תרגיל בית 3

שם: ניר רפטאר תעודת זהות: 213036635
שם: אורי אנור תעודת זהות: 212910921

חלק יבש

1. נבצע jmp בתחילת poll אל הפונקציה hookPoll. הקפיצה אליה תתבצע ע"י דריסה וקפיצה, והפונקציה תבצע את הפעולות הבאות:
   a. תשחזר את הבתים של ה-jmp שהיא דרסה בתוך poll.
   b. תדחוף למחסנית את הארגומנטים שקיבלה כפי שהיא קיבלה אותם בזימון אל poll.
   c. תבצע call אל poll.
   d. תבצע pop לארגומנטים שדחפה.
   e. תשנה את ערך הזיכרון הגלובלי כרצונה.
   f. תכתוב מחדש את ה-jmp בתחילת poll אל ה-hookPoll.
   g. תבצע ret.

   הפונקציה גורמת לכך שהתדירות בה הפונקציה ניגשת לשרת משתנה שכן שינינו את מספר המילישניות שהיא מחכה בין בקשה לבקשה (נוכל להשיג את כתובת המשתנה הגלובלי ע"י מחקר הקוד של poll).
   בנוסף, אנחנו מתייחסים אל poll כקופסא שחורה ולא משנים אותה ולכן לא משנה שיש לה מספר נקודות יציאה אפשריות.
   נשים לב כי בשביל שלבים a,f נצטרך הרשאת write לאותו אזור בזיכרון, ניתן לקבל הרשאות אלה ע"י שינוי ההרשאות עם VirtualProtect למשל.

2. נבצע jmp לקראת סוף mine אל הפונקציה hookMine. הקפיצה אליה תתבצע ע"י דריסה וקפיצה, והפונקציה תבצע את הפעולות הבאות:
   a. תבדוק את ערך החזרה ששמור במחסנית, את הפעולה שלפני זו שנמצאת בערך החזרה (פקודת ה-call) וכך תבין מהי נקודת הכניסה של הפונקציה.
   b. תבחן את הארגומנט x, תבצע לו היפוך ביטים ותמצא את $\bar{x}$.
   c. תשחזר את הבתים ש-jmp דרסה בסוף mine.
   d. תקרא ל-mine עם הארגומנט $\bar{x}$.
   e. תכתוב מחדש את ה-jmp בתחילת poll אל ה-hookMine.
   f. תעשה pop לארגומנט x, ותשים בערך החזרה (eax) את $\bar{x}$.
   g. תבצע ret.

   באמצעות בדיקת ערך החזרה, הפונקציה מוצאת את נקודת הכניסה של התוכנית mine, קוראת לה עם הארגומנט המתאים וכך מבצעת את החישוב הדרוש, ולבסוף מחזירה אותו שמבוקש.
   בנוסף, מכיוון שה-hook ממוקם בנקודת היציאה היחידה של הפונקציה הוא יתרחש ללא תלות בנקודות הכניסה השונות.
   נשים לב כי בשלבים c, e נצטרך הרשאת write לאותו אזור בזיכרון, ניתן לקבל הרשאות אלה ע"י שינוי ההרשאות עם VirtualProtect למשל.

3. נבצע jmp בתחילת sendf אל הפונקציה hookSendf. הקפיצה אליה תתבצע ע"י דריסה וקפיצה, והפונקציה תבצע את הפעולות הבאות:
   a. תציב את הארגומנטים ש-sendf קיבלה כפרמטרים בתוך מחרוזת הפורמט שהיא קיבלה (כמו שקורה ב-printf לפני שהיא מדפיסה).

b. תצפין את המחרוזת שנוצרה באמצעות פונקציית ההצפנה הרצויה.
c. תבצע pop מהמחסנית למחרוזת הפורמט הישנה ואל הארגומנטים המועברים אליה, ותדחוף את המחרוזת החדשה שנוצרה מפונקציית ההצפנה.
d. תבצע את הפקודה/פקודות שנדרסו, כולל פקודות חלקיות שנדרסו. למשל אם נדרסו פקודה וחצי, יתבצעו שתי הפקודות הראשונות של sendf.
e. תקפוץ לאחרי הפקודות שביצעה ב-sendf.

נשים לב כי מכיוון שמחרוזת היא סוג של מחרוזת פורמט ושניתן להעביר 0 ארגומנטים sendf תקבל ארגומנטים חוקיים גם לאחר השינוי, ותשלח ב-socket את המחרוזת הרצויה מוצפנת.

4. נשים קפיצה ל-connectHook בתחילת connect (למשל על ידי דריסה וקפיצה).
connectHook תבצע את הדברים הבאים:
1) תיצור thread נוסף. נקרא ל-thread המקורי oThread ול-thread החדש nThread.
לאחר יצירתו, nThread יבצע את הדברים הבאים:
1) ישחזר את הפקודות ב-connect שנדרסו.
2) יקרא ל-connect עם הארגומנטים המקוריים שהיא הייתה אמורה לקבל באותו הסדר.
3) יכתוב מחדש את הקפיצה שנמצאת בתחילת hook ל-connectHook
4) יעשה pop לארגומנטים שדחף ויסיים (ה-thread ימות לאחר פקודה זו).
לאחר שלב 1, oThread יבצע את הדברים הבאים:
1) יעשה ret (בעצם בדיוק לקוד שקורא ל-parse)
בעצם אחד ה-threads ישר קורא ל-connect והשני ישר קורא ל-parse.
יש לשים לב כי יש התעלמות מערך החזרה של connect אך אין דרך להתייחס גם לערך החזרה של parse וגם של connect בתוכנית המקורית שכן היא מתבצעת בחוט אחד במקור (אך השינויים ש-parse ו-connect עושים בזיכרון התוכנית כן יקרו וישפיעו).

5. ה-jmp בתחילת calc יהיה לפונקציה calcHook. ניתן להחדיר את הקוד של calcHook לתהליך ע"י DLLInjection למשל.
הקפיצה ל-calcHook מתבצעת ע"י דריסה וקפיצה.
calcHook תבצע את הפעולות הבאות:
1) תשחזר את הבתים של ה-jmp שנמצאים בתחילת calc להיות הבתים המקוריים שהיו שם.
2) תדחוף למחסנית את הפרמטרים שהיא קיבלה (הפרמטרים שהיו מיועדים ל-calc) באותו סדר שהיא קיבלה אותם.
3) תקרא ל-calc (בעצם קוראים ל-calc מתוך calcHook עם אותם הפרמטרים ש-calc הייתה אמורה לקבל).
3.5) תעשה pop לארגומנטים שדחפה בשביל calc
4) תעשה log לערך ש-calc חישבה.
5) תכתוב מחדש את ה-jmp ל-calcHook שהיה בתחילת calc
6) תבצע ret (בחזרה לקוד שקרא ל-calc בפעם הראשונה).
יש לשים לב כי בשביל שלב 1,5 נצטרך הרשאת write לאותו אזור בזיכרון, ניתן לקבל הרשאות אלה ע"י שינוי ההרשאות עם VirtualProtect למשל.
הסיבה שה-hook המתואר יעבוד היא כי קוראים ל-calc באותה דרך, אך מוודאים שכתובת החזרה ממנה היא לקוד שלנו, וכך ניתן לראות את ערך החזרה בקלות.
כמו כן, מוודאים שבזמן ש-calc באמת רצה, הקוד שלה שלם ותקין ולכן הערך שתחזיר אינו "לא מוגדר", כפי שרצינו, וכן השינוי בשלב 1 קורה לפני שהפונקציה בודקת את שלמותה (לפני כל הפונקציה).

6.
a. נשים קפיצה בתחילת solve לפונקציית solveHook (על ידי דריסת הבתים הראשונים ממש בתחילת solve), שהיא תהיה הפונקצייה של ההוק.

solvehook יכול להיות למשל ב-dll ונעשה dll injection
solvehook תבצע את הדברים הבאים:
1) תדחוף את כתובת X (ראו בהמשך) למחסנית.
2) תבצע את הפקודה/פקודות שנדרסו ב-solve, באותו אופן שנעשה בשאלה 3 שלב d
3) תקפוץ ל-solve, בדיוק בפקודה הראשונה שלא נדרסה. למשל אם נדרסו פקודה וחצי, תקפוץ לתחילת הפקודה השלישית של ב-solve
4) תכפיל את eax ב-2, ותעשה ret.
כתובת X היא הכתובת בה מתחילה הפקודה הראשונה של שלב 4
הסבר:
שלב 1 נועד כדי לדחוף ret addres. פונקציית solve שתרוץ בשלב 3 תחזור ל-X כשתסיים.
הסיבה שעבדנו כך היא כדי שנוכל לוודא ש-solve שתרוץ בשלב 3 תתבצע בצורה רציפה עם ה-control flow המקורי (מלבד ה-jmp שהוספנו).
חשוב לציין כי ה-jmp שקופץ בחזרה ל-solve לא אמור לשנות את התנהגות solve, שכן פקודת jmp לא משנה דגלים, לכן גם אם יש מקרה בו הפקודה הראשונה של solve שלא נדרסה תלויה בפקודה האחרונה שנדרסה, הן יקרו (כמעט) בצורה רציפה, והתנהגות solve תשאר זהה (מלבד הכפלת הערך המוחזר לאחר מכן ב-solveHook).

b. נשים קפיצה בתחילת solve לפונקציית solveHook (על ידי דריסת הבתים הראשונים ממש בתחילת solve), שהיא תהיה הפונקצייה של ההוק.
solvehook תבצע את הדברים הבאים:
1) תשחזר את הבתים של ה-jmp שנמצאים בתחילת solve להיות הבתים המקוריים שהיו שם.
2) תדחוף למחסנית את הפרמטרים שהיא קיבלה (הפרמטרים שהיו מיועדים ל-solve) באותו סדר שהיא קיבלה אותם.
3) תקרא ל-solve (בעצם קוראים ל-solve מתוך solveHook עם אותם הפרמטרים ש-solve הייתה אמורה לקבל).
4) תכפיל את הערך שחזר מ-solve פי 2 ותשמור אותו.
4.5) תעשה pop לארגומנטים שדחפה למחסנית בשביל solve
5) תכתוב מחדש את ה-jmp ל-solveHook שהיה בתחילת solve (בשביל קריאות עתידיות).
6) תבצע ret ותחזיר את הערך שנשמר בשלב 4 (בחזרה לקוד שקרא ל-solve בפעם הראשונה, בקריאה שאינה רקורסיבית).
בעצם, בגלל שהסרנו את ה-jmp ל-hook שבתחילת solve אפשר לקרוא לה ולקבל את הערך הסופי מהקריאה החיצונית ביותר בלבד, ואז אפשר להחזיר את ערכו פי 2.

חלק רטוב

חלק ראשון - ניתוח דינאמי

התחלנו מלהריץ את keygen.exe, וניסינו להכניס כל מיני קלטים ולראות אילו פלטים נפלטים.
ראינו שכל אות מתורגמת לאות אחרת, ללא תלות באותיות האחרות או באינדקס שלה בקלט.
פתחנו את keygen.exe ב-IDA, עברנו קצת על הקוד. ראינו קריאה "call eax".
על מנת לעקוב אחרי הקוד ולבצע ניתוח דינמי פתחנו את WinDBG ופתחנו בו את keygen.exe (עם ארגומנט ארביטרי "aaaa" כדי שלא יקרוס). ראינו את הכתובת של ה-"call eax" ב-IDA ולכן יכולנו לשים בכתובת זו breakpoint ב-WinDBG (ע"י bp 004014EA).
לאחר הקריאה, ראינו קוד שחלק ממנו כותב טבלה (הרבה בתים בצורה רציפה) לזיכרון. הבתים נראו כמו קידוד תווי ascii.
בהמשך המעקב על הקוד ראינו שיש גישה לתא בטבלה באינדקס כלשהו (הנחנו שהערך הזה זה התו שנקרא בקלט). ניחשנו שזה צופן פרמוטציה אבל עוד לא היינו בטוחים.
ראינו שהקידוד של 'a' הוא '?'. חיפשנו את הקידוד של '?' בטבלה, וראינו שהקידוד שנכתב מיד אחריו לזיכרון הוא של '$'. בדקנו ואכן זה היה הקידוד של 'b'. ראינו שביחס לקידוד של 'a', הטבלה מתחילה בקידוד של ' ' ונגמרת בקידוד של '~' (בעצם כל התווים ה-printable).
העתקנו את כל הטבלה (לפי הפקודות שרושמות אותה,
למשל עבור פקודה "mov byte ptr [ebp-2Fh], 7Eh" העתקנו רק את 7E לתוך קובץ). כתבנו תוכנית שמבצעת את הפרמוטציה ההפוכה על קלט ומדפיסה את התוצאה:

```
int main(int argc, char** argv)
{
    //reading the mapping
    char printable['~' - ' ' + 1];
    char mapped['~' - ' ' + 1];
    for (int i = ' '; i <= '~'; i++)
    {
        printable[i - ' '] = i;
    }
    ifstream mapping("extras\\mapping.txt");
    string mapstr;
    char idx = ' ';
    if (mapping.is_open())
    {
        while (mapping && idx <= '~')
        {
            mapping >> mapstr;
            std::stringstream ss;
            ss << "0x" << mapstr[0] << mapstr[1];
            char mapped_c = std::stoul(ss.str(), nullptr, 16);
            mapped[mapped_c - ' '] = idx++;
        }
    }
    //reversing the mapping
    string key(argv[1]);
    for (int i = 0; i < key.length(); i++)
    {
        cout << mapped[key[i] - ' '];
    }
    return 0;
};
```

(יש לשים לב כי קובץ זה משתמש בקובץ mapping.txt שמכיל את הטבלה. mapping.txt בתיקיית extras, וקובץ keygen_rev מניח שהוא נמצא בתוך התיקייה הזאת ביחס אליו).

הכנסנו כקלט את הסיסמה לאתר "CRR0W8SYYUI9D5H4", קיבלנו את הפלט

V11`Q*[88s&k0!]\

וכשהכנסנו אותו בתור סיסמה לגישה לדף Tools הסיסמה התקבלה וקיבלנו גישה.

<u>חלק שני - הוקינג</u>

התחלנו ממעבר על הקוד של client. זיהינו את המקום בו הוא שולח ומקבל בקשות מהשרת (לפי send, recv).

עברנו על הקוד של secure_pipe. ראינו שרוב הקריאות הן לפונקציות שקשורות ל-cpp (כנראה), וראינו פונקצייה אחת שהיינו יכולים לקרוא. אחרי קריאת הקוד ופיענוחו הבנו איך ההצפנה עובדת:

ההצפנה עובדת באופן הבא:

היא עוברת תו-תו על המחרוזת המתקבלת, מפרקת כל תו (char) ל-2 מספרי ה-hex המתארים אותו ומצפינה כל hex כך:

1. אם הוא 0, בהצפנה הוא יסומן כמספר רנדומלי כלשהו x פחות עצמו, כלומר כרצף התווים "x-x".
2. אם הוא 1, בהצפנה הוא יסומן כ-'A'.
3. אם הוא ספרה בין 2-9, נשאיר אותו כמו שהוא.
4. אם הוא 10, בהצפנה הוא יסומן כ-'J'.
5. אם הוא 11, בהצפנה הוא יסומן כ-'Q'.
6. אם הוא 12, בהצפנה הוא יסומן כ-'K'.
7. אחרת (כלומר, 13/14/15), המספר (אותו נסמן ב-n) יסומן ע"י 2 ספרות שסכומן הוא המספר, כלומר כרצף התווים "x-n+x".

על בסיס ניתוח זה, כתבנו פונקציית C הופכית, שמפענחת את ההודעה המוצפנת.

כתבנו injector ו-dll (לפי איך שלמדנו בתרגולים ובסדנא, ועם שימוש ב-templates שבאתר).

הוספנו את הפונקציה של ה-decryption לקוד של ה-dll, שינינו את ה-templates להתאים למקרה שלנו. כמו כן ביצירת client.exe ב-injector אנחנו מעבירים ל-client ישר את DMSG (כחלק מיצירתו).

שמנו הוק בפונקצית puts, מכיוון שהיא הפונקציה שמדפיסה את ההודעה המוצפנת, לכן יכולנו לוודא שאנחנו עושים decrypt להודעה בדיוק לפני שמדפיסים אותה. מכיוון ש-puts היא פונקצית ספרייה (שאינה טעונה מראש ב-exe) הקריאה אליה היא דרך IAT לכן בחרנו לעשות IAT hook.

ב-IAT בכניסה של puts (ידענו מה הכתובת לפי ida) כתבנו קפיצה לפונקציה שלנו (פונקציית ההוק). פונקציה זו מבצעת את הדברים הבאים:

1) מסירה את הקפיצה לפונקציית ההוק ב-IAT. (קראנו את הבתים שהיו בטבלה מראש, וכך ידענו אילו בתים לכתוב כדי להסיר את ההוק)

2) קוראת ל-decrypt עם ה-buffer ש-puts קיבלה כפרמטר (decrypt משנה את ה-buffer ישירות)

3) קוראת ל-puts עם אותו buffer

4) שומרת את ערך החזרה שקיבלנו מ-puts

5) משחזרת את הקפיצה להוק ב-IAT

6) יוצאת ומחזירה את ערך החזרה מ-puts

כמו כן, יש לשים לב ש-decrypt לא מבצעת כלום אם המחרוזת שהיא מקבלת היא אחת המחרוזות שמודפסות בתור ההוראות. כלומר בהוק הספציפי הזה ל-client.exe, המחרוזת היחידה ש-decrypt תרוץ עליה היא ההודעה המוצפנת.

העלנו zip לשרת ל-client שמכיל את ה-injector בשם toolfix.exe, את ה-dll ואת client.exe. הרצנו וקיבלנו את הפלט הבא:

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 C2. 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.

<u>חלק שלישי - שחרור השודד</u>

התחלנו ממעבר על רוב הקבצים שבעמוד tools.

ראינו שבהודעה של גובלין הוזכרו קודים ותוצאת 7 של קוביות, לכן החלטנו לאחר זמן מה להתמקד ב-codes.exe וב-dice.exe.

לאחר שעברנו על codes, הבנו שיש בו קריאה לפונקציה הבאה:

```
codes = Code.query.filter_by(code = '%s').all(); \nprint('NO SUCH CODE' if len(codes) <= 0 else codes[0].event.key)
```

פונקציה זו שומרת ב-buffer את ה-event (ניחוש) שמתאים ל-OLD_KEY (הארגומנט השני ש-codes.exe מקבל), ומדפיסה "NO SUCH CODE" אם לא קיים קוד OLD_KEY במערכת.

לאחר מכן ב-codes, יש פונקצייה שמדפיסה (כנראה בנוסף לפעולות נוספות) את הקוד שמשוייך ל-CODE_KEY (הארגומנט הראשון של codes.exe), אם ה-event שנשמר ב-buffer זהה ל-CODE_KEY (ואחרת יוצא):

```
codes = [code for code in Event.query.filter_by(key = '%s').first().codes if not code.used];
\nprint(" if len(codes) <= 0 else codes[0].code)
```

כלומר, היא עושה זאת ע"י סידור כל הפונקציות המתאימות ל-CODE_KEY במערך והחזרת הפונקציה הראשונה בו.

ניחשנו ש-CODE_KEY מייצג event.

הבנו שכדי להגיע להדפסה של הקוד אנחנו צריכים לדלג על הבדיקה של הפונקציה הראשונה ועל הקפיצה בפונקציה השנייה, מכיוון שאין לנו את ה-OLD_CODE שמתאים ל-CODE_KEY.

כדי לדלג על הבדיקה בפונקציה הראשונה עשינו hook בתחילת הפונקצייה (בדיוק אחרי ה-push-ים) הראשוניים. עשינו את ההוק באמצעות dll injection, מכיוון שיכולנו להעלות zip בשביל לעדכן את codes.exe, ולכן הכי נוח היה לעשות dll injetion ולהעלות dll ו-injector.

כמו כן עשינו דריסה וקפיצה, כי הקריאה לפונקציה שבחרנו לעשות לה הוק לא מצבעת דרך IAT ולא היתה אפשרות בה ל-hot patching.

מה שפונקציית ה-hook עושה זה בדיוק את הפעולות המשלימות שהתבצעו לפני הקפיצה להוק, ואז ret.

כלומר הפקודות בפונקציה:

```
push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
add     esp, -80h
```

דרסנו את השורה האחרונה שבתמונה (add esp, -80h) עם jmp להוק, שמבצע:

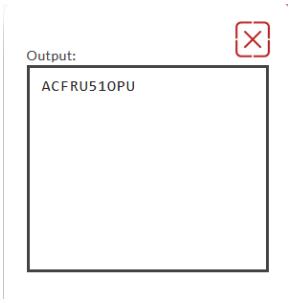
```
pop     ebx
pop     esi
pop     edi
pop     ebp
ret
```

ככה אנחנו מוודאים שאנחנו מדלגים בדיוק על הפונקציה בלי לפגוע בשאר פעילות התוכנית.

(היה אפשר באותה מידה לעשות override ל-4 השורות הראשונות עם jmp ואז לעשות רק ret בפונקציית ה-hook, אך זה שקול מבחינת התנהגות התוכנית).
לאחר מכן רצינו לעבור את הבדיקה השנייה.
ראינו שבצורה ספציפית מה שמתבצע זה strcmp בין CODE_KEY ל-buffer, והתוכנית יוצאת אם הם שונים. כדי לעבור את הבדיקה הזאת, ב-dll injection רשמנו קוד שמשנה את הקידוד של הפקודות כדי שהפרמטרים ש-strcmp תקבל יהיו מצביע לאותה מחרוזת.
כלומר בקוד הבא:

```
mov     eax, [ebp+10h]
mov     [esp+4], eax     ; Str2
mov     eax, [ebp+0Ch]
mov     [esp], eax       ; Str1
call    strcmp
```

שינינו את השורה השלישית מ-mov eax, [ebp+0Ch] ל-mov eax, [ebp+10h].
וכך הבטחנו ש-strcmp תמיד מחזיר שהמחרוזות שוות והבדיקה עוברת בהצלחה.
כדי לוודא שאנחנו מקבלים את הקוד שמתאים לאירוע ROBBER_CAPTURED (כפי שהגובלין אמר) העברנו את "ROBBER_CAPTURED" בתור הארגומנט הראשון ל-codes.exe (שה-injector מריץ) והעברנו מחרוזת ארביטרית בשביל הארגומנט השני.
העלנו את הקבצים המתאימים לאתר (injector בתור toolfix.exe, ה-dll ו-codes.exe) וקיבלנו:



מההודעה של הגובלין, הבנו שהמשבצת שלנו היא C2.
נזכרנו בקוד שהיה בתרגיל בית הקודם ובמבנה שלו, וניסינו להכניס את ACFRU51OPU-C2 לאזור הקודים בעמוד ה-game board באתר, וקיבלנו גיף של ג'ים מ-The Office אומר "so close".

לאחר שעברנו על dice.exe ראינו שבהתחלה מגרילים תוצאה לקוביות (בהסתברות ממושקלת לפי הסיכוי של כל תוצאה לצאת) ואז קוראים לפונקציה שמפצלת את הערך לשתי קוביות (למשל לפצל 4 ל-2,2 או 3,1). נקרא לפונקציה זו split_dice.
ראינו שכאשר dice מקבלת ארגומנט, היא מעבירה אותה ל-split_dice בארגומנט השני שלה. כמו כן ראינו ש-split_dice מקבלת בארגומנט הראשון שלה גם את הערך של הקוביות שהוגרל.
ראינו ש-split_dice מבצעת קריאה לפונקציה כלשהי (שמורצת ע"י קוד פייתון) ואם פונקצייה זו מחזירה "NO_ROBBER", נכנסים ללולאה שמוודאת שלא יוצא 7 בקוביות.
ראינו שהקריאה לפונקציית פייתון זו מתבצעת רק הארגומנט ש-split_dice קיבלה (ארגומנט שורת הפקודה הראשון) אינו NULL.
לאחר הכוונה מקבוצת הווטסאפ עם הסגל הבנו שאנחנו צריכים לגרום לקוביות לצאת 7 תמיד אחרי שמשתמשים בקוד שיש לנו. כמו כן הבנו שהארגומנט ש-dice מקבלת הוא NULL (בהטלה רגילה) או הקוד שהוכנס אם הוכנס קוד.
מכיוון שאפשר לעלות לאתר רק קובץ exe בשביל dice הבנו שאנחנו צריכים לעשות הוק פיזי, שכן אין לנו אפשרות להעלות dll ל-dll injection (או injector). כמו כן ההוק נקרא ע"י דריסה וקפיצה כי הפונקציה שעשינו לה הוק (מוסבר בהמשך) לא נקראת דרך IAT ולא ראינו אופציה ל-hot patching.
תכננו לעשות hook ל-split_dice, מכיוון שבפונקציה זו נכלל הקוד שקובע האם יכול לצאת בקוביות 7, לכן נוכל לשנות את הארגומנטים כרצוננו כדי לוודא שיצא 7 וכדי לוודא שהתנאי שגורם לקובייה להיות שונה -7 בוודאות נכשל.
התחלנו מלכתוב את הקוד של הפונקצייה שאליה נקפוץ בהוק.
הקוד בודק אם split_dice קיבלה בארגומנט השני שלה NULL (כלומר אם dice קיבלה ארגומנט).
אם כן, משווה את הארגומנט ל-"ACFRU51OPU" ואם הם שווים משנה את הערך של הארגומנט הראשון (הערך הכולל של הקוביות) ל-7, ומשנה את הערך של הארגומנט השני ל-0 (כדי לדלג על הפונקציה שקוראת לקוד הפייתון כדי שלא נגיע ללולאה שמוודאת שהתוצאה שונה מ-7). אחר כך מבצעים את הפקודה שדרסנו מ-split_dice וקופצים להמשכה.

אם הארגומנט השני הוא NULL או שהוא שונה מהקוד הנכון, הפונקציה מבצעת את הפקודה שנדרסה ב-split_dice וחוזרת לבצע את המשכה (הפעם בלי לשנות את הערך הכולל של הקוביות).
באמצעות CFF Explorer הוספנו section ריק חדש בגודל שמתאים לפקודות (קצת יותר בעצם אבל מילאנו את מה שלא השתמשנו ב-nop), ובאמצעות CFF Explorer ו-ida כתבנו את הקידוד של הפקודות ב-section החדש. כמו כן היינו צריכים לשנות ערכים ב-offsets של קפיצות.
העלנו את ה-dice.exe המעודכן, ולאחר הכנסת הקוד "ACFRU51OPU-C2" קיבלנו את השודד.