אישור תוכניות פנט"ה- נבות בן לולו

פירוק המשימה לחלקים:

רציונל:

המשימה שלי הינה מימוש שלושה מנגנוני הגנה שאינם תלויים אחד בשני. על כן מצאתי לנכון שהפיתוח שלהם יהיה באופן טורי (מימוש המנגנונים אחד אחרי השני). לאחר הפיתוח הראשוני אציג את תוצרי למומחה התוכן ואקבל ממנו הערות ותיקונים למנגנון הנ"ל. לאחר מכן, במקביל, אוסיף תיקונים למנגנונים הקיימים ואתחיל לחקור על המנגנון הבא ולממשו.

חלקי המשימה:

1) אפיון ומימוש ראשוניים של stack canary ו dep- אחקור כיצד פועלים מנגנונים אלו, באיזו דרך כדאי לי לממש אותם, באיזה משלבי ההידור ארצה להתערב וכתיבת ניתוח האיומים עבורם.

2) הצגת המימושים לשני המנגנונים שהוצגו בחלק 1 למומחה התוכן, קבלת ביקורת ותיקונים בהתאם.

3) אפיון ומימוש ראשוניים של מנגנון ה ASLR- לחקור איך פועל המנגנון, באיזה שלב בתהליך ההידור ארצה להתערב, מימוש וניתוח איומים.

4) הצגת המימוש הראשוני של ה ASLR והתיקונים לשני המנגנונים שהוזכרו בחלק 1 למומחה התוכן, קבלת ביקורת ותיקונים להמשך.

גאנט עבודה:

<table>
<tr><td>שבועות/ימים</td><td>ראשון</td><td>שני</td><td>שלישי</td><td>רביעי</td><td>חמישי</td></tr>
<tr><td>שבועות 3-5</td><td colspan="5">- אפיון stack canary
- מימוש ואוטומיזציה ראשוניים של ה canary
- אפיון dep
- מימוש ראשוני של dep</td></tr>
<tr><td>שבוע 6</td><td colspan="3">תיקונים dep ו canary</td><td colspan="2">חקירה על ASLR</td></tr>
<tr><td>שבוע 7</td><td colspan="5">רגילה</td></tr>
<tr><td>שבוע 8</td><td colspan="3">אפיון ASLR</td><td colspan="2">מימוש ASLR</td></tr>
<tr><td>שבוע 9</td><td colspan="5">חתך אמצע- הצגת התוצרים ותיקונים ל ASLR</td></tr>
<tr><td>שבוע 10</td><td colspan="5" rowspan="2">אוטומיזציה לכלל המנגנונים ותיקונם לפי הערות מומחה התוכן</td></tr>
<tr><td>שבוע 11</td></tr>
<tr><td>שבוע 12</td><td colspan="5">אישור מקצועי אצל מומחה התוכן</td></tr>
<tr><td>שבוע 13</td><td colspan="5">התכוננות להצגה- הכנת מצגת</td></tr>
</table>

הערה- הכנסתי באפרים גדולים יחסית בין משימה למשימה (לפי הערכתי הראשונית) כך שיש לי מקום לטעויות והתעכבויות לא צפויות.

אפיון Stack Canary:

כדי לממש את ה canary החלטתי להתערב בקוד בשתי רמות:

א) <u>רמת קוד ה c:</u>
הוספתי בתחילת הקוד את הפונקציה initCanary שמאתחלת משתנה CANARY גלובלי בגודל 8 בתים לערך רנדומלי.

ב) <u>רמת קוד האסמבלי:</u>
קראתי לפונקציה initCanary בתחילת ה prologue של פונקציית ה main.
את קוד ה c קימפלתי לקובץ המכיל קוד אסמבלי. את קוד האסמבלי ערכתי כך שכתבתי 2 פונקציות חדשות- canaryStart ו canaryEnd.
לפונקציה הראשונה, קראתי בתחילת כל prologue של פונקציה שהמתכנת כתב. הפונקציה הזו דוחפת למחסנית את ה CANARY הגלובלי, ואת ה 4 בתים הנמוכים של ה CANARY, היא משנה כך שיכילו את תוצאת ה xor בין ה canary לבין כתובת החזרה של הפונקציה.
לפונקציה השנייה, קראתי בסוף כל epilogue של פונקציה שהמתכנת כתב. הפונקציה הזו מוציאה מהמכסנית את ה canary שהוכנס בפונקציה canaryStart, עושה שוב xor לארבעה בתים הנמוכים שלה עם כתובת החזרה של הפונקציה (כך שבמצב תקין בו גם ה canary וגם כתובת החזרה לא נדרסו, ה canary שנדחף למחסנית יחזור להיות זהה ל canary הגלובלי), ובודק האם ה canary הנ"ל שווה לcanary הגלובלי. אם כן, התוכנית ממשיכה כרגיל. אם לא, התוכנית מדפיסה הודעת שגיאה ויוצאת.

<u>ה canary שמימשתי מגן מפני התקיפות הבאות:</u>

- אם תוקף מנסה לבצע bufferoverflow סטנדרטי שמטרתו לדרוס את כתובת החזרה של הפונקציה, אזי הוא ידרוס גם את ה canary בדרך ובפונקציה canaryEnd נדע ונתריע על כך.
- הכנסתי ל canary שלי בית אחד של NULL באמצעה, כך שאם תוקף ירצה להעתיק את ה canary (כדי שיוכל לא לשנות אותה בהמשך) ע"י פונקציות כמו strcpy, הוא לא יצליח לעשות כן, שכן פונקציות כאלו מסיימות כאשר הן נתקלות בבית שהוא NULL.
- אם תוקף משנה את כתובת החזרה מבלי לשנות את ה canary, עדיין נוכל לזהות זאת ולהתריע (תוצאת ה xor בין ה canary החדש לכתובת החזרה לא תתן את ה canary הגלובלי).

<u>דרכים בהן ניתן לעקוף את הגנת ה canary שלי:</u>

- אם לתוקף יש יכולת לשמור את ה canary שנדחף (ע"י שימוש בפונקציות כמו memcpy ולא strcpy), הוא יכול לעשות כן, וכשהוא מממש התקפת buffer overflow הוא יכול לדאוג לכך שה canary שבמחסנית יידרס עם ערך שהוא
$$new\ canary = (old\ canary) \oplus (old\ ret) \oplus (new\ ret)$$
כך הפונקציה canaryEnd לא תתריע על השינוי.

אפיון מנגנון ה DEP:

רקע:

את מנגנון ה dep מימשתי ברמת קוד ה c בלבד. מטרתו של מנגנון זה היא למנוע הרצת קוד מתוך ה stack.

כשקראתי איך המנגנון עובד, גיליתי שמערכת ההפעלה מסמנת page-ים שונים של זיכרון הממופים לזיכרון הוירטואלי של תהליך עם הרשאות גישה שונות (ביניהם הרשאת הרצת קוד). כשביצעתי strace לתוכנה פשוטה גיליתי שתי syscalls מעניינות בהקשר של מיפוי וסימון הזיכרון הוירטואלי:

ה syscall הראשון הינו mmap, שמבקש למפות מספר page-ים מהזיכרון הפיזי אל כתובת מסויימת בזיכרון הוירטואלי של התהליך. ה syscall השני הוא mprotect, שמקבל את כתובת זיכרון כלשהי במרחה הכתובות של התהליך אליה מופה page מסויים, את הגודל של זיכרון זה והרשאות גישה שונות ומחיל את הרשאות אלה על ה page-ים שניתנו לו כפרמטר.

כדי לראות את כל הזיכורון הממופה אל תהליך מסויים קראתי את הקובץ /proc/{PID}/maps, שמכיל את כל הכתובות במרחב הכתובות של התהליך, אליהן מופו page-ים של זיכרון. ליד כל כתובת כזו ישנו שדה המתאר מה השימוש של כתובת זו. גיליתי שישנה רשומה בקובץ זה שהינה הכתובת של ה stack של התהליך (מסומנת במחרוזת "[stack]").

כשניסיתי לקמפל קוד פשוט עם gcc, פעם אחת עם אפשרות ה dep ופעם אחת בלעדיה, גיליתי שאכן ההרשאות של ה page-ים המתאימים למחסנית משתנות.

המימוש שלי למנגנון:

הוספתי פונקציה בקוד ה c שנקראת protectStack. הפונקציה נקראת בתחילת ה main. פונקציה זו מפרסרת את הקובץ /proc/{PID}/maps המתאים לתהליך הנוכחי שרץ, מוצאת את הרשומה המתאימה ל stack, מחלצת את הכתובת של תחילת המחסנית ואת גודלה וקוראת ל syscall – mprotect, עם פרמטרים אלו והרשות גישה של קריאה וכתיבה בלבד (לא הרשאות הרצה).

מבדיקות שעשיתי, אכן קובץ ה /proc/{PID}/maps מראה שההרשאות השתנו עבור המחסנית.

המנגנון שלי מגן מפני התקיפות הבאות:

- הרצת קוד מתוך המחסנית (למשל shellcode ששמור במחסנית).

דרכים בהן ניתן לעקוף את המנגנון שלי:

- הרצת קוד מתוך אזורים אחרים מה stack, למשל- heap, code segment, data segment.
- לגרום לתוכנית לקפוץ אל mprotect עם הפרמטרים המכילים את כתובת וגודל המחסנית, ועם הרשאות הכוללות הרשאות הרצה (למשל ע"י buffer overflow).