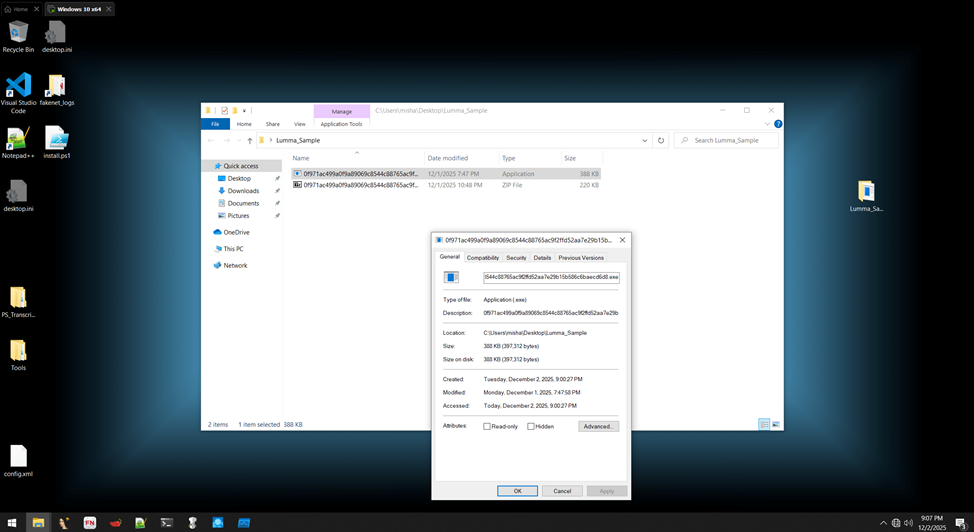


Рисунок 1 Установка Flare-VM для анализа вредоносного ПО

Рисунок 2 Установка вредоноса (Lumma-Stealer)

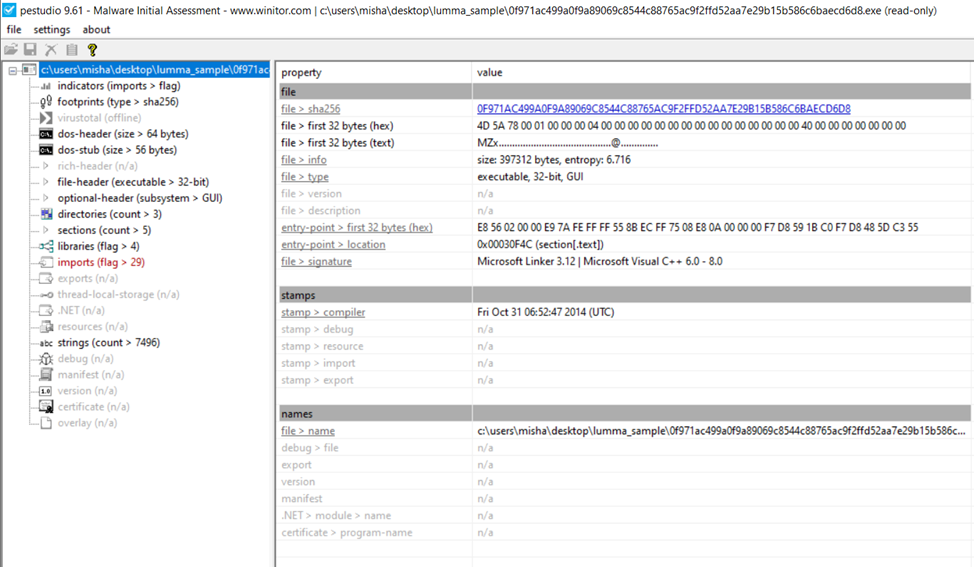


Рисунок 3 Анализ Вредоноса в PEStudio

1)Первое что мы видим, это то что файл 32-битный PE. File type = executable (обычный EXE-файл). По сути, малварь запускается сама, без необходимости загрузки другой программой.

2) Subsystem = GUI. Это значит: файл заявляет себя как GUI-приложение, а не консольное, но вредоносы обычно не показывают окно → они скрывают GUI и работают тихо. Это типичное поведение стиллеров.

3) Компилятор = Microsoft Visual C++ 6.0

Дата компиляции 2014 год, это фейковая дата. Малварь почти всегда подделывает timestamp на старые годы, для того чтобы маскироваться и уходить от сигнатур.

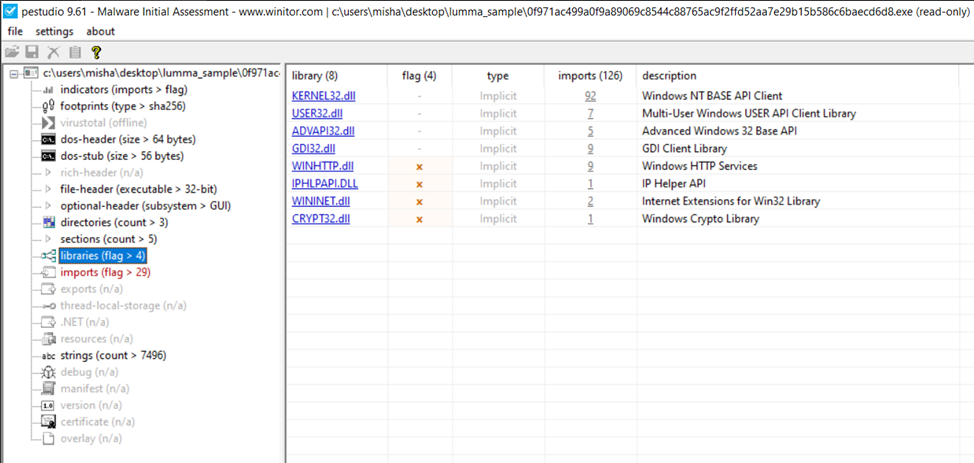


Рисунок 4 Библиотеки файла

В списке indicators видно:

· WINHTTP.dll (Windows HTTP Services)

· IPHLPAPI.DLL (IP Helper API)

· WININET.dll (Internet Extensions for Win32 Library)

· CRYPT32.dll (Windows Crypto Library)

Пояснение по каждой:

· WINHTTP / WININET

o сетевые библиотеки для работы по HTTP/HTTPS;

o для обычного приложения ничего криминального, но в контексте неизвестного исполняемого файла это явный намёк на сетевую активность / C2-коммуникацию.

· IPHLPAPI

o библиотека для работы с сетевой конфигурацией: интерфейсы, IP-адреса, статистика;

o малварь может использовать её, чтобы:

§ определить сетевую среду,

§ собрать инфу о системе,

§ реализовать какие-то проверки (VPN, прокси и т.д.).

· CRYPT32

o криптографическая библиотека Windows;

o часто используется:

§ для шифрования/дешифрования конфигов, строк, украденных данных,

§ для расшифровки сохранённых паролей (например, через CryptoAPI).

Сочетание WINHTTP + WININET + CRYPT32 очень характерно для инфостиллеров, которые:

1. собирают данные,

2. шифруют/кодируют их,

3. отправляют на удалённый C2.

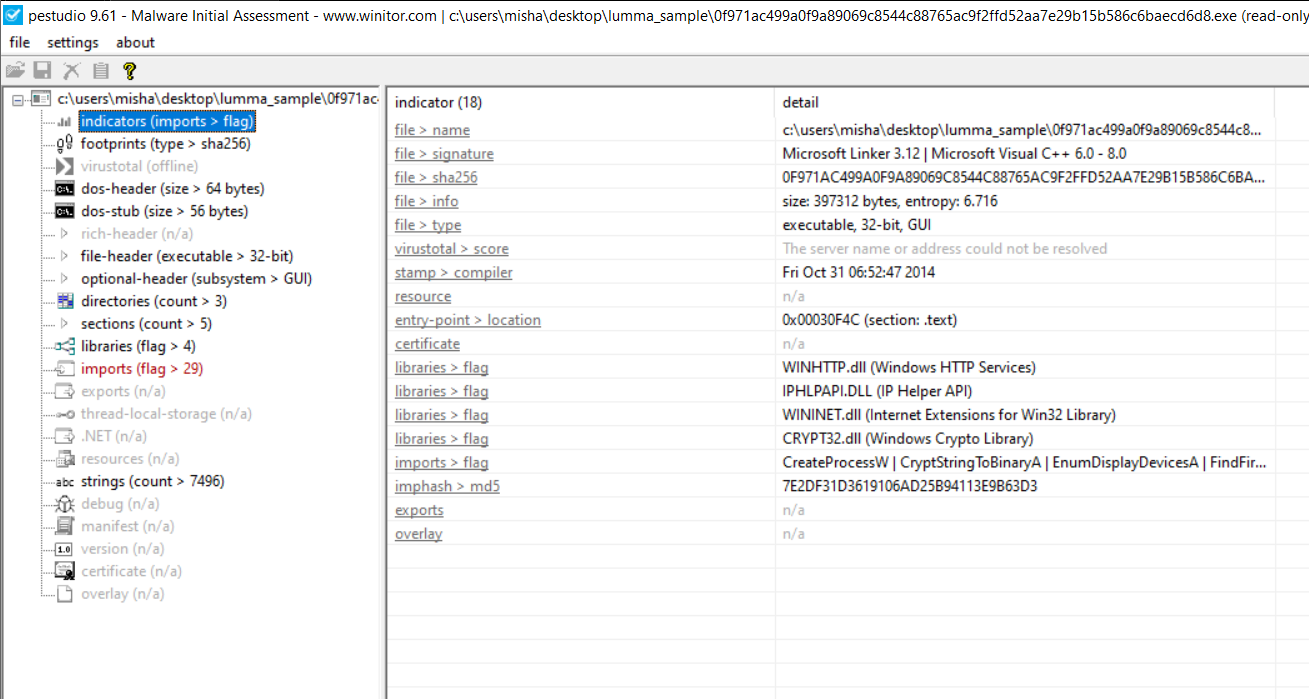


Рисунок 5. Индикаторы файла

В indicators видно строку:

* **imports > flag: CreateProcessW | CryptStringToBinaryA | EnumDisplayDevicesA | FindFir…**

Это обрезанный список, но уже по первым функциям видно:

* **CreateProcessW**
  + создание новых процессов;
  + малварь может:  
    - запускать доп. компоненты,
    - перезапускать саму себя,
    - запускать «легит» процессы, куда потом внедряется.
* **CryptStringToBinaryA**
  + функция из Crypt32 для преобразования строк в бинарный вид;
  + часто используется, когда конфиг/код закодирован в Base64 или другом текстовом виде, а потом переводится в бинарный перед расшифровкой/исполнением.
* **EnumDisplayDevicesA**
  + функция для получения инфы об устройствах отображения (мониторы, видеокарта);
  + иногда её используют:  
    - для сборки системного профиля,
    - или в анти-VM проверках (в виртуальных машинах видеоустройство может выглядеть «подозрительно»).

**Вывод по indicators:** PEstudio показывает классический набор признаков **сетевого вредоноса с шифрованием и возможной анти-VM логикой**. Для стиллера типа Lumma — это ожидаемо.

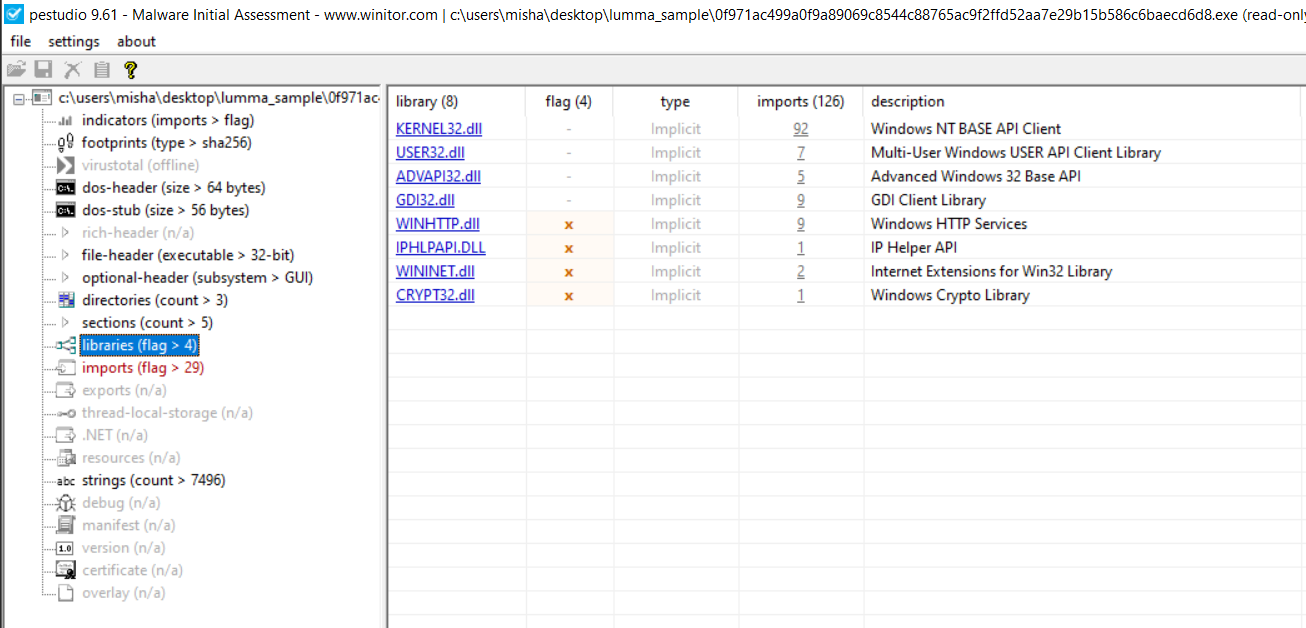


Рисунок 6. Вкладка Libraries

### **Базовые системные DLL**

* **KERNEL32** – доступ к файловой системе, памяти, процессам, потокам.
* **USER32** – окна, взаимодействие с пользователем (в том числе клавиатура/мышь).
* **ADVAPI32** – реестр, права, сервисы.
* **GDI32** – графическое отображение, реже интересно для анализа малвари.

Это есть почти в любом приложении Windows, сами по себе не подозрительны.

### **Подозрительные (для неизвестного EXE)**

* **WINHTTP / WININET** – HTTP/HTTPS запросы, работа с браузерным стеком.
* **IPHLPAPI** – информация о сети.
* **CRYPT32** – криптооперации.

**Количество импортов (126 всего)** довольно большое → это ещё один аргумент, что **файл не выглядит как маленький упакованный stub**, в котором импортов 5–10 штук. Тут видно, что внутри действительно находится рабочая логика.

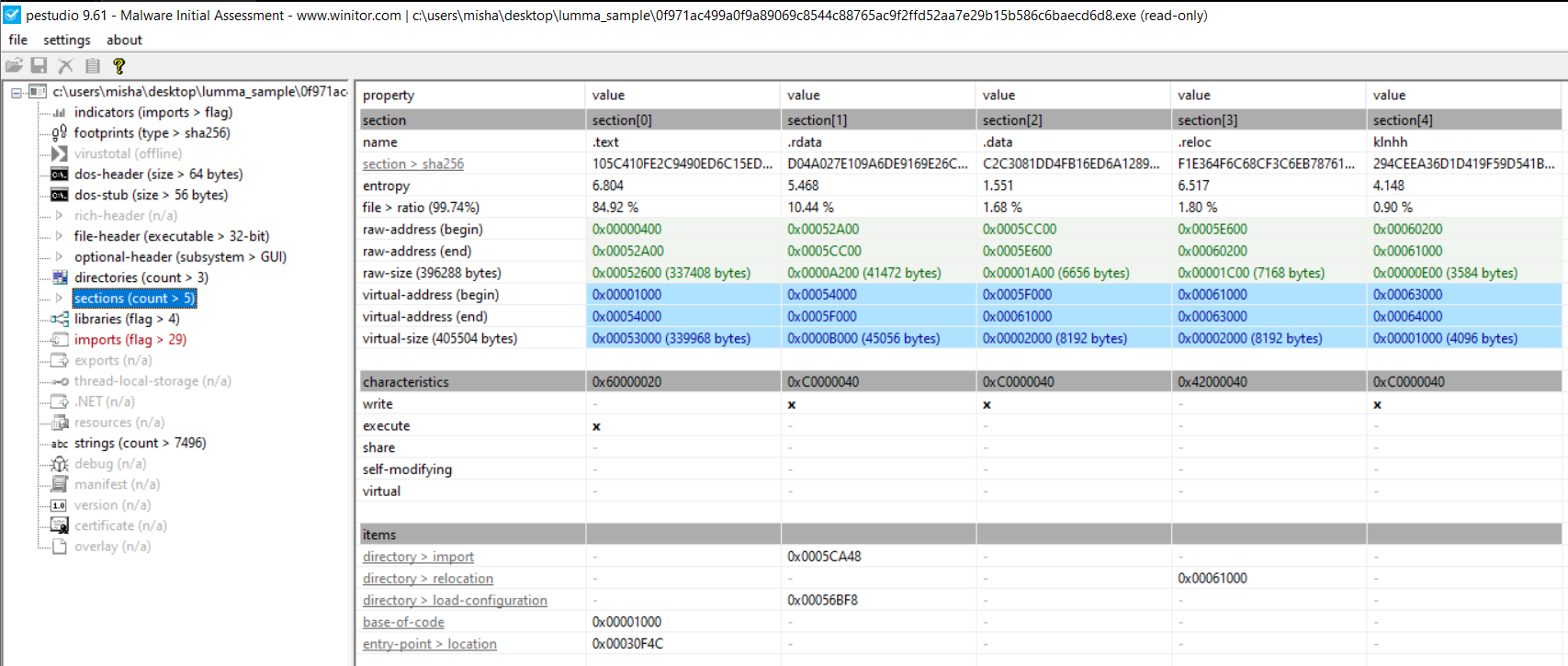


Рис 7. Вкладка Sections

В ходе статического анализа файла с использованием инструмента PEStudio обнаружено наличие пяти секций: .text, .rdata, .data, .reloc и кастомной секции klhh. Каждая из них имеет характерные особенности, которые позволяют оценить структуру и потенциальные методы сокрытия вредоносной активности.

1. **.text — основная секция исполняемого кода.** Её объём (≈396 КБ) значителен, что указывает на отсутствие типичной упаковки (например, UPX), для которой характерно минимальное содержание в .text. Повышенная энтропия (6.804) свидетельствует о возможном наличии обфусцированного или зашифрованного кода. Это типично для современных стиллеров, стремящихся усложнить анализ.
2. **.rdata — неизменяемые данные.** Эту секцию отличает средняя энтропия и реалистичный размер (≈41 КБ). Здесь могут находиться строки, таблицы импорта и константы. Секция выглядит стандартно, без аномалий.
3. **.data — глобальные переменные.** Низкая энтропия (1.55) означает наличие большого числа статических структур и нулевых байтов. Данная секция не содержит следов шифрования и не представляет интереса как место хранения конфигурации вредоносной программы.
4. **.reloc — таблица релокаций.** Секция служебная, используется системой при загрузке исполняемого файла. Характеристики соответствуют стандартному формату PE.
5. **klhh — нестандартная кастомная секция.** Наиболее интересная область. Нестандартное имя и небольшой размер (≈3—4 КБ) свидетельствуют о том, что данная секция создана вручную автором вредоносной программы. В ней могут быть размещены конфигурационные данные стиллера, зашифрованные строки, таблицы путей или ключи. Энтропия средней величины (4.148) подтверждает возможность частичной обфускации. Для семейства Lumma Stealer использование подобных кастомных секций является характерным приёмом скрытия конфигурации.

.

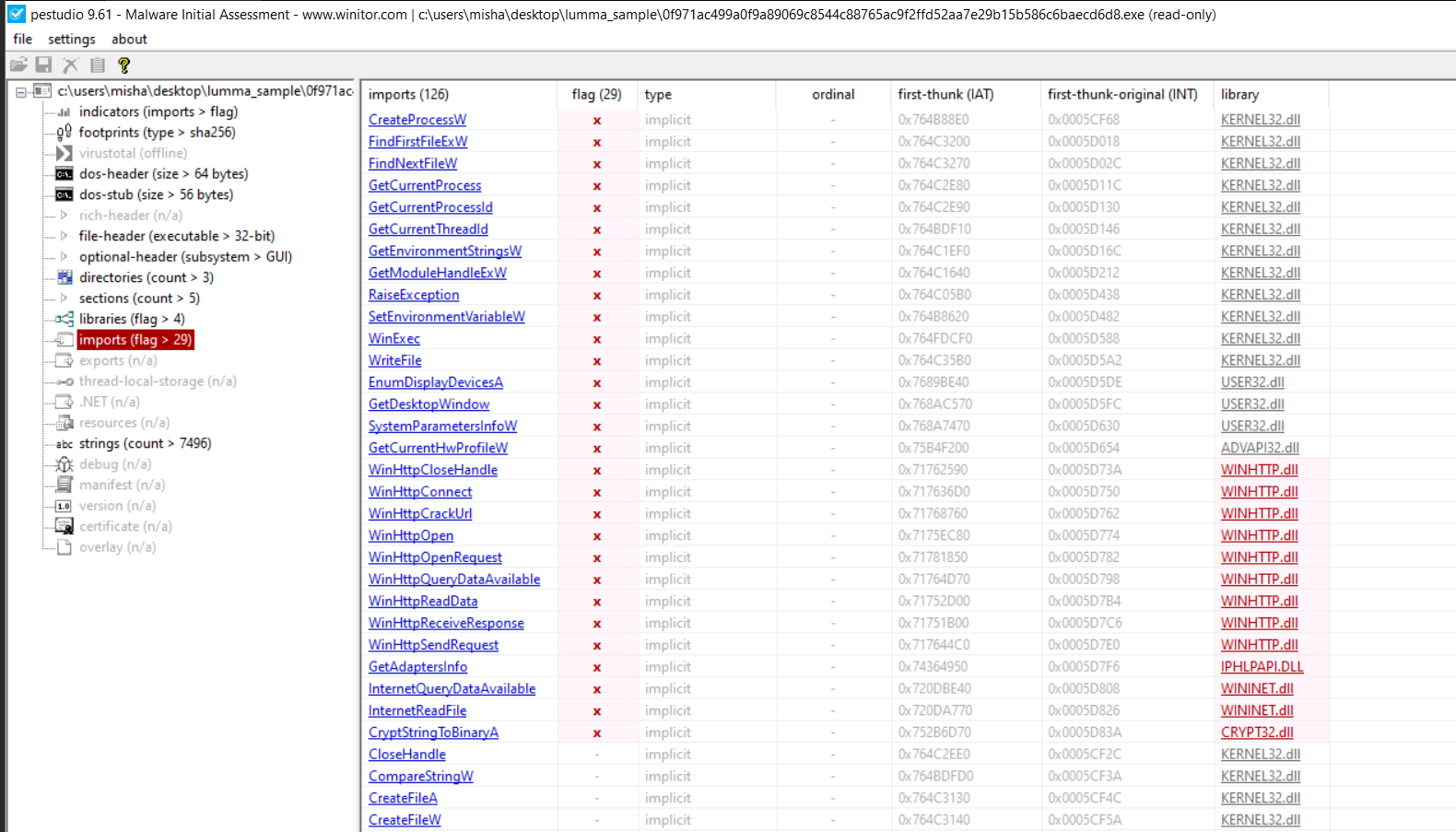


Рисунок 8. Вкладка Imports

Анализ импорта показал, что исследуемый образец активно использует библиотеки WinHTTP, WinInet, Kernel32, Advapi32 и Crypt32, что характерно для вредоносных программ класса *инфостиллер*. Наличие функций WinHttpOpen, WinHttpConnect, WinHttpSendRequest и WinHttpReceiveResponse указывает на способность программы устанавливать HTTP(S)-соединение с удалённым сервером, отправлять собранные данные и получать команды управления (C2).

Импорты CryptStringToBinaryA и функции из библиотеки Crypt32 свидетельствуют о наличии механизмов шифрования или декодирования данных, что используется для сокрытия конфигурации или подготовки украденной информации перед передачей.

Наличие системных API, таких как CreateProcessW, FindFirstFileExW, GetAdaptersInfo и EnumDisplayDevicesA, подтверждает возможность программы:

* сканировать файловую систему,
* собирать данные о сетевой конфигурации,
* выполнять анти-VM проверки,
* запускать дополнительные процессы.

Совокупность импортов недвусмысленно указывает, что файл выполняет функции сбора информации о системе и передачи данных на внешний C2-сервер, что полностью соответствует поведению вредоносных стиллеров типа **Lumma Stealer**.

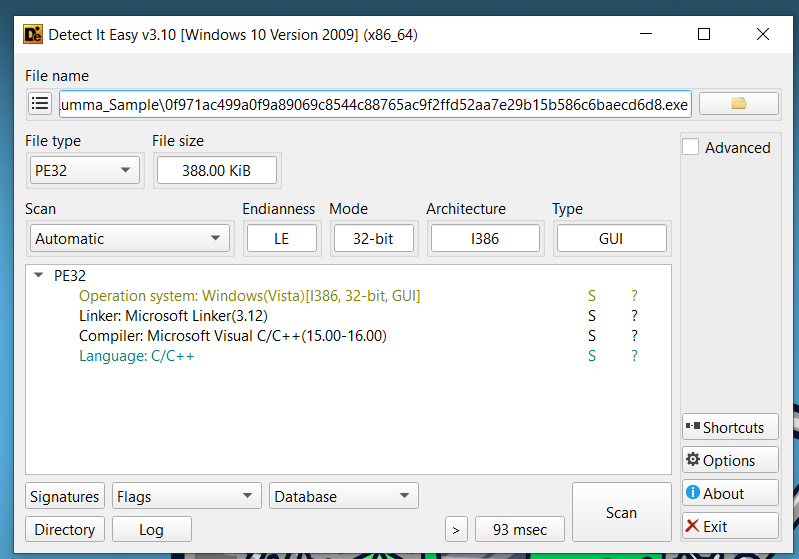


Рисунок 9. Анализ вредоноса через приложение DIE

По скрину мы можем заметить что файл не упакован. Если бы файл был упакован:

* DIE подсветил бы UPX, MPRESS, VMProtect, Themida
* В базе было бы видно packer signature
* Режим entropy был бы намного выше
* .text была бы очень маленькой

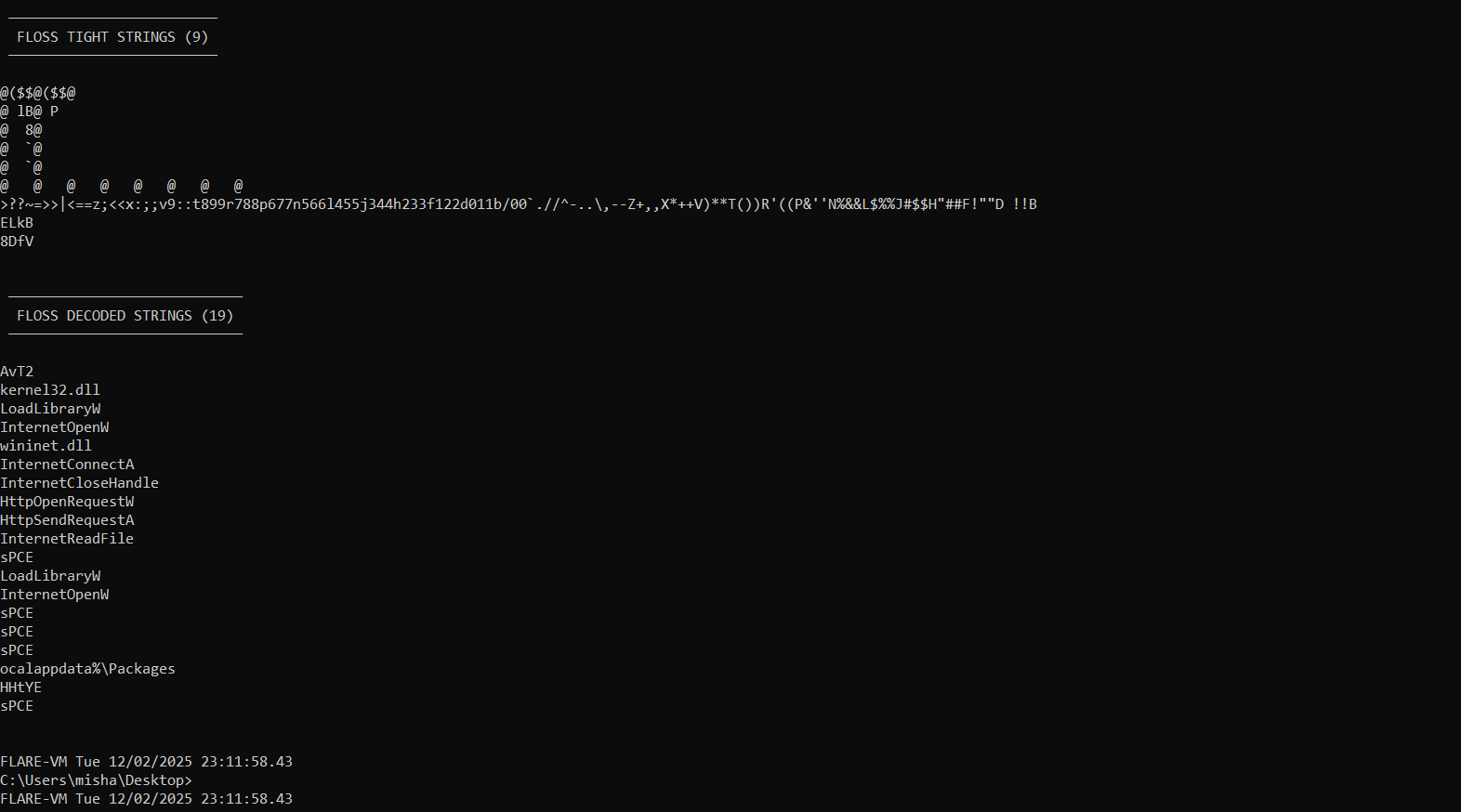


Рисунок 10. Запуск инструмента FLOSS для поиска скрытых/зашифрованных строк

Дешиврованные строки:   
kernel32.dll

LoadLibraryW

InternetOpenW

wininet.dll

InternetConnectA

HttpOpenRequestW

HttpSendRequestA

InternetReadFile

И самое важное, что есть строка %localappdata%\Packages. Это путь к данным приложений Windows, откуда стилер и украдет данные браузера.



Рисунок 11. Строки

Мы видим строку ‘’/c2sock’’ которая является частью URL пути на C2 сервере (например <http://185.xxx.xxx.xxx/c2sock>). И также видим множество обфусцированных/зашифрованных строк

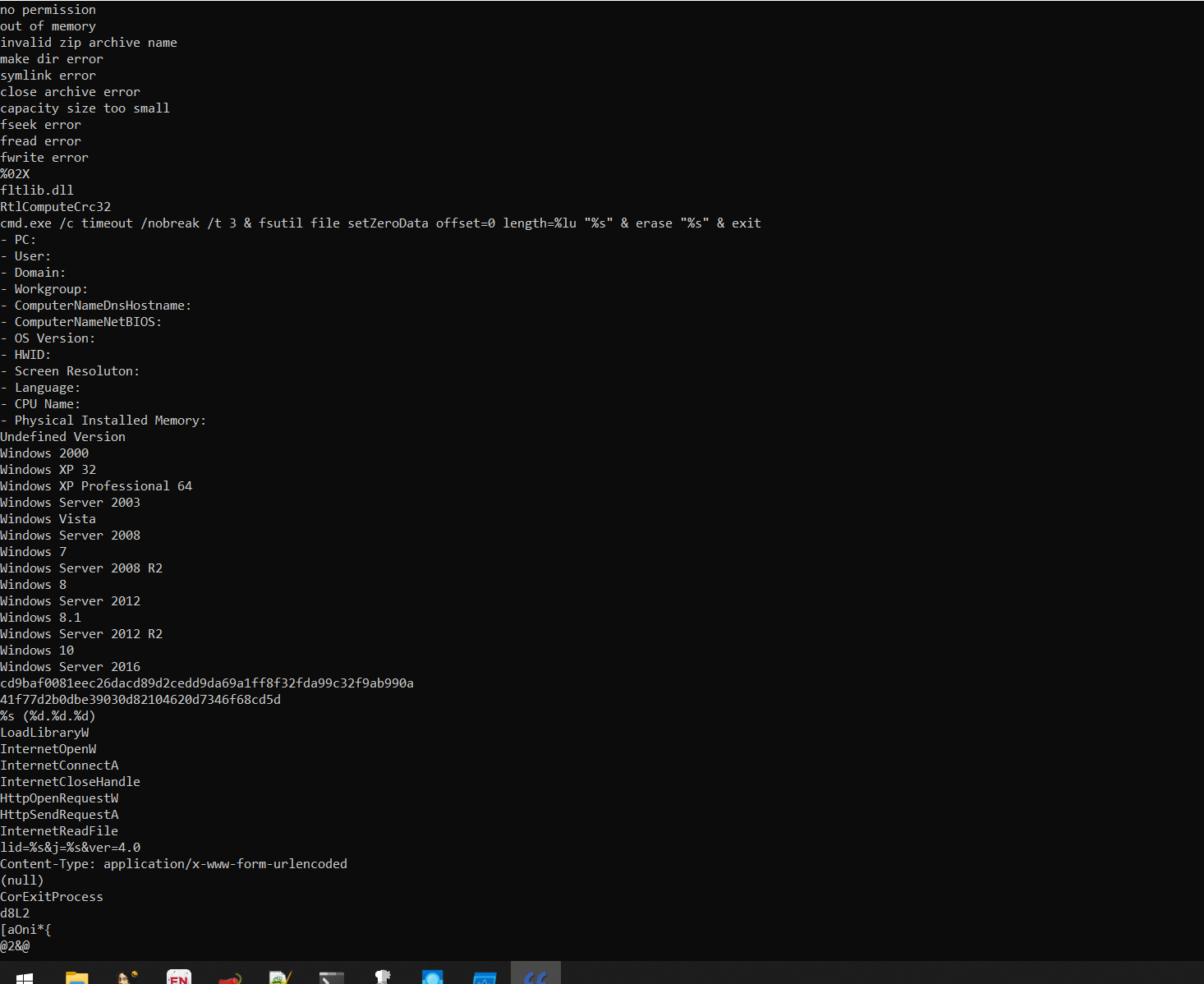


Рис 12.

Имеются строки:

OS Version:

CPU Name:

Screen Resolution:

User:

Domain:

Workgroup:

ComputerNameDnsHostname:

Installed Memory:

Это значит malware собирает идентификацию системы перед отправкой автору — стандартно для stealers.

Есть ещё строки:

InternetOpenA

InternetConnectA

HttpOpenRequestA

HttpSendRequestA

InternetReadFile

➜ это WinInet API = классический способ **эксфильтрации данных**.

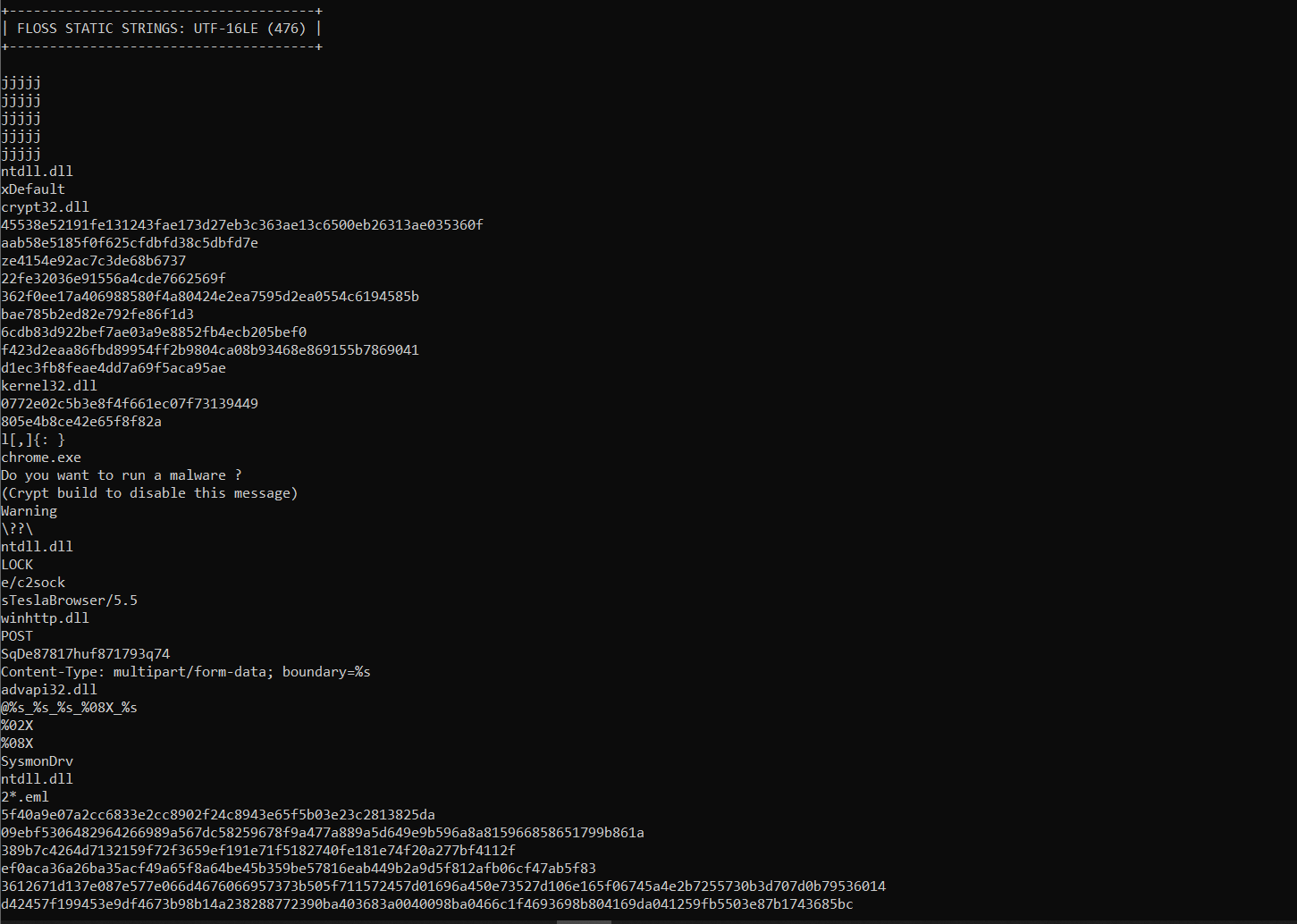


Рисунок 13

Есть строки:

POST

Content-Type: application/x-www-form-urlencoded

multipart/form-data; boundary=%s

➜ Malware делает **HTTP POST** на сервер злоумышленника.

Также сбор данных с Chrome

Есть явные признаки:

chrome.exe

LocalAppData%\Packages

Screen.png

DISPLAY

System.txt

Это говорит, что malware ворует:

* cookie
* стиллер делает скриншоты
* собирает список ПО
* ищет браузеры
* выгружает ввод / кэш
* пароли
* сессии авторизации
* autofill данные
* возможно: расширения и токены Discord, Telegram, Steam

## **Поддерживаемые браузеры / юзер-агенты**

ОЧЕНЬ важная строка:

TeslaBrowser/5.5

Это относится к семейству **Lumma Stealer / MetaStealer / Redline-подобные**.

Также есть замечательная строка:

cmd.exe /c timeout /t 3 & fsutil file setZeroData offset=0 length=%lu "%s" & erase "%s" & exit

Это чистейший **self-delete**:

1. файл затирается нулями,
2. удаляется,
3. malware закрывается.

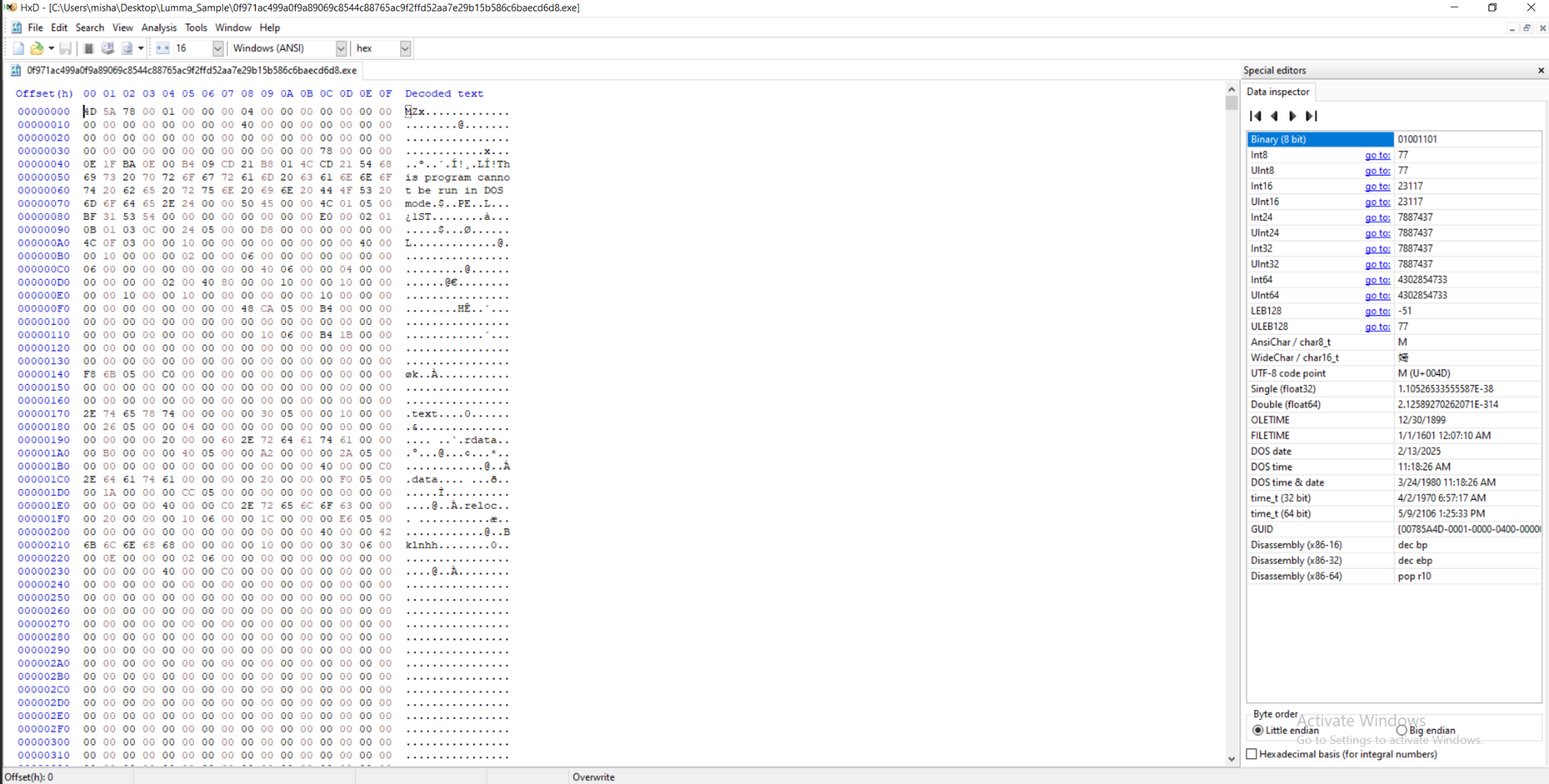
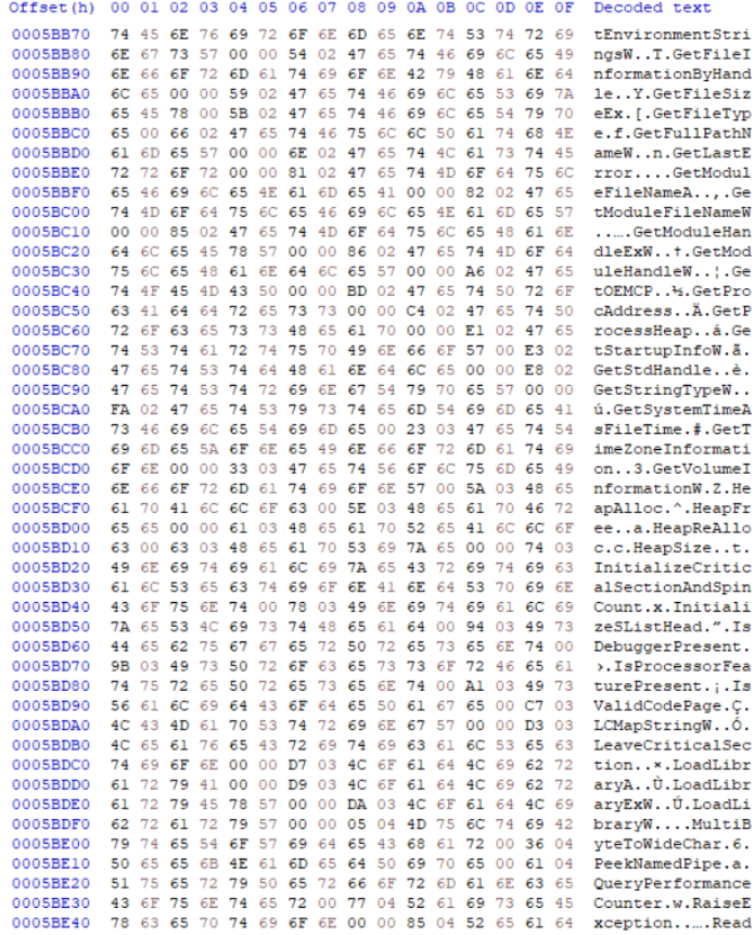
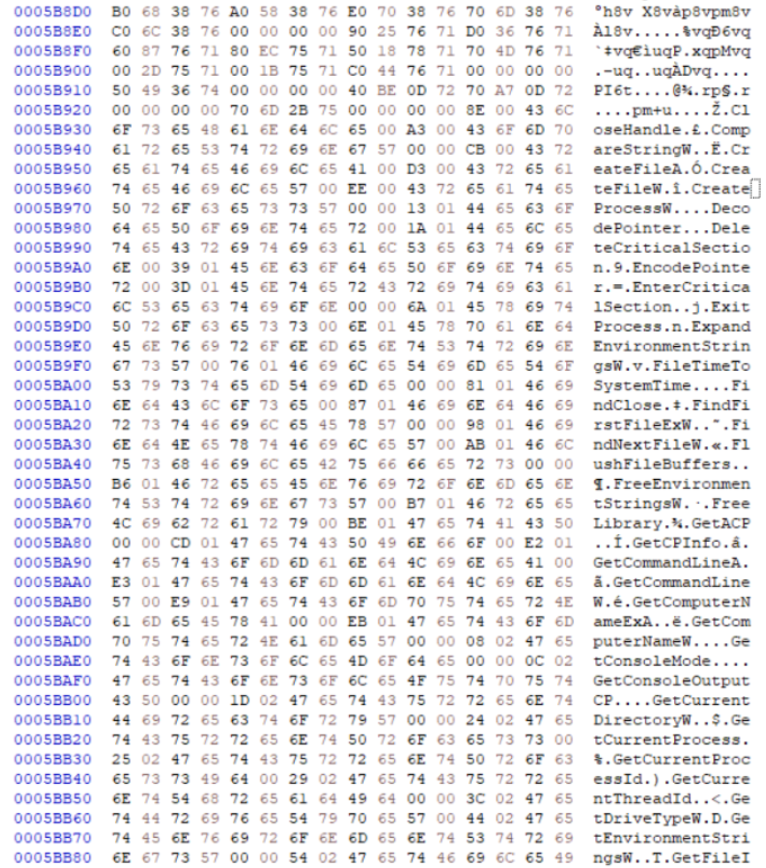


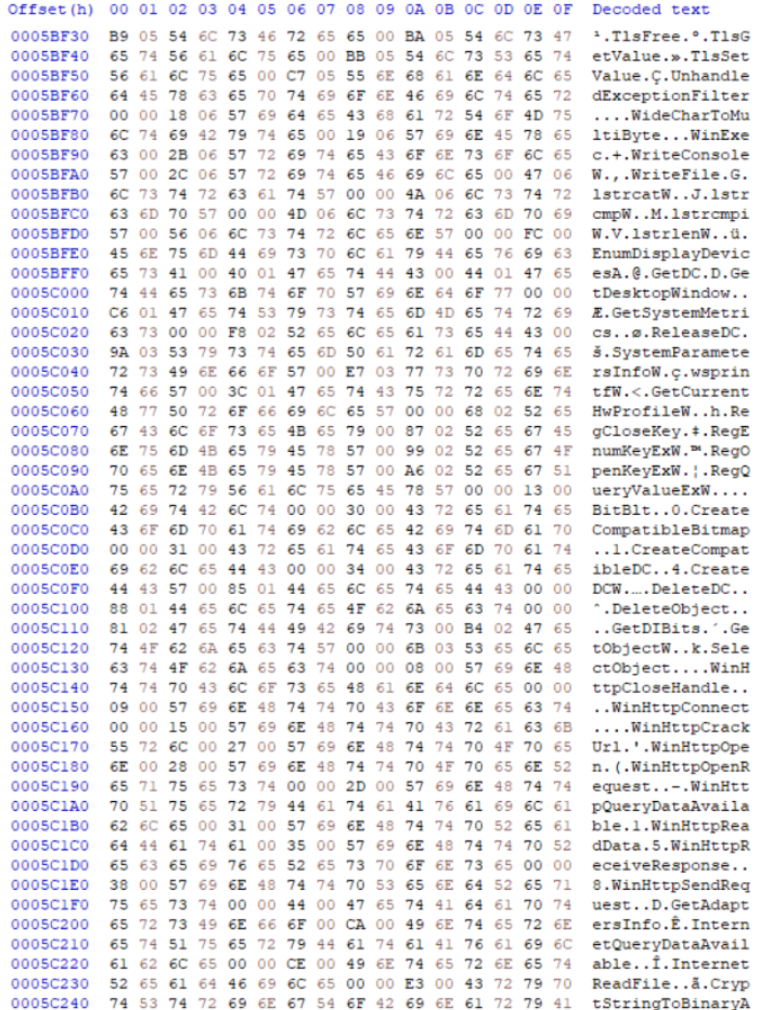
Рисунок 14 Анализ через программу HxD

## **Вывод по анализу файла в HEX-редакторе**

1. Файл начинается с сигнатуры MZ ( байты 4D 5A ), что подтверждает наличие DOS-заголовка и соответствие формату Windows Portable Executable.
2. В поле e\_lfanew (смещение 0x3C) содержится указатель на PE-заголовок по адресу 0xE0.
3. В DOS-stub присутствует стандартное сообщение “This program cannot be run in DOS mode.”.
4. Структура секций включает как стандартные .text, .rdata, .data, .reloc, так и нестандартную секцию klhnh, характерную для вредоносных программ.
5. Структура файла указывает, что он не упакован — секции имеют типичные размеры, данные читаемы, присутствуют строки.









## **Статический анализ строк в hex-редакторе HxD**

После первичной проверки PE-файла с помощью PE-Studio и Detect It Easy был выполнен детальный просмотр содержимого исполняемого файла в hex-редакторе **HxD**. Целью этого этапа было выявить значимые текстовые строки и имена API-функций, которые не были очевидны из стандартной таблицы импорта, а также сделать выводы о функциональности и уровне сложности вредоносной программы.

В нижней части файла, в области, соответствующей секции данных (.rdata), были обнаружены многочисленные строки с именами функций WinAPI. Часть из них дублировала уже известный список импортов (работа с файлами, процессами, окружением и т.п.), однако присутствовал и ряд **новых групп функций**, которые позволяют точнее охарактеризовать поведение образца.

### **1. Использование TLS (Thread Local Storage)**

Впервые были обнаружены строки:

* TlsGetValue
* TlsSetValue
* TlsAlloc

**Thread Local Storage (TLS)** — это механизм в программировании, который позволяет каждому потоку многопоточного процесса иметь собственное изолированное хранилище данных. Для вредоносного ПО это важно по двум причинам:

1. **Скрытая инициализация кода.** В PE-формате существует понятие *TLS-callbacks* — специальных функций, которые вызываются загрузчиком ещё до выполнения обычной точки входа программы. Малварь может разместить ключевую часть логики именно в таком callback-е, что усложняет анализ: отладчик, запущенный «с main», просто не увидит важный участок кода.
2. **Хранение чувствительных данных в рантайме.** Через TLS удобно хранить ключи шифрования, временные токены и другие данные, которые не должны быть явно видны в общей памяти процесса.

**Вывод:** наличие функций работы с TLS указывает на более высокий уровень сложности образца и позволяет предположить использование скрытой инициализации кода до основной точки входа.

### **2. Обработка исключений через SEH (Structured Exception Handling)**

Дополнительно были найдены строки:

* UnhandledExceptionFilter
* SetUnhandledExceptionFilter
* RaiseFailFastException
* RtlUnwind

Эти функции относятся к механизму **Structured Exception Handling (SEH)** — структурированной обработки исключений в Windows (Позволяет программистам контролировать обработку программных и аппаратных исключений, а также служит отладочным средством).

В контексте вредоносного ПО SEH чаще всего используется для:

* реализации **антиотладочных приёмов** (создание намеренных исключений и отслеживание реакции отладчика);
* маскировки реального потока исполнения (часть переходов реализуется через обработчики исключений);
* контролируемого завершения процесса при обнаружении неудобного окружения (антивирус, песочница).

**Вывод:** использование SEH-функций подтверждает, что малварь оснащена защитными механизмами от анализа и, вероятно, пытается усложнить трассировку своего кода.

### **3. Сетевое взаимодействие через WinHTTP**

В отдельном блоке строк были обнаружены функции библиотеки WinHTTP:

* WinHttpOpen
* WinHttpConnect
* WinHttpSendRequest
* WinHttpReceiveResponse
* WinHttpReadData

**WinHTTP** — это API для осуществления HTTP(S)-запросов из приложений Windows (работа с прокси, SSL, заголовками запросов и т.д.).

Для вредоносной программы такой набор функций типичен для:

* связи с **C2-сервером** (Command and Control) — сервером управления, с которым малварь обменивается командами и данными;
* загрузки дополнительного вредоносного компонента (second stage / payload);
* отправки на сервер собранной информации (системные характеристики, учётные данные и т.п.).

**Вывод:** анализ строк однозначно показывает, что образец реализует сетевое взаимодействие по HTTP(S) и, скорее всего, имеет архитектуру *клиент–сервер* с удалённым центром управления.

### **4. Использование криптографических функций**

Среди новых строк были обнаружены криптографические функции:

* CryptStringToBinaryA
* CryptBinaryToStringA
* CryptProtectData
* CryptUnprotectData

Эти функции относятся к **CryptoAPI** и **DPAPI (Data Protection API)**:

* CryptStringToBinary / CryptBinaryToString используются для преобразования данных между текстовым и бинарным представлением (например, Base64);
* CryptProtectData и CryptUnprotectData задействуют встроенный в Windows механизм шифрования данных, привязанный к конкретному пользователю или машине.

Для малвари это может означать:

* шифрование конфигурационных данных (адресов C2, токенов, ключей), чтобы они не хранились в открытом виде;
* шифрование передаваемой на сервер информации;
* использование DPAPI как простого способа хранить «секреты» на заражённой системе.

**Вывод:** образец использует системные средства шифрования, что позволяет ему скрывать части своей конфигурации и передаваемых данных, усложняя статический анализ.

### **5. Работа с графической подсистемой (GDI)**

В блоке импортов присутствует набор GDI- и GUI-функций:

* CreateCompatibleBitmap
* CreateDCW
* EnumDisplayDevicesA
* GetDIBits
* GetDeviceCaps

Эти функции относятся к **GDI (Graphics Device Interface)** — подсистеме Windows для работы с графикой и устройствами вывода.

Для обычных утилит использование GDI характерно для отрисовки интерфейса. В контексте вредоносного ПО такой набор чаще всего встречается в следующих сценариях:

* создание **скриншотов рабочего стола** или отдельных окон и последующая их отправка на C2-сервер;
* сбор информации о конфигурации мониторов, разрешении экрана и т.д. (например, для проверки, не запущена ли программа в виртуальной машине с нетипичными параметрами).

**Вывод:** по обнаруженным строкам можно предположить, что образец обладает функциональностью съёма скриншотов или, как минимум, активно взаимодействует с графической подсистемой для сбора дополнительных данных о системе.

### **6. Механизмы антиотладки и проверки окружения**

Отдельного внимания заслуживает группа функций, связанная с проверкой окружения:

* IsDebuggerPresent
* CheckRemoteDebuggerPresent
* GetVersionExW
* GetSystemMetrics
* GetStartupInfoW и др.

**IsDebuggerPresent** и **CheckRemoteDebuggerPresent** позволяют программе обнаружить наличие отладчика.  
 **GetSystemMetrics** и **GetVersionExW** используются для получения информации о версии операционной системы, параметрах экрана, наличии специфичных устройств и т.п.

Для вредоносного ПО это, как правило, служит для:

* определения факта запуска в виртуальной машине или песочнице (нестандартные параметры экрана, небольшое количество памяти, отсутствие пользовательской активности);
* блокировки выполнения при обнаружении отладчика;
* выбора конкретной ветки поведения в зависимости от версии ОС.

**Вывод:** образец содержит развитый набор проверок окружения, что подтверждает наличие антианализных механизмов и попыток скрыть вредоносную активность при запуске в исследовательской среде.

### **7. Косвенные признаки использования C++ и стандартной библиотеки**

В теле файла также встречаются строки вида:

* bad allocation
* string too long
* invalid string position

Такие сообщения характерны для исключений стандартной библиотеки **C++ (STL)**, в частности классов std::string и контейнеров, выбрасывающих исключения при ошибках работы с памятью или строками.

Наличие этих строк говорит о том, что:

* программа, с высокой долей вероятности, написана на C++ с использованием STL;
* структура кода более сложная, чем у простых «ручных» шеллкодов или минималистичных троянов: имеются классы, контейнеры, система обработки исключений.

**Вывод:** использование C++ и STL косвенно подтверждает, что перед нами не примитивный образец, а более сложное и развитое вредоносное ПО.