**תרגיל בית 2 בהנדסה לאחור**

בעזרת רשימת המשתמשים והסיסמאות נכנסנו לאתר, אצל כל אחד מהמשתמשים יש 3 כספות:

* משותפת – מוצפנת ע"י סיסמה
* ציבורית – המכילה:
  + תרשים המתאר כיצד ניתן לפתוח את הכספת המשותפת
  + analyzer.exe המתואר בתרגיל.
* פרטית – מכילה קבצים שונים אצל כל משתמש.

**חלק א'**

לפי התרשים שמצאנו התחלנו לפצח את קבצי הsafe.exe של המשתמשים השתמשנו בניתוח סטטי בעזרת Ida.

 Giant

איתרנו את פונק' ה- main של הקובץ safe.exe ע"פ כלל האצבע שהיא הקריאה האחרונה לפני הקריאות ל- cexit ול- ExitProcess. פונקציית main מבצעת קריאות ל4 פונק' אחת אחרי השניה – נסמן אותן f1,f2,f3,f4. לאחר מכאן חיפשנו איפה התוכנית מקבלת קלטים בעזרת הלשונית Imports.

ראינו שיש 4 קריאות לscanf:

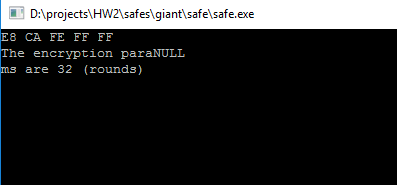
1. בפונק' sub\_401683:
   1. הקלט המבוקש הוא מהצורה "%02X %02X %02X %02X %02X", כלומר 5 מספרים בגדול byte בייצוג הקסה.
   2. לאחר קבלת הקלט, יש קריאה לפונק' אשר הערך המוחזר שלה הוא ebp+4, כלומר כתובת החזרה של הפונק'. לכתובת זו מוסיפים 98h מה שמביא אותנו לכתובת 4017B4h שהיא שורה בקוד של הפונק' sub\_401683. הקוד החל משורה זו מכיל 6 פקודות nop.
   3. בעזרת קריאה VirtualProtect משנים את הרשאות של הכתובת שחישבנו בסעיף קודם ל- PAGE\_EXECUTE\_READWRITE ולאחר מכן בעזרת memcpy מעתיקים את 5 הבתים מהקלט אל הכתובת.
   4. הבנו הקלט שאנחנו מחפשים הוא פקודה בינארית שאמורה להחליף את פקודות הnops. המשכנו לחקור את הפונק' וראינו שיש שקריאה רקורסיבית (החלק המתואר בסעיפים 1.2-1.3 מוגן בעזרת דגל גלובאלי כך שיתבצע רק פעם אחת). לאחר הקריאה הרקורסיבית נראה שמכינים שוב פרמטרים לקריאה רקורסיבית אבל במקום הקריאה יש nops, הבאנו שהקלט צריך להכיל את הפקודה call עם הoffset הנכון כך שתתבצע קריאה רקורסיבית.
   5. חישבנו את ה-offset: 401683h – 4017B4h = -131h = FFFFFECF, בעזרת האתר defuse.ca תרגמנו את הפקודה call 0xFFFFFECF 🡨 **E8 CA FE FF FF**
2. בפונק' f3
   1. בתחילת הפונק' יש קריאה ל- SetUnhandledExceptionFilter לפי התיעוד בMSDN היא משמשת לאיתחול פונק לטיפול בחריגות. הפונק מקבלת כפרמטר כתובת של פונק' לטיפול בחריגות. הפונק' לטיפול בחריגות (ExceptionHandler) מקבלת כארגומנט מצביע ל- struct \_EXCEPTION\_POINTERS המכיל מידע על ה- exception. היא מבצעת את הפעולות הבאות:
      1. ניגשים לשדה המתאים ובודקים עם החריגה היא מסוג EXCEPTION\_ACCESS\_VIOLATION, אם לא מחזירים 0 ויוצאים.
      2. ניגשים לשדה המתאים וקוראים אם הכתובת אליה ניסו לגשת היא כתובת 0 , אם לא מחזירים 0 ויוצאים.
      3. מבצעים השמה למשתנה גלובאלי (נקראה לו – isExceptionOccurred) ושמים בו את הערך 1.
      4. ניגשים לשדה המתאים וקוראים את הכתובת של הפקודה שבה התבצעה הגישה הלא חוקית לזיכרון – dest
      5. בעזרת קריאה ל-VirtualProtect משנים את ההרשאות של destו-24h הכתובות העוקבות ל- PAGE\_EXECUTE\_READWRITE.
      6. בעזרת קריאה ל-memset מציבים ב- destו-24h הכתובות העוקבות את הערך 90h (פקודת nop)
      7. בעזרת VirtualProtect משנים שוב את ההרשאות של destו-24h הכתובות העוקבות ל- PAGE\_EXECUTE\_READ.
      8. מחזירים את הערך EXCEPTION\_CONTINUE\_EXECUTION שמסמן שהטיפול בחריגה בוצע ורוצים להמשיך מהמקום בו היא התבצעה.
   2. לאחר מכן בעזרת scanf קולטים 4 תווים (כל אחד בגודל בית) c1,c2,c3,c4 ומציבים את הערך המספרי שלהם לתוך אוגר eax בסדר הבא : c1c3c2c4 לאחר מכן מחסרים מהמספר שיצא את המספר 4E4C554Ch, נשים לב שערכי ה-ascii של רצף הבתים יוצר את המחרוזת "NLUL".
   3. ניגשים לכתובת שמתקבלת מתוצאת החיסור ומנסים לקרוא משם ערך. העובדה שמחסרים את המחרוזת "NLUL" ושמצפים לטפל בחריגה בגישה לכתובת 0, הבנו שהקלט הנדרש הוא **"NULL"**
3. שתי קריאות ב-f4
   1. קריאה ראשונה מבקשת קלט בפורמט "%x" כלומר מספר בייצוג הקסהדצימלי (input)  
      אם הקלט שלילי מבצעים לו היפוך סימן.
   2. קריאה שניה מבקשת קלט בפורמט "%d %d %d" כלומר 3 מספרים (x1,x2,x3)
   3. לאחר מכן יש מספר הסתעפויות בקוד ע"פ ערכי הקלטים. כדי "לעבור בהצלחה" את כל ההסתעפויות ולהגיע לחלק של הפונ' שמבצע משהו הקלטים צרכים לעמוד בתנאים הבאים:

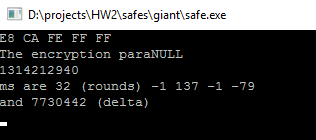
x1 >= 0  
x1 % 256 == 137  
y = -9164\*x3 -58\*x3^2 – 361977 = 1 🡪 x3 == -79  
x2 < 0  
x2 - y > 0  
Input < 0

ניתן לראות שיש סתירה בתנאים של input ושל x2 ולכן לכל קלט הפונק' תמיד תצא בלי לעשות כלום.

בגלל שלא מצאנו פתרון לשלב זה השתמשנו בדיבאגר כדי לגרום לקוד להגיע לחלק שמדפיס את הפרמטר השני. הכנסנו קלט x1=137, x3=-79, שני הקלטים האחרים ערכים שרירותיים. הערה: לא הצלחנו לדבג את הקוד אחרי ה- EXCEPTION בשלב קודם, לכן דיבאגנו מהתחלה וגרמנו לקוד לרוץ ללא EXCEPTION ע"י שינוי ערכי רגיסטרים ושינוי הקוד עצמו בעזרת הדיבאגר.

הרצנו את safe.exe:  
קלט ראשון : E8 CA FE FF FF.  
פלט למסך: The encryption para  
קלט שני: NULL  
קלט שלישי: -1 137 -1 -79  
פלט למסך: ms are 32 (rounds)





 wizard

איתרנו את פונק' ה- main של הקובץ safe.exe ע"פ כלל האצבע שהיא הקריאה האחרונה לפני הקריאות ל- cexit ול- ExitProcess. פונקציית main מבצעת קריאות ל3 פונק' אחת אחרי השניה – נסמן אותן f1,f2,f3. לאחר מכאן חיפשנו איפה התוכנית מקבלת קלטים בעזרת הלשונית Imports.

ראינו שיש 2 קריאות לscanf:

1. בפונק' f2:  
   בתחילת הפונק' בודקים את ערכו של משתנה גלובאלי (נקרא לו f2\_call\_clounter). ערכו של המשתנה גדל באחד כל פעם לפני יציאה תקינה מהפונק' (שחזור מחסנית וקריאה לret).
   1. בפעם הראשונה שמגיעים לפונק' f2, נכנסים לסנף שמצבע את הפעולות הבאות:
      1. שומרים למשתנים גלובאלים את ערכי: כתובת החזרה מהפונק (ebp+4), ערך ebp הקודם, ebp, esp. **זה מרמז לנו שכנראה ערכים אלו ידרסו וצריך לשמור אותם כדי לאפשר המשך ריצה תקינה של הקוד**.
      2. קריאה לקלט מהמשתמש בפורמט מחרוזת.
      3. 28 התווים הראשונים של מחרוזת מתורגמים לערך מספרי ע"י החסרה של התו 'A', כלומר כל תו מקבל את ערכו הלקסיקוגרפי בABC. **זה מרמז שהקלט כנראה צריך להכיל רק אותיות ABC גדולות**.
      4. 28 הערכים הראשוני של הקלט (לאחר ההמרה מסעיף קודם) מעודכנים להיות תוצאת הxor של כל ערך עם הערך הבא אחריו. האחרון הוא תוצאת xor בינו לבין הראשון.
      5. 28 הערכים מומרים שוב להיות תווי asci ע"י הוספת הערך של התו 'A'
      6. 28 הערכים מושווים בעזרת strcnmp למחרוזת "CIONAPGNOGKGEGPFJHEDKNIGNIBK".  
         כדי לשחזר את הקלט הנדרש כתבנו תוכנית אשר מבצעת את הפעולה ההפוכה על מחרוזת הקלט ומחפשת קלט אפשרי תקין.
2. #define SIZE 28
3. #define F 15
4. void findKey()
5. {
6. const char\* ref = "CIONAPGNOGKGEGPFJHEDKNIGNIBK";
7. int ref\_num[SIZE];
8. int key[SIZE];
9. const char A = 'A';
10. if (strlen(ref) != SIZE)
11. {
12. return;
13. }
14. for (int i = 0; i < SIZE; i++)
15. {
16. ref\_num[i] = ref[i] - A;
17. }
18. for (int k = 0; k <= F; k++)
19. {
20. key[1] = k;
21. key[0] = key[1] ^ ref\_num[0];
22. for (int i = 2; i < SIZE; i++)
23. {
24. key[i] = key[i - 1] ^ ref\_num[i - 1];
25. }
26. if ((key[SIZE - 1] ^ ref\_num[0]) == ref\_num[SIZE - 1])
27. {
28. //found input key !!
29. break;
30. }
31. }
32. //print result
33. for (int i = 0; i < SIZE; i++)
34. {
35. printf("%c", key[i] + 'A');
36. }
37. printf("\n");
38. }

בעזרת הפונק' מצאנו את הקלט הנדרש "NPHJEELNAOICEAGJMFCGFPCKMBJI"

* + 1. לאחר הכנסת הקלט התקין מגיעים לחלק בקוד שמעתיק (בעזרת sprintf) את ערכי הקלט (ללא הגבלת גודל) ולאחריו את כתובת הפונק' f2 לכתובת שנמצאת על המחסנית. לאחר מכן מבצעים בדיקה אם כתובת החזרה (ebp+4) זהה לכתובת הפונק' עצמה (f2). כדי שמצב זה יקרה אנחנו צריך לבצע דריסה של כתובת החזרה ע"י כך שניתן קלט באורך מתאים ואז sprintf תדרוס את כתובת החזרה בכתובת של f2. חישבנו שכדי שזה יקרה צריך להוסיף לקלט 16 בתים כלשהם ולכן הכנסו את הקלט: "**NPHJEELNAOICEAGJMFCGFPCKMBJI1234123412341234**".
    2. לאחר מכן מגיעים לחלק שבמצע עדכון של f2\_call\_clounter, ניקוי מחסנית וחזרה, מכיוון שדרסנו את כתובת החזרה מגיעים שוב לתחילת הפונק' f2.
  1. בפעם השניה שמגיעים לf2 הפונק' מבצעת הדפסה למסך של הפלט :

**"The hashed password is: a60b"**

לאחר מכן מתבצע שחזור של ערכי המחסנית שנדרסו בשלבים הקודמים וחזרה לmain.

1. הפונק' f3 מגדירה שני מערכים הראשון מאותחל בערכים כך:

int arr[12] = { 6,10,1,2,8,5,12,7,4,11,3,9 };

* 1. המערך השני (int input[12]) מאותחל ע"י קלט מהמשתמש באופן הבא:

for (int i = 0; i < 12; i++)

{

int x;

scanf("%d", &x);

input[i] = arr[x];

}

לאחר מכן נבדקים התנאים הבאים:

1. no duplicate value in input.
2. input [1]+ input [2]+ input [3]+ input [4] == 16
3. input [7]+ input [8]+ input [9]+ input [10] == 16
4. input [0]+ input [2]+ input [5]+ input [7]) == 16
5. input [0]+ input [3]+ input [6]+ input [10]) == 16
6. input [1]+ input [5]+ input [8]+ input [11]) == 16
7. input [4]+ input [6]+ input [9]+ input [11]) == 16
8. input [0]+ input [1]+ input [4]+ input [7]+ input [10]+ input [11]) == 16

השתמשנו בקוד הבא ע"מ למצוא את הקלט הנדרש:

#define ARR\_SIZE 12

void solverf3()

{

int arr2[ARR\_SIZE] = { 6,10,1,2,8,5,12,7,4,11,3,9 };

bool valid[ARR\_SIZE];

for (int i = 0; i < ARR\_SIZE; i++)

{

valid[i] = true;

}

for (int a = 0; a < ARR\_SIZE; a++) if(valid[a])

{

int x0 = arr2[a];

valid[a] = false;

for (int b = 0; b < ARR\_SIZE; b++) if (valid[b])

{

int x1 = arr2[b];

valid[b] = false;

for (int c = 0; c < ARR\_SIZE; c++) if (valid[c])

{

int x2 = arr2[c];

valid[c] = false;

for (int d = 0; d < ARR\_SIZE; d++) if (valid[d])

{

int x3 = arr2[d];

valid[d] = false;

for (int e = 0; e < ARR\_SIZE; e++) if (valid[e])

{

int x4 = arr2[e];

valid[e] = false;

for (int f = 0; f < ARR\_SIZE; f++) if (valid[f])

{

int x5 = arr2[f];

valid[f] = false;

for (int g = 0; g < ARR\_SIZE; g++) if (valid[g])

{

int x6 = arr2[g];

valid[g] = false;

for (int h = 0; h < ARR\_SIZE; h++) if (valid[h])

{

int x7 = arr2[h];

valid[h] = false;

for (int i = 0; i < ARR\_SIZE; i++) if (valid[i])

{

int x8 = arr2[i];

valid[i] = false;

for (int j = 0; j < ARR\_SIZE; j++) if (valid[j])

{

int x9 = arr2[j];

valid[j] = false;

for (int k = 0; k < ARR\_SIZE; k++) if (valid[k])

{

int x10 = arr2[k];

valid[k] = false;

for (int l = 0; l < ARR\_SIZE; l++) if (valid[l])

{

int x11 = arr2[l];

bool B1 = x1 + x2 + x3 + x4 == target; bool B2 = x7 + x8 + x9 + x10 == target;

bool B3 = x0 + x2 + x5 + x7 == target;

bool B4 = x0 + x3 + x6 + x10 == target;

bool B5 = x1 + x5 + x8 + x11 == target;

bool B6 = x4 + x6 + x9 + x11 == target;

bool B7 = x0 + x1 + x4 + x7 + x10 + x11 == target;

if (B1 && B2 && B3 && B4 && B5 && B6 && B7)

{

printf("%d %d %d %d %d %d %d %d %d %d %d %d\n", a, b, c, d, e, f, g, h, i, j, k, l);

return;

}

}

valid[k] = true;

}

valid[j] = true;

}

valid[i] = true;

}

valid[h] = true;

}

valid[g] = true;

}

valid[f] = true;

}

valid[e] = true;

}

valid[d] = true;

}

valid[c] = true;

}

valid[b] = true;

}

valid[a] = true;

}

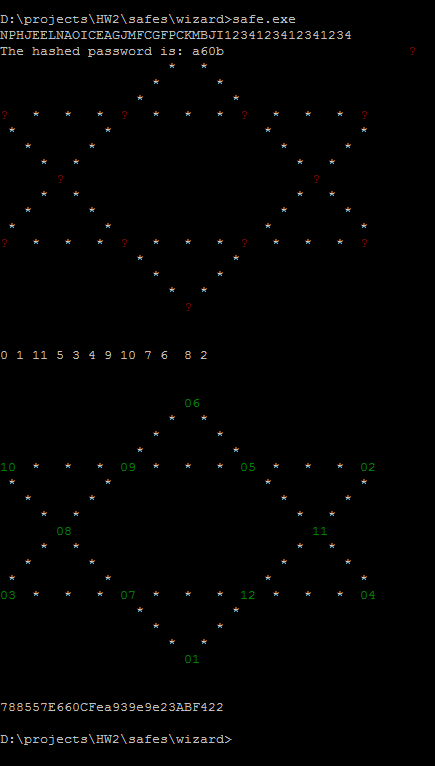
}

בעזרת הפונק' מצאנו את הקלט הנדרש "**0 1 11 5 3 4 9 10 7 6 8 2**"

לאחר הכנסת הקלט קיבלנו את המשך הפלט:

The hashed password is: a60b788557E660CFea939e9e23ABF422

**וציור של מגן דוד המכיל מספרים בקודקודיו.**





איתרנו את פונק' ה- main של הקובץ safe.exe ע"פ כלל האצבע שהיא הקריאה האחרונה לפני הקריאות ל- cexit ול- ExitProcess. בפונקציית main יש קריאה לפונק אתחול ואחר מכן לולאה מרכזית עם 8 איטרציות.  
בפונק' האתחול, מאתחלים 3 משתנים a=0, b=7, c='L' ובנוסף מערך בגדול 64 בתים (arr64) בהמשך נראה שפונים למערך בצורה הבאה arr64[a\*8+b] לכן זהו מערך דו-מימדי.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | . | . | 7 | F | . | 4 | . | . |
| 1 | X | X | X | X | X | X | X | X |
| 2 | . | . | . | . | . | . | . | . |
| 3 | X | X | X | X | X | X | X | X |
| 4 | F | . | . | . | . | . | . | . |
| 5 | 2 | . | . | . | . | . | . | . |
| 6 | X | X | X | X | X | X | X | X |
| 7 | . | 3 | . | . | . | . | . | . |

לאחר מכאן חיפשנו איפה התוכנית מקבלת קלטים בעזרת הלשונית Imports וראינו שיש 2 קריאות לscanf בתוך אותה הפונק' סימנו אותה func\_with\_scanf. היא מקבלת מstdin קלט הראשון בפורמט "%c". הקלט חייב להיות אחד מהתוים: C,D,U אחרת התוכנית מסתיימת. אם הקלט הראשון היה C אז ממשיכים לשני גם בפורמט "%c", הקלט השני חייב להיות תו המייצג ספרה 0-9 אחרת התוכנית מסתיימת. את התו השני ממירים לערך שהוא מייצג (ספרה integer). שני הקלטים מוחזרים בעזרת שני ארגומנטים שהפונ' קיבלה (מצביעים כפולים).

חזרנו לנתח את main, בכל אחת מהאיטרציות של main מתבצעים השלבים הבאים:

1. קריאה לפונק' func\_with\_scanf . ובדיקת הקלט הראשון: 🡪 🡨 ← ⬆ ↓⇓
   1. אם הקלט הראשון היה 'C' מבצעים סריקה של המערך arr64[a\*8+b] לפי החוקיות הבאה:

start

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | ⇓ | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 |
| 1 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | ⇓ |
| 2 | ⇓ | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 |
| 3 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | ⇓ |
| 4 | ⇓ | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 |
| 5 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | ⇓ |
| 6 | ⇓ | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 | 🡨 |
| 7 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | 🡪 | ⇓ |

* + 1. מספר הצעדים בסריקה הוא לפי הקלט השני.
    2. אם מגיעים לאחד הקצוות של השורה, יורדים שורה וכיוון הסריקה מתהפך.
    3. אם נוחתים על X התוכנית מסתיימת
    4. סופרים את מספר הפעמים שעברנו דרך F
  1. אם הקלט הראשון הוא U קופצים מספר שורות למעלה לפי הערך הנוכחי של arr64[a\*8+b].
  2. אם הקלט הראשון הוא D קופצים מספר שורות למטה לפי הערך הנוכחי של arr64[a\*8+b].

1. אם a==7 ו- b==7 כלומר הגענו לפינה ימנית תחתונה ואם עברנו פעמיים דרך F אז הלולאה מסתיימת ומודפס הפלט עם התשובה למסך.
2. אם לא סיימנו את הסעיף הקודם תוך 8 איטרציות, התוכנית מסתיימת.

הרצנו את safe.exe ועם הקלט הבא:

C 5

U

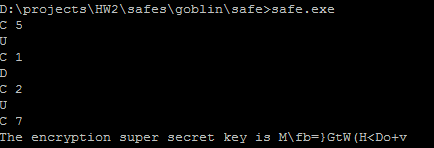
C 1

D

C 2

U

C 7



**חלק ב' - decrypt.exe**

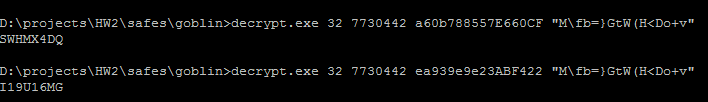
לאחר שיש בידינו את הקלטים הנדרשים ל decrypt:

* The encryption params are 32 (rounds) and 7730442 (delta)
* The hashed password is: a60b788557E660CFea939e9e23ABF422
* The encryption super secret key is M\fb=}GtW(H<Do+v

ביצענו ניתוח סטטי של decrypt.exe כדי להבין איך להשתמש בקלטים.

1. בפונק' main בודקים את מספר הארגומנטים שהתוכנית מקבלת וצריך שיהיו בדיוק 4 ארגומנטים.
2. ארגומנט 4 – צריך להיות בגודל 16 בתים אחרת מודפסת ההודעה: "Error: Key must be 128 bits in length." לכן הבנו שהוא ה – key.
3. ארגומנטים 1,2 מומרים למספרים לכן הם הפרמטרים.
4. ארגומנט 3 (הסיסמה) צריכה להיות באורך זוגי, ומתייחסים רק לחצי הראשון של הססמה.

הרצנו את ה decrypt.exeפעמיים ובכל פעם הצגנו לה חצי מהסיסמה.



איחדנו בחזרה את הסיסמה שקיבלנו : **SWHMX4DQI19U16MG** ופתחנו בעזרתה את הכספת ה משותפת. בתוך הכספת מצאנו שני קבצים:

* codes.pdf – מסביר איך אפשר להפוך אריח.
* sheep.jpg – קובץ תמונה, בדקנו אותה בעזרת analyzer.exe והתקבל הקובץ sheep.out.

פתחנו את sheep.out לעריכה בעזרת notpad וראינו שהוא מתחיל ב-"MZ" מה משאיד שהוא מכיל תוכנית מקומפלת. שינינו את השם שלו ל- sheep.exe ופתחנו אותו לניתוח בעזרת ida .