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

«Исследование трояна»

Выполнили:

,

(подпись)

Проверил:

,

ФБИТ

(отметка о выполнении)

(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – в рамках данной лабораторной работы проводится исследование методов обфускации и скрытия, применяемых в троянском программном обеспечении. Особое внимание уделяется изучению механизмов усложнения статического и динамического анализа, способам обхода систем обнаружения и анализу функциональных возможностей вредоносной программы.

ИНСТРУМЕНТЫ АНАЛИЗА:

- IDA Pro 7.7 (Interactive Disassembler) - для статического анализа кода
- x64dbg - для динамического анализа и трассировки
- PEiD - для определения типа упаковщика
- Process Monitor - для мониторинга операций с файлами и реестром
- Wireshark - для анализа сетевого трафика

1 ИССЛЕДОВАНИЕ УПАКОВКИ И ОБФУСКАЦИИ

Начальным этапом исследования является проверка наличия упаковщика или криптора. Данный этап критически важен, так как большинство современных троянов используют различные способы сокрытия кода для защиты от анализа.

1.1 Проверка наличия упаковщика

Исследование началось с проверки файла с помощью инструмента PEiD. Результаты анализа показали следующее:

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

Дальнейший анализ структуры PE-файла подтвердил, что исследуемый образец НЕ использует известные упаковщики, такие как UPX, ASPack, VMProtect или другие распространенные крипторы. Исходный код доступен для статического анализа непосредственно в среде IDA Pro, что существенно упрощает процесс исследования.

Характеристики, указывающие на отсутствие упаковщика:

- Присутствуют стандартные имена секций (.text, .data, .idata)
- Весь код может быть проанализирован статически
- Точка входа ведет напрямую к функциональному коду программы
- Размеры секций соответствуют реальному объему данных

Параметры секций 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: стандартная структура импортов

1.2 Изучение точки входа

Точка входа программы располагается по адресу 0x00401410 и соответствует функции start. При первичном анализе в IDA Pro становится очевидным, что данный адрес не содержит кода распаковки, а сразу переходит к выполнению основной логики программы.

Адрес точки входа: 0x00401410

Исходный код функции start (из файла lst):

```
.text:00401410 start           proc near
```

```
.text:00401410                  push    ebp
.text:00401411                  mov     ebp, esp
.text:00401413                  sub     esp, 280h
.text:00401419                  mov     [ebp+var_21C], 0
.text:00401423                  mov     [ebp+var_238], 0C8h
.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
```

Анализ точки входа демонстрирует, что программа не использует стандартный механизм распаковки, а сразу приступает к выполнению основных функций, что подтверждает отсутствие упаковщика.

1.3 Исследование методов обфускации

Несмотря на отсутствие упаковщика, в программе обнаружены техники обфускации, применяемые на уровне отдельных функций. Ярким примером служит функция копирования памяти, реализация которой усложнена за счет дополнительных операций.

Функция: Memcpy

Расположение: 0x00401350

Код функции 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
.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]
.text:0040137F                  jge     short loc_4013C7
.text:00401381                  mov     edx, [ebp+arg_4]
.text:00401384                  add     edx, counter
.text:0040138A                  movzx   eax, byte ptr [edx]
.text:0040138D                  mov     [ebp+var_C], eax
.text:00401390                  mov     [ebp+var_8], 23E4CCh
.text:00401397                  mov     ecx, [ebp+var_C]
.text:0040139A                  add     ecx, [ebp+var_8]
.text:0040139D                  mov     edx, [ebp+arg_0]
.text:004013A0                  add     edx, counter
.text:004013A6                  mov     [edx], cl
.text:004013A8                  mov     eax, [ebp+arg_0]
.text:004013AB                  add     eax, counter
.text:004013B1                  movzx   ecx, byte ptr [eax]
.text:004013B4                  sub     ecx, 23E4CCh
```

```
.text:004013BA                  mov     edx, [ebp+arg_0]
.text:004013BD                  add     edx, counter
.text:004013C3                  mov     [edx], cl
.text:004013C5                  jmp     short loc_401369
.text:004013C7 loc_4013C7:
.text:004013C7                  mov     esp, ebp
.text:004013C9                  pop     ebp
.text:004013CA                  retn
```

Функция выполняет операции с константой 0x23E4CC, которые в конечном итоге компенсируют друг друга, оставляя результат копирования неизменным. Это создает дополнительную сложность для автоматизированных средств анализа, хотя не скрывает функциональность полностью.

1.4 Анализ восстановления таблицы импорта

В программе реализован механизм ручного восстановления таблицы адресов импорта (IAT) через специализированные функции FindKernel32dll и FindGetProcAddress.

Это позволяет скрыть список используемых функций от статического анализа.

Функция FindKernel32dll:

Адрес: 0x00401160

Реализация:

```
.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]    ; Получение PEB
.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
.text:00401184                  mov     esi, [esi]           ; Следующий
элемент
.text:00401188                  mov     al, 'k'
.text:0040118A                  cmp     [edi], al
.text:0040118C                  jz      short loc_401196
.text:0040118E                  mov     al, 'K'
.text:00401190                  cmp     [edi], al
.text:00401192                  jz      short loc_401196
.text:00401194                  jmp     short loc_401179
.text:00401196 loc_401196:
.text:00401196                  mov     eax, [ebp+DllBase]
.text:00401199                  pop     edi
```

```
.text:0040119A                  pop     esi
.text:0040119B                  pop     ebx
.text:0040119C                  mov     esp, ebp
.text:0040119E                  pop     ebp
.text:0040119F                  retn
```

Механизм работы: Функция обходит список загруженных модулей через структуру PEB (Process Environment Block) и находит kernel32.dll путем сравнения первого символа имени модуля. После этого получается базовый адрес библиотеки, которы используется для дальнейшего поиска экспортируемых функций.

Подтверждающие данные:

1. PEiD не обнаружил упаковщик
2. В IDA Pro присутствуют стандартные секции
3. Код поддается статическому анализу
4. Точка входа (0x00401410) ведет к функциональному коду
5. Обнаружены функции восстановления IAT (FindKernel32dll, FindGetProcAddress)
6. Применяется обфускация на уровне функций

Анализ подтвердил, что программа не использует стандартный упаковщик, что облегчает процесс исследования. Однако обнаружены следующие техники усложнения анализа:

1. Обфускация на уровне кода - функция Memcpy использует операции с константой
2. Ручное восстановление IAT - скрывает зависимости от статического анализа
3. Отсутствие стандартной таблицы импорта - усложняет определение используемых функций

Несмотря на отсутствие упаковщика, применение обфускации на уровне кода создает определенные трудности при реверс-инжиниринге.

2 МЕХАНИЗМЫ СКРЫТИЯ ОТ СРЕДСТВ ОБНАРУЖЕНИЯ

В ходе исследования выявлено несколько техник, направленных на скрытие от антивирусных систем и средств анализа.

2.1 Механизм скрытия импортов

Программа не использует стандартную таблицу импорта PE-файла. Вместо этого применяется механизм динамической загрузки всех необходимых функций через GetProcAddress после ручного поиска kernel32.dll в памяти процесса.

Последовательность восстановления IAT:

1. Локализация kernel32.dll путем обхода PEB (FindKernel32dll)

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

3. Динамическая загрузка всех требуемых функций по строковым именам

Данный подход делает невозможным статический анализ списка импортируемых функций - средства обнаружения не могут просто просмотреть таблицу импорта для определения зависимостей программы.

2.2 Методы обнаружения отладчиков

При исследовании кода выявлено применение антиотладочных техник.

Метод 1: Использование функции IsDebuggerPresent

Расположение: 0x00402068

Реализация:

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

Принцип действия: Функция IsDebuggerPresent проверяет значение флага PEB.BeingDebugged, который устанавливается операционной системой при присоединении отладчика к процессу. В случае обнаружения отладчика программа может изменить свое поведение или прекратить выполнение.

Метод 2: Обработка исключений

В программе применяется механизм структурированной обработки исключений (SEH), который может использоваться для обнаружения отладчиков. Различные отладчики обрабатывают исключения по-разному, что позволяет их обнаружить путем анализа контекста выполнения.

2.3 Применение обфускации кода

Как было продемонстрировано в разделе 1.3, в программе используется обфускация на уровне функций. Функция Memcpy выполняет операции с константой 0x23E4CC, что усложняет понимание логики копирования данных при первичном анализе.

2.4 Взаимодействие с системным реестром

Обнаружен доступ к системному реестру Windows с целью проверки окружения выполнения.

Операция: Открытие ключа реестра

Ключ: HKEY_CURRENT_USER\TypeLib

Реализация (адрес 0x004014CC):

```
.text:004014B6                  push    offset phkResult
.text:004014BB                  push    20019h              ; KEY_READ |
KEY_WRITE
.text:004014C0                  push    0
.text:004014C2                  push    offset SubKey       ; "TypeLib"
.text:004014C7                  push    80000000h           ;
HKEY_CURRENT_USER
.text:004014CC                  call    ds:RegOpenKeyExW
.text:004014D2                  test    eax, eax
.text:004014D4                  jz      short loc_4014E0
```

Назначение данного действия может включать:

- Проверку наличия определенных системных компонентов
- Сохранение настроек или данных конфигурации
- Получение информации о среде выполнения

2.5 Отсутсвие продвинутых техник

Детальный анализ показал, что в программе не используются следующие продвинутые техники:

- Детекция виртуальных машин (VM detection)
- Проверка времени выполнения (timing checks)
- Проверка количества процессоров
- Сложные методы обхода песочниц (sandboxes)

Это свидетельствует о среднем уровне сложности применяемых техник обфускации.

Исследование выявило применение базовых техник скрытия:

1. Ручное восстановление IAT с целью скрытия импортов
2. Проверка наличия отладчика через IsDebuggerPresent

3. Обфускация кода на уровне функций

4. Работа с реестром для проверки окружения

Уровень обфускации можно оценить как средний - используются стандартные техники, которые могут быть обойдены опытным исследователем, но эффективны против автоматизированных средств анализа.

3 АНАЛИЗ ФУНКЦИОНАЛЬНОСТИ И ПОЛЕЗНОЙ НАРУЗКИ

Исследование функциональных возможностей программы выявило следующие особенности.

3.1 Взаимодействие с ситемными файлами

Программа выполняет операции с системной утилитой Windows.

Целевой файл: %SystemRoot%\system32\net.exe

Процесс работы с файлом (адрес 0x0040151F):

```
.text:004014E0                  push    104h
.text:004014E5                  lea     ecx, [ebp+var_218]
.text:004014EB                  push    ecx
.text:004014EC                  push    offset aSystemroot   ; "SystemRoot"
.text:004014F1                  call    [ebp+GetEnvironmentVariableW]
.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]
.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]
.text:0040151E                  push    eax
.text:0040151F                  call    [ebp+CreateFileW]
.text:00401525                  mov     [ebp+var_234], eax
```

Интерпретация: Программа открывает системный файл net.exe с правами на чтение.

Это может указывать на следующие сценарии использования:

- Чтение оригинального файла перед его возможной заменой (техника trojan horse)

- Анализ структуры системных файлов

- Создание резервной копии для маскировки

3.2 Обработка данных с применением обфускации

После открытия файла программа копирует данные, используя обфусцированную функцию Memcpy (описание которой представлено в разделе 1.3). Данные обрабатываются посредством циклов с применением XOR-операций.

3.3 Выделение памяти и процесс расшифровки

Для хранения обрабатываемых данных программа выделяет память через функцию

VirtualAllocEx.

Реализация:

```
.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
.text:004015A7                  call    Alloc
```

После выделения памяти выполняется процесс расшифровки данных через циклы с XOR-операциями (код функции start, строки 790-812 в lst файле).

3.4 Использование графических библиотек

Обнаружен импорт множества функций из библиотеки GDI32.dll:

- BitBlt
- CreateCompatibleDC
- CreateDIBitmap
- TextOutW
- SetTextColor
- И другие функции работы с графикой

Это может указывать на следующие возможности:

- Создание графических элементов интерфейса (окон, диалогов)
- Модификацию графического интерфейса системы
- Создание поддельных диалоговых окон для фишинга

3.5 Работа с библиотеками пользовательского интерфейса

Программа импортирует функции из библиотеки USER32.dll:

- CreateDialogParamW
- ShowWindow
- UpdateWindow
- MessageBox (через wsprintfW)
- Функции работы с буфером обмена (OpenClipboard, GetClipboardData)

Особое внимание привлекает использование функций работы с буфером обмена, что может применяться для:

- Перехвата данных из буфера обмена
- Кражи паролей и учетных данных

- Получения конфиденциальной информации

3.6 Механизмы проверки условий выполнения

Перед выполнением основных функций программа проверяет результаты операций с файлами и реестром.

Проверка 1: Результат работы с реестром

```
.text:004014D2                  test    eax, eax
.text:004014D4                  jz      short loc_4014E0
.text:004014D6                  mov     eax, 37h            ; Код ошибки 55
.text:004014DB                  jmp     loc_4018E7
```

Проверка 2: Результат открытия файла

```
.text:0040152B                  cmp     [ebp+var_234], 0FFFFFFFFh
.text:00401532                  jz      short loc_40153D
.text:00401534                  cmp     [ebp+var_234], 0
.text:0040153B                  jnz     short loc_40153F
.text:0040153D loc_40153D:
.text:0040153D                  int     6                   ; Неопределенная
инструкция
```

Особенность: При ошибке открытия файла выполняется инструкция "int 6", которая вызывает неопределенное исключение и приводит к аварийному завершению программы. Это может быть преднамеренным механизмом затруднения анализа или способом проверки окружения выполнения.

На основании проведенного анализа функциональности можно сделать следующие выводы о полезной нагрузке программы:

1. Программа работает с системным файлом net.exe, что может указывать на реализацию функциональности trojan horse - замены или модификации системных файлов.
2. Применение функций GDI32.dll и USER32.dll указывает на наличие графического интерфейса, возможно предназначенного для создания поддельных диалогов или фишинговых окон.
3. Использование функций работы с буфером обмена может применяться для перехвата паролей и другой конфиденциальной информации.
4. Обфускация данных через XOR-операции указывает на необходимость сокрытия обрабатываемой информации.
5. Наличие проверок условий выполнения позволяет программе адаптировать свое поведение в зависимости от окружения.

ЗАКЛЮЧЕНИЕ

Проведенное исследование троянской программы ED2D527EA3A55212D09AD5BD6ED5010E позволило выявить методы, применяемые для усложнения анализа и обхода систем защиты.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ:

1. УПАКОВКА И ОБФУСКАЦИЯ:
   - Программа не использует стандартный упаковщик
   - Код доступен для статического анализа
   - Применяется обфускация на уровне функций
2. МЕХАНИЗМЫ СКРЫТИЯ:
   - Реализовано ручное восстановление IAT
   - Используется проверка на отладчик (IsDebuggerPresent)
   - Применяется обфускация кода через XOR-операции
   - Выполняется работа с системным реестром
3. ФУНКЦИОНАЛЬНОСТЬ:
   - Взаимодействие с системным файлом net.exe
   - Использование графических библиотек (GDI32.dll, USER32.dll)
   - Возможный перехват данных через буфер обмена
   - Обфускация обрабатываемых данных

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ