גירסה 1.00 – 1.8.2004

Format String Vulnerabilities

מסמך זה הינו סיכום של קטעים מהמסמך Exploiting Format String Vulnerabilities מאת scut המופץ כשנתיים באינטרנט. כל הזכויות למסמך המקורי שייכות ל-scut.

Nir Adar

Email: underwar@hotmail.com

Home Page: http://underwar.livedns.co.il

Format String Vulnerabilities

Format String Vulnerabilities הינה בעית אבטחה הנוצרת עקב שימוש לא נכון בפקודות המקבלות **מחרוזת בקרה**, כגון printf, sprintf וכדומה. מחרוזת בקרה היא מחרוזת ASCIIZ המכילה טקסט וסימני בקרה.

דוגמא לשימוש לא נכון בפונקציות אלה:

```
int func (char *user)
{
            printf(user);
}
```

דוגמא לשימוש נכון:

```
int func (char *user)
{
            printf("%s", user);
}
```

מבנה המחסנית בעת שימוש ב-printf

עבור הפקודה:

```
printf ("Number %d has no address, number %d has: %08x\n", i, a, &a);
```

המחסנית תראה כך:

stack top
. . .
<&a>
<i>
A
. . .
stack bottom

כאשר A הוא הכתובת של מחרוזת הבקרה.

שימושים להתקפה

גרימה לקריסת התוכנית:

```
printf ("%s%s%s%s%s%s%s%s%s%s%s");
```

%s קורא כתובת מהמחסנית ומתחיל לקרוא ממנה עד להגעה ל-'\0'. אם ניגש הרבה פעמים לכתובות אקראיות (אלו שנמצאות על המחסנית) סביר להניח שבמהרה נגיע לכתובת שאינה ממופה לתוכנית שלנו, והתוכנית תבצע גישה לא חוקית לזכרון.

שיטה נוספת – שימוש ב-%n. %n הוא פרמטר בקרה המקבל מצביע ל-int ורושם לתוך מצביע זה כמה תווים הודפסו עד כה. על ידי שימוש ב-%n נוכל לכתוב ערכים לכתובות בזכרון, שלא בהכרח קיימת הרשאת כתיבה אליהם.

צפיה בתוכן המחסנית:

```
printf("%08x\t%08x\t%08x\t%08x\t%08x\n");
```

מכיוון שהערכים עבור printf נלקחים מהמחסנית, שורה זו תדפיס מספר ערכים מהמחסנית עצמה.
פירצת האבטחה שהצפיה במחסנית גורמת היא אפשרות לחשוף את ערכם של משתנים לוקליים, ושל מצב התוכנית, ובכך לתת לתוקף מידע שימושי עבור התקפות המשך.

צפיה בזכרון התוכנית:

אחרי מספר שליפות מהמחסנית נגיע אל המקום בו מאוכסנת המחרוזת עצמה. נביט בפקודה הבאה:

```
printf ("AAA0AAA1_%08x.%08x.%08x.%08x.%08x");
```

אם נוסיף בסוף המחרוזת %s, יש סיכוי סביר שתתבצע קפיצה לכתובת AAA0.
על ידי החלפה של קטע זה בכתובת השייכת למרחב הכתובות של התוכנית, ניתן בשיטתיות לקרוא את כל זיכרון התוכנית:

address = 0x08480110
address (encoded as 32 bit string): "\x10\x01\x48\x08"

```
printf ("\x10\x01\x48\x08_%08x.%08x.%08x.%08x.%08x.|%s|");
```

התקפת Buffer Overflow:

אם הפקודה הבעייתית הינה sprintf, פקודה הכותבת את התוצאה לתוך כתובת בזיכרון, נוכל לתת מחרוזת בקרה הגדולה מגודל החוצץ, ולבצע התקפת Buffer Overflow.

יתכנו בעיות גם במצבים הנראים לכאורה בטוחים:

```
{
        char      outbuf[512];
        char      buffer[512];

        sprintf (buffer, "ERR Wrong command: %400s", user);
        sprintf (outbuf, buffer);
}
```

לכאורה בקוד זה קיימת הגנה – יקראו עד 400 תווים, אולם ניתן להערים על הגנה זו על ידי סיפוק המחרוזת: "%497d\x3c\xd3\xff\xbf<nops><shellcode>".

ההתקפה זהה ל-buffer overflow, מלבד ה-%497d היוצר מחרוזת באורך 497. בשילוב עם המחרוזת "ERR Wrong command:" מתרחשת חריגה של 4 בתים מגבולות המערכת, וכתובת החזרה נדרסת על ידי הכתובת 0xbfffd33c.