-Xor הוא עם הערך החדש של האיבר הראשון, ולכן:

$$result[28] = result[0]\hat{}str[28]$$

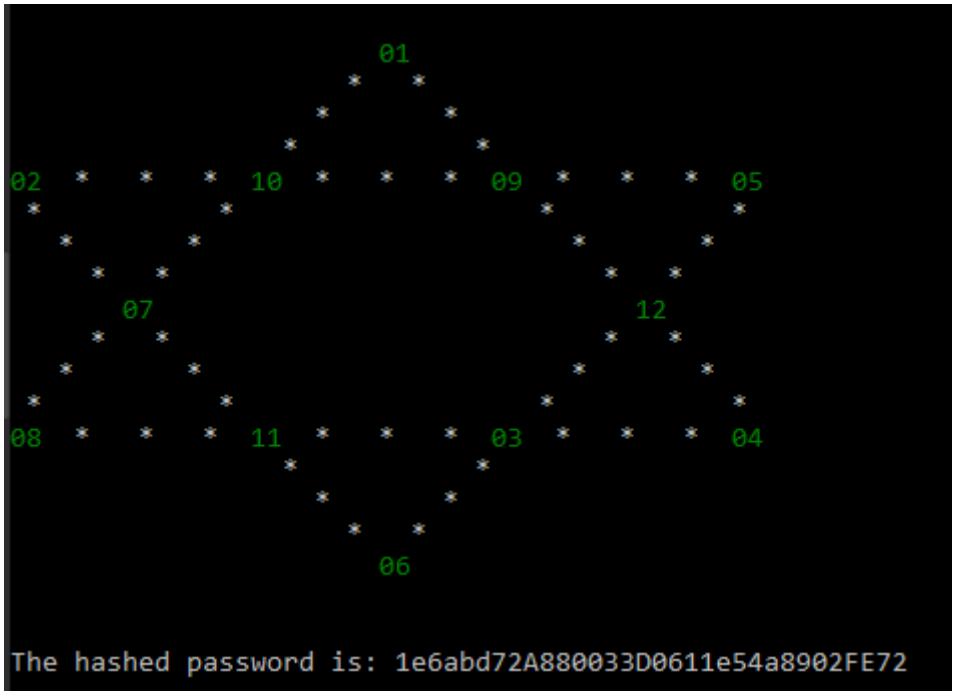
מכיוון ש-result ידוע, ולפי תכונות של XOR, אפשר לחלץ את האיבר במחרוזת הקלט באינדקס 28. אחר כך, בלולאה:

$$result[i] = result[i+1]\hat{}str[i]$$

מכיוון שresult ידוע, אפשר לקבל את מחרוזת הקלט במקום הi.

בעזרת script פייתון (findStr.py) מצאנו את כל המחרוזת והצלחנו לעבור את הבדיקה.

כעת היינו רוצים לחזור על הפונקציה, כי כאשר עדכנו את ערך dword_409024, הפונקציה עוברת ל-branch אחר אשר מדפיס את שארית הפלט, לבסוף משחזר את כתובת החזרה המקורית ואת ה-ebp, ויוצא כמו שצריך. לשם כך, הוספנו למחרוזת הקלט ריפוד של 16 בתים (זה יכול היה להיות כמעט כל ערך). כעת, בגלל שהמחרוזת מגיעה עד סוף ה-frame, הכתיבה לאחר המחרוזת דורסת את כתובת החזרה והפונקציה אכן תיקרא שנית. קיבלנו את שארית משפט הפלט וסיימנו את wizard.



הקובץ **decrypt.exe**:

הבנו שבקובץ זה יש לנו שתי משימות: להבין את סדר הכנסת ארבע הארגומנטים, והאם להכניס אותם כמו שהם או לשנותם.

פענוח סדר הארגומנטים:

מפתח ההצפנה – ראינו כי יש פונקציה שבודקת אורך מפתח. הארגומנט שנבדק הוא הארגומנט הרביעי שמועבר לתכנית. מכאן שהארגומנט הרביעי הוא מפתח ההצפנה.

```
loc_401453:
mov     eax, [ebp+argv4]
mov     [esp+14h+Buffer], eax ; Str
call    strlen
cmp     eax, 10h
jz      short loc_401497
```

```
mov     eax, ds:_iob
add     eax, 40h ; '@'
mov     [esp+14h+Stream], eax ; Stream
mov     [esp+14h+ElementCount], 27h ; ''' ; ElementCount
mov     [esp+14h+ElementSize], 1 ; ElementSize
mov     [esp+14h+Buffer], offset aErrorKeyMustBe ; "Error: Key must be 128 bits in length."...
call    fwrite
mov     [esp+14h+Buffer], 1 ; Code
call    exit
```

מספר ה-rounds – ראינו כי באחת הפונקציות ב-decrypt יש לולאה שמתרחשת מספר פעמים כארגומנט הראשון שניתן לתכנית. הסקנו שזה מתאים לכך שהארגומנט הראשון הוא מספר ה-rounds.

תחילת הלולאה:

```
loc_401BFF:
movzx   edx, [ebp+var_D]
mov     eax, ds:argv1_long
cmp     edx, eax
jl      loc_401B7F
```

סוף הלולאה:

```
mov     [ebp+var_18], eax
mov     eax, ds:argv2long
sub     [ebp+var_C], eax
movzx   eax, [ebp+var_D]
add     eax, 1
mov     [ebp+var_D], al
```

ה-delta – הנחנו שפרמטר ההצפנה הנוסף יופיע לצד מספר ה-rounds. ואכן הוא הופיע במהלך אותה לולאה – שמו argv2_long (לאחר קריאה ל-strtol על [argv[2).

לפי אלימינציה, הסקנו כי הסיסמה המוצפנת היא הארגומנט השלישי לתכנית.

מהתבוננות בפונקציות בקובץ decrypt.exe, הבנו כי בריצת התכנית מתייחסים רק ל-16 תווים הראשונים של הסיסמה. הסיסמה שקיבלנו מ-wizard באורך 32 תווים. לכן, חשבנו להריץ את decrypt.exe פעמיים: כל פעם על חצי אחר של הסיסמה.

הסיסמה המלאה הייתה: 1e6abd72A880033D0611e54a8902FE72

הרצה על חצי הסיסמה הראשונה:

```
C:\Users\lydia\OneDrive\Desktop\Studies\spring 23\reverse\Reverse2023\HW2\goblin>decrypt.exe
8 125549079 1e6abd72A880033D ~]{r.E\MEy:c!#;[
RDANUSAN
```

התקבל RDANUSAN.

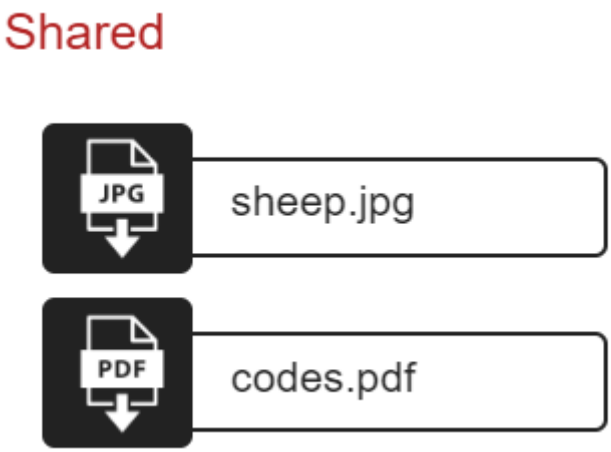
הרצה על חצי הסיסמה השנייה:

```
C:\Users\lydia\OneDrive\Desktop\Studies\spring 23\reverse\Reverse2023\HW2\goblin>decrypt.exe
8 125549079 0611e54a8902FE72 ~]{r.E\MEy:c!#;[
2D7D4ERB
```

והתקבל 2D7D4ERB.

חיבור שני החצאים הביא RDANUSAN2D7D4ERB.

הסיסמה הזו עבדה על ה-shared safe באתר, וכך נראה תוכן הכספת:



ראינו כי ב-codes.pdf, נותנים הנחיות לגבי שימוש בקוד בהקשר של לוח המשחק קטאן. הפעלנו את analyzer.exe על הקובץ sheep.jpg וקיבלנו קובץ פלט sheep.out. ניחשנו ש-sheep.out הוא קובץ ריצה שעלינו לנתח, כי זה קורס הנדסה לאחור. לכן, שינינו את הסיומת שלו לקובץ exe, וטענו אותו ב-IDA.

Sheep.exe

בקובץ מצאנו את main כמו בקבצים הקודמים וראינו שיש שתי פונקציות רלוונטיות:

הפונקציה הראשונה כותבת בתים לזיכרון, לכן קראנו לה MemWrite. בין הדברים שהיא כותבת לזכרון – ניתן לראות כאן מעין לוח, ומערך של structs.
כמובן שרק מאוחר יותר הבנו שמדובר בלוח 6x6, ובמבט ראשון לא ידענו לפרש אותו.
מאוחר יותר, ראינו בפונקציה השנייה, שמחשבים משהו שדומה לאינדקס במערך דו מימדי -

$$index = 6 * i + j$$

אז הבנו שהלוח הוא 6 על 6.

הלוח:

F	F	D		C	A
		D	X	C	A
	E	E	X		
R	Q	Q	Q	B	B
R		O	O	O	
R		P	P	P	

לאחר התבוננות בלוח, וחקירה של פונקציה הקוראת קלט משתמש ומבצעת פעולות על הלוח, הגענו למסקנה שיש בקובץ משחק Rush Hour ™, ושעלינו לפתור אותו בעזרת הכנסת קלט מתאים.

מערך של structs בזכרון: מתחיל ב-0x004080E0, ומסתיים ב-0x4081BF.

```
004080D0   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
004080E0 A 05 00 00 00 00 00 00 00  02 00 00 00 56 00 00 00  ............V...
004080F0 B 04 00 00 00 03 00 00 00  02 00 00 00 48 00 00 00  ............H...
00408100 C 04 00 00 00 00 00 00 00  02 00 00 00 56 00 00 00  ............V...
00408110 D 02 00 00 00 00 00 00 00  02 00 00 00 56 00 00 00  ............V...
00408120 E 01 00 00 00 02 00 00 00  02 00 00 00 48 00 00 00  ............H...
00408130 F 00 00 00 00 00 00 00 00  02 00 00 00 48 00 00 00  ............H...
00408140   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00408150   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00408160   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00408170 G 02 00 00 00 04 00 00 00  03 00 00 00 48 00 00 00  ............H...
00408180 P 02 00 00 00 05 00 00 00  03 00 00 00 48 00 00 00  ............H...
00408190 Q 01 00 00 00 03 00 00 00  03 00 00 00 48 00 00 00  ............H...
004081A0 R 00 00 00 00 03 00 00 00  03 00 00 00 56 00 00 00  ............V...
004081B0 X 03 00 00 00 01 00 00 00  02 00 00 00 56 00 00 00  ............V...
```

כל struct בגודל 16 בתים, ובכל struct, הסקנו כי השדות הם (לפי הסדר):

א. עמודה נוכחית (int)
ב. שורה נוכחית (int)
ג. אורך המכונית (int)
ד. האם המכונית אופקית או אנכית ('H' או 'V')

במהלך ריצת התכנית, נקראת fscanf כאשר הקובץ שקוראים ממנו הוא stdin. זה שקול להפעלת scanf רגילה – קריאת קלט מהמשתמש.
לאחר קריאת קלט של שלושה תווים מהמשתמש, נקראת פונקציה שמבצעת XOR בין הקלט

של המשתמש לתוכן בזכרון, ושומרת אותו שם – מאוחר יותר גילינו כי כאשר נכניס את הקלט הנכון, אז ייכתב לזכרון הפלט המבוקש והקוד, ובסוף יודפס אם הקלט מתאים.
הפונקציה הבאה מבצעת switch על התו הראשון – הסקנו שזה התו שבוחר איזו מכונית להזיז, ואחר כך סימנו את ה-structs השונים בזכרון לפי האותיות – ניתן לראות, שבהתאם ללוח, מתקיים:

המכוניות A B C D E F הן באורך 2, המכוניות O P Q R הן באורך 3, והמכונית X באורך 2.
בנוסף, שדות השורה והעמודה מתייחסים למשבצת העליונה ביותר \ שמאלית ביותר שהמכונית נמצאת בה, בהתאם להאם היא אופקית או אנכית.

חוקי המשחק:

כל מהלך מורכב מ3 תווים:

1. אות מבין "ABCDEFOPQRX": כל אות מציינת "מכונית" על הלוח.
2. כיוון: L/R, U/D. כל מכונית יכולה לנוע בכיוון אופקי או אנכי. צריך להתאים את הקלט למכונית. בחירת L/R מתאימה רק למכוניות האופקיות – אלו הן כל המכוניות שמסומן עליהן H, שהן B,E,F,O,P,Q. כל האחרות הן אנכיות, ומסומן עליהן V, לכן מתאימים להן כיווני U/D.
3. מספר צעדים: מסמל כמה המכונית תתקדם בכיוון הספציפי. זהו מספר בין 0 ל-9 (הקלט הוא תו בין '0' ל-'9').

תנאי הנצחון של המשחק:

המכונית אשר מסומנת ב-X (על הלוח היא נראית כזוג תווי 'X' אחד מעל השני), צריכה להגיע למשבצת בשורה החמישית ובעמודה הרביעית. היא מתחילה בשורה השנייה ובעמודה הרביעית – לכן עליה רק לרדת שלוש שורות, כדי לנצח במשחק.

פתרון: (מהסוף להתחלה)
המכוניות QQQ, OOO, PPP חוסמות את דרכה של XX. נהיה חייבים להזיז את QQQ, OOO, ו-PPP שמאלה עד הסוף כדי ש-XX תוכל לנוע מטה. על מנת להזיז את שלושתן יש להזיז את RRR למעלה עד הסוף – מה שדורש להזיז את FF ימינה, אך FF חסומה מימין על ידי DD, ו-DD חסומה על ידי גבול הלוח ועל ידי EE מלמטה.
בחרנו להזיז את XX למעלה בתור התחלה – זוהי הפקודה "XU1".
אחר כך, את EE עד הסוף ימינה, כדי ש-XX תוכל לחזור מטה – הפקודה "ER3".
המשכנו כך עד שהקלט שלנו נראה כך:

XU1ER3DD1FR1RU3QL1OL2PL2XD4

קיבלנו את הפלט הבא מהתכנית:

```
C:\Users\lydia\OneDrive\Desktop\Studies\spring 23\reverse\Reverse2023\HW2\sharedsafe>sheep.exe
XU1ER3DD1FR1RU3QL1OL2PL2XD4
You won the gamd!       Herd js a single use codd:#DLQ5W2AMT2. Use it wiselx.
```

שמנו לב שהפלט מסולף בחלקים ממנו – וניחשנו שזה אומר שגם הקוד שנפלט לא בטוח נכון.
בחנו האם יש דרכים אחרות לפתור את המשחק, והגענו למסקנה כי בעת הזזת המכוניות PPP, OOO, ו-QQQ, אפשר להזיז את שלושתן באיזה סדר שרוצים. זה כבר נותן לנו $3! = 6$ קלטים שפותרים את המשחק, מתוכם ניסינו רק אחד. לאחר נסיון כל 6 הקלטים, הגענו למסקנה שהקלט הבא גורם לפלט תקין להיפלט:

הקלט: XU1ER3DD1FR1RU3PL2OL2QL1XD4

הפלט:

```
E:\technion\reverse\Reverse2023\HW2\sharedsafe>sheep.exe
XU1ER3DD1FR1RU3PL2OL2QL1XD4
You won the game!
Here is a single use code: DLQ4W1AMT2. Use it wisely.
```

קיבלנו קוד לשימוש חד-פעמי. מוקדם יותר, קיבלנו רמז מתמונת QR שהפעלנו עליה את analyzer.exe –

_ _ _ _ _ _ _ _ _ _ _ - C 3

הסקנו שהקוד שקיבלנו יחד עם הרמז מהתמונה תואמים את ההנחיות בקובץ codes.pdf.

ניגשנו לאתר, ופתחנו את הלוח. לחצנו על כפתור *"Use Code"* והזנו את הקוד הבא:

DLQ4W1AMT2-C3

ארבע משבצות הכבשים החסרות נחשפו, והשלמנו את המשימה.

(אחלה אנימציה ☺)