Методы и средства антивирусной защиты

Лекция 8. Лекция 8. Распространение вирусов в Windows. Изучение PE-формата

Литература

- Климентьев К.Е. Компьютерные вирусы и антивирусы взгляд программиста
- **C++ немного практики**

http://www.interface.ru/home.asp?artId=26300

Это полноценный PE-файл, однако, большинство PE-файлов содержат большее количество элементов. Для простоты объяснения опущены.

версия 1.00 RU, 03.07.2012

Процесс загрузки

❶ Заголовки

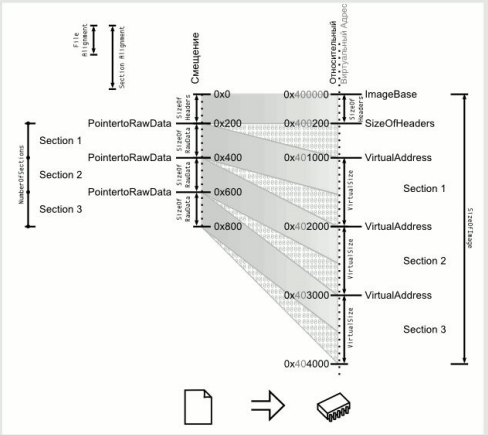
разбор *DOS заголовка*
разбор *PE заголовка*
(поле *e_lfanew* в *DOS заголовке* указывает на *PE заголовок*)
разбор *Опционального заголовка*
(он следует сразу за PE заголовком)

❷ Таблица секций

Разбор Таблицы секций
(она расположена по смещению: offset(*Optional Header*) + *SizeOfOptionalHeader*)
она содержит *NumberOfSections* элементов
она проверяется на корректность выравнивания:
FileAlignments и *SectionAlignments*

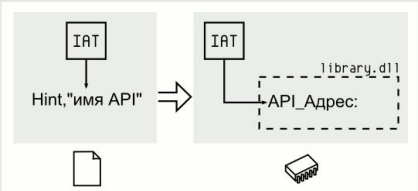
❸ Проецирование

файл проецируется в память в соответствии с:
ImageBase
SizeOfHeaders
Таблицей секций (Sections table)



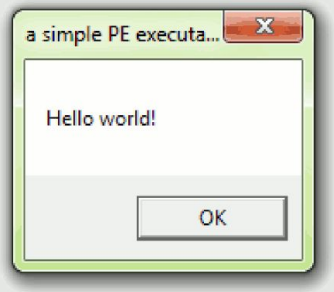
❹ Таблица импорта

Разбор директории данных (*DataDirectories*)
Она следует за *Опциональным заголовком*
Количество элементов *NumOfRVAAndSizes*
Директория импорта всегда имеет №2
Разбор Директории импорта
каждый дескриптор определяет имя DLL библиотеки
эта DLL загружается в память
разбор IAT и INT происходят одновременно
для каждой API функции в INT
записывается адрес этой функции в соответствующую запись в IAT



❺ Запуск

Исполнение начинается с точки входа (*EntryPoint*)
вызовы API функций в коде происходят через IAT



PE-формат

Рассмотри пример чтения структуры pe-файла на языке C++.

PE-формат

```
//библиотека ввода-вывода для вывода информации в консоль
#include <iostream>
//библиотека для работа с файлами
#include <fstream>
//вспомогательная библиотека для выравнивания, форматирования вывода и т.д.
#include <iomanip>
//конечно, нам потребуются структуры из Windows.h //но ничто, в общем-то, не мешает их перенести прямо в код и скомпилировать это под линукс :)
#include <Windows.h>
```

PE-формат

Далее - несколько макросов, которые предоставил Крис Касперски в своей статье про формат PE. Мы будем чаще всего использовать ALIGN_UP - макрос для выравнивания числа на заданную границу.

```
#define Is2power(x) (!(x & (x - 1)))
#define ALIGN_DOWN(x, align) (x & ~(align - 1))
#define ALIGN_UP(x, align) ((x & (align - 1)) ? ALIGN_DOWN(x, align) + align : x)
```

PE-формат

Итак, тело главной функции. В качестве единственного аргумента задается путь к исполняемому файлу для анализа.

```
int main(int argc) {
char* argv = "C:\\1.exe";
```

PE-формат

Теперь пришла пора открыть файл, имя которого нам передали через консоль.

```
//откроем файл формата PE в бинарном режиме
std::ifstream pefile;
pefile.open(argv[1], std::ios::in | std::ios::binary);
if(!pefile.is_open()) {
//если вдруг его открыть не удалось, то выведем ошибку и выйдем
std::cout << "Can't open file" << std::endl; return 0; }
```

PE-формат

```
//определим размер файла, он нам
   пригодится дальше
pefile.seekg(0, std::ios::end);
//для этого переведем файловый указатель
   чтения в самый конец файла, получим его
   позицию
std::streamoff  filesize = pefile.tellg();
 //это и будет размер файла в байтах
//затем вернем файловый указатель в
   начало файла
pefile.seekg(0);
```

PE-формат

В самом начале файла должна лежать структура **IMAGE_DOS_HEADER**. Считаем ее и немного проверим.

```
IMAGE_DOS_HEADER dos_header;
pefile.read((char*)(&dos_header), sizeof(IMAGE_DOS_HEADER));
if(pefile.bad() // pefile.eof()) { //если вдруг считать не удалось...
std::cout << "Unable to read IMAGE_DOS_HEADER" << std::endl; return 0; }
```

IMAGE_DOS_HEADER

Эта структура описывает заголовок **MS DOS** для программ **Windows**, которые имеют обозначение **PE - Portable Executable**. Данный заголовок сделан для того, чтобы программы созданные для **Windows** не запускались из **MS DOS**. И этого не происходит. Вместо запуска вы видите примерно вот что:

This program cannot be run in DOS mode

Эта программа не может быть запущена в DOS

Общая структура начала программы для **Windows** выглядит так:

MS-DOS MZ Сигнатура и заголовок

MS-DOS MS DOS программа

PE File PE Сигнатура и заголовок

IMAGE_DOS_HEADER

Структура **IMAGE_DOS_HEADER** как раз и описывает этот заголовок. Этот заголовок используется начиная с **MS DOS 2.0**. Он занимает 64 байта и вот его описание из **WinNT.h**:

```
typedef struct _IMAGE_DOS_HEADER {
USHORT e_magic; // Сигнатура заголовка
USHORT e_cblp; // количество байт на последней странице файла
USHORT e_cp; // количество страниц в файле
 USHORT e_crlc; // Relocations
USHORT e_cparhdr; // Размер заголовка в параграфах
USHORT e_minalloc; // Минимальные дополнительные параграфы
USHORT e_maxalloc; // Максимальные дополнительные параграфы
USHORT e_ss; // начальное относительное значение регистра SS
USHORT e_sp; // начальное значение регистра SP
USHORT e_csum; // контрольная сумма
USHORT e_ip; // начальное значение регистра IP
USHORT e_cs; // начальное относительное значение регистра CS
 USHORT e_lfarlc; // адрес в файле на таблицу переадресации
USHORT e_ovno; // количество оверлеев
USHORT e_res[4]; // Зарезервировано
USHORT e_oemid; // OEM идентифкатор
USHORT e_oeminfo; // OEM информация
USHORT e_res2[10]; // Зарезервировано
LONG e_lfanew; // адрес в файле нового .exe заголовка (PE) } IMAGE_DOS_HEADER,
    *PIMAGE_DOS_HEADER;
```

IMAGE_DOS_HEADER

```
//Первые два байта структуры должны быть MZ, но, так как в x86 у нас обратный порядок следования байтов,
//мы сравниваем эти байты со значением 'ZM'
if(dos_header.e_magic != 'ZM')
{ std::cout << "IMAGE_DOS_HEADER signature is incorrect" << std::endl; return 0; }
```

PE-формат

```
//Начало заголовка самого PE-файла (IMAGE_NT_HEADERS) должно быть
//выровнено на величину двойного слова (DWORD)
//убедимся, что это так
if((dos_header.e_lfanew % sizeof(DWORD)) != 0) {
//а иначе наш PE-файл некорректен
std::cout << "PE header is not DWORD-aligned" << std::endl; return 0; }
```

PE-формат

Теперь необходимо считать структуру IMAGE_NT_HEADERS.

Читать будем, соответственно, структуру IMAGE_NT_HEADERS32 (это 32-разрядная версия IMAGE_NT_HEADERS, они все определены в глубине Windows.h).

Сейчас я пропускаю множество необходимых проверок полей заголовка PE-файла (например, не проверяю выравнивания), потому что они сейчас не являются критичными.

PE-формат

Вот эта структура:

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD Signature;
  IMAGE_FILE_HEADER FileHeader;
  IMAGE_OPTIONAL_HEADER32
  OptionalHeader; }
  IMAGE_NT_HEADERS32,
  *PIMAGE_NT_HEADERS32;
```

PE-формат

IMAGE_FILE_HEADER.

Machine - архитектура, на которой может запускаться файл;

NumberOfSections - количество секций в PE-файле. Допустимое значение - от 1 до 0x60. Секция - это некая область памяти, обладающая определенными характеристиками и выделяемая системой при загрузке исполняемого файла;

SizeOfOptionalHeader - размер идущего за этой структурой опционального заголовка в байтах;

Characteristics - поле флагов характеристик PE-файла. Тут содержится информация о том, имеет ли файл экспортируемые функции, перемещаемые элементы, отладочную информацию и т.д.
Остальные поля при загрузке ни на что не влияют.

PE-формат

Magic - для 32-разрядных PE-файлов это поле должно содержать значение 0x10B, а для 64-разрядных - 0x20B.

AddressOfEntryPoint - адрес точки входа относительно базового адреса загрузки файла (ImageBase).

ImageBase - базовый адрес загрузки PE-файла. В памяти по этому адресу после загрузки будет располагаться вышеописанная структура **IMAGE_DOS_HEADER**.

FileAlignment и ***SectionAlignment*** - файловое и виртуальное выравнивание секций. В обязательном порядке должны быть выполнены следующие условия:

1. SectionAlignment >= 0x1000;
2. FileAlignment >= 0x200;
3. SectionAlignment >= FileAlignment.

SizeOfImage - это поле содержит размер в байтах загруженного образа PE-файла, который должен быть равен виртуальному адресу последней секции плюс ее виртуальный выровненный размер.

SizeOfHeaders - размер всех заголовков. Это поле говорит загрузчику, сколько байт считать от начала файла, чтобы получить всю необходимую информацию для загрузки файла.

CheckSum - контрольная сумма файла, которая проверяется загрузчиком только для самых важных системных файлов.

PE-формат

```
//Переходим на структуру
  IMAGE_NT_HEADERS и готовимся
  считать ее
  pefile.seekg(dos_header.e_lfanew);
  if(pefile.bad() || pefile.fail())
 { std::cout << "Cannot reach
  IMAGE_NT_HEADERS" << std::endl;
  return 0; }
```

PE-формат

```
//читать будем только часть структуры
  IMAGE_NT_HEADERS без дата
  директорий  они нам и не понадобятся
  сейчас
IMAGE_NT_HEADERS32 nt_headers;
  pefile.read(reinterpret_cast<char*>(&nt_hea
  ders), sizeof(IMAGE_NT_HEADERS32) -
  sizeof(IMAGE_DATA_DIRECTORY) * 16);
if(pefile.bad() || pefile.eof()) { std::cout <<
  "Error reading IMAGE_NT_HEADERS32"
  << std::endl; return 0; }
```

PE-формат

```
//Проверяем, что наш файл - PE
//сигнатура у него должна быть "PE\0\0"
//помним про обратный порядок байтов
   и проверяем…
if(nt_headers.Signature != 'EP')
{ std::cout << "Incorrect PE signature" <<
   std::endl; return 0; }
```

PE-формат

```
//Проверяем, что это PE32
if(nt_headers.OptionalHeader.Magic !=
   0x10B)
{ std::cout << "This PE is not PE32" <<
   std::endl; return 0; }
```

PE-формат

Теперь нам необходимо переместиться к таблице секций, которую мы и будем читать, чтобы получить информацию о секциях исполняемого файла. Можно было бы воспользоваться макросом IMAGE_FIRST_SECTION, но я сделал это руками, чтобы было понятнее:

PE-формат

```
//позиция в файле таблицы секций - это
    размер всех заголовков полностью
    (включая дос-стаб, если он есть и все
    дата директории, если они есть)
    DWORD first_section =
    dos_header.e_lfanew +
    nt_headers.FileHeader.SizeOfOptionalHe
    ader + sizeof(IMAGE_FILE_HEADER) +
    sizeof(DWORD) /* Signature */;
```

PE-формат

```
//переходим на первую секцию в
   таблице секций
   pefile.seekg(first_section);
if(pefile.bad() || pefile.fail())
{ std::cout << "Cannot reach section
   headers" << std::endl; return 0; }
```

PE-формат

Немного подготовим консоль для удобного вывода информации. Выставим выравнивание текста по левому краю и вывод чисел в 16-ричной системе счисления. std::showbase добавит перед 16-разрядными числами "0x" автоматически.

std::cout << std::hex << std::showbase << std::left;

PE-формат

Теперь начнем читать таблицу секций.
Количество секций лежит в IMAGE_NT_HEADERS.FileHeader.NumberOfSections.

```
for(int i = 0; i < nt_headers.FileHeader.NumberOfSections; i++)
{ //готовим заголовок секции
  IMAGE_SECTION_HEADER header;
```

PE-формат

```
typedef struct _IMAGE_SECTION_HEADER
{ BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
union { DWORD PhysicalAddress; DWORD VirtualSize; } Misc;
DWORD VirtualAddress;
DWORD SizeOfRawData;
DWORD PointerToRawData;
DWORD PointerToRelocations;
DWORD PointerToLinenumbers;
WORD  NumberOfRelocations;
WORD  NumberOfLinenumbers;
DWORD Characteristics; }
IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

```
Name                    db 8 dup(?) ; +00h - Наименование секции
VirtualSize             dd ?        ; +08h - Размер секции в памяти
VirtualAddress          dd ?        ; +0Ch - RVA секции в памяти
SizeOfRawData           dd ?        ; +10h - Размер секции на диске
PointerToRawData        dd ?        ; +14h - Файловое смещение секции
PointerToRelocations    dd ?        ; +18h - Не используется
PointerToLinenumbers    dd ?        ; +1Ch - Не используется
NumberOfRelocations     dw ?        ; +20h - Не используется
NumberOfLinenumbers     dw ?        ; +22h - Не используется
Characteristics         dd ?        ; +24h - Битовые флаги, характеризующие секцию
```

PE-формат

```
//и читаем его
pefile.read(reinterpret_cast<char*>(&head
   er), sizeof(IMAGE_SECTION_HEADER));
if(pefile.bad() || pefile.eof())
{ std::cout << "Error reading section
   header" << std::endl; return 0; }
```

PE-формат

Всевозможные проверки корректности таблицы секций. Разберем их. Во-первых, "сырой" размер данных и виртуальный размер секции не могут быть одновременно нулевыми

```
if(!header.SizeOfRawData &&
   !header.Misc.VirtualSize)
{ std::cout << "Virtual and Physical sizes of
   section can't be 0 at the same time" <<
   std::endl; return 0; }
```

PE-формат

```
//если размер инициализированных данных ("сырых") не равен нулю…
if(header.SizeOfRawData != 0) {
//Проверим, что инициализированные данные секции также не вылетают за пределы нашего PE-файла
if(ALIGN_DOWN(header.PointerToRawData, nt_headers.OptionalHeader.FileAlignment) + header.SizeOfRawData > filesize) { std::cout << "Incorrect section address or size" << std::endl; return 0; }
```

PE-формат

//в этой переменной мы сохраним выровненный виртуальный размер секции

DWORD virtual_size_aligned;

PE-формат

```
//если виртуальный размер секции был выставлен в ноль,
 if(header.Misc.VirtualSize == 0)
//то ее выровненный виртуальный размер равен ее реальному размеру инициализированных данных,
//выровненному на границу
SectionAlignment virtual_size_aligned = ALIGN_UP(header.SizeOfRawData, nt_headers.OptionalHeader.SectionAlignment);
else
//а иначе он равен ее виртуальному размеру, //выровненному на границу
SectionAlignment virtual_size_aligned = ALIGN_UP(header.Misc.VirtualSize, nt_headers.OptionalHeader.SectionAlignment);
```

PE-формат

```
//Проверим, что виртуальное пространство секции не вылетает за пределы виртуального пространства всего PE-файла
if(header.VirtualAddress + virtual_size_aligned > ALIGN_UP(nt_headers.OptionalHeader.SizeOfImage, nt_headers.OptionalHeader.SectionAlignment)) { std::cout << "Incorrect section address or size" << std::endl; return 0; } }
```

PE-формат

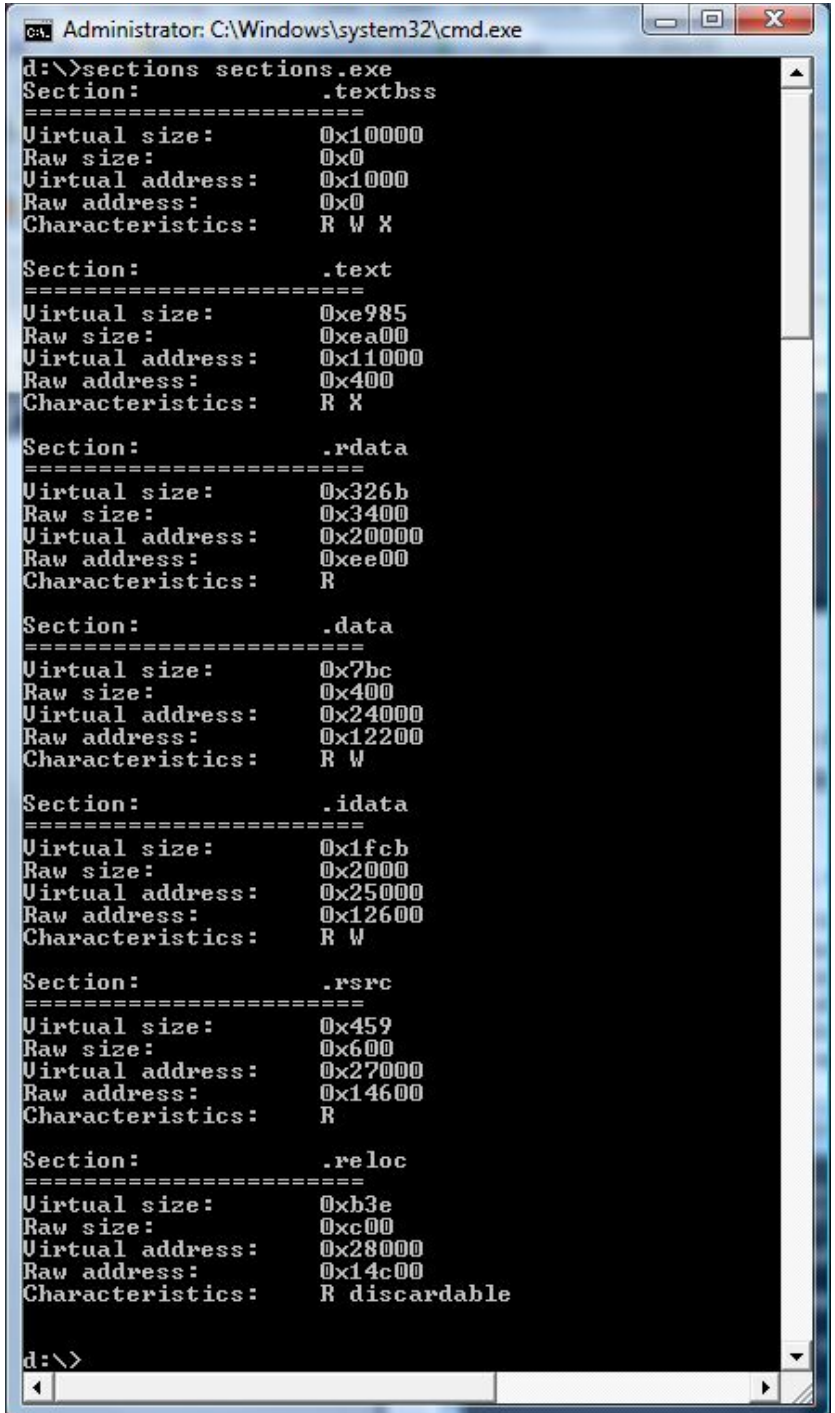
Пришло время вывести информацию о секции - раз уж она прошла все проверки :)

```
//имя секции может иметь размер до 8 символов char name[9] = {0}; memcpy(name, header.Name, 8); //выводим имя секции std::cout << std::setw(20) << "Section: " << name << std::endl << "======================" << std::endl;

//ее размеры, адреса

std::cout << std::setw(20) << "Virtual size:" << header.Misc.VirtualSize << std::endl;

std::cout << std::setw(20) << "Raw size:" << header.SizeOfRawData << std::endl;

std::cout << std::setw(20) << "Virtual address:" << header.VirtualAddress << std::endl;

std::cout << std::setw(20) << "Raw address:" << header.PointerToRawData << std::endl;
```

PE-формат

```
//и самые важные характеристики std::cout <<
    std::setw(20) << "Characteristics: ";
    if(header.Characteristics &
    IMAGE_SCN_MEM_READ) std::cout << "R ";
    if(header.Characteristics &
    IMAGE_SCN_MEM_WRITE) std::cout << "W ";
    if(header.Characteristics &
    IMAGE_SCN_MEM_EXECUTE) std::cout << "X
    "; if(header.Characteristics &
    IMAGE_SCN_MEM_DISCARDABLE) std::cout
    << "discardable "; if(header.Characteristics &
    IMAGE_SCN_MEM_SHARED) std::cout <<
    "shared";   std::cout << std::endl << std::endl; }
    return 0; }
```

PE-формат


Administrator: C:\Windows\system32\cmd.exe
d:\>sections sections.exe
Section: .textbss
Virtual size: 0x10000
Raw size: 0x0
Virtual address: 0x1000
Raw address: 0x0
Characteristics: R W X
Section: .text
Virtual size: 0xe985
Raw size: 0xea00
Virtual address: 0x11000
Raw address: 0x400
Characteristics: R X
Section: .rdata
Virtual size: 0x326b
Raw size: 0x3400
Virtual address: 0x20000
Raw address: 0xee00
Characteristics: R
Section: .data
Virtual size: 0x7bc
Raw size: 0x400
Virtual address: 0x24000
Raw address: 0x12200
Characteristics: R W
Section: .idata
Virtual size: 0x1fcb
Raw size: 0x2000
Virtual address: 0x25000
Raw address: 0x12600
Characteristics: R W
Section: .rsrc
Virtual size: 0x459
Raw size: 0x600
Virtual address: 0x27000
Raw address: 0x14600
Characteristics: R
Section: .reloc
Virtual size: 0xb3e
Raw size: 0xc00
Virtual address: 0x28000
Raw address: 0x14c00
Characteristics: R discardable
d:\>

PE-формат

Теперь код!