תרגיל בית 3

חלק יבש

1. בסעיף זה נשתמש בדריסה וקפיצה (וכן נשתמש ב-DLL Injector).
אנו נדרוס את הפקודות הראשונות ונבצע קפיצה אל פונק' ה-HOOK שלנו, ובה אנו נשנה את הזכרון הגלובלי להיות המספר שאנו רוצים שיהיה שם.
בפונק' ה-HOOK שלנו ניעזר בדגל גלובלי שיסמן לנו מתי יש לשנות את הזכרון הגלובלי – כאשר הטיימר הגיע ל-0, אנו יודעים שהפונק' poll הולכת לשנות את הערך, כך שתישמר התדירות הרגילה, ולכן נסמן לנו בדגל שעלינו לשנות את הטיימר לתדירות הרצויה שלנו.
בקריאה הבאה של poll, בזכות הדגל, נוכל לדרוס את הערך שהיא כתבה ולכתוב את הערך הרצוי.
כמובן, נדאג לשחזר את כל הפקודות האחרות שקורות ודרסנו במידת הצורך.
להלן פסאודו-קוד המסביר את אופן הפעולה שלנו:

```
poll_hook() {
        if (change_next_time)
                t = 30;
        }

        if (t == 0) {
                change_next_time = true;
        }
        else {
                change_next_time = false;
        }

        restored_instructions();
        jmp to poll + 5;
}

poll() {
        jmp to poll_hook //set_after_dll_is_loaded
        more_important_stuff();
        send_things();

        if (t == 0) {
                do_actions();
                t = 20;
        }

        other_stuff();
}
```

2. בסעיף זה נשתמש בדריסה וקפיצה וכן נשתמש ב-DLL Injector.
השימוש בדריסה וקפיצה יהיה על-מנת לקפוץ אל הפונק' ה-HOOK שלנו שתיכתב ב-DLL. אנו נבצע את הדריסה בסוף הפונקציה, לפני נק' היציאה היחידה (וכך נוודא שה-HOOK שלנו ייקרא תמיד, גם אם ניכנס לפונקציה מנק' כניסה שונות).
ב-DLL שלנו יהיה משתנה גלובלי שיהווה דגל (ומיד נסביר את השימוש בו), והוא יאותחל ל-0.
בפונק' ה-HOOK נפעל בצורה שונה לפי ערך הדגל. אם ערך הדגל הוא 0 (המשמעות שזוהי הקריאה הראשונה לפונק'), נשנה את הדגל להיות 1, ואז נוכל להכין מחסנית בצורה הבאה: נשנה את הארגומנט x להיות $\bar{x}$ (נתון שהפונק' משנה רגיסטרים ומשתנים מקומיים בלבד, ולכן אנו מניחים שגם בסוף הפונק' הארגומנט נשאר x), נחסיר מינוס 5 מערך החזרה ששמור על המחסנית (הסיבה למינוס 5, היא שפקודת call <offset> היא באורך 5 בתים, ובצורה זו אנחנו נבצע שוב את הקריאה לפונק' mine ברגע שנחזור

מהפונק' וכעת הארגומנט הוא $\bar{x}$). לאחר מכן, נבצע חזרה מפונקציית ה-HOOK אל ערך החזרה ששונה.
אם ערך הדגל הוא 1, המשמעות היא שכבר ביצענו את השינויים (וכיוון שאנחנו לא רוצים שהם יקרו רקורסיבית שוב ושוב, עלינו לדאוג לא לבצע זאת שוב), ולכן במקרה הזה, נשנה את $\bar{x}$ להיות x (למקרה שיש שימוש במשתנה הנ"ל במחסנית, אנו רוצים להחזיר את המצב לקדמותו), ואז אנו נבצע חזרה רגילה מהפונק'.
כמובן שגם נשחזר כחלק מה-HOOK את הפקודות שדרסנו (כחלק מדריסה וקפיצה).

3. נבצע דריסה וקפיצה (או HOT PATCHING במידה וקיים), ושימוש ב-DLL Injector.
אנו ניגש לארגומנט השני (שמגדיר את פורמט ההודעה), ונבנה את ההודעה על-פי הפורמט והארגומנטים הנוספים (נשתיל את הארגומנטים במקומות המתאימים). לאחר מכן, נצפין את ההודעה, ונקפוץ אל הפונק' המקורית בנק' בה מסתיימת בניית ההודעה שתישלח ומבצעים שליחה שלה.

4. נבצע דריסה וקפיצה (או HOT PATCHING במידה וקיים), ושימוש ב-DLL Injector.
נבצע זאת בפונק' connect שנקראת קודם. בפונק' ה-HOOK שלנו, אנו ניצור thread (בעזרת ספריה מתאימה, או פונק' winapi ליצירת thread-ים), שיריץ את הפונק' connect (עם הארגומנטים שלה, במידה ויש כאלו). את ההרצה של connect נוכל לבצע על-ידי קריאה מפורשת במידה והפונק' נגישה לנו (ניתן לייבא אותה ולקרוא לה), אחרת נוכל לחלץ את הכתובת שלה על-פי ערך החזרה השמור במחסנית ופקודת הקפיצה שהתבצעה 5 בתים קודם לכן (וחישוב כתובת הפונק' connect על-פי ה-offset). אחר כך, נבצע return ובכך נחזור למקום בקוד שקורא לפונק' parse – היא תתבצע בתהליך הראשי של התוכנית, ולכן לאחר סיומה, התוכנית תמשיך לרוץ (נתון כי אין צורך לחכות לסיום שתי הפונק', אלא מספיק שאחת מהן מסיימת).

5. נבצע דריסה וקפיצה (או HOT PATCHING במידה וקיים), ושימוש ב-DLL Injector.
בפונק' ה-HOOK שלנו נבצע תיקון של הפונק' המקורית (נוריד את השתלת ה-HOOK שביצענו, כך שהפונק' לא תזהה את השינוי ולא תחזיר ערך אקראי), לאחר מכן נדחוף את הארגומנטים שהועברו שוב למחסנית, נבצע קריאה (ולא קפיצה) לפונק' המקורית, כך שבסיומה השליטה תחזור אלינו.
לאחר חזרת הפונק' נבצע הדפסה של הערך המוחזר, וכן נשתיל את ה-HOOK שלנו שוב לטובת הקריאה הבאה.
להלן פסאודו-קוד המתאר זאת:

```
hook {
      restore_original_func; //clean the seeding of the hook
      push_args; //for the original func
      ret_val = call original_func(); //return address of hook
is on stack
      pop_args; //so the stack will be restored
      printf(ret_val);
      seeding_the_hook() //seeding the hook again so it will be
called next time
      return_to_where_original_func_was_called; //ret address
is on stack
}
```

6. נתייחס לשני המקרים:
   a. נבצע דריסה וקפיצה (או HOT PATCHING במידה וקיים), ושימוש ב- DLL Injector.
   בפונק' ה-HOOK שלנו נדאג לקיומם של הארגומנטים על המחסנית, ונבצע קריאה (ולא קפיצה) לפונק' המקורית (כאשר נדאג לבצע את הפקודות המשוחזרות במידה ויש צורך), כדי שהשליטה תחזור אלינו בעת חזרת הפונק'.

בחזרת הפונק', נכפיל את ערך החזרה פי 2, ונבצע חזרה אל המיקום ממנו הפונק' נקראה (במקרה הרקורסיבי – אל אחת הקריאות הקודמות).
נשים לב, שפונק' ה-HOOK שלנו תיקרא בכל קריאה רקורסיבית ותבצע אותו דבר.
פסאודו-קוד המסביר הזאת:

```
hook {
      push_args; //for the original func
      restored_instructions();
      ret_val = call original_func(); //return address of hook
is on stack
      pop_args; //so the stack will be restored
      ret_val = ret_val * 2
      return_to_where_original_func_was_called; // ret address
is on stack
}
```

b. נבצע דריסה וקפיצה (או HOT PATCHING במידה וקיים), ושימוש ב- DLL Injector.

בדומה לסעיף 5, בפונק' ה-HOOK שלנו נבצע תיקון של הפונק' המקורית (נוריד את השתלת ה-HOOK שביצענו, כך שפונק' ה-HOOK שלנו לא תתבצע בכל קריאה רקורסיבית, אלא רק בקריאה הראשונה), לאחר מכן נדחוף את הארגומנטים שהועברו שוב למחסנית, נבצע קריאה (ולא קפיצה) לפונק' המקורית, כך שבסיומה השליטה תחזור אלינו בעת חזרת הפונק'.
בחזרת הפונק' (כלומר בקיפול כל הרקורסיה, כיוון שרק בקריאה הראשונה בוצעה פונק' ה-HOOK), נכפיל את ערך החזרה פי 2, ונבצע חזרה אל המיקום ממנו הפונק' נקראה (המיקום בקוד שביצע את הקריאה הראשונה לפונק', ולא הקריאות הרקורסיביות).
לפני החזרה, נדאג להשתיל את ה-HOOK בשנית, כדי לוודא שייקרא בקריאה הבאה לפונק' הרקורסיבית.
פסאודו-קוד המסביר הזאת:

```
hook {
      restore_original_func; //clean the seeding of the hook
      push_args; //for the original func
      ret_val = call original_func(); //return address of hook
is on stack
      pop_args; //so the stack will be restored
      ret_val = ret_val * 2
      seeding_the_hook() //seeding the hook again so it will be
called next time
      return_to_where_original_func_was_called; // ret address
is on stack
}
```

חלק רטוב

חלק ראשון

הורדנו את הקובץ מאתר הקורס והתחלנו לבצע ניתוח של קובץ ההרצה בעזרת IDA.
איתרנו את פונק' ה-main, והתחלנו לנתח את הפעולות המתבצעות בה.
ראינו כי מתבצעות פעולות שונות, המשנות ומעתיקות מחרוזות אל המחסנית. בעזרת דיבוג הצלחנו להבין כי בעצם הועתקו פקודות שונות אל המחסנית, ולאחר מכן מתבצעת קפיצה על איזור זה במחסנית ובעצם מבצעים את הפקודות השונות שנכתבו במחסנית.
התחלנו לנתח את הפקודות הנ"ל וראינו כי הן אלו שאחראיות לג'ינרוט הסיסמה. שמנו לב, כי ישנו מערך גדול, ממנו נלקחים תווים שונים באופן הבא:
עוברים תו-תו ממחרוזת הקלט – עבור תו כלשהו מבצעים חיסור של ערכו ב-0x20, ולאחר מכן משתמשים בערך לאחר החיסור כאינדקס גישה למערך התווים הגדול, וממנו נבחר התו המג'ונרט.
כדי לכתוב את התוכנית ההופכית, העתקנו מערך זה, וביצענו פעולה הפוכה – לכל תו בדקנו מהו האינדקס שלו במערך, ולאינדקס ביצענו חיבור של 0x20, וכך קיבלנו את תו המקור ששימש עבור ג'ינרוט התו.

חלק שני

התחלנו לנתח את הקובץ client.exe כפי שהוזכר בהוראות התרגיל. מצאנו כי מתבצעת הדפסה של מחרוזות המבקשות הקשת בחירה של הפעולה הרצויה. לאחר מכן מתבצעת קליטה של אותה בחירה ולאחר מכן עוברים לביצוע הפעולה עצמה.
ראינו את חלק הקוד בו מתבצעות הפעולות השונות, ובפרט DMSG, ולא הבחנו במשהו חריג – מתבצעת קליטה של ההודעה בעזרת recv, ולאחר מכן העתקה של המחרוזת למיקום הרצוי והדפסתה.
בהתאם להוראות התרגיל, התחלנו לנתח גם את הקובץ secure_pipe.exe ואכן מצאנו בו את פונק' ההצפנה:
לכל בית בהודעה המתקבלת, מחלקים אותו ל-4 ביטים עליונים ותחתונים, כאשר לכל אחד מהם מבצעים הצפנה בצורה זהה, כשמתחילים מהביטים העליונים, ואז עוברים לתחתונים.
כתלות בערך הביטים מבצעים את אחת מהאפשרויות:

1. מבצעים השמה של 'A' במידה וערך הביטים הוא 1.
2. מבצעים השמה של 'J' במידה וערך הביטים הוא 10.
3. מבצעים השמה של 'Q' במידה וערך הביטים הוא 11.
4. מבצעים השמה של 'K' במידה וערך הביטים הוא 12.
5. במידה וערך הביטים הוא בין 2 ל-9 (כולל), מבצעים השמה של [ערך הביטים]+'0'.
6. במידה וערך הביטים הוא 0, מבצעים השמה של מס' רנדומלי (עם מודולו), לאחר ביצוע חיבור של ערכים כלשהם, לאחר מכן השמה של התו '-' ואז עוד השמה של מס' כלשהו (מקרה זה ניתן לזיהוי על-פי התו '-').
7. אחרת, מבצעים שני חישובים ארתימטי כלשהו עם ערך רנדומלי וערכי הביטים, מבצעים השמה של הראשון מביניהם, לאחריו השמה של '+', ואחר כך השמה של הערך החישוב השני (מקרה זה ניתן לזיהוי על-פי התו '+') – במקרה זה ניתן לחלץ את ערך הביטים על-ידי חיסור שני הערכים שחושבו אריתמטית (וחיסור של הערך '0' פעמיים).

לאחר שהבנו את אופן ההצפנה, יכולנו לכתוב קוד המפענח את ההצפנה (והוא נמצא ב-ClientHook.cpp).

כיוון שאנו רוצים לפענח את ההודעות המגיעות, עלינו להשתיל את הפענוח כאשר ה-client מקבל את ההודעות. כפי שאמרנו הוא מבצע אותן בעזרת הפונק' recv, ולכן בחרנו לבצע לה hook.
בגלל שאנו רוצים לבצע פענוח לאחר קבלת ההודעה, ולא לשנות משהו בתחילת הפונק', בחרנו לבצע IAT hook, בו נשנה את הכניסה בטבלה לפונק' ה-hook שלנו, בה נבצע קריאה רגילה ל-

recv, ולאחר מכן פענוח של ההודעה שהתקבלה ב-buffer, ולאחר מכן נחזיר את מס' התווים שנקראו (לאחר פענוח).
נעזרנו בטמפלייטים מהאתר עבור ה-injector, כאשר דאגנו לשנות את שמות הקבצים המתאימים ודאגנו להעביר ל-client.exe את הארגומנט DMSG, וכן עבור ה-DLL המתאים לביצוע IAT hook.
לאחר חיפוש ברשת ראינו כי הפונק' recv נמצאת ב-DLL בשם Ws2_32.dll, ולכן ביצענו חיפוש של הפונק' בו (בתהליך הנוכחי – לכן העברנו ארגומנט NULL עבור קבלת ה-handle של התהליך), ומצאנו את ה-offset של הכניסה המתאימה בטבלה בעזרת IDA כפי שלמדנו בסדנה.
לאחר השלמת ה-hook אכן מצאנו את ההודעה:

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.
Goblin

<u>חלק שלישי</u>

מההודעה שהתקבלה, וכן על-פי הודעתו של אלעד בקבוצת הוואטסאפ כיוון שהאתר סירב לקבל את קבצי ה-client שלנו, עברנו לניתוח הקובץ codes.exe.
ראינו כי מתבצעת תחילה בדיקה שניתנו 2 ארגומנטים לתוכנית, ובמידה ולא מתבצעת הדפסה שמלמדת אותנו מהם משמעות הארגומנטים: <OLD_CODE> <CODE_KEY>. מההודעה שקיבלנו בשלב הקודם ניחשנו כבר כי נצטרך להזין CODE_KEY = ROBBER_CAPTURED.

לאחר מכן מתבצעות 2 פונק'. לראשונה מביניהן מועבר ה-OLD_CODE שהועבר לתכנית, ומתבצעת בדיקה על-פי query שמוגדר שם האם הקוד הנ"ל קיים בדאטה-בייס, ואז מתבצעת בדיקה של ערך החזרה של ה-query על-ידי strcmp (במידה והקוד לא קיים, מתבצעת יציאה מהתוכנית).
לפונק' השנייה מועבר ה-CODE_KEY. מתבצעת השוואה על-ידי strcmp, ובמידה וה-CODE_KEY שווה לתוצאה שחזרה מה-query בפונק' הקודמת אנו נבצע יציאה מהתוכנית. במידה ועברנו את ההשוואה, מתבצע query המחלץ מהדאטה-בייס NEW_CODE חדש שלא היה בשימוש בעבר, ומתאים ל-CODE_KEY שהוזן.

לאחר הניתוח הבנו כי הבעיה שלנו היא לעבור את שתי הבדיקות של strcmp (בעיקר לאור העובדה שלא ידוע לנו אף OLD_CODE עבור הבדיקה הראשונה), ולכן נרצה לבצע hook ל-strcmp.
דרך נוחה לעשות זאת היא IAT hook, כיוון שאין לנו צורך בפעולה הרגילה של הפונק', ואנו פשוט רוצים להחזיר באופן דטרמניסטי 1 בקריאה הראשונה ו-0 בקריאה השנייה, ולכן דריסה של הפונק' ב-IAT יסייע לנו לבצע זאת.
נעזרנו בטמפלייטים מהאתר עבור ה-injector, כאשר דאגנו לשנות את שמות הקבצים המתאימים ודאגנו להעביר ל-codes.exe את הארגומנטים ROBBER_CAPTURED whatever, וכן עבור ה-DLL המתאים לביצוע IAT hook.
לאחר חיפוש ברשת ראינו כי הפונק' strcmp נמצאת ב-DLL בשם msvcrt.dll, ולכן ביצענו חיפוש של הפונק' בו (בתהליך הנוכחי – לכן העברנו ארגומנט NULL עבור קבלת ה-handle של התהליך), ומצאנו את ה-offset של הכניסה המתאימה בטבלה בעזרת IDA כפי שלמדנו בסדנה.

לאחר העלאת הקבצים לאתר קיבלנו את הקוד: W6V60LQR1E.

בשלב זה ניגשנו לקובץ dice.exe, כיוון שבהודעה שהתקבלה נאמר שבשימוש בקוד שקיבלנו, אמורה להתקבל התוצאה 7 בהטלת הקוביות (ובהכנסת הקוד לאתר סתם ככה, לא הצלחנו לשחרר את השודד).
ראינו כי בתחילתו מתבצעת הגרלה של ערך אקראי שקובע את ערך זריקת הקוביות, לאחר מכן יש

קריאה לפונק' בה מתבצעת בדיקה של הקוד שהוכנס (אם הוכנס), ובמידה והוא משויך ל-event של NO_ROBBER מתבצע וידוא שהערך שהתקבל אינו 7 (ואם הוא כן, אז הוא מוחלף בערך אחר), ולבסוף מתבצעת חלוקה של תוצאת הקוביות לשתי קוביות נפרדות (לדוגמה, אם התקבל הערך 8 לתוצאת הקוביות, הוא יחולק בין הקוביות באופן רנדומלי ל-6 ו-2, או 4 ו-4 וכו').
תחילה ניסינו לכתוב injector שרק מריץ את התוכנית, בלי לבצע עדיין hook כלשהו, והעלאנו את ה-zip לאתר, וכך גילינו כי עלינו להעלות קובץ exe ולא zip. המסקנה – יש לבצע hook פיזי כדי לשנות את קובץ ההרצה עצמו!

המטרה שלנו הייתה ברגע זה להוסיף בדיקה של הקוד שהוכנס (אם הוכנס), ברגע הכניסה לפונק' בה מתבצעת הקריאה לפונק' בה ערך הקוביות מחולק ל-2 קוביות, ובמידה והוא שווה לזה שקיבלנו (W6V60LQR1E), אז נשנה את ערך הקוביות (שהועבר כארגומנט לפונק', ולכן נמצא על המחסנית) ל-7.

על-מנת לבצע זאת, כתבנו קוד אסמבלי שבמידה והוכנס ארגומנט לתוכנית – מכניס את המחרוזת "W6V60LQR1E" למחסנית, ואז את המצביע למחרוזת הזאת (esp) מכניס למחסנית כארגומנט.
בנוסף, מכניס את המצביע לארגומנט של התוכנית שהועבר, ומבצע השוואת מחרוזות בעזרת strcmp (הפונק' כבר מיובאת בקובץ dice.exe, אז יכולנו להשתמש בה). במידה ובהשוואה התקבל כי המחרוזות שוות, שינינו את הארגומנט של ערך הקוביות שעל המחסנית ל-7, ובכל מקרה ביצענו שחזור של הפקודה אותה נדרוס בפונק' (כדי לקפוץ ל-hook שלנו) וקפיצה חזרה לפקודה שאחריה.

כדי להוסיף את קטע הקוד שכתבנו לקובץ dice.exe הוספנו section חדש בעל הרשאות הרצה, שבו ייכתב קטע הקוד.
כדי לבצע זאת נעזרנו ב-CFF Explorer: בתפריט השמאלי בחרנו ב-Section Headers, ועל-ידי לחיצה על הכפתור הימני באיזור ה-sections בחרנו ב-(Add Section (Empty Space. עבור גודל ה-section בחרנו באקראי ב-1000 (על-אף שזה יותר ממה שהיינו צריכים), ודאגנו לשנות את ההרשאות של ה-section (Characteristics) להיות כמו אלו של ה-text section.

לאחר הוספת ה-section יכולנו לערוך את הבתים שלו (ב-Hex) בעזרת CFF Explorer או בעזרת IDA בתצוגת Hex. לכן, נעזרנו באתר שפורסם בתרגול השני להמרת פקודות אסמבלי לקידוד שלהן, והעתקנו את הקידוד ל-section החדש.

לאחר מכן פתחנו את הקובץ ב-IDA, והלכנו לאיזור של ה-section שהוספנו. ה-IDA כשל בלפענח את האיזור הנ"ל כקוד, ולכן לחצנו על האיזור הנ"ל ואז על המקש 'C', וכך גרמנו לו לפענח את האיזור כקוד, ואכן ראינו את כל פקודות האסמבלי שרשמנו.

בשלב זה, נותר לנו לתקן את כל הקפיצות היחסיות בקטע הקוד שלנו. לשם כך ביצענו חישוב של המרחק היחסי (הסתכלנו מהן הכתובות מהן אנו קופצים, ומהן כתובות היעד של הקפיצה, וביצענו חיסור), ותיקנו בפקודות הקפיצה את ה-offset לפי מה שחישבנו (שינינו את ה-offset שהיה כתוב בעזרת patch bytes).
שמרנו את כל השינויים וביצענו הרצה מקומית תוך כדי דיבוג כדי לוודא שקטע הקוד שלנו אכן מתבצע כפי שציפינו.

לאחר מכן, העלנו את הקובץ לאתר, ניגשנו ללוח המשחק והכנסנו את הקוד שקיבלנו, הטלנו את הקוביות, והופיעה אנימציה של השודד בריצתו לחופשי.