הנדסה לאחור 236496 – תרגיל בית 3

מגישים: אלון פליסקוב 315468116 שליו ריסין 211578794

חלק יבש

1. בתחילת הפונקציה נזריק פקודת jmp. יעד ה-jmp הוא קוד שלנו. בקוד שלנו יתבצעו הדברים הבאים:
   א. נחלק את הזמן בזיכרון הגלובלי ב-2.
   ב. שחזור הקוד המקורי ב-poll (החזרת הפקודות שהיו במקום ה-jmp שהזרקנו).
   ג. קריאה ל-poll, על מנת שתרוץ כרגיל ותגרום לכל ה-side effects הצפויים, כולל כתיבה לזכרון הגלובאלי.
   ד. שחזור הhook.
   ה. חזרה מהפונקציה.

   הקטנו את הזמן בין בקשות.

2. מזריקים לסוף הפונקציה פקודת jmp לקוד שלנו.
   בקוד שלנו:
   א. נשמור את תוצאת הריצה של mine. נסמנה ב-y.
   ב. נשחזר את הקוד המקורי.
   ג. הארגומנט x זמין על המחסנית ולא השתנה לפי ההנחה בשאלה שהפונקציה mine משפיעה רק על משתנים לוקליים ורגיסטרים. לכן, ניקח את x ונריץ את mine מהקוד שלנו, כל פעם מנקודת כניסה אחרת.
   ד. אם אחת הריצות של mine הניבה תוצאה ששווה ל-y, אז נקרא ל-mine מנקודת הכניסה המתאימה לריצה זו, עם הארגומנט $\bar{x}$.
   ה. נשחזר את הhook.
   ו. נבצע ret כשה-return value מהפונקציה הוא התוצאה מ- ג'.
3. בתחילת הפונקציה נזריק פקודת jmp. יעד ה-jmp הוא קוד שלנו. בקוד שלנו יתבצעו הדברים הבאים:
   א. ניתן להסיק על פי מחרוזת הפורמט מהו מספר הארגומנטים. נשתמש בsprintf על מנת להדפיס את המחרוזת לאחר הformatting לתוך buffer.
   ב. נצפין את המחרוזת שבתוך הbuffer.
   ג. נשחזר את הקוד המקורי של sendf.
   ד. נקרא לsendf עם הhandle והמחרוזת המוצפנת בתור הformat string (ואין ארגומנטים נוספים).
   ה. נשחזר את הhook.
   ו. חזרה מהפונקציה.
4. בתחילת הפונקציה connect נזריק פקודת jmp. יעד ה-jmp הוא קוד שלנו. בקוד שלנו יתבצעו הדברים הבאים:
   א. נשחזר את הקוד המקורי של connect.
   ב. נריץ בשני threads נפרדים את connect ואת poll.
   ג. נחכה ששתיהן יסיימו

ד. נשחזר את הhook בconnect
ה. נוסיף לערך כתובת החזרה מconnect את גודל הפקודה "call connect".
ו. נחזור מהפונקציה

5. בתחילת הפונקציה נזריק פקודת jmp. יעד ה-jmp הוא קוד שלנו. בקוד שלנו יתבצעו הדברים הבאים:
   א. נשחזר את הקוד המקורי
   ב. נריץ את הפונקציה
   ג. נדפיס את ערך החזרה משלב ב'.
   ד. נשחזר את הhook
   ה. נחזיר את ערך החזרה משלב ב'.
6. מקרה א':

הרעיון הוא לשמור מחסנית של כתובות חזרה. כל פעם לדרוס את הreturn address עם כתובת של קטע קוד שקופץ לקטע קוד שמכפיל את rax ב-2 ואז קופץ לכתובת החזרה המקורית.

א. נדחוף את כתובת החזרה של הפונקציה למחסנית כתובות החזרה.
ב. נחליף את כתובת החזרה לכתובת של קטע קוד (שיהיה כתוב גם בhook) שמכפיל את eax ב-2, ואז חוזר לכתובת החזרה שבראש מחסנית כתובות החזרה ועושה לה pop.
ג. הhook יבצע את קטע הקוד שהוא דרס (נדאג לכך שגודל קטע הjmp יהיה מיושר ולא ידרוס פקודה באמצע) ולאחר מכן יקפוץ חזרה לפונקציה solve לקטע שאחרי הדריסה.

מקרה ב':

א. נשחזר את הקוד המקורי
ב. נריץ את הפונקציה
ג. נשחזר את הhook
ד. נכפיל ב-2 את ערך החזרה משלב ב' ונחזיר אותו

השבת השודד – חלק ראשון – ניתוח דינמי

משימתינו הייתה לפצח את keygen.exe כך שנוכל ליצור תכנית הופכית אליו.
ראשית, פתחנו את keygen ב-IDA, תוך כדי נתינת שמות סטנדרטיים למשתנים ולפונקציות כגון main.
שמנו לב, שמתבצעת פקודת call eax כאשר ב-eax שמורה כתובת על המחסנית.
לכן הסקנו שבמהלך התכנית, התכנית כותבת קוד למחסנית, ואז קוראת לו.
בשלב הבא, הרצנו את התכנית בדיבאגר והסקנו כי עד לרגע של call eax, התכנית בסה"כ כותבת למחסנית ולא מבצעת שום פעולה שקשורה ישירות ללוגיקה שאנו מחפשים.
לפיכך, בחרנו להתמקד בקוד שנכתב למחסנית. בניתוח דינמי ב-IDA פרשנו את הקוד שנכתב למחסנית והתחלנו לפענחו. ראינו כי הקוד הוא פונקציה המקבלת שלושה פרמטרים: argv[1], כתובת של printf, וכתובת של strlen. הפונקציה כותבת בתים למחסנית באזור חדש שהיא מקצה, שגודלו כגודל argv[1] עד כדי alignments.
כמו כן, בראשית הפונקציה, מוגדר מערך שנקרא לו *array* של 0x5F (במספר) תווי ASCII יחודיים.
למעשה, אלה הם כל תווי ה-ASCII הניתנים להדפסה, החל ברווח ' ' וכלה בטילדה '~', אך אין ביניהם סדר.
לכל תו x בקלט (argv[1]), הפונקציה מדפיסה את [x-0x20]*array*, כאשר 0x20 הוא התו רווח – התו הקטן ביותר שניתן להדפסה. כמו כן, x מיוצג כאן כערך מספרי על ידי התאמה בין התו x למספר ה-ascii שלו.
לאור המידע שאספנו, כתבנו תכנית פייתון קצרה שעושה את הפעולה ההפוכה –
בהינתן תו y בסיסמה, אנחנו מסתכלים על האינדקס שלו ב-*array*, נקרא לו i.
מדפיסים את התו הניתן לכתיבה ה-i לפי סדר (הראשון הוא רווח). שורת קוד המתארת זאת:

```
ascii_order[keygen_order.index(password[i])]
```

הכנסנו את הסיסמה הראשונית שקיבלנו לאתר כקלט לתכנית.

חלק שני – הוקינג

בחלק השני, התבקשנו להשתמש בהוקינג על אחד הקבצים ב-tools, שהוא client.exe. מטרתינו היא לגרום לכך שכאשר נריץ את client.exe עם הקלט "DMSG", אז תיפלט הודעה מפוענחת, במקום מוצפנת.

כדי להבין איך לעשות זאת, ניגשנו לקבצים secure_pipe.exe ול-client.exe. על ידי ניתוח ב-IDA, הבנו באיזה שלב client.exe מקבל את ההודעה המוצפנת, ובעזרת איזו פונקציה (recv) הוא מקבל את ההודעה המוצפנת. בנוסף, על ידי ניתוח של secure_pipe.exe, מצאנו את הפונקציה האחראית על ההצפנה בקובץ, ופענחנו את האופן בו ההצפנה עובדת. כעת נסביר על כך:

בהינתן תו בקלט (הלא מוצפן):
התו הוא 8 ביטים. הפונקציה מפצלת את התו לשני חלקים שווים – ארבעה הביטים העליונים, וארבעה הביטים התחתונים של התו. נקרא להם החלק העליון והחלק התחתון.
כל אחד מהחלקים הללו, בעל ערך מספרי בין 0 ל-15. הפונקציה מתאימה את ערכי החלקים הללו לתווים של קלפי משחק – המספר 1 מקבל אס "A", המספר 10 מקבל נסיך "J", המספר 11 מקבל מלכה "Q", והמספר 12 מקבל מלך "K". המספרים בין 2 ל-9 נשארים איך שהם. המספר 0 והמספרים בין 13 ל-15 הם מקרים מיוחדים.
אם המספר הוא 0, אז הפלט הוא "x-x" כאשר x ספרה רנדומלית גדולה מ-5 (מוגרל על ידי פונקציה

rand, כאשר לפני כן התבצעה קריאה ל-srand עם ערך אקראי time).
אם המספר בין 13 ל-15, אז הפלט הוא "y+z" כאשר y הוא ספרה רנדומלית, ו-z הוא המספר (בין -13 15) שחיסרו ממנו את הספרה הרנדומלית.
למשל, נצפין את התו "O". ערך ה-ASCII של "O" הוא 0x41. בבינארי נכתוב 01001111_b. פירוק הייצוג הבינארי לחלק עליון ותחתון – 1111_b , 0100_b.
החלק העליון, 0100, בעל ערך דצימלי 4, לכן, לפי מה שתיארנו מעלה, הפונקציה תפלוט עבור חלק זה את התו "4".
החלק התחתון, 1111, בעל ערך דצימלי 15. לכן, הפונקציה תגריל ספרה גדולה מ-5, למשל 6, ואז נקבל "6+9", כי מתקיים 9 = 6 – 15.
בסה"כ עבור הקלט "O", הפלט המוצפן הוא "46+9".
בדקנו את נכונות ההבנה שלנו על ידי כתיבת קוד קצר שמבצע את הפענוח לפי ההצפנה שפירטנו.
לקחנו הודעה מוצפנת בעזרת פקודת DMSG. ההודעה המוצפנת:

4927-77468+767+86Q25-574686527-77267+86262657224-4636A77-77469766527+728-8486529-9697324-468656K6424-46967+729-9443A28+6 496627-76668+77224-47366+968+56528-872656A7368+766+828-8776528-8736867+8756K6424-46672656524-474686524-46775792K 67+865+96524-464+975737427-7666967+76424-46A27-76368+7646528-86A737367+863696A74656424-47769746824-474686524-45247+84242455258+7434A54-4545552454425-565766566+87426+8 57686566+822-27468697324-46366+9646527-7697328-8757365642K24-47268+76K6K6967+76728-874686529-96469636524-4736868+7756K6424-4726573756K7429-97769746829-9637562657324-474686A7425-5737565+827-77467+822-27365766568+627+7

ההודעה המפוענחת:

I took the robber captive. He is held in D1.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

ניגשנו לכתיבת קוד ה-hook.

בחרנו לבצע hook על הפונקציה recv ששייכת ל-socket api. בחרנו בה כי היא הפונקציה שנקראת ב-client אשר מקבלת את ההודעה המוצפנת מהשרת. יש לציין, כי הפונקציה נקראת פעמיים, והקריאה המעניינת אותנו היא הקריאה בה מועבר דגל 0.

בחרנו בשיטת ה-Hook הרגילה, זאת כי לא ידענו אם יש תמיכה ב-hot-patching בפונקציה recv. גם אם יש, עדיין אפשר להשתמש ב-hook רגיל. היה אפשר להשתמש גם ב-IAT Hooking כיוון ש-recv נטענת מתוך DLL.
אנו מזריקים את ה-hook באופן דינאמי בעזרת DLL injection.

חילקנו את ה-hook לכמה שלבים.
השתמשנו בטמפלייטים מהאתר injector_template.cpp, ו-hook_override_jump_template.cpp.
ב-injector_template.cpp, שינינו את שורת ההרצה כך שתכלול את הפקודה שאנחנו רוצים לבצע – DMSG.
ב-hook_override_jump_template.cpp, הוספנו פונקציות שאחראיות על פענוח ההודעה המוצפנת.
בנוסף, כתבנו את הקוד של funcHook, שהיא ה-hook code.
ריצת הפונקציה funcHook:

א. שחזור הקוד המקורי על ידי ()remove_hook.
ב. הכנת ארגומנטים לקריאה חוזרת של הפונקציה recv.
ג. קריאה חוזרת ל-recv.
ד. פענוח ההודעה המוצפנת שהתקבלה (ושמירת ההודעה המפוענחת על ה-buffer ש-recv כותבת אליו).
ה. שחזור ה-hook, והחזרת ה-return value שחזר מ-recv.

הכנו zip שמכיל את ה-injector.exe, את ה-client.exe ואת ה-DLL והעלינו לשרת.

<u>השבת השודד – חלק שלישי</u>

הוטל עלינו לשחרר את השודד. לפי ההודעה המפוענחת של Goblin, השודד מוחזק ב-D1, ועלינו למצוא את הקוד שקשור ל-event שנקרא ROBBER_CAPTURED, ובנוסף, שהטלת הקוביות תצא 7 כשנשתמש בקוד זה.

התחלנו בניתוח סטטי לא מעמיק של כל קבצי ה-exe ב-tools.
חיפשנו קצה חוט שקשור ל-ROBBER_CAPTURED.
מצאנו כי הקובץ codes.exe מקבל event וקוד, ומריץ שתי שאילתות פייתון.
הראשונה מקבלת את הקוד ופולטת את ה-event הקשור לקוד זה, ואם אין קוד כזה במערכת, אז השאילתה מחזירה
"NO SUCH CODE".
אחרי השאילתה הראשונה, מתבצעת השוואה בין ה-event שחזר לבין "NO SUCH CODE". במידה וה-events זהים, התכנית מסיימת את ריצתה.
לפני השאילתה השנייה, מתבצעת השוואה בין ה-event שניתן כארגומנט לתכנית, ובין ה-event שחזר מהשאילתה הראשונה, אם קיים כזה. אם שני ה-events זהים, אז מודפס הקוד הקשור ל-event הזה, במידה והוא לא שומש כבר.

כלומר, על מנת להשיג את הקוד הקשור ל-ROBBER_CAPTURED, עלינו לעבור שתי השוואות ולהגיע לשאילתה השנייה כאשר הארגומנט הוא ROBBER_CAPTURED.

לטובת מטרה זו, ביצענו hooking על הקובץ codes.exe.
למטרות דיבוג ו-logging, בחרנו את הפונקציה strcmp להיות מטרת ה-hook. פונקציה זו היא פונקציית ספריה, ולכן בחרנו בשיטת ה-IAT hook. לקחנו תבנית מתאימה ל-IAT hook מהאתר וקובץ injector.
רצינו בהוק הראשוני לגלות על אילו מחרוזות מתבצעת השוואה, בעת שתי הקריאות ל-strcmp (קריאה אחת אחרי השאילתה הראשונה, וקריאה נוספת לפני השאילתה השנייה).
העלינו את הגרסה המעודכנת של codes.exe לאתר, עם הארגומנטים:

“codes.exe ROBBER_CAPTURED DLQ4W1AMT2”
כאשר DLQ4W1AMT2 הוא הקוד המתאים להחזרת הכבשים.
ראינו על ידי ההדפסות שהצבנו ב-hook, שה-event המתקבל על ידי הקוד DLQ4W1AMT2 הוא SHEEP_HIDING. ובכל מקרה, לא קיבלנו את הפלט “NO SUCH CODE” ולכן עברנו את ההשוואה הראשונה.
בהשוואה השנייה, רצינו שהיא תעבור בהצלחה (כלומר שהמחרוזות יהיו זהות), ולפי הארגומנטים שבחרנו, המחרוזות שמועברות ל-strcmp הן SHEEP_HIDING, ROBBER_CAPTURED.
לכן לא ייתכן כי strcmp תחזיר 0.
רצינו שבמקרה הספציפי הזה – שבו הארגומנט הראשון לתכנית הוא ROBBER_CAPTURED, בקריאה השנייה ל-strcmp, נדלג על ההשוואה עצמה, ונקפוץ לשאילתה השנייה. ללא דילוג זה, התכנית תסיים את ריצתה בלי להדפיס דבר.
עשינו זאת על ידי שינוי כתובת החזרה של strcmp **רק** בקריאה השנייה. כתובת החזרה החדשה היא לתחילת ביצוע השאילתה השנייה.
כעת, עברנו את שתי ההשוואות, והגענו לשאילתה השנייה כאשר הארגומנט שמועבר הוא ROBBER_CAPTURED. התכנית הדפיסה את הקוד הבא:

WV434KPRXO

בזאת השלמנו חלק מהמשימה – יש בידינו את הקוד הקשור ל-ROBBER_CAPTURED, ואנו יודעים כי המשבצת הרלוונטית היא D1.
נותר לטפל בהטלת הקוביות. ע"י ניחוש מושכל ניגשנו לקובץ dice.exe.

בקובץ dice.exe, לאחר ניתוח סטטי זריז, מצאנו כי הסכום של שתי הקוביות מוגרל על ידי srand(time), אחרי זה rand, ובסוף עוד מניפולציות כדי לגרום לתוצאה להיות מספר בין 2 ל-12, תוך כדי שמירה על התפלגות המתאימה לשתי קוביות.

עם תוצאה זו, קוראים לפונקציה נוספת, שנקרא לה func. בפונקציה, מתבצעת הפרדה למקרים לפי האם נשלח ארגומנט נוסף לתכנית, או לא.
חזרנו לאתר, והזנו את הקוד WV434KPRXO-D1 כמו שהגובלין הנחה אותנו לעשות בהודעה שלו.
ה-board השיב לנו ב-GIF של ג'ים הלפרט מ"המשרד", שנראה מאוכזב, אך לא מופתע.
הבנו מזה שאנחנו בכיוון, ושככל הנראה, כשמכניסים קוד תקף לאתר, זה מפעיל את dice.exe עם ארגומנט נוסף שהוא הקוד.
עם הידע החדש, חזרנו ל-dice.exe, וחקרנו מה קורה כשמריצים את הפונקציה func במקרה שבו יש ארגומנט נוסף לתכנית.
ראינו כי במקרה זה, מתבצעת שאילתה בפייתון שמכילה את הקוד, בדומה לקובץ codes.exe.
השאילתה מוציאה את ה-event הקשור לקוד.
לאחר השאילתה, מתבצעת השוואה בין ה-event שהתקבל לבין ה-event שנקרא “NO_ROBBER”.
במידה וה-event הוא אכן NO_ROBBER, מתבצעת לולאה שמגרילה מספר חדש לסכום הקוביות כל עוד סכום הקוביות הוא 7.
הבנו על פי הודעת הגובלין, שעלינו לשנות את ריצת dice.exe כך שבמקרה שה-event שהתקבל הוא ROBBER_CAPTURED, אז סכום הקוביות יוצא 7.
לשם כך, שינינו את הלוגיקה של func באופן הבא:
במידה וה-event הוא **ROBBER_CAPTURED**, מתבצעת לולאה שמגרילה מספר חדש לסכום הקוביות כל עוד סכום הקוביות הוא **לא** 7.

את החלק של ROBBER_CAPTURED עשינו על ידי כך שהוספנו את המחרוזת ROBBER_CAPTURED ל-rdata section. בנוסף, שינינו את הפונקציה כך שה-event המתקבל מושווה ל-ROBBER_CAPTURED במקום ל-NO_ROBBER.
שינינו את לוגיקת הפונקציה כך שאם ההשוואה לעיל יוצאת חיובית (מחרוזות שוות), אז עוברים ללולאה שמוודאת שסכום הקוביות הוא 7. עשינו זאת על ידי שינוי ה-jz בתחילת הלולאה ל-jnz (קודם לכן, היינו נכנסים ללולאה במקרה של jz), כך שניכנס ללולאה כל עוד סכום הקוביות אינו 7.
עדכנו את dice.exe על ידי byte patching.

העלינו את הקובץ החדש לאתר. הכנסנו את הקוד WV434KPRXO-D1 וקיבלנו את האנימציה של השודד, באופן קונסיסטנטי.

הסוף.

...or is it?