תרגיל בית 3 בהנדסה לאחור
Hooking

דרור כרמון 036861292

עידו מנגר 212324313

חלק יבש

א)נרצה לשנות את הזמן בין פניות של הפונקציה poll לשרת , לשנות את המשתנה הגלובלי בתחילת התכנית לא יעזור, אבל אם נוסיף קטע קוד שלוקח הרבה זמן, יותר מכמה שהתכנית כבר מחכה, אז הוא יגדיל את כמות הזמן בין בקשות של התכנית, לכן נבנה קטע קוד שפשוט ישן לתקופה שהיא יותר גדולה מהמשתנה הגלובאלי, ונדרוס בתחילת הפונקציה קפיצה אליו, את הקטע עצמו נוסיף לקובץ הריצה ואז כאשר יקראו לפונקציה, היא תחכה בהתחלה שלה יותר זמן ממה שהיא אמורה לחכות בין הבקשות, ולכן התדירות פניה שלה לשרת תשתנה.

ב) כלומר נרצה לעשות הוק בהתחלה של הפונקציה ולהפוך את x להיפוך שלו ואז להמשיך את פעולת התוכנית כאילו הכנסנו את ההיפוך של x, אבל אנחנו יודעים שזה בעייתי להכניס בתחילת הפונקציה הזו קוד, ולכן את ההוק נרצה לעשות בסוף הפונקציה,מה שנרצה לעשות זה להפוך את x ואז להפעיל עליו שוב את הפונקציה, מכיוון שאנחנו לא יודעים כמה מקום יש לנו להוק נרצה פשוט להפוך את x ואז לקרוא שוב לפונקציה, לכן מה שנרצה לעשות זה לקחת את כתובת החזרה של הפונקציה, להקטין אותה כך שהיא תחזור שורה אחורה, כלומר למקום שבו קראו לפונקציה, ולפני זה להפוך את x ולוודא שאנחנו עושים את זה רק פעם אחת, כלומר שבקריאה החדשה לפונקציה לא נכנס שוב להוק, בעזרת רגיסטר שהפונקציה לא משתמשת בו או על המחסנית. את כל הקוד הזה נצטרך להכניס איפשהו, ולקפוץ אליו עם הדריסה בסוף הפונקציה, נוכל להכניס אותו בsection נוסף בקובץ הריצה שאליו ההוק יקפוץ, וכמובן שנוסיף גם את הקוד שאנחנו דורסים לשם.

ג)הפונקציה sendf ככל הנראה מחברת את ההודעה למחרוזת אחת ואז שולחת אותה לשרת, כדי לשלוח אותה לשרת היא תצטרך להשתמש בפונקציית ספריה לשליחה על גבי סוקט לשרת(ככל הנראה send מwinsocket) לכן מה שנרצה לעשות זה לעשות הוקינג לפונקציה הזו כך שלפני שהיא שולחת לשרת היא מצפינה את ההודעה, במקרה והפונקציה משתמשת בהוט פטצינג, נוכל להשתמש באפשרות הזו כדי להזריק פונקציה משלנו שנכתוב על dll (כמו שעשינו בסדנה), או שנוכל להשתמש בIAT ונחליף את הכתובת של הפונקציה לפונקציה שמצפינה את ההודעה ואז קוראת לפונקציית הספריה המקורית.

ד) נרצה בעצם לגרום לפונקציה connect לחכות עד שתופעל הפונקציה parse לפני שהיא מתחילה את התכנית, זה דומה למה שאנחנו עושים כשאנחנו רוצים לדבג ומחכים לדיבאגר ולכן נשתמש באותה השיטה, connect תחכה לדיבאגר ופארס יפעיל דיבאגר על connect כשהוא מתחיל (הוא לא באמת ידבאג אותו, הוא יכבה אותו ישר) וכך שניהם יתחילו לפעול במקביל. נשתמש בפונקציות של DebugActiveProcess וDebugActiveProcessStop כדי להפעיל את הדיבאגר ולכבות אותו מparse וconnect נחכה לדיבאגר כמו שלמדנו בתרגול, כך ברגע שהדיבאגר יופסק שני הפונקציות יחזרו להתחלה, את הקוד הזה נכניס לשתי פונקציות, אחת לconnect ואחת לparse שנוסיף לקובת הריצה ונדרוס בתחילתconnect וparse קפיצה אליהן(נוסיף להן גם את השורה שאנו דורסים).

ה)הפונקציה בודקת במהלך ריצתה את שלמותה ולכן אם לפני ריצתה נכניס את ה hookשלנו ואז נדרוס חזרה את השורה היא לא תזהה את השינוי, מה שנרצה לעשות זה לגרום לפונקציה לפני חזרתה להדפיס את הערך חזרה שלה, נעשה את זה על ידי כך שנשנה את כתובת החזרה מהפונקציה לכתובת משלנו ונשמור את הכתובת שמחקנו, ואז בסוף הפונקציה היא תחזור לפונקציה שלנו, בה נדפיס את ערך החזרה(כנראה בeax) ואז נחזור לכתובת חזרה המקורית ששמרנו, את שתי קטעי הקוד האלה נוכל גם להוסיף לקובץ הריצה(קטע אחד שדורס חזרה את הקפיצה ומשנה את כתובת החזרה ושומר אותה, וקטע אחר שמדפיס את ערך החזרה וחוזר)

ו)נכתוב קטע קוד שמכפיל את eax ונרצה שהפונקציה תפעיל אותו לפני שהיא מחזירה בכל פעם(כי הערך חזרה יהה בeax) לכן עבור א בתחילת הפונקציה נוסיף קטע קוד שמשנה את כתובת החזרה לכתובת של הפונקציה הזאת ושומר את ערך החזרה הקודם גם במחסנית, ואז בסוף הפונקציה היא תקפוץ לקטע שלנו שמכפיל את eax ומשם נוסיף קפיצה לערך החזרה האמיתי ששמרנו על המחסנית, כך עבור אופציה א בכל פעם שהפונקציה תפעל היא תחזיר ערך כפול. עבור ב, נרצה לעשות אותו הדבר, אבל נרצה שזה יקרה רק בקריאה הראשונה לפונקציה, בשביל לבדוק אם זה הקריאה הראשונה, נשמור על המחסנית גם ערך מסוים שיסמן לנו שהתבצעה קריאה הפונקציה ונשים אותו במחסנית בכל פעם שאנחנו דוחפים לשם ערך חזרה בפונקציה הראשונה, ואז בפונקציה השנייה לפני שאנו מכפילים, נבדוק שאין מעלינו במחסנית את הערך הזה יותר מפעם אחת וככה נדע שזוהי הקריאה הראשונה של הפונקציה.

חלק רטוב

השבת השודד – חלק ראשון – ניתוח דינמי

התחלנו בניתוח סטאטי של keygen.exe וראינו שהתוכנית מקבלת ארגומנט יחיד (key), מעתיקה רצף של בתים אל המחסנית ולאחר מכן מבצעת call לבית הראשון שהועתק אל המחסנית.
המשמעות היא שהתוכנית עוברת לרוץ מקוד על המחסנית. כדי לראות את הקוד החל מקריאה זו עברנו לניתוח דינאמי בעזרת דיבאגר.
הפונקציה שרצה מהמחסנית (נקראה לה keygen) מקבלת את key כפרמטר. מאתחלת ע"ג המחסנית מערך תווים (arr) בגודל 95. המערך נראה כך:

```
static const char arr[95] = {'.', '5', '`', '>', 'A', 'd', 'l', 'G', '-', '*', '8', 'E', 'e', '}', '@', 'B', 'D', 'R', '
    F', 'k', 'z', 'w', ']', 'Z', 'Y', 'g', 'r', 's', 'm', '1', 'q', '\\', '2', '|', '3', 'J', 'j', '(', 'K', 'u', 'X', '
    ', '!', '\', ':', '_', 'v', '6', 'a', 'W', 'f', '{', 'y', '\"', 'C', ';', '+', 'L', '<', 'S', '4', 'H', 'O', '[',
    '0', '?', '$', 'b', 'h', '%', '/', 'M', 't', 'o', 'T', '9', ',', 'N', ')', '&', 'I', '=', 'x', 'U', 'p', 'V', '^', '
    7', '~', 'n', '#', 'c', 'i', 'Q', 'P'}
```

לאחר מכן הפונק' מקצה מקום לפלט (לפי גודל הקלט) ומתרגמת את הקלט בצורה הבאה:

```
1.  len ← strlen(key)
2.  for i ←0 to len :
3.     index ← key[i] - ' '
4.     output[i] ← arr[index]
```

כתבנו את התוכנית keygen_rev.exe אשר מבצעת את הפעולה ההפוכה.
הרצנו אותה עם קלט סיסמת הגישה לאתר שקיבלנו בתרגיל הראשון **CMX3GBV11Q7R7OK9**
וקיבלנו את סיסמת הגישה לדף הtools : **VgHB'/u==}w1w^Fk**

השבת השודד – חלק שני - hooking

בעזרת הסיסמה שמצאנו נכנסנו לדף ה-tools .

הורדנו את client.exe והרצנו אותו, שיחקנו עם אפשרויות הקלט השונות ובמיוחד עם DMSG שנתנה לנו את ההודעה המוצפנת:

4929-97468+766+96Q29-974686524-47268+76262657224-4636A78-87469766528+627-7486524-4697329-968656K6426-66967+729-9423327+7

496629-96668+77225-57367+866+76523-372656A7367+867+725-5776527-7736867+8756K6429-96672656526-674686524-46775792K

67+868+66525-567+675737424-4666966+86424-46A28-86366+9646527-76A737367+863696A74656429-97769746826-674686524-45246+94242455256+9434A56-6545552454422-265766565+97426+8

57686568+625-57468697325-56367+8646524-4697323-3757365642K24-47266+96K6K6968+66727-774686524-46469636529-9736867+8756K6424-4726573756K7429-97769746828-8637562657326-674686A7422-2737566+727-77466+922-27365766566+826+8

4767+8626K6968+6

בעזרת ida ביצענו ניתוח סטאטי ודינאמי של התוכנית ועקבנו אחר הפעולות שהיא עושה ובמיוחד איפה מתקבל הקלט באמצעות scanf איך הוא מתורגם לקוד של הפעולה שנבחרה (DMSG מתורגם לקוד 1) ומשם מתורגם להודעה לשרת (" :MESSAGE"). מצאנו גם איפה מתבצעת ההדפסה למסך של תשובת השרת באמצעות `puts`. ראינו גם שכאשר קוראים לputs עדיין נמצא במחסנית הקוד להודעה לשרת והיא נמצאת בebp+23. החלטנו לבצע hooking על הפונק' puts אשר נמצאת ב-msvcrt.dll. כדי להבין איזה סוג של hooking אנחנו צרכים נכנסנו בעזרת הdebuger לראות איך נראית הפונק' וראינו שהיא לא מתאימה לhot paching מכיוון שאין לה את ההכנה המתאימה (nops + mov edi,edi). לכן החלטנו לבצע דריסה וקפיצה באופן הבא: בכתובת הכניסה לputs נדרוס את הקוד הקיים בפקודת jmp לפונק' ה-hook שלנו. בפונק ה-hook ניגש אל ebp+23 ונבדוק בעזרת strncmp האם ההודעה לשרת היא " :MESSAGE". אם כן נקרא לפונק `decrypt_text` (שנכתוב בהמשך) ונעביר לה את המחרוזת שputs קיבלה כפרמטר ונמצא ב`esp + 4`. לאחר מכן (בכל מקרה) נשחזר את הפקודות שדרסנו מpustn ונבצע קפיצה לשורה הבאה אחרי הדריסה שלנו כדי שה- flow של הקוד ימשיך באופן תקין ו-puts תעשה את העבודה שלה.

כדי להבין מה `decrypt_text` צריכה לעשות הורדנו את secure_pipe.exe ופתחנו אותו באמצעות ida. זיהינו שבmain יש מספר קריאות לפונקציות שמטפלות בstreams ו- basic_strings (פונק' חיצוניות) ופונק' אחת (נקרא לה encryptor) של התוכנית שמקבלת כפרמטר מצביע למערך תוים. ומצפינה אותו באופן הבא על מחרוזת חדשה:

1. each char (c) decompose to its low 4 bits and high 4 bits.
```
low = c & 0xF;
high = c >> 0xF;
```
2. each 4 bit (val) is convert as follow (same for low and high):

4 bit value	**new value**	
val = 0	"x-x"	`x is random number`
1 < `val` < 10	val + '0'	ascii representation of val
val = 1	'A'	Ace
val = 10	'J'	Jack
val = 11	'Q'	Queen
val = 12	'K'	King
`val` > 12	"x+y"	x,y are random number that sum to val

3. concatenate high new value and low new value to the end of the new (output) string.

כדי לפענח את הקלט המתקבל מ-client מימשנו את `decrypt_text` אשר עושה את הפעולה ההפוכה והוספנו אותה ל ClientHook.cpp .

על מנת להזריק את ה-hook השתמשנו ב Injector.cpp בדומה למה שעשינו בסדנא.
בנוסף מצאנו גם ש-client.exe יכולה לקבל את הפקודה לשרת גם כארגומנט ולכן הוספנו את הפקודה "DMSG" כארגומנט להפעלה של client באמצעות ClientInjector.

הרצנו את ClientInjector בצורה הבאה וקיבלנו את הפלט :

```
D:\projects\HW3\part2\myTests\Debug>ClientInjector.exe client.exe

D:\projects\HW3\part2\myTests\Debug>
What would you like to do?
[1] ECHO - ping the server with a custom message, receive the same.
[2] DMSG - Download message from the server.
[3] TIME - Get local time from server point of view.
[4] HNKH - Request a spinning top for Hanukkah!
your choice (4 letters command t):
I took the robber captive. He is held in B3.
If for some reason we should free the guy,
one must find a code associated with the ROBBER_CAPTURED event.
When this code is used, rolling the dice should result with cubes that sum to seven.

Goblin
```

השבת השודד – חלק שלישי

עברנו על הכלים הנוספים בעמוד הTools באתר וראינו ש-Codes מתאים להקשר של code ו events בדומה לרמז. הורדנו את codes.exe ופתחנו אותו בעזרת ida ע"מ לבצע ניתוח סטאטי ודינאמי לתוכנית. Codes מקבלת שני ארגומנטים לריצה: <CODE_KEY>, <OLD_CODE> .

במידה ומתקבלים בדיוק שני ארגומנטים התוכנית מפעילה בעזרת CreateProcessA תהליך python בצורה הבאה:

```
python -c "from db_models import*; codes = Code.query.filter_by(code =
<OLD_CODE> ).all(); print('NO SUCH CODE' if len(codes) <= 0 else
codes[0].event.key)";
```

כלומר מבצעת שאילתה על מסד נתונים כלשהו המכיל קודים (Code), השאילה מבקשת את כל הקודים התואמים לקוד הקלט <OLD_CODE>. אם אין קודים תואמים מדפיסה 'NO SUCH CODE' אחרת מדפיסה event.key של הקוד הראשון מבין אלה שהתקבלו. ההדפסה print אינה מדפיסה לפלט הסטנדרטי (stdout/stderr) אלה לpipe שיצרה codes.exe והיא קוראת ממנו את התוצאה. הפלט המתקבל מpython נבדק באמצעות strcmp ובמידה והתקבל 'NO SUCH CODE' התוכנית מסתיימת.

אם התקבל פלט אחר (key) , התוכנית ממשיכה ובודקת (בעזרת strcmp) שהkey שהתקבל מpython תואם את <CODE_KEY> - ובמידה ולא התוכנית מסתיימת.

אחרת, התוכנית ממשיכה לבדיקת את הקלט <CODE_KEY> בדומה לשאילתה הקודמת, בצורה הבאה:

```
python -c "from db_models import*; codes = [code for code in
Event.query.filter_by(key = <CODE_KEY>).first().codes if not code.used];
.print('' if len(codes) <= 0 else codes[0].code)
```

כלומר השאילתה מבקשת את כל הקודים אשר תואמים ל- Event עם key זהה לארגומנט <CODE_KEY>.
לאחר מכן python מדפיסה (אם נמצא) את הקוד הראשון ברשימה שהתקבלה.
שוב codes.exe קוראת מה-pipe את הפלט והפעם מדפיסה אותו באמצעות printf אל הפלט הסטנדרטי.

ניסינו לדבג ולהריץ את codes על המחשב שלנו אבל הפלט שהתקבל מpython היה שגיאה שקשורה ל- db_models ולכן הבנו שנהיה חייבים להריץ אותה על השרת.

הבנו שכנראה אנחנו צרכים להריץ את codes עם ROBBER_CAPTURED = <CODE_KEY> ע"מ שהיא תדפיס את הקוד התואם. הבעיה שאנחנו לא יודעים את ה- <OLD_CODE> המתאים ע"מ לעבור את שתי הבדיקות שהתוכנה מבצעת.

כדי להתגבר על זה ננסה לבצע hooking לפונק' strcmp אשר נמצאת ב-msvcrt.dll ולהחליף את המימוש שלה באופן הבא:

```
int strcmp(char* str1, char* str2) :

        if str2 is equal to "NO SUCH CODE" :

                // if compering to "NO SUCH CODE" than this is firs strcmp of
                "codes.exe"
                // return 1 to pass it

                return 1

        else:

                // this is second strcmp of "codes.exe"
                // return 0 to pass it

                retutn 0
```

כדי להשוות את המחזורות נשתמש ב-strncmp (ע"מ לא ליצור קריאה רקורסיבית ב-hook).

כדי להבין איזה סוג של hooking אנחנו צרכים נכנסנו בעזרת הdebuger לראות איך נראית הפונק' וראינו שהיא לא מתאימה לhot paching מכיוון שאין לה את ההכנה המתאימה (nops + mov edi,edi). לכן החלטנו לבצע דריסה וקפיצה באופן הבא: בכתובת הכניסה לstrncmp נדרוס את הקוד הקיים בפקודת jmp לפונק' ה-hook שלנו. בפונק ה-hook ניגש אל esp + 8 שם נמצא (str2).

על מנת להזריק את ה-hook השתמשנו ב Injector.cpp בדומה למה שעשינו בסדנא.
כדי לקבל את הקוד אנחנו רוצים להריץ את codes בצורה הבאה:

```
Codes.exe ROBBER_CAPTURED whatever
```

הקלט השני יבדק מול ה-db אבל הה-hook ידלג על התוצאה שתתקבל.

ע"פ ההנחיות בעמוד tools , יצרנו קובץ zip המכיל את Codes.dll ו- CodesInjector.exe (שינינו לו את השם ל-toolfix.exe). העלנו את הzip וביצענו update . קיבלנו לפלט את הקוד:

0Q4T3K85KL

חזרנו אל הרמז, השורה הבאה "I took the robber captive. He is held in B3" הזכירה לנו את מפת האריחים שקיבלנו בתרגיל הבית הקודם. עברנו אל לוח המשחק וניסינו להזין את הקוד:

0Q4T3K85KL-B3

האריח התהפך אבל הבחור בווידאו סימן לנו שאנחנו ממש קרובים.

חזרנו שוב אל הרמז :

```
When this code is used, rolling the dice should result with cubes that sum to
seven.
```

הבנו שאנחנו צרכים להתעסק עם הקוביות ככה שתוצאת הזריקה כאשר מזינים את הקוד הנכון תצא 7 ....

בדף הtool מצאנו את Dice אשר מבצעת הגרלה של הקוביות. את תוצאת ההגרלה, dice מחזירה באמצעות ערך ההחזרה של התוכנית בצורה הבאה:

```
int d1 = // drow dice no 1 generates number in the range 1..6
int d2 = // drow dice no 2 generates number in the range 1..6
int res =(d1 << 16) | d2;
```

```
return res;
```

בנוסף שראינו שdice מקבלת ארגומנט ומשתמשת בו כדי לבצע שאילתה בדומה לcodes, מתוכן השאילה הבנו שהארגומנט הוא code. החלטנו נסות לכתוב hook שבאמצעותו נבדוק את הערך של הארגומנט, ובמידה ומתקבל הקוד **0Q4T3K85KL** אז נגרום לקוביה "להטיל" את הסכום 7.

בתחילת התוכנית מגרילים את הסכום שיצא בהטלת שתי הקוביות (2-12) ומעבירים את המספר הזה (sum) יחד עם הארגומנט שהתקבל בהרצת התוכנית (str) אל הפונק' שמגרילה את הקוביות (נקרא לה drow_dice) . במידה ואין ארגומנט לmain אז יש branch נוסף בקוד בו קוראים ל-drow_dice עם str=0. לא נטפל בbrach הזה כי אין לנו צורך לשנות את ההתנהגות שלו.

נבצע הוק פיזי (ניתן להעלות רק קובץ מעודכן של dice) על הקריאה לdrow_dice (רק בbranch המטפל במקרה שיש ארגומנט בהרצת התוכנית). נערוך את התוכנית באמצעות ida כדי שנוכל להוסיף labels ולכתוב פקודת jmp לida תתרגם אותם לקפיצה לoffset המתאים.

תיאור ה-hook:

נדרוס את הפקודה call drow_dice בפקודה jmp myHook (נצטרך למצוא מקום פנוי ל- myHook).

ב- myHook נבדוק (בעזרת strcmp) את המחרוזת שהתקבלה כארגומנט ב- esp+4. אם היא שווה ל- "0Q4T3K85KL" אז נעדכן את תוכן המחסנית ב esp בערך 7. לאחר מכן נקרא ל- drow_dice וקפוץ בעזרת jmp לשורה בקוד שאחרי הקריאה המקורית ל-drow_dice .

הקוד של hook מתואר בקובץ המצורף dice_hook.txt:

```
robber_code     db '0Q4T3K85KL',0

myHook:
                mov     eax, [esp+4]
                push    eax             ; Str2
                mov     eax, offset robber_code ; "0Q4T3K85KL"
                push    eax             ; Str1
                call    strcmp
                add     esp, 8          ;clear the stack
                test    eax, eax
                jnz     short orig_call
                mov     eax, 7
                mov     [esp], eax
orig_call:
                call    drow_dice
endMyHook:
                 jmp     retFromHook
```

כדי למצוא מקום עבור הhook פתחנו את dice.exe באמצעות CFF Explorer. בטבלת Section Headers, הסתכלנו על הגודל של text. וראינו שהגודל הפיזי שלו הוא 3800 בתים והגודל הוירטואלי הוא 36A4. כלומר יש מקום בקובץ dice.exe שאינו מנוצל ואינו נטען לזיכרון כחלק מהתוכנית. ערכנו את השדה והגדלנו את הגודל הוירטואלי ל-3800 (יכלנו גם פחות אבל לא היינו מדוייקים.). כעת פתחנו שוב את הקובץ בעזרת ida ובסוף text. גילינו איזור מלא ב0. הכנסנו את הקוד של MyHook שמותר למעלה.

חזרנו לפונ' main ודרסנו את הפקודה call drow_dice ב- jmp myHook והוספנו lable retFromHook על השורה שאחרי.

חזרנו אל לוח המשחק. זרקנו כפה פעמים את הקוביות כדי לוודא שהם עדיין תקינות.

הכנסנו את הקוד: **0Q4T3K85KL-B3. ומצאנו את השודד !!!**


Give me your money!
You can't do this, I'm a US Congressman!
In that case, give me MY money!
memecenter.com