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

«Разбор трояна»

Выполнили:

,

_____
(подпись)

Проверил:

,

ФБИТ

_____
(отметка о выполнении)

_____
(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – комплексный анализ техник обфускации, скрытия и функциональности троянской программы. Работа направлена на изучение методов, применяемых современным вредоносным программным обеспечением для усложнения статического и динамического анализа, обхода систем защиты и выполнения вредоносной функциональности.

ИНСТРУМЕНТЫ АНАЛИЗА:

- IDA Pro 7.7 (Interactive Disassembler) - основной инструмент для статического анализа и дизассемблирования
- x64dbg - отладчик для динамического анализа и трассировки выполнения
- PEiD - инструмент для определения типа упаковщика и компилятора
- Process Monitor - инструмент для мониторинга файловых операций и реестра
- Wireshark - анализатор сетевого трафика для отслеживания сетевых подключений

1 РАСПАКОВКА ТРОЯНА

Первый этап анализа любой троянской программы - это определение наличия упаковщика, криптора или обфускатора. Современные трояны активно используют различные техники сокрытия кода для защиты от обнаружения и анализа.

1.1 Определение типа установщика

Для определения типа упаковщика был использован инструмент PEiD. При анализе файла было обнаружено следующее:

PEiD Результат: "Nothing found" / "Microsoft Visual C++"

Анализ структуры PE-файла показывает, что троян НЕ использует стандартный упаковщик типа UPX, ASPack или VMProtect. Код доступен для статического анализа в IDA Pro без необходимости распаковки. Однако троян использует техники обфускации на уровне кода для усложнения анализа.

Характерные признаки отсутствия упаковщика:

- Стандартные имена секций (.text, .data, .idata)
- Код доступен для статического анализа в IDA Pro
- Entry Point указывает на реальный код, а не на распаковщик
- Размер секций соответствует размеру данных

Анализ структуры PE-файла:

- Секция .text: Virtual size = 0x00001792 (6034 байт), Size in file = 0x00001800 (6144 байт)
- Секция .data: Virtual size = 0x0000B18C (45452 байт), Size in file = 0x00000200 (512 байт)
- Секция .idata: содержит стандартные импорты

1.2 Анализ точки входа

Entry Point трояна находится по адресу 0x00401410 (функция start). При анализе кода в IDA Pro сразу видна структура программы без признаков упаковщика.

Адрес Entry Point: 0x00401410

Код точки входа из lst файла:

```
.text:00401410 start           proc near
.text:00401410                  push    ebp
.text:00401411                  mov     ebp, esp
.text:00401413                  sub     esp, 280h           ; Выделить место
в стеке (640 байт)
```

```
.text:00401419                  mov     [ebp+var_21C], 0
.text:00401423                  mov     [ebp+var_238], 0C8h  ; 200 в
десятичной
.text:0040142D                  mov     [ebp+var_224], 4
.text:00401437                  mov     [ebp+var_8], 0
.text:0040143E                  mov     [ebp+var_22C], 0
.text:00401448                  xor     ecx, ecx
.text:0040144A                  mov     [ebp+var_248], 0
.text:00401454                  push    offset ModuleName   ; "kernel32.dll"
.text:00401459                  call    ds:GetModuleHandleA ; Получить
базовый адрес kernel32.dll
```

Код точки входа сразу начинает выполнение основной логики трояна, а не кода распаковки. Это подтверждает отсутствие стандартного упаковщика.

1.3 Обфускация на уровне кода

Хотя троян не упакован, он использует обфускацию на уровне отдельных функций.

Пример - функция Memcpy с обфускацией XOR:

Адрес: 0x00401350

Код из lst файла:

```
.text:00401350 Memcpy          proc near
.text:00401350                  push    ebp
.text:00401351                  mov     ebp, esp
.text:00401353                  sub     esp, 0Ch
.text:00401356                  mov     [ebp+var_4], 23E4CCh  ; Константа для
обфускации
.text:0040135D                  mov     counter, 0
.text:00401367                  jmp     short loc_401376
.text:00401369 loc_401369:
.text:00401369                  mov     eax, counter
.text:0040136E                  add     eax, 1
.text:00401371                  mov     counter, eax
.text:00401376 loc_401376:
.text:00401376                  mov     ecx, counter
.text:0040137C                  cmp     ecx, [ebp+arg_8]     ; Сравнить с
размером
.text:0040137F                  jge     short loc_4013C7     ; Если больше
или равно - выход
.text:00401381                  mov     edx, [ebp+arg_4]     ; Адрес
источника
.text:00401384                  add     edx, counter
.text:0040138A                  movzx   eax, byte ptr [edx]  ; Загрузить байт
.text:0040138D                  mov     [ebp+var_C], eax
.text:00401390                  mov     [ebp+var_8], 23E4CCh  ; Константа
обфускации
.text:00401397                  mov     ecx, [ebp+var_C]
.text:0040139A                  add     ecx, [ebp+var_8]     ; Добавить
константу
.text:0040139D                  mov     edx, [ebp+arg_0]     ; Адрес
назначения
.text:004013A0                  add     edx, counter
.text:004013A6                  mov     [edx], cl            ; Сохранить байт
.text:004013A8                  mov     eax, [ebp+arg_0]
.text:004013AB                  add     eax, counter
```

```
.text:004013B1                  movzx   ecx, byte ptr [eax]
.text:004013B4                  sub     ecx, 23E4CCh          ; Вычесть
константу (обратная операция)
.text:004013BA                  mov     edx, [ebp+arg_0]
.text:004013BD                  add     edx, counter
.text:004013C3                  mov     [edx], cl             ; Сохранить
результат
.text:004013C5                  jmp     short loc_401369      ; Повторить
.text:004013C7 loc_4013C7:
.text:004013C7                  mov     esp, ebp
.text:004013C9                  pop     ebp
.text:004013CA                  retn
```

Функция выполняет XOR-подобные операции с константой 0x23E4CC для обфускации копирования данных. Это делает анализ кода более сложным, но не скрывает его полностью.

1.4 Ручное восстановление IAT

Троян использует ручное восстановление Import Address Table через функции FindKernel32dll и FindGetProcAddress, что позволяет скрыть список импортируемых функций от статического анализа.

Функция FindKernel32dll (адрес 0x00401160):

```
.text:00401160 FindKernel32dll proc near
.text:00401160                  push    ebp
.text:00401161                  mov     ebp, esp
.text:00401163                  push    ecx
.text:00401164                  push    ebx
.text:00401165                  push    esi
.text:00401166                  push    edi
.text:00401167                  xor     ecx, ecx
.text:0040116B                  mov     esi, fs:[ecx+30h]     ; Получить PEB
(Process Environment Block)
.text:0040116F                  mov     edx, edx
.text:00401171                  mov     esi, [esi+0Ch]        ; PEB_LDR_DATA
.text:00401174                  mov     edx, edx
.text:00401176                  mov     esi, [esi+1Ch]        ;
InInitializationOrderModuleList
.text:00401179 loc_401179:
.text:00401179                  mov     eax, [esi+8]          ; Базовый адрес
DLL
.text:0040117C                  mov     [ebp+DllBase], eax
.text:00401181                  mov     edi, [esi+20h]        ; BaseDllName
(имя DLL)
.text:00401184                  mov     esi, [esi]            ; Следующий
элемент списка
.text:00401188                  mov     al, 'k'               ; Проверка
первого символа
.text:0040118A                  cmp     [edi], al             ; Сравнить с 'k'
(kernel32.dll)
.text:0040118C                  jz      short loc_401196      ; Если совпало -
нашли
.text:0040118E                  mov     al, 'K'               ; Проверка
заглавной буквы
.text:00401190                  cmp     [edi], al
.text:00401192                  jz      short loc_401196
```

```
.text:00401194                  jmp     short loc_401179     ; Продолжить
поиск
.text:00401196 loc_401196:
.text:00401196                  mov     eax, [ebp+DllBase]   ; Вернуть
базовый адрес
.text:00401199                  pop     edi
.text:0040119A                  pop     esi
.text:0040119B                  pop     ebx
.text:0040119C                  mov     esp, ebp
.text:0040119E                  pop     ebp
.text:0040119F                  retn
```

Проведенный анализ показывает, что троян НЕ использует стандартный упаковщик. Код доступен для статического анализа в IDA Pro без необходимости распаковки. Однако троян использует несколько техник усложнения анализа:

1. Обфускация на уровне кода - функция Memcpy использует XOR-операции с константой

2. Ручное восстановление IAT - скрывает список импортируемых функций

3. Отсутствие стандартной таблицы импорта - затрудняет статический анализ

Отсутствие упаковщика упрощает анализ, но обфускация на уровне кода создает дополнительные трудности при реверс-инжиниринге.

2 СКРЫТИЕ ОТ ОБНАРУЖЕНИЯ

Троян использует несколько техник для скрытия от антивирусных систем и анализаторов.

2.1 Ручное восстановление IAT

Троян не использует стандартную таблицу импорта. Вместо этого все функции загружаются динамически через GetProcAddress после ручного поиска kernel32.dll.

Процесс восстановления IAT:

1. Поиск kernel32.dll через PEB (FindKernel32dll)

2. Поиск GetProcAddress в Export Directory kernel32.dll (FindGetProcAddress)

3. Динамическая загрузка всех необходимых функций

Это делает статический анализ импортов невозможным - антивирусы не могут просто посмотреть список импортируемых функций.

2.2 Антиотладочные техники

Троян использует несколько техник для обнаружения и обхода отладчиков:

ТЕХНИКА 1: Проверка через IsDebuggerPresent

Адрес: 0x00402068

Код из lst файла:

```
.text:00402068                  call    ds:IsDebuggerPresent
.text:0040206E                  mov     dword_40DEA0, eax     ; Сохранить
результат
```

Функция IsDebuggerPresent проверяет флаг PEB.BeingDebugged. Если отладчик обнаружен, троян может изменить свое поведение или завершить выполнение.

ТЕХНИКА 2: Обработка исключений

Троян использует структурированную обработку исключений (SEH) для усложнения отладки. При возникновении исключения троян может проверять контекст и обнаруживать отладчик.

2.3 Обфускация кода

Как было показано в разделе 1.3, троян использует обфускацию на уровне функций.

Функция Memcpy выполняет операции с константой 0x23E4CC, что усложняет понимание логики копирования данных.

2.4 Работа с реестром

Троян открывает ключ реестра для проверки окружения:

Адрес: 0x004014CC

Код из lst файла:

```
.text:004014B6                  push    offset phkResult    ; Указатель на
handle ключа
.text:004014BB                  push    20019h              ; samDesired =
KEY_READ | KEY_WRITE
.text:004014C0                  push    0                   ; ulOptions = 0
.text:004014C2                  push    offset SubKey       ; "TypeLib"
.text:004014C7                  push    80000000h           ; hKey =
HKEY_CURRENT_USER
.text:004014CC                  call    ds:RegOpenKeyExW    ; Открыть ключ
реестра
.text:004014D2                  test    eax, eax            ; Проверить
результат
.text:004014D4                  jz      short loc_4014E0    ; Если успешно -
продолжить
```

Это может использоваться для:

- Проверки наличия определенных компонентов системы

- Сохранения настроек или данных

- Проверки окружения выполнения

2.5 Отсутсвие сложных техник скрытия

При детальном анализе были обнаружены следующие отсутствующие техники:

1. Нет проверки на виртуальные машины (VM detection)

2. Нет проверки времени выполнения (timing checks)

3. Нет проверки количества процессоров

4. Нет сложных техник обхода песочниц

Это указывает на относительно простой уровень обфускации трояна.

Троян использует базовые техники скрытия:

1. Ручное восстановление IAT для скрытия импортов

2. Проверка на отладчик через IsDebuggerPresent

3. Обфускация на уровне кода

4. Работа с реестром для проверки окружения

Уровень обфускации средний - используются стандартные техники, которые могут быть обойдены опытным аналитиком, но эффективны против автоматизированных систем анализа.

3 ПОЛЕЗНАЯ НАГРУЗКА ТРОЯНА

Анализ кода трояна показывает его основную функциональность.

3.1 Работа с файлов NET.EXE

Троян читает файл %SystemRoot%\system32\net.exe:

Адрес: 0x0040151F

Код из lst файла:

```
.text:004014E0                  push    104h                ; Размер буфера
(260 байт)
.text:004014E5                  lea     ecx, [ebp+var_218]  ; Адрес буфера
.text:004014EB                  push    ecx
.text:004014EC                  push    offset aSystemroot  ; "SystemRoot"
.text:004014F1                  call    [ebp+GetEnvironmentVariableW] ;
Получить %SystemRoot%
.text:004014F7                  push    offset aSystem32Net_ex ;
"\\system32\\net.exe"
.text:004014FC                  lea     edx, [ebp+var_218]
.text:00401502                  push    edx
.text:00401503                  call    [ebp+lstrcatW]      ; Объединить
пути
.text:00401509                  push    0
.text:0040150B                  push    80h                  ;
FILE_ATTRIBUTE_NORMAL
.text:00401510                  push    3                    ; OPEN_EXISTING
.text:00401512                  push    0
.text:00401514                  push    3                    ;
FILE_SHARE_READ | FILE_SHARE_WRITE
.text:00401516                  push    1                    ; GENERIC_READ
.text:00401518                  lea     eax, [ebp+var_218]  ; Путь к файлу
.text:0040151E                  push    eax
.text:0040151F                  call    [ebp+CreateFileW]   ; Открыть файл
net.exe
.text:00401525                  mov     [ebp+var_234], eax  ; Сохранить
handle
```

Троян открывает системный файл net.exe. Это может использоваться для:

- Чтения оригинального файла перед его заменой (trojan horse)

- Анализа структуры системных файлов

- Создания копии для маскировки

3.2 Копирование данных

После чтения данных троян копирует их с использованием обфусцированной функции Memcpy (см. раздел 1.3). Данные обрабатываются через циклы с операциями XOR.

3.3 Выделение памяти и расшифровка

Троян выделяет память через VirtualAllocEx для хранения обработанных данных:

```
.text:00401589                  push    offset aVirtualallocex ;
"VirtualAllocEx"
.text:0040158E                  mov     eax, [ebp+k32_image_base]
.text:00401594                  push    eax
.text:00401595                  call    [ebp+GetProcAddress]
.text:0040159B                  mov     VirtualAllocEx, eax
.text:004015A0                  mov     ecx, [ebp+var_24C]
.text:004015A6                  push    ecx                     ; Размер
.text:004015A7                  call    Alloc                   ; Выделить
память
```

После выделения памяти троян выполняет расшифровку данных через циклы с XOR-операциями (см. код функции start, строки 790-812 в lst файле).

3.4 Возможное использование GDI32.DLL

Троян импортирует множество функций из GDI32.dll:

- BitBlt
- CreateCompatibleDC
- CreateDIBitmap
- TextOutW
- SetTextColor
- И другие графические функции

Это может указывать на:

- Создание графических элементов (окон, диалогов)
- Модификацию графического интерфейса
- Создание fake-интерфейса для фишинга

3.5 Возможное использование USER32.DLL

Троян импортирует функции из USER32.dll:

- CreateDialogParamW
- ShowWindow
- UpdateWindow
- MessageBox (через wsprintfW)
- Clipboard функции (OpenClipboard, GetClipboardData)

Это может указывать на:

- Создание диалоговых окон
- Перехват данных из буфера обмена
- Взаимодействие с пользователем через GUI

3.6 Проверка условий выполнения

Троян проверяет результат открытия файла и работы с реестром перед выполнением основных функций:

```
.text:004014D2                  test    eax, eax            ; Проверить
результат RegOpenKeyExW
.text:004014D4                  jz      short loc_4014E0    ; Если успешно -
продолжить
.text:004014D6                  mov     eax, 37h            ; Иначе код
ошибки
.text:004014DB                  jmp     loc_4018E7          ; Выйти

.text:0040152B                  cmp     [ebp+var_234], 0FFFFFFFFh ; Проверить
handle файла
.text:00401532                  jz      short loc_40153D    ; Если ошибка
.text:00401534                  cmp     [ebp+var_234], 0
.text:0040153B                  jnz     short loc_40153F    ; Если успешно -
продолжить
.text:0040153D loc_40153D:
.text:0040153D                  int     6                   ; Неопределенная
инструкция (crash)
```

На основе анализа кода можно сделать следующие выводы о payload трояна:

1. Троян работает с системным файлом net.exe, что может указывать на trojan horse функциональность - замена или модификация системных файлов.

2. Использование функций GDI32.dll и USER32.dll указывает на графический интерфейс, возможно для создания fake-диалогов или фишинговых окон.

3. Перехват данных из буфера обмена может использоваться для кражи паролей и другой конфиденциальной информации.

4. Обфускация данных через XOR указывает на необходимость скрытия обрабатываемой информации.

5. Проверка условий выполнения позволяет трояну адаптировать свое поведение в зависимости от окружения.

Точная функциональность payload требует дополнительного динамического анализа, так как часть кода может быть зашифрована и расшифровываться только во время выполнения.

ЗАКЛЮЧЕНИЕ

Проведенный анализ троянской программы позволил получить представление о методах, используемых для усложнения анализа и обхода систем защиты.

ОСНОВНЫЕ ТЕХНИКИ:

1. РАСПАКОВКА:
   - Троян НЕ использует стандартный упаковщик
   - Код доступен для статического анализа
   - Обфускация на уровне отдельных функций
2. СКРЫТИЕ ОТ ОБНАРУЖЕНИЯ:
   - Ручное восстановление IAT
   - Проверка на отладчик (IsDebuggerPresent)
   - Обфускация кода через XOR-операции
   - Работа с реестром
3. ПОЛЕЗНАЯ НАГРУЗКА:
   - Работа с системным файлом net.exe
   - Использование графических функций (GDI32.dll, USER32.dll)
   - Возможный перехват данных из буфера обмена
   - Обфускация данных

Троян использует средний уровень обфускации. Отсутствие упаковщика упрощает анализ, но использование ручного восстановления IAT и обфускации кода создает дополнительные трудности.