הנדסה לאחור – ת"ב 1

חלק 2

הבעיות בקובץ partb.nopt.s הן:

1. חסרים פרמטרים לקריאה לפונקציה scanf לכן העברנו כארגומנט ראשון את המחרוזת "%d", שלפני כן הייתה שמורה כ-"%c" ב-L2, וכפרמטר שני את המצביע למשתנה שהוקצה לטובת הניחוש.
2. במקרה של ניחוש לא נכון גם ב-L2 וגם ב-L6 יש קפיצה ל-L3, ונרצה לאפשר ניחוש מחדש לכן הוספנו קפיצה חוזרת ל-L6 לצורך תמיכה בקריאה רקורסיבית לניחוש חוזר.
3. ב-L2 במקרה שהניחוש נכון מבצעים קפיצה ל-L4. אם הניחוש נכון נרצה לצאת מהפונקציה, לכן הוספנו קפיצה ל-L8 שמבצעת יציאה מהפונקציה.

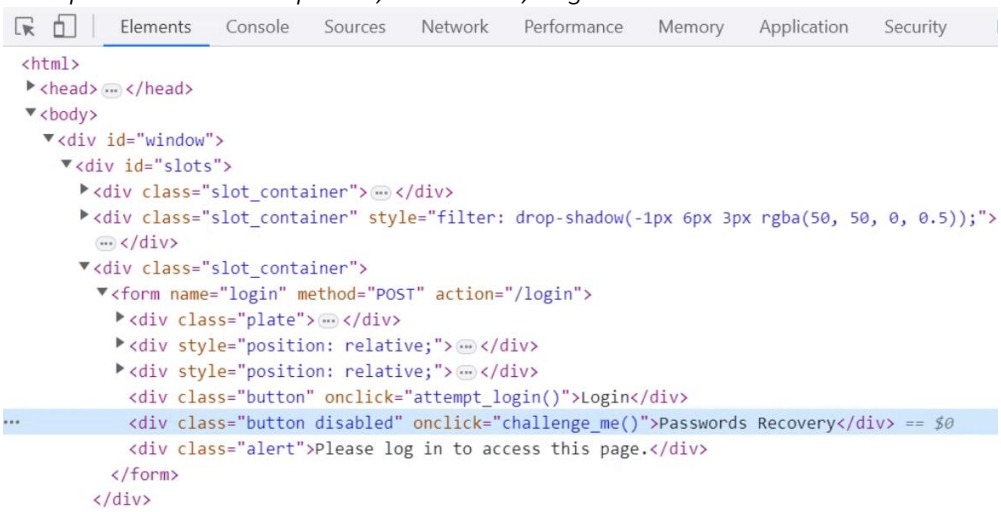
הבעיות בקובץ partb.pt.s הן:

1. לא קוראים לפונקציה srand_ מה שאומר שאנחנו משתמשים ב-seed דיפולטי או ב-seed קודם, לכן לא מובטח שבכל הרצה של הפונקציה נקבל ערך שונה בפונקציה rand_.
2. לפני הקריאה ל-printf_ מעבירים מצביע לתווית LC0 שהודפסה לפני כן בקריאה ל-puts_, ועלינו להעביר מצביע לתווית LC1 על מנת להדפיס למסך את המחרוזת המבוקשת.
3. L2 אחראית לקבלת ניחוש מהמשתמש וביצוע השוואה. לאחר הבדיקה של jge, במקרה בו המספר שהמשתמש ניחש קטן מידי, נרצה להדפיס למסך הודעה שהמספר אכן קטן, או אם הניחוש היה נכון אז נדפיס הודעה מתאימה, ולכן לא נכון לקפוץ ישר לניחוש נוסף אלא לקפוץ ל-L3 שבודקת זאת.

חלק 3

1. כיווץ קובץ הרצה

לפי המלצת הצוות הלכנו אל ה-Developer Tools, עברנו על ה-HTML של האתר שנמצא בלשונית Elements וחפשנו כל מיני מילות מפתח כמו "login", "user" ועוד, כדי לקבל כל מיני רמזים איך לפרוץ לאתר.



מצאנו כפתור שלא גלוי, "button disable", ותיקנו אותו (מחקנו את "disable") כך שהתגלה כפתור שהוביל אותנו לשלב הבא.
בשלב הבא, כתבנו תוכנית מתאימה (לפי ההוראות) על-מנת לצלוח את האתגר, אבל לצערנו כשהעלנו את הקובץ גילינו שהוא גדול מידי והבנו שצריך לבצע דחיסה.
ע"מ לבצע את הדחיסה הלכנו ל-Compiler Explorer והמרנו את קוד ה-C שכתבנו לקוד assembly, לאחר מכן בצענו את שלבי הדחיסה כפי שהוצג לנו בתרגול והשתמשנו בקישור דינאמי בזמן ריצה עבור הפונקציות scanf ו-printf.

2. ניתוח קובץ ASSEMBLY

ראינו שעלינו לנתח את הקובץ crackme.S הכתוב ב-assembly לכן רצינו להפוך אותו לקובץ הרצה ולהיעזר ב-IDA לצורך הניתוח (לשם נוחות). כאשר ניסינו לקמפל אותו ראינו שחסרה לו הספרייה הדינאמית sqlite3.dll, לכן הורדנו אותה מהאינטרנט ויצרנו את קובץ ההרצה.
התחלנו לנתח את הקובץ וראינו בתחילתו שיש דריסה של הסינגל SIGFPE של חלוקה ב-0 עם פונקציה בשם _handler. אחר-כך, לפני הקריאה ל-level_1, מכינים את המחסנית כך שהפרמטרים הם:
__dummy_, argv[1], argv[0], argc (כשהשניים הראשונים הינם הפרמטרים לפונקציה שנקראת, והשאר נשמרים במחסנית כמשתנים מקומיים – על-פי השימוש ב-esp או ebp לאחסון שלהם).
נשים לב שפרמטרים ב-shell מועברים כמחרוזות, וניתן לראות שהמירו, לפני העברתו במחסנית, את argv[1] ל-integer בעזרת הפונקציה atoi.
ב-Level 1 אנחנו רואים שהוא בודק שהפרמטר הראשון שהעברנו לפונקציה גדול מ-1. הפרמטר הראשון שמועבר לפונקציה הוא argc, שהוא מספר הערכים במערך argv.
כידוע argv[0] הוא שם התוכנית, ומהדרישה שרוצים שגודל המערך argv יהיה גדול מ-1 אנחנו מסיקים שעל מנת לעבור את level_1 עלינו לוודא שמועבר לפונקציה לפחות פרמטר 1.
לפני הקריאה ל-level_2 שמים את argv[1] כפרמטר הראשון שיועבר לפונקציה.
ב- Level 2 ראינו התייחסות לשני חלקים במחסנית, החלק הראשון מוגדר מראש (מתבצעת השמה של בתים מסוימים למחסנית). לאחר מכן, ממלאים את החלק השני של המערך בערכים רנדומליים בעזרת הפרמטר ששלחנו (שמהווה את ה- seed לפונקציה הרנדומלית).
ראינו שאחרי שלב זה נקראת הפונקציה scanf לקליטת 2 מספרים מהמשתמש. שני מספרים אלה משמשים כדי להגדיר טווח על המחסנית אותו נוכל להדפיס (ובהמשך לשנות), החל מכתובת בסיס מוגדרת (שהיא תחילת החלק השני של המחסנית – זה שהתמלא בערכים רנדומליים).
הפרמטר הראשון מגדיר את ה-offset מכתובת הבסיס שעלינו להתחיל ממנו (עד כדי יישור ל-4 בתים), והפרמטר השני מגדיר כמה בתים עלינו להדפיס (מגדיר את הגבול העליון של המחסנית עליה נסתכל).
לאחר מכן, מתבצע מעבר על האיברים אותם הדפסנו ומבצעים bitwise XOR עם ערכים שנסרקים ע"י המשתמש.
לבסוף מתבצעת השוואה בין שני חלקי המחסנית ובמידה והם שווים נעבור את שלב 2.
לפיכך דאגנו שבביצוע ה-XOR על החלק השני של המערך נשנה אותו להיות זהה לחלק הראשון (על-ידי בחירת ערכים מתאימים איתם יתבצע XOR) ובכך נעבור את level_2.
Level 3 חיפשנו בתוכנית מיקום בו מודפסת מחרוזת הכוללת את "Level 3" וראינו שזה מתבצע בפונקציה __dummy_, אליה אין קריאה מפורשת בקוד, ונזכרנו בשיטת ROP (מאת"מ) שדורסת את כתובת החזרה ומשנה אותה.
מקודם ראינו שהכתובת של __dummy_ נמצאת על המחסנית ולכן אם נגדיל את טווח ההדפסה בשלב 2 נוכל לראות אותה. כעת כשידענו היכן שמורה כתובת החזרה במחסנית ומהי הכתובת של __dummy_, יכולנו לשנות את כתובת החזרה כמו שביצענו בשלב 2 (ביצוע XOR עם ערך מתאים) ולקפוץ ל- __dummy_.

Level 4 כפי שראינו בתחילת התרגיל מתבצעת דריסה של ה-SIGHANDLER של SIGFPE ובשגרת הטיפול החדשה מודפס שעברנו את שלב 4 (ולכן נרצה לחלק ב-0 כדי להיכנס ל-handler הזה).
בפונקציה _dummy__ מתבצעת חלוקה עם איזשהו ערך שנמצא ב- divider_, וראינו ההשמה היחידה בקוד ל- divider_ מתבצעת בהמשך פונקציית ה-main ומתחשבת ב-seed שאנחנו מעבירים אליה (בארגומנט של התוכנית ב-shell).
לכן, הבנו שעלינו לקפוץ לפונקציה _dummy__ רק לאחר החזרה מה-main ולא בחזרה מהפונקציה של level_2.
לפיכך, הסתכלנו שוב על המחסנית ומצאנו היכן שמורה כתובת החזרה מה-main (לשם כך היינו צריכים להגדיל את הטווח שהגדרנו בשלב 2), והחלפנו אותה לכתובת של _dummy__.
כעת, ניסינו מספר אפשרויות שונות ל-seed עד שקיבלנו הדפסה שצלחנו את שלב 4 (כלומר, seed של-ידי שימוש בו הוכנס למשתנה הגלובלי divider_ המספר 0, לאחר החישוב שקורה בפונק' ה-main).
מציאת הסיסמאות ראינו שבפונקציה db_access_ אליה מגיעים לאחר מכן, מתבצעת הכנה של פרמטרים לשליחה לפונקציה sqlite3_exec, ומבדיקה באינטרנט ראינו שהפרמטר השני מכיל את השאילתה שתתבצע על הטבלה, ושאילתה זו תלויה בין היתר בארגומנט השני שמועבר לקובץ הריצה ב-shell (ב-main ארגומנט זה מוכנס למשתנה הגלובלי arg_).
השאילתה שנכניס תהיה תלויה בתבנית שמוטבעת בקוד ומבדיקה באינטרנט מצאנו איך להשלים אותה (על-ידי קריאה אודות sql injection).
התבנית היא :

select username, password from users where username='<_arg>'

ועבור:

_arg = ' OR 'a'='a

קבלנו את כל היוזרם והסיסמאות (בעצם דאגנו שעבור כל כניסה בטבלה יתקבל ב-predicate שהוגדר ב-where TRUE ע"י 'a'='a').