



הפלא ופלט - `_IO_2_1_stdout`

מאת שנהב מור

הקדמה

קומפיילרים מודרניים, בשילוב עם מערכות הפעלה מודרניות, מסוגלים לספק לקבצים בינארים, בעת יצירתם, ובעת הרצתם הגנות רבות. ביניהן: ASLR, RELRO, Canaries, Pie ועוד. נשים לב כי בכל מערכת הפעלה מודרנית, שמופעל עליה ASLR נזדקק לדליפה קלה של זיכרון בכדי לגשת לכתובות מסוימות. אם נרצה להשתמש בפונקציה מסוימת מהספרייה Libc בזמן ריצה, נצטרך קודם כל למצוא כתובת שנמצאת בחלק של Libc בזיכרון, לשחק עם ההיסטים ולחשב את הכתובת של הפונקציה הנ"ל. לשם כך, נצטרך פונקציונליות מסוימת של קריאה מהזיכרון. לדוגמא, אם נמצא מצביע של תווים שאנחנו יכולים להדפיס ולשלט על תכולתו, נוכל לשחק עם הקלט כפי שנהוג ולהדליף כתובת שתוכל לעזור לנו בהמשך.

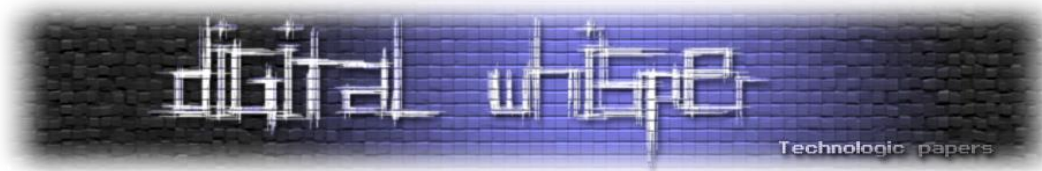
אבל רגע... מה נעשה אם לא תהיה לנו פונקציונליות של קריאה מהזיכרון? איך בכלל זה אפשרי להדליף כתובות מהזיכרון אם אין לנו אפילו יכולת קריאה מהזיכרון שבשליטתנו? במאמר זה נלמד טכניקה שמנצלת את `_IO_2_1_stdout`. בסוף המאמר תוכלו, במקרים מסוימים, גם אתם להדליף כתובות זיכרון בשמחה ובששון ללא פונקציונליות של קריאה 😊 מלהיב נכון? עכשיו בואו נצלול פנימה.

במאמר זה אניח כי לקוראים ידע בנושאים הבאים:

- [מבנה ה-Heap](#) - מאמר מאת עידן בני
- ידע כללי על Binary Exploitation

File Structures

לפני שנצלול פנימה כמו שהבטחתי נצטרך קצת להכיר מושגים ותהליכים פנימיים בספריית Glibc בכדי שחוך מלממש דברים נוכל גם להבין ולהפנים כיצד הם קורים. מבנה הנתונים העיקרי במערכת ההפעלה Linux לניהול קבצים הוא ה-`_IO_FILE` או בקיצור: FILE, ניתן למצוא אותו בספרייה הסטנדרטית ל-IO. הוא נפתח בצורה אוטומטית. ברגע שהתוכנה קוראת ל-`fopen`, הוא יוקצה בזיכרון ה-HEAP, ומשם נוכל לנהל אותו בעזרת `fread`, `fwrite`, `fclose` וכו'...



מבנה הנתונים FILE / _IO_FILE מוגדר בקובץ libio.h בספריית glibc וזהו קוד המקור שלו:

```
struct _IO_FILE {
  int _flags; /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

  /* The following pointers correspond to the C++ streambuf protocol. */
  /* Note: Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
  char* _IO_read_ptr; /* Current read pointer */
  char* _IO_read_end; /* End of get area. */
  char* _IO_read_base; /* Start of putback+get area. */
  char* _IO_write_base; /* Start of put area. */
  char* _IO_write_ptr; /* Current put pointer. */
  char* _IO_write_end; /* End of put area. */
  char* _IO_buf_base; /* Start of reserve area. */
  char* _IO_buf_end; /* End of reserve area. */
  /* The following fields are used to support backing up and undo. */
  char *_IO_save_base; /* Pointer to start of non-current get area. */
  char *_IO_backup_base; /* Pointer to first valid character of backup area */
  char *_IO_save_end; /* Pointer to end of non-current get area. */

  struct _IO_marker *_markers;

  struct _IO_FILE *_chain;

  int _fileno;
#if 0
  int _blksize;
#else
  int _flags2;
#endif
  _IO_off_t _old_offset; /* This used to be _offset but it's too small. */

#define __HAVE_COLUMN /* temporary */
  /* 1+column number of pbase(); 0 is unknown. */
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];

  /* char* _save_gptr; char* _save_egptr; */

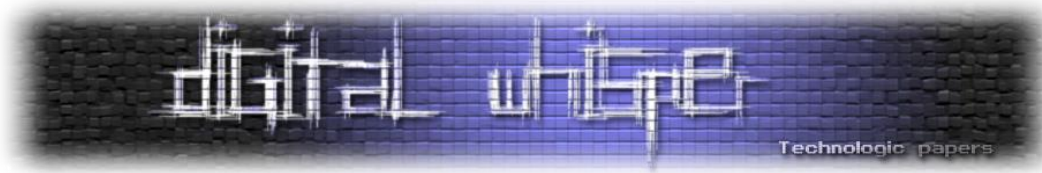
  _IO_lock_t * lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

אני יודע, הוא קצת מאיים, אבל אני מבטיח לכם שהוא לא כל כך מסובך כמו שהוא נראה, הסבר מלא ומפורט על קוד המקור ניתן למצוא ב**מאמר** המעולה של עמית שמואל שיסביר לכם את רוב המבנה, ואני אעמיק בחלק שנדרש על מנת ביצוע השיטה.

דגלים

לדגלים יש מטרה אחת די פשוטה: במהלך זמן הריצה הדגלים הם אלו שיחליטו אילו חלקים בקוד ירוצו. בכדי לבדוק האם דגל מסוים דלוק מתבצעת פקודת AND (&). לשדה זה יש ערך חשוב מאוד בהדלפת כתובות LIBC בשימוש של _IO_2_1_stdout.

הערה: יש שוני בין הדגלים בכל LIBC ולכן צריך לשים לב עם איזה LIBC עובדים, אבל בעקרון הדגלים עצמם הם אותו הדבר.



להלן רשימת הדגלים מתוך קוד המקור:

```
#define _IO_MAGIC 0xFBAD0000 /* Magic number */
#define OLD_STDIO_MAGIC 0xFABC0000 /* Emulate old stdio. */
#define _IO_MAGIC_MASK 0xFFFF0000
#define _IO_USER_BUF 1 /* User owns buffer; don't delete it on close. */
#define _IO_UNBUFFERED 2
#define _IO_NO_READS 4 /* Reading not allowed */
#define _IO_NO_WRITES 8 /* Writing not allowed */
#define _IO_EOF_SEEN 0x10
#define _IO_ERR_SEEN 0x20
#define _IO_DELETE_DONT_CLOSE 0x40 /* Don't call close(_fileno) on cleanup. */
#define _IO_LINKED 0x80 /* Set if linked (using _chain) to streambuf::_list_all.*/
#define _IO_IN_BACKUP 0x100
#define _IO_LINE_BUF 0x200
#define _IO_TIED_PUT_GET 0x400 /* Set if put and get pointer logicly tied. */
#define _IO_CURRENTLY_PUTTING 0x800
#define _IO_IS_APPENDING 0x1000
#define _IO_IS_FILEBUF 0x2000
#define _IO_BAD_SEEN 0x4000
#define _IO_USER_LOCK 0x8000
```

יש מספר קסם שתמיד משתמשים בו שהוא 0xfbad0000 ה-2 בתים העליונים משמשים לאינדיקציה על איזה קובץ מדובר ו-2 הבתים התחתונים מאבחנים כל מיני אפשרויות. בהמשך אפרט לעומק איזה דגלים ולמה נצטרך לזייף.

מה זה IO Buffering?

אם לא נשתמש ב-Buffering בכדי לרשום דברים לדיסק נצטרך לחכות זמן רב. המעבד שלנו הרבה יותר מהיר מרכיבי החומרה שעימם הוא מתקשר. ולכן במקום לרשום byte לדיסק בכל פעם שנרצה לרשום כפלט או לקרוא כקלט, אנו מאחסנים את המידע ב-buffer ותלוי מצב ה-Buffering שאנחנו נמצאים בו המידע ירשם לדיסק.

ל-Buffering יש 3 מצבים:

1. **Full buffering** - ה-buffer מתנקה בכל פעם שהוא מתמלא לחלוטין או כשמשתמשים בפונקציות ניקוי כגון fflush וכו'...
2. **New Line Buffering** - בכל פעם שנשתמש ב-enter ה-buffer יתנקה או מתי כשנשתמש בפונקציות ניקוי.
3. **Unbuffered** - אין buffer ובכל אפשרות של הקרנל להוציא פלט הוא ישר יוציא אותו.

הסבר FWRITE

לפני שנצלול לקוד המקור חשוב להסביר כמה מצביעים חשובים שנמצאים במבנה הנתונים FILE:

- `_IO_buf_base` - כתובת ההתחלה של ה-Buffer I/O
- `_IO_buf_end` - כתובת הסיום של ה-Buffer I/O
- `_IO_write_base` - כתובת ההתחלה ל-Buffer הפלט
- `_IO_write_ptr` - התו האחרון שנכנס ל-Buffer הפלט
- `_IO_write_end` - סוף Buffer הפלט

ישנם כמה חוקים ברורים, ה-Buffer הנ"ל נוצר בעזרת פונקציה בשם `doallocbuf`, ה-Buffer יכול לשמש לקבלת מידע (input) ולפליטת מידע (output).

במאמר זה נתמקד רק על פליטת המידע. ברגע שה-Buffer משמש לפליטת מידע אז המצביעים שלו יעבדו כך: כתובת ההתחלה תמצא ב-`_IO_write_base` וכתובת הסיום תמצא ב-`_IO_write_end`, הכתובת של התו הנוכחי תמצא ב-`_IO_write_ptr`. בין `_IO_write_base` ל-`_IO_write_ptr` ימצא כל ה-Buffer שכבר מילאנו במידע, ובין `_IO_write_ptr` לבין `_IO_write_end` תמצא כל שארית ה-Buffer שפנויה לנו.

נתמקד בפונקציה שהעיקרית ברצף העבודה של ביצוע הפונקציה `fwrite`, `_IO_new_file_xsputn`.

ניתן לחלק אותה לארבעה חלקים:

1. בדיקת המקום הפנוי ב-Buffer והשוואה אל כמות הפלט שנרצה לרשום. אם נשאר מספיק אז ישירות להעתיק את ה-Buffer שרצינו לרשום ל-Buffer הפלט. לדוגמה אם נרשום `puts("hello world")`, אז אנו רוצים להעתיק ל-Buffer פלט של `STDOUT` את הפלט `hello world`, נבדוק האם יש ב-Buffer פלט של `STDOUT` מעל 12 תווים פנויים ואם יש מיד נעתיק את `hello world` ל-Buffer פלט של `STDOUT`. כפי שהסברנו הדרך לבדוק האם נשאר מקום זה לבצע (`_IO_write_ptr` - `_IO_write_end`) אם התוצאה גדולה מ-0 וגם גדולה ממה שנרצה להעתיק ל-Buffer (בדוגמה הזו 12). בנוסף כמובן שגם המצביעים ישונו בהתאם (PTR יגדל).
2. אם עדיין יש מידע שלא רשמנו, משמע: ה-Buffer פלט עוד לא נוצר או שאין מספיק מקום לרשום את מה הפלט אליו. במקרה זה הפונקציה `_IO_new_file_overflow` תקרא ע"י השדה `IO_OVERFLOW_` בטבלה הוירטואלית של ה-File Stream בכדי ליצור או לנקות את ה-Buffer.
3. לאחר שה-Buffer פלט יתנקה יש לבדוק האם יש מספיק מקום או אם קיימת חריגה מגודל ה-Buffer (גודל הבלוק). הבדיקה בודקת האם הגודל של מה שנרצה לרשום גדול מ-`end-base`. אם כן נקרא ישירות לפונקציה `new_do_write` שכותבת בלי להשתמש ב-Buffer, פשוט כותבת ישירות בבלוקים.
4. לבסוף - נקרא לפונקציה `_IO_default_xsputn` שתעתיק את המידע ל-Buffer הפלט.

הסבר על הפונקציה `_IO_new_file_overflow`:

המטרה הראשית של פונקציה זו היא לנקות את ה-`buffer` או ליצור את ה-`buffer`. נסתכל על קוד המקור ונמצא שם בדיקה שמטרתה היא לראות אם הדגל `_IO_NO_WRITES` דולק. אם הוא דולק הפונקציה ישר חוזרת מכיוון שבעצם לא ניתן לרשום.

לאחר מכן הפונקציה תבדוק האם `_IO_write_base` ריק. אם כן היא קוראת ל-`doallocbuf`. הדבר היחיד שמעניין אותנו היא שהפונקציה הזו יוצרת את ה-`buffer`, לא מעבר לזה. לאחר שנחזור מ-`doallocbuf` הפונקציה `overflow` תאתחל את ה-`buffer` הפלט. אחר כך נקרא לפונקציה `_IO_do_write`.

הפונקציה `_IO_do_write` תקרא לפונקציה `new_do_write` ובתוך פונקציה זו אנו נשתמש בפונקציית המערכת: `WRITE`, בכדי להדפיס את כל מה שנמצא בין `_IO_write_base` ל-`_IO_write_ptr`. בנוסף היא מאפסת את ה-`buffer`.

לאחר הסיכום הארוך נתחיל להיכנס לחלק מחתיכות הקוד החשובות שנצטרך לשנות, בסופו של דבר המטרה הסופית שלנו היא לגרום לפסיקת המערכת `Write` לפעול.

```
IO_new_file_overflow (_IO_FILE *f, int ch)
{
    if (f-> flags & IO_NO_WRITES) /* SET ERROR */
    {
        f-> flags |= IO_ERR_SEEN;
        _set_errno (EBADF);
        return EOF;
    }
    /* If currently reading or no buffer allocated */
    if ((f-> flags & IO_CURRENTLY_PUTTING) == 0 || f-> IO_write_base == NULL)
    {
        /* Allocate a buffer if needed. */
        if (f-> IO_write_base == NULL)
        {
            IO_doallocbuf (f);
            IO_setg (f, f-> IO_buf_base, f-> IO_buf_base, f-> IO_buf_base);
        }
        /* Otherwise must be currently reading.
        If IO read_ptr (and hence also IO read_end) is at the buffer end,
        logically slide the buffer forwards one block (by setting the
        read pointers to all point at the beginning of the block). This
        makes room for subsequent output.
        Otherwise, set the read pointers to IO_read_end (leaving that
        alone, so it can continue to correspond to the external position). */
        if (_glibc_unlikely (!IO_in_backup (f)))
        {
            size_t nbackup = f-> IO_read_end - f-> IO_read_ptr;
            IO_free_backup_area (f);
            f-> IO_read_base -= MIN (nbackup,
                f-> IO_read_base - f-> IO_buf_base);
            f-> IO_read_ptr = f-> IO_read_base;
        }

        if (f-> IO_read_ptr == f-> IO_buf_end)
            f-> IO_read_end = f-> IO_read_ptr = f-> IO_buf_base;
        f-> IO_write_ptr = f-> IO_read_ptr;
        f-> IO_write_base = f-> IO_write_ptr;
        f-> IO_write_end = f-> IO_buf_end;
        f-> IO_read_base = f-> IO_read_ptr = f-> IO_read_end;

        f-> flags |= IO_CURRENTLY_PUTTING;
        if (f-> mode <= 0 && f-> flags & (_IO_LINE_BUF | _IO_UNBUFFERED))
            f-> IO_write_end = f-> IO_write_ptr;
        if (ch == EOF)
            return IO_do_write (f, f-> IO_write_base,
```

בקוד מעלינו ניתן לראות את הפונקציה `_IO_new_file_overflow`. המטרה שלנו בפונקציה הזו היא להגיע למצב בו אנו קוראים לפסיקת המערכת - `Write`, היא נקראת באמצעות `_IO_do_write` (כפי שניתן לראות מסומן בוורוד). אנו רוצים להגיע לקטע קוד זה מכיוון שנרצה להדליף מידע, כלומר נצטרך לכתוב מידע



והפונקציה `_IO_do_write` תוביל לכך שנדפיס את `buffer` הפלט שלנו. הפרמטרים שנשלחים לפונקציה זו הם:

1. המצביע `_IO_write_base`, בשביל לסמן את ההתחלה של ה-`buffer` שאנו רוצים להתחיל להדפיס ממנו.

2. גודל ה-`buffer` שנרצה להדפיס, מחושב על ידי: `_IO_write_ptr - _IO_write_base`.

הסיבה שדבר זה חשוב הוא בגלל שאם נשלח את המצביע בפרמטרים לפונקציה `_IO_write_base`, ונשנה אותו להצביע קצת לפני איפה שהוא מצביע, נדפיס את כל מה שנמצא בינו לבין `_IO_write_ptr`, אם יש שם כתובות `libc` נוכל לקרוא אותם ולהנות ☺

שימו לב, בכדי להגיע לפונקציה זו יש כמה בדיקות שנצטרך לעבור:

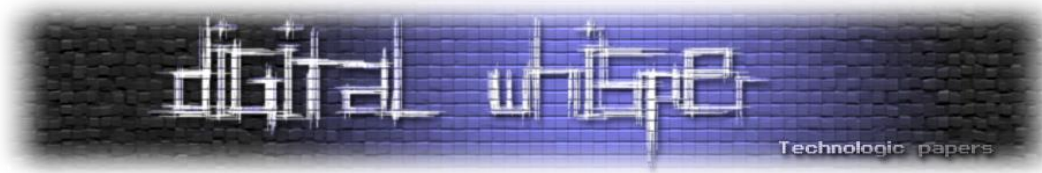
1. מתבצעת בדיקה שמטרתה היא לבדוק האם הרשאות הקובץ שפתחנו מאפשרות לנו לרשום בו, בדיקה זו מבוצעת כפי שנאמר בהתחלה בעזרת פעולת `AND`, אם נבצע את ה-`AND` עם הדגל ונקבל 1 נדע כי הדגל הזה קיים. אם הדגל הזה קיים אנו לא נוכל להמשיך ולהגיע למטרה שלנו `_IO_do_write`, ולכן בדגלים נצטרך לשים לב כי הביט שערכו 8 (הרביעי מימין) צריך להיות כבוי. או במילים אחרות:

```
_flags & 0x8 = 0
```

2. הבדיקה השנייה באה לבדוק האם ה-`buffer` ריק, אם ה-`buffer` ריק אנחנו נקצה לו זיכרון ונאתחל את המצביע שלנו... דמיינו שכבר ערכנו את `io_write_base` לכתובת נמוכה יותר ואז מגיעה הבדיקה המבאסת הזו ומאתחלת לנו את הפוינטרים ☹, לא נשמע משהו שנרצה להכנס אליו, בכדי לא להכנס לזה. נצטרך לגרום ל-2 הבדיקות להיות שליליות גם הדגל הנ"ל יצטרך להיות דלוק וגם המצביע לא יכול להיות ריק. הדגל הוא `_IO_CURRNETLY_PUTTING`. שמו מעיד על מעשיו ☺. נדאג לכך שהדגל יהיה דלוק בכך שהביט ה-12 יהיה דלוק או בפשטות:

```
_flags & _IO_CURRNETLY_PUTTING = 1
```

ערכו של דגל זה הוא `0x800`.



מ-`_IO_do_write` נגיע לפונקציה נוספת לפני פסיקת המערכת (`syscall`) שמה ברבים היא:
: `new_do_write` זו הפונקציה שבסוף קוראת ל-`SYSWRITE`:

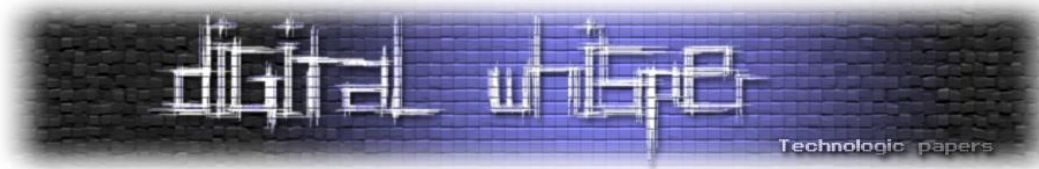
```
new_do_write (_IO_FILE *fp, const char *data, _IO_size_t to_do)
{
    _IO_size_t count;
    if (fp-> flags & _IO_IS_APPENDING)
        /* On a system without a proper O_APPEND implementation,
         * you would need to sys_seek(0, SEEK_END) here, but is
         * not needed nor desirable for Unix- or Posix-like systems.
         * Instead, just indicate that offset (before and after) is
         * unpredictable. */
        fp-> offset = _IO_pos_BAD;
    else if (fp-> _IO_read_end != fp-> _IO_write_base)
    {
        _IO_off64_t new_pos
        = _IO_SYSSEEK (fp, fp-> _IO_write_base - fp-> _IO_read_end, 1);
        if (new_pos == _IO_pos_BAD)
            return 0;
        fp-> _offset = new_pos;
    }
    count = _IO_SYSWRITE (fp, data, to_do);
    if (fp-> _cur_column && count)
        fp-> _cur_column = _IO_adjust_column (fp-> _cur_column - 1, data, count) + 1;
    _IO_setg (fp, fp-> _IO_buf_base, fp-> _IO_buf_base, fp-> _IO_buf_base);
    fp-> _IO_write_base = fp-> _IO_write_ptr = fp-> _IO_buf_base;
    fp-> _IO_write_end = (fp-> _mode <= 0
        && (fp-> flags & (_IO_LINE_BUF | _IO_UNBUFFERED))
        ? fp-> _IO_buf_base : fp-> _IO_buf_end);
    return count;
}
```

יש לנו עוד שתי בדיקות נוספות, רק אחת מהן יכולה להתבצע מכיוון והן מסוג if-else. ולכן נצטרך להבין איזה אחד מזרמי הקוד הללו ישבש לנו פחות את המשימה.

▪ **בדיקה ראשונה:** בדיקה פשוטה למדי של דגלים, בדיקה של הדגל `_IO_IS_APPENDING`, אם דגל זה דלוק - זרימת התוכנית תכנס לקטע הקוד שלאחר בדיקה זו, מה שיביא לכך שאת שדה ה-`offset` של מבנה הקובץ נשווה לאיזשהו ערך שלא יהרוס לנו את התוכנית.

▪ **בדיקה שנייה:** ניתן להימנע גם מבדיקה זו כל עוד `_IO_read_end = _IO_write_base` אבל זה לא כל כך פשוט. בנוסף אם נכנס לקטע הקוד בהינתן ועברנו את הבדיקה הזו, הפונקציה `_IO_SYSSEEK` תופעל; פונקציה זו היא ה-`lseek`. אם נשווה את `_IO_read_end` ל-0 כמתוכנן נגיע למצב מביך בו הפרמטר השני ל-`lseek` יהיה ממש גדול מכיוון ואנחנו מבצעים `(0 - _IO_write_base)` - מה שבסוף יגרום לחריגה מתחום הנתונים ויציאה מהתוכנית. מה שלא יביא אותנו כל כך רחוק לפי מטרתנו...

אז מה נוכל לעשות על מנת להתגבר על 2 בדיקות אלו ולהגיע ל-`SYSWRITE`? מכיוון וסוגי בדיקות אלה הן מסוג if-else, כלומר - אם הבדיקה הראשונה תעבור אז לא נכנס לקטע הקוד של הבדיקה השנייה (קטע הקוד שמבלגן לנו את המשימה ומסובך יותר). ולכן נסיף את הדגל `_IO_IS_APPENDING` לדגלים שלנו (ערכו הוא `0x1000`) וכך נקבל את מספר הקסם הנ"ל - `0xfbad1800`.



אני מניח שבמהלך קריאת כל התאוריה הזו שאלתם את עצמכם: רגע! אני לא אוכל להדליף כתובות זיכרון אם לא ישתמשו ספציפית ב-Fwrite בתוכנית? אז בואו נבחן פעולות פלט אחרות:

:Puts

```
[#0] Id 1, Name: "puts", stopped 0x7ffff7ed0060 in __GI__libc_write (), reason: BREAKPOINT
[#0] 0x7ffff7ed0060 - __GI__libc_write (fd=0x1, buf=0x555555592a0, nbyte=0xc)
[#1] 0x7ffff7e50e8d - _IO_new_file_write (#=0x7ffff7faf6a0 <_IO_2_1_stdout_>, data=0x555555592a0, i=0xc)
[#2] 0x7ffff7e52951 - _IO_new_file_write (co_dc=0xc, data=0x555555592a0 "hello world\n", fp=0x7ffff7faf6a0 <_IO_2_1_stdout_>)
[#3] 0x7ffff7e52951 - _IO_new_file_write (co_dc=0xc, data=0x555555592a0 "hello world\n", fp=0x7ffff7faf6a0 <_IO_2_1_stdout_>)
[#4] 0x7ffff7e52951 - _IO_new_file_write (fp=0x7ffff7faf6a0 <_IO_2_1_stdout_>, data=0x555555592a0 "hello world\n", co_dc=0xc)
[#5] 0x7ffff7e52e93 - _IO_new_file_write (#=0x7ffff7faf6a0 <_IO_2_1_stdout_>, cl=0xa)
[#6] 0x7ffff7e4659a - _IO_puts (str=0x55555556004 "hello world")
[#7] 0x5555555515d - main ()
```

כפי שניתן לראות גם פה באותה התצורה נקרא ל-`_IO_new_file_overflow` מה שמזכיר לנו פחות או יותר את `fwrite`, כלומר גם בעזרת `puts`, `printf` ועוד תוכלו לממש את שיטה זו.

ניצול החולשה

קעת ניגש לעניין: כיצד נדליף מידע בעזרת מה שלמדנו עכשיו? אמרנו שבכל פעם שנדפיס את ה-`buffer` יודפס כל מה שהיה רושם בין `__IO_write_base` ל-`__IO_write_ptr`. אם נוכל לשלוט על הדגלים בכדי שנוכל להדפיס למסך + נשנה את המצביע של `_IO_write_base` להיות בכתובת נמוכה יותר בעזרת דריסה חלקית נוכל להדפיס כתובות זיכרון בלי פונקציונליות של שליטה על קריאה מהזיכרון ☺ אני יודע... מטורף.

אז בוא נתחיל להבין מה השלבים בכלל לדבר האדיר הזה: בכדי לרשום ל-`_IO_2_1_stdout` נצטרך דרך לרשום לשם, אבל רגע אחד... בכל מערכת הפעלה שמכבדת את עצמה ה-`ASLR` יהיה דלוק, מה שיגרום לכך שהמבנה כל פעם יהיה בכתובת אחרת... ולכן לא נוכל ישר לרשום למבנה, קודם כל נצטרך בכלל למצוא את הכתובת שלו.

התהליך:

הדרך פשוטה היא "הרעלת `tcache`", להכניס את הכתובת של ה-`unsortedbin` לאחד מה-`tcachebins`. ברגע שעשינו את זה כבר יש לנו שלב חשוב מאוד כי עכשיו נוכל לרשום ב-`main arena` אם נקצה את הצ'אנק הזה. כמו שאנחנו יודעים יש הסטים קבועים בין נתונים שנמצאים באותם החלקים בזיכרון. ולכן נוכל לבצע דריסה חלקית של הכתובת `MAIN ARENA` שכרגע שוכנת לנו ב-`Tcache` ולשנות אותה ל-`_IO_2_1_stdout`. עכשיו כשנקצה אותה נוכל לערוך את הצ'אנק ובשמחה ובששון נוכל לנצל את `stdout` ולשנות לו את התכולה למה שנרצה.

הכיף יראה כך:

```
_flags + _IO_read_base + _IO_read_ptr + _IO_read_end + partial
overwrite _IO_write_base
```




דוגמה להשמה

ניקח קובץ בינארי שאני אוהב לעשות בו בדיקות כאלו, לקובץ זה קוראים malloc_testbad זה קובץ ש-Max Kamper יצר למטרת [הקורס שלו](#) ב-HEAP exploitation ב-Udemy. בקובץ זה יש פחות או יותר את רוב חולשות ה-Heap כגון: שימוש לאחר שחרור צ'אנקים, חריגת חוצץ, הדלפת כתובות וכו'...

בקובץ זה יש גם אפשרות קריאה אבל אנו נדליף זיכרון בלי להשתמש בפונקציה זו ☺

כפי שאמרנו ניתן לחלק את השיטה הזו לשלושה שלבים:

1. הכנסת ה-Unsortedbin לתוך ה-Tcachebins.
2. שינוי כתובת ה-main_arena של ה-unsortedbin לכתובת של IO_2_1_stdout.
3. הקצאת הצ'אנק ושינוי מבנה הקובץ של stdout.

בקובץ זה, בגלל כמות החולשות, נוכל בקלות להגיע למצב שכתובת ה-unsortedbin נמצאת לנו ב-tcachebins. בסה"כ נצטרך להקצות כמה צ'אנקים בגודל ה-tcache אבל מעל גודל ה-fastbins ונשחרר 6 מהם, לאחר מכן נבצע שחרור כפול של אותו הצ'אנק באותו הגודל כך שהוא יכנס פעם אחת ל-tcache ופעם אחת ל-unsortedbin.

כפי שאנו יודעים, כאשר הצ'אנק נכנס ל-unsortedbin הכתובת של ה-unsortedbin נרשמת בשדה ה-fd ובשדה ה-bk. מכיוון והצ'אנק שנמצא ב-tcache נמצא באותה הכתובת, והכנסנו את אותו הצ'אנק לשני bins שונים אז גם הצ'אנק שב-tcache יהנה מכתובת זו בשדה ה-fd וה-bk:

```
Tcachebins[idx=16, size=0x120, count=3] - Chunk(addr=0x603a40, size=0x120, flag
s=PREV_INUSE) - Chunk(addr=0x7fff7dd2c78, size=0x0, flags=! PREV_INUSE) - C
hunk(addr=0x603c70, size=0x0, flags=! PREV_INUSE)
Fastbins for arena at 0x7fff7dd2c20
Fastbins[idx=0, size=0x20] 0x00
Fastbins[idx=1, size=0x30] 0x00
Fastbins[idx=2, size=0x40] 0x00
Fastbins[idx=3, size=0x50] 0x00
Fastbins[idx=4, size=0x60] 0x00
Fastbins[idx=5, size=0x70] 0x00
Fastbins[idx=6, size=0x80] 0x00
Unsorted Bin for arena at 0x7fff7dd2c20
[+] unsorted_bins[0]: fw=0x603a30, bk=0x603a30
- Chunk(addr=0x603a40, size=0x120, flags=PREV_INUSE)
[+] Found 1 chunks in unsorted bin.
Small Bins for arena at 0x7fff7dd2c20
[+] Found 0 chunks in 0 small non-empty bins.
Large Bins for arena at 0x7fff7dd2c20
[+] Found 0 chunks in 0 large non-empty bins.
```

כעת יש לנו כתובת של libc הישר בתוך ה-tcache. נוכל להקצות אותו ללא בעיה מכיוון ובגרסא זו של libc אין שום בדיקות שרירותיות לגבי הקצאות מה-tcache. אנחנו קרובים מתמיד להדלפת הזיכרון. רק נצטרך לשנות חלקית את הכתובת שכבר יש לנו כדי שתתאים ל-IO_2_1_stdout. אבל פה אנחנו נתקלים בבעיה קלה שקצת מבאסת את השיטה הזו...



```
gef> p 0x7ffff7dd2c78
$3 = 0x7ffff7dd2c78
gef> p &_IO_2_1_stdout_
$4 = (struct _IO_FILE_plus *) 0x7ffff7dd3720 <_IO_2_1_stdout_>
gef> x/gx 0x7ffff7dd2c78
0x7ffff7dd2c78 <main_arena+88>: 0x0000000000603c70
```

כפי שאנו רואים, הכתובת של `_IO_2_1_stdout` גדולה ביותר מ-4 ספרות הקסדצימליות (2 בתים). הבעיה שלנו היא ש-ASLR מופעל. ASLR מג'נט את רוב הבתים חוץ מכמה בתים בתחילת הכתובת ושלושת הספרות האחרונות של הכתובת. ספרות אלו נשארות קבועות, זאת אומרת שתמיד בספריית ה-`libc` הזו הכתובת שאנו רוצים (`_IO_2_1_stdout`) תסתיים ב-`0x720` אבל הספרה שלפני ה-7 כל פעם תהיה ספרה אחרת. אופ ☹️, זה אומר ששיטה זו עובדת אחת ל-16 פעמים אבל היי, שיטה עובדת זו שיטה טובה.

בכל זאת לאחר שקצת התאכזבנו אבל קיבלנו את המגבלה של השיטה נרצה לעבור לחלקה השני, לדרוס ולשנות את הכתובת לכתובת של `_IO_2_1_stdout`. בקובץ זה נוכל פשוט לערוך את הצ'אנק. מכיוון ויש פה שימוש לאחר שחרור צ'אנקים, נערוך את הצ'אנק ובשדה ה-`fd` נרשום ספרה רנדומלית כספרה הכי גדולה ב-2 בתים ואת שאר ההיסט שאנחנו מכירים כי הוא קבוע, לדוגמא, `0x3720`. וכמובן שאחרי שערכנו נקצה את הצ'אנק.

וכעת, לחלק האחרון: כל מה שנשאר לנו לעשות זה לשנות את מבנה הקובץ כפי שתכננו: בדגלים נשים את הערך `0xfbad1800`, לאחר מכן נשים 3 פעמים 0 כדי לא להתייחס ל"מצביעי הקלט", ואז נדרוס חלקית את הכתובת שרצינו לדרוס (שהיא: `_IO_write_base`), ונקטין אותה כמה שנוכל, (`\x00`) יעשה את העבודה). כך שפעם הבאה שפונקציה שמתמשת ב-`_IO_new_overflow` (לדוגמא `puts`) תקרא, אנו בתקווה נדליף כמה כתובות זיכרון.

לאחר עריכת הצ'אנק המשוחרר:

```
Tcachebins[idx=16, size=0x120, count=2] ← Chunk(addr=0x603a40, size=0x120, flags=PREV_INUSE) ← Chunk(addr=0x7ffff7dd3720, size=0x7ffff7dcf420, flags=! PREV_INUSE) ← [Corrupted chunk at 0xfbad2887]
Fastbins for arena at 0x7ffff7dd2c20
Fastbins[idx=0, size=0x20] 0x00
Fastbins[idx=1, size=0x30] 0x00
Fastbins[idx=2, size=0x40] 0x00
Fastbins[idx=3, size=0x50] 0x00
Fastbins[idx=4, size=0x60] 0x00
Fastbins[idx=5, size=0x70] 0x00
Fastbins[idx=6, size=0x80] 0x00
Unsorted Bin for arena at 0x7ffff7dd2c20
[+] unsorted_bins[0]: fw=0x603a30, bk=0x603a30
→ Chunk(addr=0x603a40, size=0x120, flags=PREV_INUSE)
[+] Found 1 chunks in unsorted bin.
Small Bins for arena at 0x7ffff7dd2c20
[+] Found 0 chunks in 0 small non-empty bins.
Large Bins for arena at 0x7ffff7dd2c20
[+] Found 0 chunks in 0 large non-empty bins.
```

לאחר הקצאת הצ'אנק שערכנו ושינוי ערכיו:

```
gef> p _IO_2_1_stdout_
$1 = {
  file = {
    _flags = 0xfbad1800,
    _IO_read_ptr = 0x0,
    _IO_read_end = 0x0,
    _IO_read_base = 0x0,
    _IO_write_base = 0x7ffff7dd3700 <_IO_2_1_stderr_+192> "",
    _IO_write_ptr = 0x7ffff7dd37a3 <_IO_2_1_stdout_+131> "\n",
    _IO_write_end = 0x7ffff7dd37a3 <_IO_2_1_stdout_+131> "\n",
    _IO_buf_base = 0x7ffff7dd37a3 <_IO_2_1_stdout_+131> "\n",
    _IO_buf_end = 0x7ffff7dd37a4 <_IO_2_1_stdout_+132> "",
    _IO_save_base = 0x0,
    _IO_backup_base = 0x0,
    _IO_save_end = 0x0,
    _markers = 0x0,
    _chain = 0x7ffff7dd29a0 <_IO_2_1_stdin_>,
    _fileno = 0x1,
    _flags2 = 0x0,
    _old_offset = 0xffffffffffffffff,
    _cur_column = 0x0,
    _vtable_offset = 0x0,
    _shortbuf = "\n",
    _lock = 0x7ffff7dd4880 <_IO_stdfile_1_lock>,
    _offset = 0xffffffffffffffff,
    _codecvt = 0x0,
    _wide_data = 0x7ffff7dd28a0 <_IO_wide_data_1>,
    _freeres_list = 0x0,
    _freeres_buf = 0x0,
    __pad5 = 0x0,
    _mode = 0xffffffff,
    _unused2 = '\000' <repeats 19 times>
  },
  vtable = 0x7ffff7dcf420 <__GI__IO_file_jumps>
}
```

סיכום השיטה

לאחר שעברנו על כל התאוריה מאחורי שיטה זו הנה מה שתצטרכו לשנות במבנה הקובץ. מבחינת דגלים: ישנם שלושה דגלים שצריכים להיות דלוקים (הכוונה בדלוקים היא שאם נבצע את פעולת ה-AND בין הדגל לבין הערך של שדה הדגלים שלנו נקבל 1, כבוי כלומר נקבל 0).

1. הדגל `_IO_NO_WRITES` צריך להיות כבוי - ערכו הוא `0x8`
2. הדגל `_IO_CURRENTLY_PUTTING` צריך להיות דלוק - ערכו הוא `0x800`
3. הדגל `_IO_IS_APPENDING` צריך להיות דלוק - ערכו הוא `0x1000`. כמו שהזכרתי מקודם תדאגו שהשדה יהיה שווה ל-`0xfbad1800` שזה מספר הקסם + הדגלים שאנו זקוקים להם בשביל השיטה תעבוד.
4. נשנה את המצביע `_IO_write_base` כך שיצביע לאזור שנרצה להדליף (בדרך כלל פשוט נקטין אותו קצת בכדי להדליף כתובות של `libc`). אבל בעקרון ניתן לשחק עם `_IO_write_base` ו-`_IO_write_ptr` כך שידפיסו משהו ספציפי שנרצה. (מזכיר: כשהתוכנית תדפיס היא תדפיס מ-`base` ל-`ptr`).



סיכום

במאמר זה למדנו על מבנה הקובץ, על I/O Buffering, כיצד עובדות רוב פונקציות הפלט של הספרייה החיצונית libc, כיצד ניתן לנצל מידע זה בכדי להביא לדליפת זכרון בלי פונקציונליות קריאה בקובץ ההרצה ובנוסף ראינו השמשה שלב אחר שלב של השיטה.

בתחום אבטחת המידע, ישנה תחרות תמידית בין הגנה לבין התקפה. לשם כך, למרות שהשיטה מביאה איתה חומרים לא הכי מוכרים, היא טובה והיא תוכל לעזור לכם כששיטות פשוטות יותר לא יעזרו. ככל שנדע יותר בתחום זה כך נוכל לתקוף מכל הכיוונים עד שלבסוף נצליח.

בשבילי, FSOP בכללי, ותקיפה בעזרת file streams לא הייתה פשוטה, אך לאחר תרגול, שבירת ראש קלה, והרבה תרגום מסינית לאנגלית (תודה לסינים על המאמרים המעולים ב-fsop), בסוף הבנתי.

כדי לתרגל ולהשתפר בשיטות דמויות file streams המלצתי היא להתחיל בלקחת אתגרים קלים עם הרבה חולשות ולנצל את שיטות אלו עליהם. ברגע שכמה פעמים תצליחו בעצמכם להשמיש "מקרים פשוטים" הידע שתצברו יעזור לכם בעת אתגרים קשים יותר.

זהו 😊, כעת גם אתם יכולים להדליף כתובות זיכרון מבלי להשתמש בפונקציונליות של קריאה מהזיכרון. וגם למדתם הרבה על תהליכים מאוד חשובים לפי דעתי.

קצת עליי

אני שנהב מור בן 21, הנדסאי ומפתח Embedded בתחום הסייבר בחיל האוויר. מתעניין ב-Binary Exploitation ואבטחת מידע כבר מעל שנה ומפתח מגיל 17.

לשאלות, הערות והארות ניתן לפנות אליי במייל: shenhavmor10@gmail.com או ב-[LinkedIn](https://www.linkedin.com/in/shenhavmor10).