**תרגיל בית 2 – פירוט**

לפי התרשים safe.pdf המטרה הסופית לפיצוח הכספת היא Safe Password, שאליה אין לנו גישה. אבל נוכל להשיג דרך הכספת של ה-wizard את הסיסמה כשהיא מוצפנת, את המקודד ומפתח ההצפנה נשיג דרך goblin ואת הפרמטרים לקידוד דרך giant

* **:wizard**

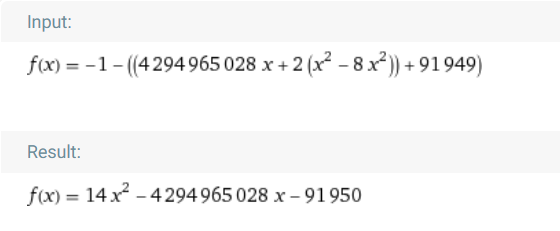
הורדנו את safe.exe מהכספת הפרטית של wizard וניתחנו את הקובץ עם IDA.

ניתן לראות כי בתחילת הקובץ קוראים לפונקציה TlsCallback, מה שלהבנתנו מסבך את המעבר עם דיבאגר, זאת היות ו-TlsCallback לוקח שליטה לפני נקודת הכניסה לתכנית. לכן, המשכנו בניתוח סטטי של הקוד ומצאנו לוגיקה המתחילה בהכנסת ה-magic value DEADBEEF למחסנית. ניתן לראות כי מצפים לקלט ראשוני, שכרגע לערכו אין חשיבות, מלבד להיותו מספר בייצוג hexa, ולאחר מכן ממשיכים. בעת הזנת המספר, בודקים אם הוא שלילי ואם כן הופכים אותו לחיובי ע"י neg. במידה ונכניס מספר אקראי, בסוף יודפס לנו ג'יבריש. הדרך לבצע unhash היא ע"י פונקציה שאליה נכנסים רק בהמשך במידה והפרמטר הנ"ל שלילי, מה שלא יקרה לרוב כי ביצעו לו neg. לכן, המספר היחיד שישאר עדיין שלילי גם לאחר ביצוע neg הוא MAX\_INT. מאחר והקלט בשלב זה הוא ב-hexa, נכניס 80000000, עליו יבוצע neg מבלי לשנותו, וכך נגיע גם לחלק שעושה unhash בהמשך.

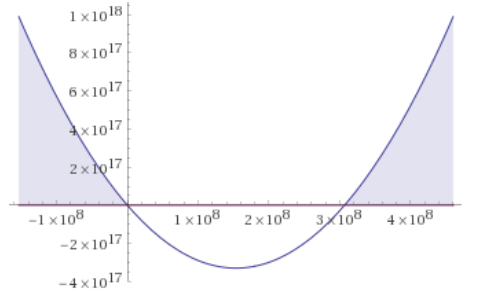
כעת קוראים שלושה מספרים למחסנית אל ebp-10, ebp-14, ebp-18. מבצעים מספר פעולות אריתמטיות, מעתיקים את המספר הראשון שנקלט ומשווים אותו ל-196 (C4). אם הוא שונה, מסתיימת התכנית. לכן המספר הראשון שנכניס הוא 196.

אחרי כן יש בדיקה שהמספר השני בקלט הוא קטן מאפס, אחרת התכנית מסיימת. לכן, המספר השני שנכניס הוא -1.

לאחר מכן, בודקים אם תוצאת הפעולה האריתמטית שביצענו, בעת חיסורה מהמספר השני שקלטנו, גדול אפס, ואם כן מסיימים את הריצה. בעזרת WolframAlpha חישבנו את הפולינום המתקבל מהאריתמטיקה וראינו מתי הוא מתאפס:



והגרף המתקבל הוא:



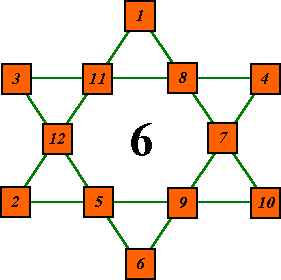
עבור מספרים בעלי ייצוג ארוך, למשל מאוד גדולים, יתכן שיחתכו חלק מהביטים בעת ביצוע הפעולות האריתמטיות, אך בקירוב ניתן לראות כי עבור מספרים גדולים מאוד נעבור את בדיקת הקלט.

במידה ועמדנו בכל תנאי הקלט הנ"ל, נקבל הדפסה של מגן דוד עם סימני שאלה מפוזרים. עם סיום ההדפסה, התוכנית מחכה לקלט של 12 מספרים בין 0 ל-11 ומבצעת בדיקת קלט עבור כל אחד מהם. בנוסף, עבור כל מספר שמוזן התכנית בודקת שלא הוזן קודם לכן, כך שיוזנו כל המספרים בין 0 ל-11.

ניתן לראות כי קודם לכן, נדחפו על המחסנית המספרים 1-12 בסדר מסוים, כך שבעת הזנת הקלט שלנו, כל מספר שנזין מוחלף במספר כלשהו מהמחסנית לפי התאמתו, כלומר – המספר 0 מתאים למספר הראשון על המחסנית וכך הלאה עד שלבסוף המספר 11 מתאים למספר האחרון מקבוצת המספרים על המחסנית.

לאחר מכן, הקוד ממשיך בוידוא שהקלט שלנו פותר את החידה כך שהוא סוכם את ארבעת המספרים בכל צלע במגן דוד ומוודא שסכומם הוא 26 כאשר על כל שורה שעמדה בתנאי הוא מגדיל מונה ב-1. לבסוף, הוא סוכם גם את כל המספרים על היקפו של המגן דוד ומוודא שגם סכומם 26, כך שאם עמדנו בתנאים המונה יעמוד על 7 (6 צלעות + היקף).

ניתן למצוא את פתרון החידה באינטרנט ולהגיע לפתרון הבא:



C:\Program Files\Microsoft VS Code\bin\codeכעת, כל שעלינו לעשות הוא למצוא את המספרים שעלינו להזין על מנת שהחלפת המספרים תביא לפתרון הנ"ל, כאשר סדר המספרים (לפי הוידוא של הקוד) הוא מלמעלה למטה ומשמאל לימין. על מנת לקבל את הרצף:

1 3 11 8 4 12 7 2 5 9 10 6, עלינו להזין 7 11 5 0 3 6 9 10 1 8 4 2 ובעת הזנת רצף זה נקבל הודעת הצלחה.

לאחר מכן על המחסנית טוענים את שתי מחרוזות,

3+U3waA$tMzvU$lX=]k>3|E , hacked this

התוכנית מחכה לקלט, וכשמקבלת משווה אותו למחרוזות האלה כל פעם עד אורך המחרוזת המקורית. אז הקלט המכיל את שתיהן מופרדות ברווח נותן

f82e1d49F17D72136a7d7b7b4017731E

* **goblin:**

מפתח הצפנה

לאחר בחינה מהירה של safe.exe ניתן לראות שהתוכנית קוראת לrand ובוחרת ערך רנדומלי מהמילון (dictionary.txt). אז רצינו לחפש באיזה מסלולי חישוב יש קריאה פונקציה שתחזיר פלט, אז חיפשנו מחרוזות והבחנו בשתי מחרוזות \\\\.\\pipe\\SuperSecretPipe \\\\.\\pipe\\SecretAnswerPip

חיפשנו את הגישות אליהן בקובץ ומתברר שהן נשלחות כפרמטר לפונקציות winapi: CreateFileA ו-CreateNamedPipeA. חיפשנו בתיעוד של מיקרוסופט והבנו שאחת יוצרת pipe server והשניה מתחילה להאזין ל-pipe server קיים כלומר התוכנית מאזינה על SuperSecretPipe ושולחת דרך SecretAnswerPipe.

כתבנו תוכנית שמבצעת אותו הדבר בתפקידים הפוכים: מאזינה ל-SecretAnswerPipe ושולחת דרך SuperSecretPipe.

(התוכנית קודם אתחלה את ה-pipe server ואחר כך בדקה אם קיים ה-pipe המרוחק)

כעת בדקנו מה מצפה לקבל התוכנית, שכן היא קודם כל מאזינה לבקשות.

שמנו לב שהיא משווה את המחרוזת שהתקבלה מה-pipe לזאת שבחרה מוקדם יותר מהמילון ואם אינן זהות היא מחכה לבקשה נוספת.

אז הסקנו שנוכל לעבור על כל המחרוזות מהמילון עד שנפגע בזו שנבחרה באקראי. בתוכנית שכתבנו פתחנו את קובץ המילון ושלחנו לתוכנית את כל המחרוזות ממנו.

הוספנו המתנה בתחילת התוכנית כדי שנוכל להפעילן במקביל, הרצנו וקיבלנו

The encryption super secret key is (?FRcQ,AfcjZrdkv

* **:giant**

התחלנו בניתוח ה-exe של הכספת. בשלב הראשוני ניתן לראות כי ישנה קריאה לפונקציה SetUnhandledExceptionFilter של winapi. חיפשנו בתיעוד של מייקרוסופט והבנו כי פונקציה זו היא מעין פילטר לחריגות ע"י כך שהיא מאפשרת לתהליך לקבל שליטה על כל חריגה שקוראת בזמן ריצתו ולא נתפסת. כאשר חריגה כזאת נזרקת, וזה בזמן שהתהליך לא מדובג, הפונקציה תקבל שליטה ותקרא ל-handler בכתובת lpTopLevelExceptionFilter שהיא מקבלת כארגומנט.

לאחר מכן, נדחפת על המחסנית המחרוזת NLUL. אחרי כן, מקבלים קלט של ארבעה תווים. הכתובות המועברות ל-scanf ניתנות לא ככתובות שלמות, אלא כרצף המספרים החל מ-ebp-11h עד ebp-14h, כך שארבעת התווים שנקלטו מתקבלים כמילה רצופה התופסת כולה את הכתובת ebp-14h לפי ייצוגי ה-ASCII של ארבעת הבתים שנקלטו. לאחר מכן, ארבעת התווים שנקלטו, שנסמנם לצורך הנוחות abcd (כאשר a הוא הראשון ו-d הוא האחרון) מועתקים אחד אחד ל-eax, ומבצעים עליהם shift ו-or כך שלבסוף נקבל מתוך הקלט את acbd.

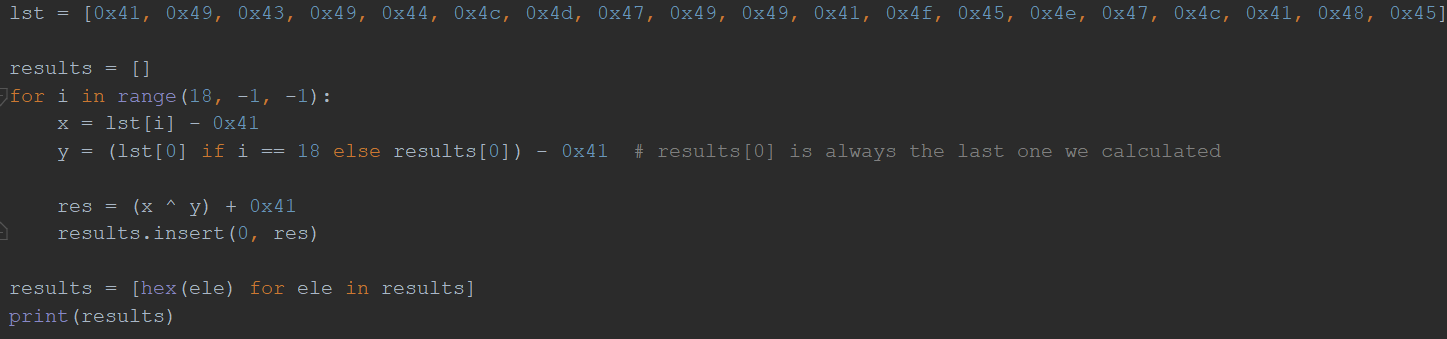
לאחר מכן, מתבצע חיסור של המחרוזת שהתקבלה מהמחרוזת שנדחפה קודם לכן למחסנית (NLUL) ומתבצעת גישה לכתובת שהתקבלה.

עבור קלטים רבים שנזין, נקבל גישה לכתובת לא חוקית. כך למשל אם נזין NULL, לאחר שינוי סדר האותיות תתקבל המחרוזת NLUL כך שהחיסור יניב 0 ובגישה לכתובת 0 נקבל חריגת גישה לכתובת לא חוקית. במקרה זה, ה-handler שהוגדר קודם לכן יקבל שליטה.

ה-handler מקבל את EXCEPTION\_POINTERS שהוא ה-struct עם פרטי החריגה שנזרקה כארגומנט, ניגש אל הכתובת שלו שהוא האיבר הראשון ב-struct והוא EXCEPTION\_RECORD ולאחר מכן ניגש אל האיבר הראשון בו שהוא קוד החריגה. הוא מוודא שקוד החריגה שנזרקה הוא 0x0C0000005, שזהו למעשה הקוד של Memory Access Violation ואחרת יוצא מה-handler. לאחר מכן, ניגש באותה דרך אל האיבר השלישי ב-EXCEPTION\_RECORD שהוא פוינטר שמצביע אל ה-EXCEPTION\_RECORD הבא ומוודא שאינו 0 – כלומר, שלא התקבלה חריגה נוספת. לאחר מכן, כותבים 1 בכתובת 408020 (שערכה הקודם היה 0) ובודקים. לאחר היציאה מה-handler, מתבצעת בדיקה לערך ב-408020 ומבצעים לו test, כך שאם הקוד לא עבר דרך ה-handler ששינה אותו מ-0, התכנית היתה עוצרת כאן. לאחר מכן ע"י VirtualProtect משנים את ה-Context של החריגה כך שהתכנית תוכל להמשיך לרוץ וממשיכים לפונקציה הבאה.

מקבלים מהמשתמש קלט של תווים עד קבלת שורה חדשה (אנטר) ועוברים בלולאה על 19 התווים הראשונים. מכל אחד מהתווים מחסרים 0x41 השקול לאות A בייצוג ASCII. לאחר המעבר על כל 19 התווים הראשונים, נכנסים שוב ללולאה שרצה על אותם 19 התווים לאחר השינוי שבוצע להם. בכל איטרציה, מכניסים את אותו הקבוע 0x6BCA1AF3 לרגיסטר ומבצעים עליו פעולות אריתמטיות ביחד עם האינדקס של האיטרציה הנוכחית. תוצאת הפעולות האריתמטיות היא אינדקס של אחד התווים במילה. לאחר קבלת האינדקס, מבצעים xor בין האינדקס שחושב לבין האינדקס התואם לאיטרציה הנוכחית על התווים ואת התוצאה שומרים במקום התו המופיע באינדקס הנוכחי. למשל, באיטרציה הראשונה מתבצע חישוב xor של התו הראשון עם התו השני והתוצאה נשמרת במקום התו הראשון. בכל איטרציה מתבצע כך על שני התווים העוקבים ואילו באיטרציה האחרונה נבדק התו האחרון עם התו הראשון ותוצאת ה-xor מחליפה את התו האחרון. לאחר מכן נכנסים ללולאה נוספת, עוברים שוב על התווים מההתחלה עד הסוף ומוסיפים לכל אחד 0x41 השקול לאות A בייצוג ASCII.

לאחר היציאה מהלולאה, משווים את התוצאה שהתקבלה מהפעולות על הקלט ומשווים את 19 התווים הראשונים ע"י strncmp (מקבל את 19 כארגומנט כך שישווה רק את 19 התווים הראשונים) למחרוזת הקבועה "AICIDLMGIIAOENGLAHE".

כתבנו סקריפט פייתון הפותר את הבעיה:

וקיבלנו את הפלט הנ"ל: JJBDLIDPJBJJHDOIDDE

לאחר שהכנסנו את הקלט הנ"ל התוכנית יוצאת, כיוון שלאחר בדיקת הקלט כותבים את הכתובת של הפונקציה הנוכחית על המחסנית (מיד אחרי הקלט) ואז בודקים האם כתובת החזרה זהה לכתובת הפונקציה הנוכחית. אז יש צורך להאריך את הקלט עד שיגיע במחסנית לebp כך שהכתיבה תדרוס את כתובת החזרה. הקלט נשמר על המחסנית ב- ebp-31h , אורך הקלט הקודם הוא 13h ואורך הכתובת הוא 3 בתים לכן סך הכל נוסיף 31h-13h-3h=10h כלומר צריך להוסיף 16 תווים ולכן הקלט הבא שניסינו היה JJBDLIDPJBJJHDOIDDE0000000000000000

לאחר מכן בתוך הפונקציה העוקבת, התוכנית קורסת שוב. בתחילתה דוחפים מספרים מ-1 עד 9 בסדר קבוע לא ממוין למחסנית (מ-עתה ה"מערך"). מבצעים פעולה על המערך בעזרת פונקציה א' (שנראית רקורסיבית מקבלת קלט של כ-5 תווים ומעורב בה שימוש בVirtualProtect) ולבסוף בודקים אם המערך ממוין. אם הוא לא ממוין התוכנית יוצאת. אז הרצנו את התוכנית עם דיבאגר (כדי לדלג על החריגה סיפקנו את התנאים להמשך התוכנית ידנית לרגיסטרים) וגילינו שפונקציה א' מקבלת 4 פרמטרים, ואם האחרון הוא 0 היא מבקשת קלט שיכתב לרצף של 6 nop-ים לקראת סוף הפונקציה. כלומר צריך לכתוב לשם פקודה באורך של עד 6 מילים.

ניתן לראות גם שלפני הפקודה הנ"ל דוחפים למחסנית 4 ערכים ולאחריה נפטרים מהם ע"י add esp 10h. כמו כן הפרמטר הראשון תמיד אותו הדבר בכל הקריאות הרקורסיביות והוא מצביע לתחילת המערך. המצביע למערך נדחף באותה הצורה גם לפני הפקודה שעלינו לכתוב. אז ניסינו לכתוב קריאה לפונקציה א' ע"י הכנסת הקלט

E8 CA FE FF FF

כלומר הפקודה call (E8) עם signed offset 32 bit מכתובת הפקודה הבאה אל תחילת הפונקציה. כלומר

2s complement(-136)=FFFFFECA

וסדר המילים ב-offset הוא little-endian אז הכנסנו בסדר הפוך את הכתובת.

מכאן התוכנית פולטת

The encryption params are 8 (rounds) and 117132623 (delta)

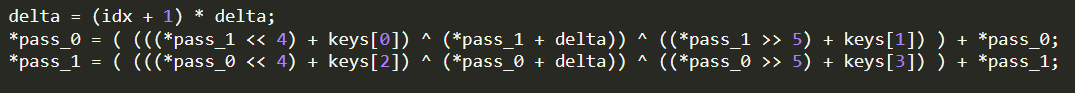
ומחזירה 0.

* **:encrypt**

כעת יש לנו את מפתח ההצפנה מ-goblin, שני הפרמטרים מ-giant ואת הסיסמא המוצפנת מ-wizard וננסה למצוא את הסיסמא הלא מוצפנת מ-encrypt.exe. מחקירת הקובץ נראה שהוא מקבל ארבעה ארגומנטים – שלושת הפרמטרים שציינו וסיסמא לא מוצפנת שאותה מצפין ופולט בסוף. ראשית, ניתן לראות כי מתבצעת בדיקה שכל הפרמטרים קיימים ואחרת התכנית יוצאת. לאחר מכן, מוודאים שאורך מפתח ההצפנה הוא 16 תווי hexa ואחרת יוצאים.

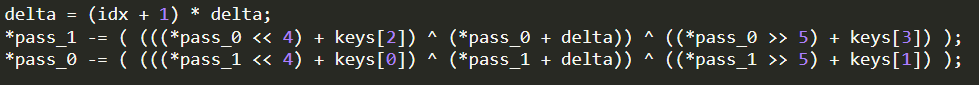
לאחר מעבר הבדיקות, מעבירים את הפרמטר השני שהתקבל ל-strtol על מנת להמיר אותו למספר בבסיס 16 ושומרים בכתובת 0x408080. מבצעים אותו דבר על הפרמטר הראשון ושומרים בכתובת 0x408088. לאחר מכן נבין כי מדובר למעשה ב-delta ו-rounds. לבסוף, מוודאים שהארגומנט הרביעי הוא סיסמה באורך 8 תווים.

כעת, נגמרו בדיקות הקלט ומתחיל שלב ההצפנה. ניתן לראות כי מאתחלים מונה לאפס, מבצעים איטרציות חישוביות ובכל איטרציה משווים את המונה לערך הארגומנט הראשון שהתקבל. מכך ניתן להסיק כי הארגומנט הראשון הוא למעשה rounds. הבנו מתוך זה שהארגומנט השני הוא delta ואכן רואים במהלך האיטרציות כי משתמשים בו לשם החישובים. בהקשר זה, ניתן לשים לב כי עקב כך שקיבלנו delta בעלת ערך גבוה, הפעלת strtol עליה מחזירה את MAX\_LONG בייצוג 32 ביט. כעת התעמקנו בכל איטרציה, וראינו שאת הסיסמה (באורך 8) מפצלים לשני בלוקים באורך 4 ואת המפתח (באורך 16) מפצלים לארבעה בלוקים בגודל זהה. לאחר מכן, ניתחנו את הפעולות האריתמטיות שמתרחשות והמרנו אותן לקוד C++ (המופיע בקובץ decrypt.cpp בפונקציה encryptRound). קיבלנו שהפעולה המתקבלת היא:



כלומר, דלתא מוכפלת באינדקס האיטרציה הנוכחית ומעדכנים את הבלוק הראשון והשני של הסיסמא ע"י חישובים המערבים את ערכיהם הישנים, ערכי המפתח, דלתא וחישובי XOR.

לאחר שוידאנו כי הבנו נכון את האלגוריתם, כל שעלינו לעשות היה להפוך את הפעולה על מנת לבצע פענוח להצפנה. לכן, המרנו כל איטרציה של encrypt לאיטרציה תואמת של decrypt ומימשנו גם אותה בקוד (המופיע בקובץ decrypt.cpp בפונקציה decryptRound). קיבלנו שהפעולה המתקבלת היא:



כעת, כל שעלינו לעשות היה להריץ את את התכנית שקיבלנו, שקומפלה ונבנתה ל-decrypt.exe עבור הסיסמה המוצפנת עם הפרמטרים התואמים.

נשים לב כי את הסיסמה המוצפנת קיבלנו כסיסמה באורך 16 תווים בייצוג hexa, ואילו encrypt ו-decrypt פועלות על סיסמה באורך 8 תווים, ולכן פיצלנו את הסיסמה המוצפנת לשני חלקים והרצנו פעמיים.

בפעם הראשונה הרצנו את decrypt עם החלק הראשון של הסיסמה, כך שהארגומנטים הם:

והפלט שהתקבל הוא O8TTNERC.

בפעם השנייה הרצנו עם החלק השני של הסיסמה עם הארגומנטים:

והפלט שהתקבל הוא 0D4HJLKD.

כך קיבלנו בסה"כ שהסיסמה היא O8TTNERC0D4HJLKD.