Digital Whisper
Technologic Papers

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.

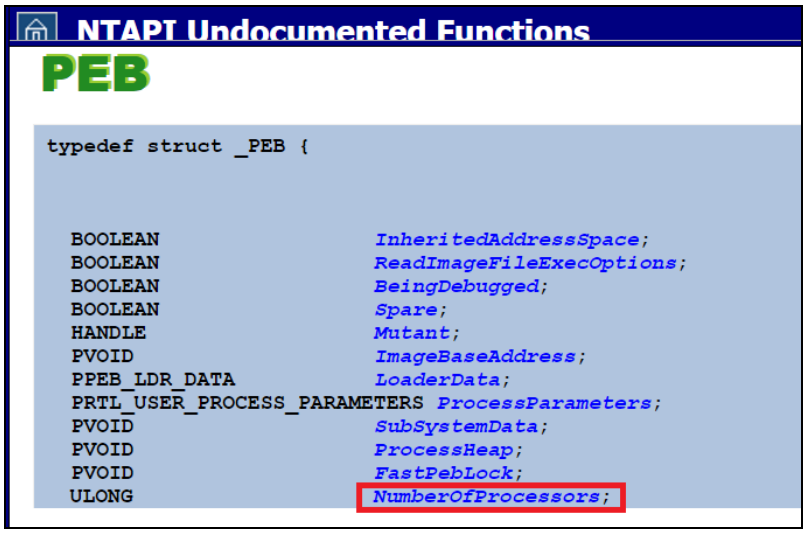
כתוצאה מהתקדמות של טכנולוגיה זו, וירוסים ופוגענים צריכים לדעת להתמודד עם המצב החדש, ולכן פיתחו לעצמם כל מיני טכניקות (ידועות יותר, וידועות פחות) עקיפה שונות שאותם אני הולך להציג לכם במהלך המאמר.

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.



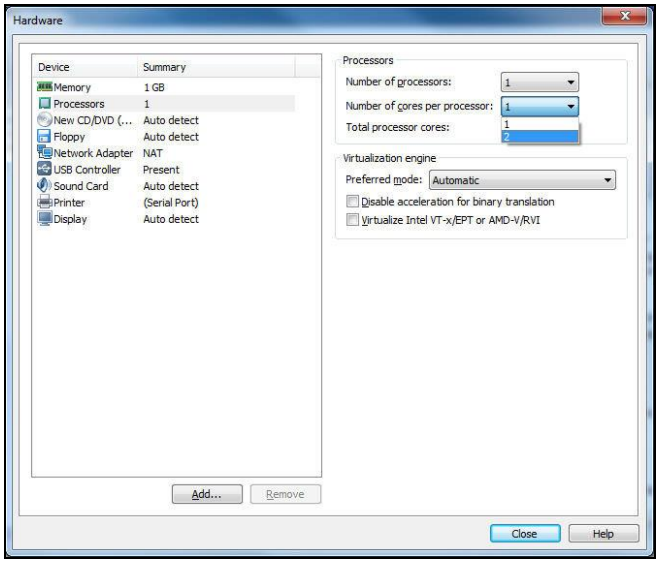
[השמטתי פרמטרים נוספים מהתמונה מפאת חוסר המקום במאמר.]

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;
}
```

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

אגב, מצדם של החוקרים, דרך ראשונה לעקיפת טכניקה זו, היא ע"י שינוי אוגר השיוויון - ZF, Zero Flag, על מנת לשבור את השיוויון הנ"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 כאן 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.

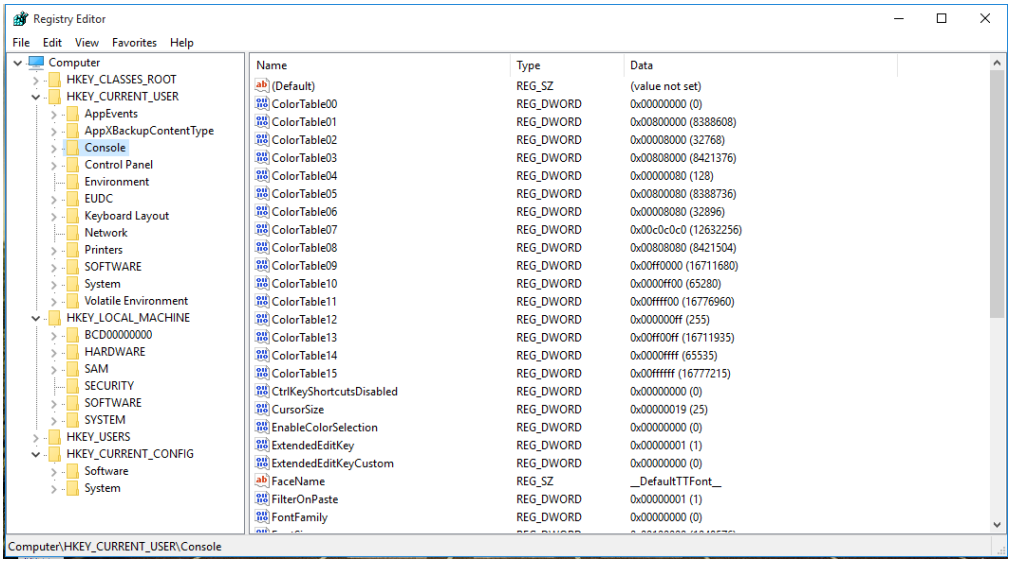
התיאוריה שעומדת מאחורי זה אומרת שבמידה והמערכת הפעלה רצה בתוך מכונת VMware, יהיה לנו זמין פורט בשם 'VX' שאיתו נוכל לבצע תקשורת בין המערכת הפעלה שמארחת אותנו לבין מ"ה הוירטואלית. לחברת VMware ישנו Magic Number שהוא 0x564D5868 (המייצג 'VMXh' ב-ASCII), אשר איתו נוכל לזהות הרצה תחת סביבה וירטואלית באמצעות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

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, RegQueryValueEx, RegCloseKey - נוכל לבדוק האם הערכים קיימים ומכילים את המילים השמורות של שמות המוצרים, כגון VMware, VirtualBox, QEMU, Xen ועוד.

- 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 יכיל 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 בפוקנציות 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 יכיל 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 השיוויון, 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.

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 "\\SAMPLE", "\\VIRUS", "\\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 ב-Windows 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 כתובת MAC קבועות התחלתיות של VMware, והן:

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

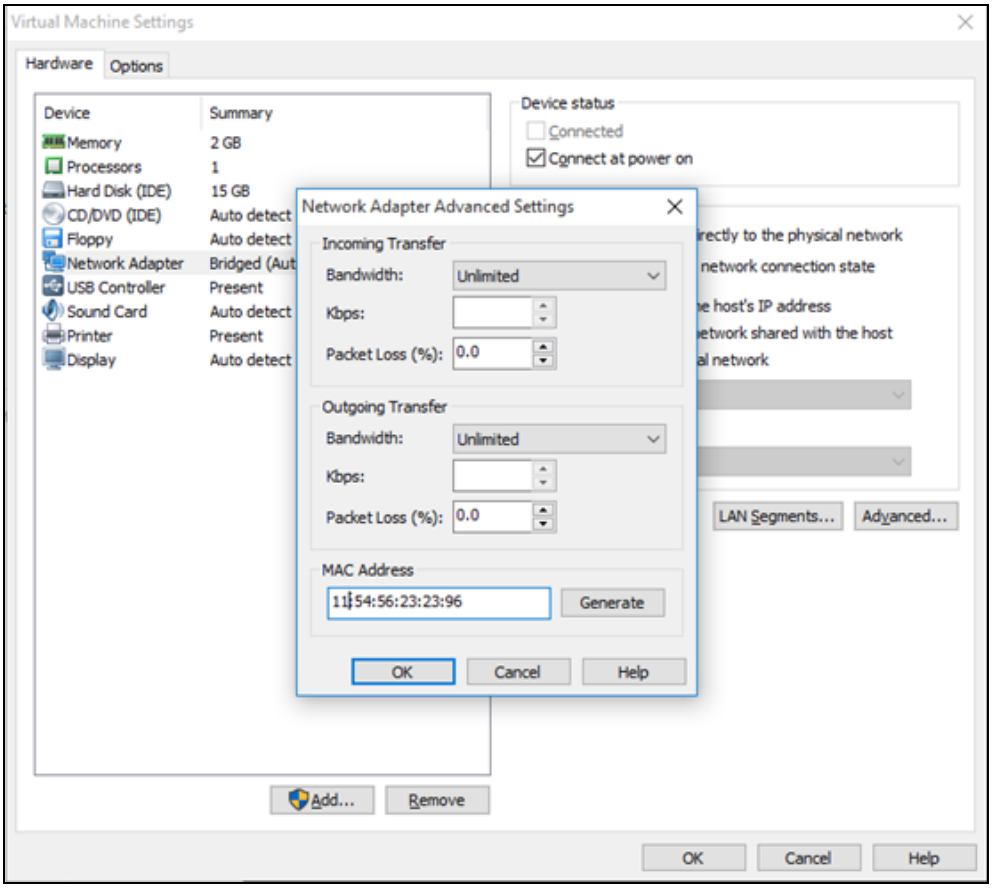
המימוש של פונקצייה זו תיראה כך:

```
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;
}
```

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, ומושוואת 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 הבוליאנים, כמובן שכדאי ואף מומלץ להגדיר כתובת MAC רנדומלית שלא תואמת את אחד מרשימת כתובת ה-MAC אשר מפרסמות חברות מוצרי הוירטואזליזציה השונים, כאשר ניתן לעשות זאת כאן בהגדרות ה-Network Adapter:



רק לא לשכוח לא ללחוץ על הכפתור Generate אלא לכתוב באופן יידני 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 וירטואלית

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 באנומרציה של פרוססים במטרה לגלות פרוססים ידועים על מנת לזהות את המוצר איתם אנו מבצעים את החקירה שלנו על הפוגען.

בפונקציות Windows API קיימות 2 פונקציות שאיתן נוכל לבצע טכניקה זו, ולקחת את רשימת הפורססים 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 הפורססים שרצים בכל מוצר וירטואליזציה שקיים כיום בשוק.

טכניקת זיהוי הDLL-ים הנטענים לזיכרון

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

אנו נוכל לזהות את ה-DLL-ים 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 אצלינו.

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

```
Select C:\Windows\System32\cmd.exe
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 {222A245B-E637-4AE9-A93F-A59CA119A75E}

Base               Size      Path
0x00000000a5360000  0x19000   C:\Windows\System32\dbghelp.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 DLL-ים ידועים של מוצרי וירטואליזציה שונים שאיתם נוכל לבצע טכניקה זו:

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

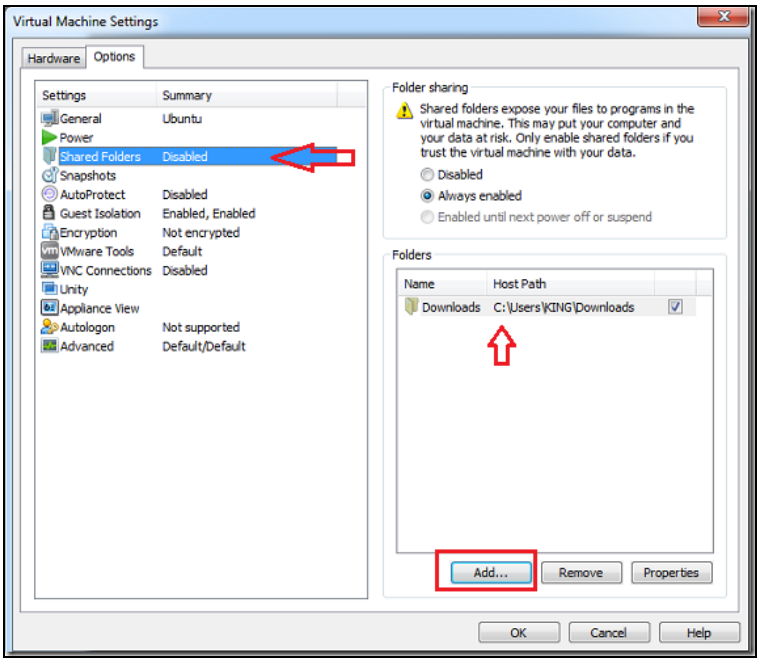
להלן קוד אשר מבצע בדיקה זו:

```
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 Virual 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 הביצועים 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.



ככותבי וירוסים, נוכל לזהות ולחפש אחר התיקייה 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 pnsize = 0x1000;
    LPWSTR provider = (LPWSTR)LocalAlloc(LMEM_ZEROINIT, pnsize);
    BOOL status = FALSE;

    int retv = WNetGetProviderName(WNNC_NET_RDR2SAMPLE, provider, &pnsize);
    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 מעניינית 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 נוסיף 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.

```
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 ספציפים לפני דגמים מסויימים, ולא האם הם רצים על מכונות 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 תיהיה 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+RDTSC1_EAX], eax
text:0042C51A                 mov     [ebp+RDTSC1_EDX], edx
text:0042C51D                 push    2000h           ; dwMilliseconds
text:0042C522                 call    ds:Sleep(x)
text:0042C528                 rdtsc
text:0042C52A                 sub     edx, [ebp+RDTSC1_EDX]
text:0042C52D                 cmp     edx, 0
text:0042C530                 jg      short return_sucess ; Return Sucess ,set eax = 1
text:0042C532                 sub     eax, [ebp+RDTSC1_EAX]
text:0042C535                 cmp     eax, 2000h
text:0042C53A                 jge     short return_sucess ; Return Sucess ,set eax = 1
text:0042C53C                 mov     eax, 0          ; return fail ,set eax = 0
text:0042C541                 retn
text:0042C542 ; ---------------------------------------------------------------------------
text:0042C542
text:0042C542 return_sucess:                          ; CODE XREF: _detect_sleep_patch+60↑j
text:0042C542                                         ; _detect_sleep_patch+6A↑j
text:0042C542                 mov     eax, 1          ; Return Sucess ,set eax = 1
text:0042C547                 retn
```

כיצד היא עובדת?

בפעם הראשונה היא עושה שימוש ב-RDTSC ושומרת את המספר, לאחר מכן ישנו שימוש בפונקציית Sleep למשך כ-2000H שניות, ולאחר מכך משתמשת עוד הפעם בפונקציית RDTSC (אשר מונה את מספר סיבובי השעון - Clock Rate מאז התחולו, 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-RDTSC 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-RDTSC 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.

טכניקה זו מתבססת על השוואת מיקום העכבר (X,Y) - לפני ואחרי האם הם נותרו באותו המקום 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 coardinate of two points
text:00401485                 jnz     short Cursor_position_changed
text:00401487                 mov     eax, [ebp+Point1.y]
text:0040148D                 cmp     eax, [ebp+Point2.y] ; comare y coardinate 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 ממקום למקום תוך כלום זמן (כי ה-30 שניות בעצם לא עבדו לנו..), וכן במידה ונמצא שיוויון 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 יידועים 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 שייתכן 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
```

```
00401AD4
00401AD4                   loc_401AD4:
00401AD4 8D 85 F0 FE FF FF 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.

כלומר, 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.

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        sldt    [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 וירטואלית 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 וירוסים חדשים המכילים רעיונות חדשים, אשר מהם ניתן לקחת את הרעיונות ולממש בוירוסים 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 לטיוטה של מאמר זה, תודה ל**ישי** ותודה ל**אור**.
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.
- ערוץ 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