FickerStealer Malware Report

Date: 22/04/2025

Written by: Guy Zwerdling

Table of Contents

MR02: FickerStealer

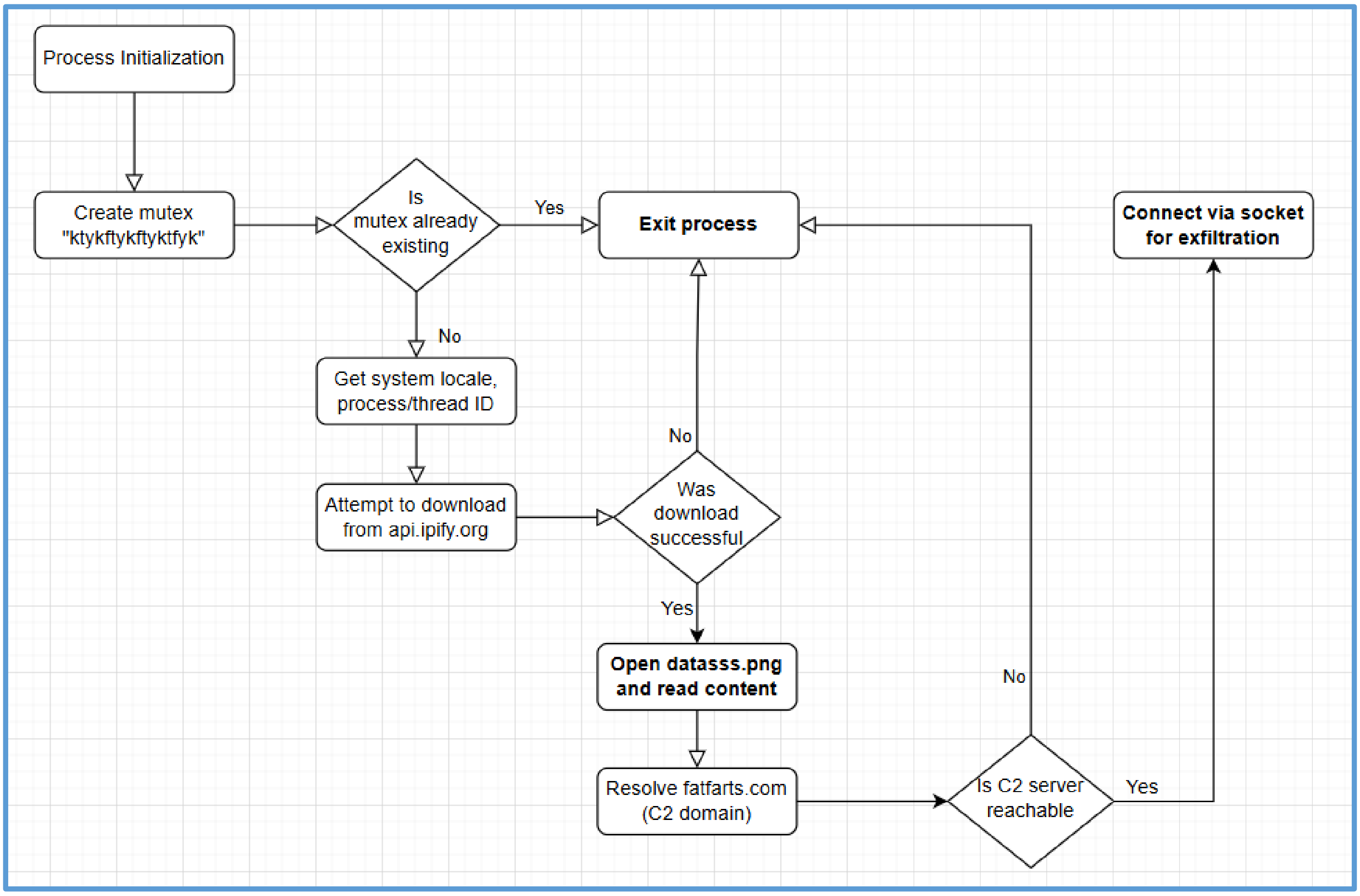
Executive Summary

The analyzed sample of the **FickerStealer** malware exhibits preliminary reconnaissance behavior without progressing to full-scale data theft. Upon execution, the malware performs basic environment enumeration, such as retrieving the system locale and querying **api.ipify.org** to determine the machine's external IP address. This IP is saved to a file named **datasss.png** in the **C:\ProgramData** directory. Shortly afterward, the malware attempts to establish a connection to its command-and-control (C2) server at **fatfarts.com**, indicating preparation for potential data exfiltration. However, the network interaction does not include any outbound transfer of sensitive data.

No evidence was found of deeper data harvesting activity. Specifically, the malware does not access known data storage locations such as browser credential databases (Login Data), password vaults (e.g., KeePass), or cryptocurrency wallets. Furthermore, it does not invoke decryption APIs like CryptUnprotectData or engage with SQLite databases, both of which are typically associated with information theft. These findings suggest that the malware either relies on a remote configuration (which was not provided in this environment), or employs conditional logic to avoid executing its payload in sandboxed or offline conditions. As such, this sample demonstrates only its initial stages, and further behavioral analysis may be required in a fully connected environment to observe its complete functionality.

High-Level Technical Summary (with diagram)

The analyzed FickerStealer sample demonstrates initial reconnaissance and network activity but does not proceed to steal or exfiltrate sensitive information. The following key actions were observed during execution:



1. **Mutex Creation**: The malware creates a uniquely named mutex ("ktykftykftyktfyk") to prevent multiple instances from running simultaneously.
2. **Environment Enumeration**: It collects basic system information, including locale settings and process/thread identifiers.
3. **External IP Acquisition**: The malware loads Urlmon.dll and uses the **URLDownloadToFileA** API to contact **api.ipify.org**. The response (external IP address) is saved to a file named datasss.png in **C:\ProgramData**.
4. **File Handling**: It reopens and reads **datasss.png**, presumably to verify or prepare it for further use.
5. **Command-and-Control Communication**: A DNS resolution and socket connection are initiated toward fatfarts.com. The malware establishes a TCP connection and waits for a response.

No Further Malicious Activity Observed: No file system access to browser credentials, password vaults, or crypto wallets was detected. No calls to sensitive Windows APIs such as CryptUnprotectData or sqlite3_open were made. No data exfiltration or module activation occurred following the C2 connection.

Basic Static Analysis

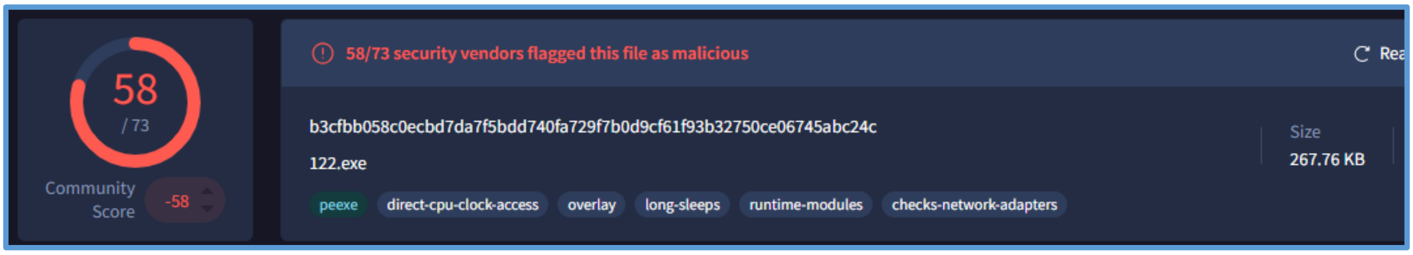
By checking that malware hash on viruses total to check and get more information about that sample as a first stage.

Filename: SecuriteInfo.com.Trojan.Packed2.42600.30573.20195.exe

MD5: 9ada122303e6dee1c0f0171bf2e59253

SHA1: b9f2cac95510c1199083504e0ae57fd14bf559d5

SHA256: b3cfbb058c0ecbd7da7f5bdd740fa729f7b0d9cf61f93b32750ce06745abc24c



We can see that the file name on viruses total in that case is 122.exe and it have tagged as malicious file by 58 vendors out of 73, and they specified this one as Trojan.

AVG	Win32:TrojanX-gen [Trj]	Avira (no cloud)	HEUR/AGEN.1341576
BitDefender	Gen:Variant.Jaik.41292	ClamAV	Win.Trojan.FickerStealer-9805476-1
CrowdStrike Falcon	Win/malicious_confidence_100% (W)	CTX	Exe.trojan.fickerstealer

Then using floss for extract string from the file itself to local txt file for allow us find related information that can give us clue on that basic static analysis be fore we go further.

```
λ floss.exe SecuriteInfo.com.Trojan.Packed2.42600.30573.20195.exe > flickerstealer.txt
INFO: floss: extracting static strings
finding decoding function features: 100%|████| 1244/1244 [00:03<00:00, 370.22 functions/s, skipped 0 library functions]
INFO: floss.stackstrings: extracting stackstrings from 1098 functions
INFO: floss.results: #0VA30VAC0VAS0VAc0VAs
INFO: floss.results: ^>>>n>>>
INFO: floss.results: ^>>>.>>>
extracting stackstrings: 100%|████████████████████████████████| 1098/1098 [00:14<00:00, 74.06 functions/s]
INFO: floss.tightstrings: extracting tightstrings from 53 functions...
```

Then we can grep out DLL files that look like being used by that malware.

```
λ grep -i "dll$" .\flickerstealer.txt
Urlmon.dll
KERNEL32.dll
msvcrt.dll
WS2_32.dll
ADVAPI32.dll
CRYPT32.dll
GDI32.dll
KERNEL32.dll
USER32.dll
NTDLL.DLL
```

In that case we can see several important DLL imports were discovered, indicating the functionality and behavior of the malware.

By reading about each one on the google we can find more interesting information about the function abilities of each, so that give us clue about the functioning of that malware and what it do, but we can tell so far, the order of the functions.

Urlmon.dll – Suggests use of HTTP functionality, likely for downloading additional payloads or communicating with a Command and Control (C2) server.

> answers.microsoft.com/en-us/windows/forum/all/urlmondll/7736927e-2f69-44f2-bd4f-2191dc6dc78a
>
> Urlmon.dll is a valid Microsoft file used by the Windows operating system to load the "Object Linking and Embedding" library on your Computer. This file is called only when a program needs it for its functionality.

KERNEL32.dll – Provides core Windows API functions such as memory allocation, process and thread manipulation, and file I/O. It is commonly used in all Windows executables.

> en.wikipedia.org/wiki/Microsoft_Windows_library_files#KERNEL32.DLL
>
> **KERNEL32.DLL** [edit]
>
> Contents hide
>
> KERNEL32.DLL exposes to applications most of the Win32 base APIs, such as memory management, input/output (I/O) operations, process and thread creation, and synchronization functions.[8]

msvcrt.dll – Indicates use of C runtime functions, possibly for memory operations, string manipulation, or system calls.

> en.wikipedia.org/wiki/Microsoft_Windows_library_files#MSVCRT.DLL_MSVCP*.DLL_and_CRTDLL.DLL
>
> **MSVCRT.DLL** is the C standard library for the Visual C++ (MSVC) compiler from version 4.2 to 6.0. It provides programs compiled by these versions of MSVC with most of the standard C library functions. These include string manipulation, memory allocation, C-style input/output calls, and others. **MSVCP*.DLL** is the corresponding C++ library.

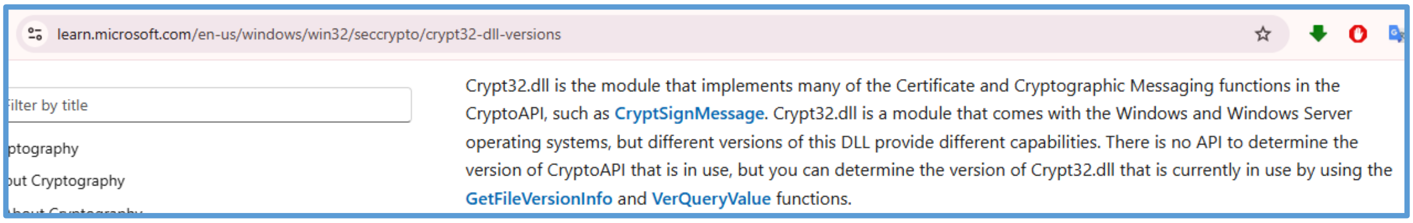
WS2_32.dll – The presence of this library shows that the malware uses Winsock for low-level TCP/IP networking. This implies active communication with remote servers, potentially for data exfiltration.

> en.wikipedia.org/wiki/Microsoft_Windows_library_files#WS2_32.DLL
>
> **WS2_32.DLL** [edit]
>
> Contents hide
>
> **WS2_32.DLL** implements the Winsock API, which provides TCP/IP networking functions and provides partial, broken compatibility with other network APIs. **wsock.dll** and **wsock32.dll** are older versions for Win3.11 and Win95 compatibility.

ADVAPI32.dll – Typically used for registry access, privilege manipulation, or cryptographic functions. Its use might indicate persistence mechanisms or access to sensitive system information.

> en.wikipedia.org/wiki/Microsoft_Windows_library_files#WS2_32.DLL
>
> **ADVAPI32.DLL**, the Advanced Windows 32 Base API DLL,[12] provides security calls and functions for manipulating the Windows Registry.

CRYPT32.dll – Supports encryption and decryption, possibly to encode stolen data before transmission to evade detection.


learn.microsoft.com/en-us/windows/win32/seccrypto/crypt32-dll-versions

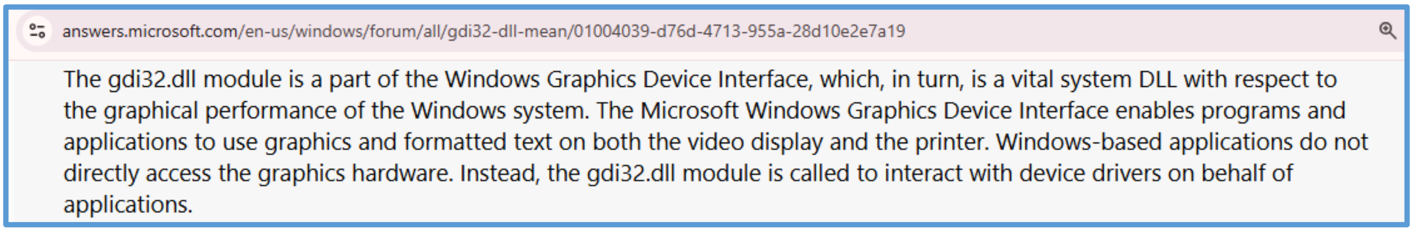
Filter by title

ptography

out Cryptography

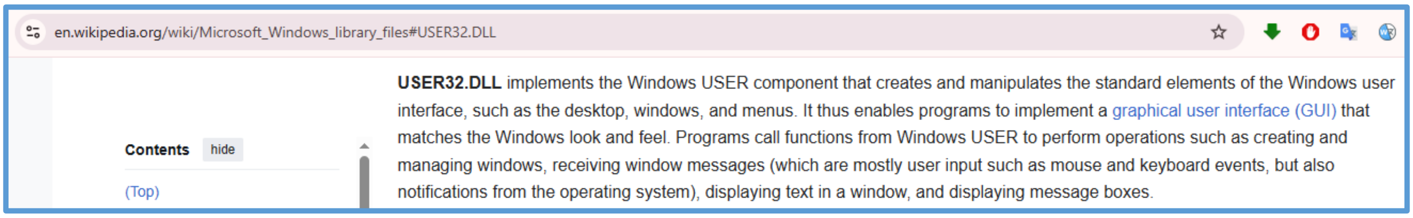
Crypt32.dll is the module that implements many of the Certificate and Cryptographic Messaging functions in the CryptoAPI, such as CryptSignMessage. Crypt32.dll is a module that comes with the Windows and Windows Server operating systems, but different versions of this DLL provide different capabilities. There is no API to determine the version of CryptoAPI that is in use, but you can determine the version of Crypt32.dll that is currently in use by using the GetFileVersionInfo and VerQueryValue functions.

GDI32.dll – Used for graphics and screen rendering. Its presence suggests the malware might perform screen captures.


answers.microsoft.com/en-us/windows/forum/all/gdi32-dll-mean/01004039-d76d-4713-955a-28d10e2e7a19

The gdi32.dll module is a part of the Windows Graphics Device Interface, which, in turn, is a vital system DLL with respect to the graphical performance of the Windows system. The Microsoft Windows Graphics Device Interface enables programs and applications to use graphics and formatted text on both the video display and the printer. Windows-based applications do not directly access the graphics hardware. Instead, the gdi32.dll module is called to interact with device drivers on behalf of applications.

USER32.dll – Provides functions for interacting with the user interface, potentially indicating keylogging or window enumeration.

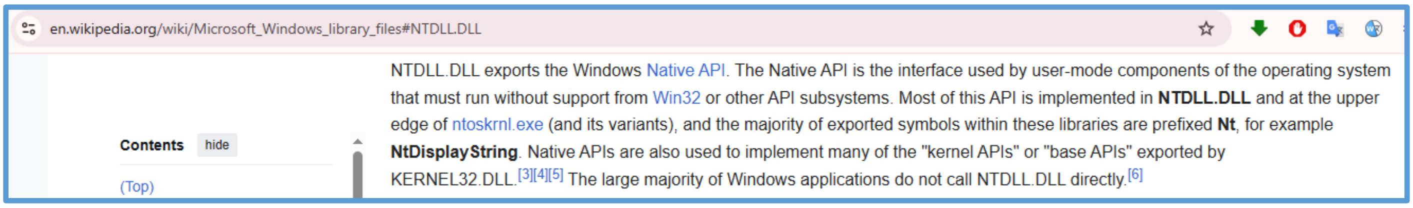

en.wikipedia.org/wiki/Microsoft_Windows_library_files#USER32.DLL

Contents hide

(Top)

USER32.DLL implements the Windows USER component that creates and manipulates the standard elements of the Windows user interface, such as the desktop, windows, and menus. It thus enables programs to implement a graphical user interface (GUI) that matches the Windows look and feel. Programs call functions from Windows USER to perform operations such as creating and managing windows, receiving window messages (which are mostly user input such as mouse and keyboard events, but also notifications from the operating system), displaying text in a window, and displaying message boxes.

NTDLL.dll – Gives access to low-level NT system calls. Malware often uses this for stealthy operations, such as manual mapping or avoiding detection by user-mode security tools.


en.wikipedia.org/wiki/Microsoft_Windows_library_files#NTDLL.DLL

Contents hide

(Top)

NTDLL.DLL exports the Windows Native API. The Native API is the interface used by user-mode components of the operating system that must run without support from Win32 or other API subsystems. Most of this API is implemented in **NTDLL.DLL** and at the upper edge of ntoskrnl.exe (and its variants), and the majority of exported symbols within these libraries are prefixed **Nt**, for example **NtDisplayString**. Native APIs are also used to implement many of the "kernel APIs" or "base APIs" exported by KERNEL32.DLL.[3][4][5] The large majority of Windows applications do not call NTDLL.DLL directly.[6]

At this stage, the presence of these DLL imports suggests that the malware may rely on various system capabilities. Specifically, the inclusion of WS2_32.dll and Urlmon.dll indicates potential network communication functionality, while the use of GDI32.dll hints at possible interaction with graphical components, such as screen rendering — which may support screenshot capabilities. Other libraries, such as ADVAPI32.dll, USER32.dll, and CRYPT32.dll, suggest broader system interaction including registry access, user interface manipulation, and possible data encryption. However, while these imports reveal potential areas of functionality, they do not confirm any specific malicious behavior.

Also, by searching for windows API, let's say these that end up with W we can find the following.

```
C:\Users\zwerd\Desktop
λ grep "........W$" "FickerStealer.txt"
WSASocketW
RegEnumKeyExW
RegOpenKeyExW
RegQueryInfoKeyW
RegQueryValueExW
GetObjectW
CreateDirectoryW
CreateFileW
FindFirstFileW
FindNextFileW
FormatMessageW
GetComputerNameW
GetEnvironmentVariableW
GetLocaleInfoW
GetModuleFileNameW
GetModuleHandleW
WriteConsoleW
EnumDisplayDevicesW
```

The extracted wide-character API calls strongly indicate that the malware performs host profiling (e.g., **GetComputerNameW**, **GetLocaleInfoW**), registry interaction, file system traversal (e.g., **FindFirstFileW**), and network socket creation (**WSASocketW**). These functions suggest that the malware likely gathers system information, searches for files of interest, and communicates with a remote server. However, while the presence of these functions implies certain behaviors, confirmation requires dynamic execution and code flow analysis.

By searching the work password we can see the following, which may be some variable that used on the functioning of that executable file.

```
C:\Users\zwerd\Desktop
λ grep -i "password" FickerStealer.txt
password_value
```

We also can search for protocols used, like http and ftp, in the following case, I have found several URL, several seems valid and may be used during the malware execution, we may get to know about them on the other part of the assessment like the dynamic analysis part.

```
C:\Users\zwerd\Desktop
λ grep -Ei "http|https|ftp|tcp|udp" FickerStealer.txt
https://sectigo.com/CPS0
2http://crl.sectigo.com/SectigoRSACodeSigningCA.crl0s
2http://crt.sectigo.com/SectigoRSACodeSigningCA.crt0#
http://ocsp.sectigo.com0#
?http://crl.usertrust.com/USERTrustRSACertificationAuthority.crl0v
3http://crt.usertrust.com/USERTrustRSAAddTrustCA.crt0%
http://ocsp.usertrust.com0
https://www.digicert.com/CPS0
2http://crl3.digicert.com/DigiCertAssuredIDCA-1.crl08
2http://crl4.digicert.com/DigiCertAssuredIDCA-1.crl0w
http://ocsp.digicert.com0A
5http://cacerts.digicert.com/DigiCertAssuredIDCA-1.crt0
.http://www.digicert.com/ssl-cps-repository.htm0
http://ocsp.digicert.com0C
7http://cacerts.digicert.com/DigiCertAssuredIDRootCA.crt0
4http://crl3.digicert.com/DigiCertAssuredIDRootCA.crl0:
4http://crl4.digicert.com/DigiCertAssuredIDRootCA.crl0
is_httponly
```

We can see here four URL's that repeating them selfs:
http://ocsp.digicert.com
http://crl.sectigo.com
http://crt.usertrust.com
https://sectigo.com/CPS

These URLs are not C2 indicators or signs of malicious communication. They are most likely:

- Embedded in the digital signature of the malware (even if it's invalid or expired), or
- Part of the OS validating certificate chains when the binary is loaded.

By searching other files that can be execute on windows as part of website, I have found the following.

```
C:\Users\zwerd\Desktop
λ grep -E "\.php|\.asp|\.jsp|\.xyz|\.top|\.ru|\.cc" FickerStealer.txt
tramplink-msk@rambler.ru0
```

One potential indicator of malicious intent was found:

- Email address: **tramplink-msk@rambler.ru**. This could be used for attacker identification, exfiltration, or embedded metadata.

The following is just way of extracting the domains and subdomain and order what we have found so far.

```
λ grep -Eo "[a-zA-Z0-9.-]+\.(com|net|xyz|ru)" FickerStealer.txt | sort | uniq.exe
cacerts.digicert.com
crl.sectigo.com
crl.usertrust.com
crl3.digicert.com
crl4.digicert.com
crt.sectigo.com
crt.usertrust.com
fatfarts.com
ocsp.digicert.com
ocsp.sectigo.com
ocsp.usertrust.com
rambler.ru
sectigo.com
www.digicert.com
```

By using PEview we can see the magic starting value for executable file which is 4D 5A the MA sign, the we can see on the other headers

```
FickerStealer.exe
  IMAGE_DOS_HEADER
  MS-DOS Stub Program
  IMAGE_NT_HEADERS
  IMAGE_SECTION_HEADER .text
  IMAGE_SECTION_HEADER .data
  IMAGE_SECTION_HEADER .rdata
  IMAGE_SECTION_HEADER
  IMAGE_SECTION_HEADER .bss
  IMAGE_SECTION_HEADER .idata
```

pFile	Raw Data	Value
00000000	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	MZ..............
00000010	B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	........@.......
00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....
00000030	00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00	....
00000040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68	........!..L.!Th
00000050	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno
00000060	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS
00000070	6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00	mode....$.......
00000080	50 45 00 00 4C 01 08 00 00 00 00 00 0C 04 00 00	PE..L...........

We also have indication that this malware file used for 32bit system architecture.

0008	IMAGE_FILE_LOCAL_SYMS_STRIPPED
0020	IMAGE_FILE_LARGE_ADDRESS_AWARE
0100	IMAGE_FILE_32BIT_MACHINE

On PEStudio we can see the same indication for 32bit malware.

size-of-optional-header	0x00E0	224 bytes
signature	0x00004550	PE00
machine	0x014C	Intel-386

file > size	274184 bytes	+
file > type	executable, 32-bit, GUI	+
compiler > stamp	Thu Jan 01 00:00:00 1970	+

PEStudio also can suggest what each DLL is used for.

library (8)	duplicate (1)	flag (2)	first-thunk-original (INT)	first-thunk (IAT)	type (1)	imports (118)	group (0)	description
KERNEL32.dll	-	-	0x000430B4	0x000432AC	implicit	10	-	Windows NT BASE API Client
msvcrt.dll	-	-	0x000430E0	0x000432D8	implicit	29	-	Microsoft C Runtime Library
WS2_32.dll	-	x	0x00043158	0x00043350	implicit	13	network	Windows Socket Library
ADVAPI32.dll	-	-	0x00043190	0x00043388	implicit	5	-	Advanced Windows 32 Base API
CRYPT32.dll	-	x	0x000431A8	0x000433A0	implicit	1	crypto	Windows Crypto Library
GDI32.dll	-	-	0x000431B0	0x000433A8	implicit	7	-	GDI Client Library
KERNEL32.dll	x	-	0x000431D0	0x000433C8	implicit	47	-	Windows NT BASE API Client
USER32.dll	-	-	0x00043290	0x00043488	implicit	6	-	Multi-User Windows USER API Client Library

This tool gives us the following indication of another windows API's that being used and also suggest the technique that this malware may do during execution.

imports (118)	flag (26)	first-thunk-original (INT)	first-thunk (IAT)	hint	group (0)	technique (9)	type (4)
GetDesktopWindow	x	0x00043BD6	0x00043BD6	270 (0x010E)	windowing	-	implicit
GetCurrentProcessId	x	0x000434B8	0x000434B8	457 (0x01C9)	reconnaissance	T1057 | Process Discovery	implicit
WSACleanup	x	0x000436C6	0x000436C6	27 (0x001B)	network	-	implicit

Each **X** in the "**Flag**" column next to an imported function indicates that this particular API is:

- Known to be **commonly used by malware**
- Possibly **risky or suspicious** in a behavioral context
- Flagged by PEStudio's internal ruleset based on **threat intelligence and heuristics**

ascii	36	section:.rdata	-	-	-	Partial loss of significance (PLOSS)
ascii	26	section:.rdata	-	-	-	Mingw-w64 runtime failure:
ascii	31	section:.rdata	-	-	-	Address %p has no image-section

The presence of the string Mingw-w64 runtime failure: in the .rdata section suggests that the malware was compiled using the MinGW-w64 toolchain. This compiler is **commonly used in Linux environments for cross-compiling** Windows executables. While this does not confirm the development platform, it increases the likelihood that the malware was built on a non-Windows system, such as Linux.

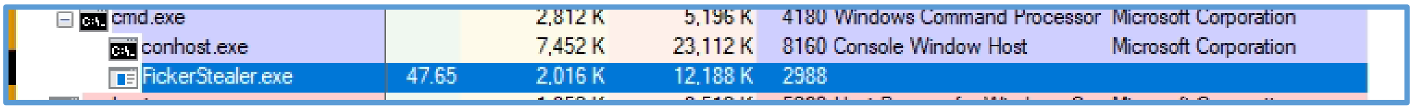
So, if Mingw-w64 have being used for compile that file, is an indication that the source code may be **C/C++** and not **.NET**, we can also check die tool that can give us more indication what is used for that malware.

```
▾ PE32
    Operation system: Windows(95)[I386, 32-bit, GUI]          S   ?
    Linker: GNU Linker ld (GNU Binutils)(2.30)[GUI32,signed]  S   ?
    Compiler: MinGW                                           S   ?
    (Heur)Language: C                                         S   ?
  ▾ Overlay: Binary[Offset=0x00040c00,Size=0x2308]
        Data: BitRock installer data                          S   ?
```

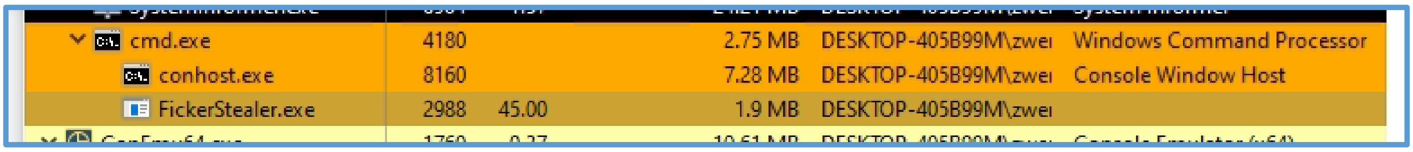
So, we can be sure that this malware was written in C and was compiled with MinGW.

Basic Dynamic Analysis

I have ran that malware under CMD that run as administrator, then with several tools check it's activities, on the procexp I was able to see the following which tell us that this malware indeed have being run from cmd process.

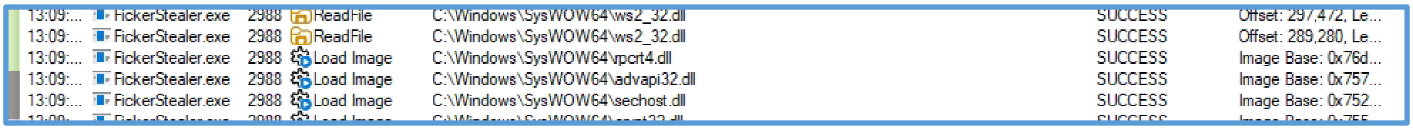


Also, on System Informer we can see the same

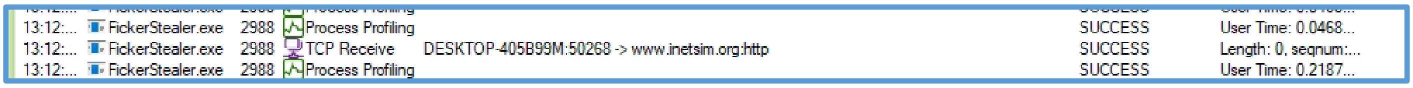


On Procmon I was able to see that this file sample do several stuff related to registry and directory

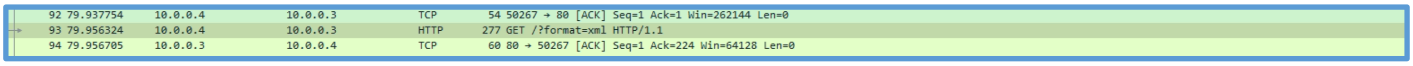
We can see the following DLL files in used related to that sample



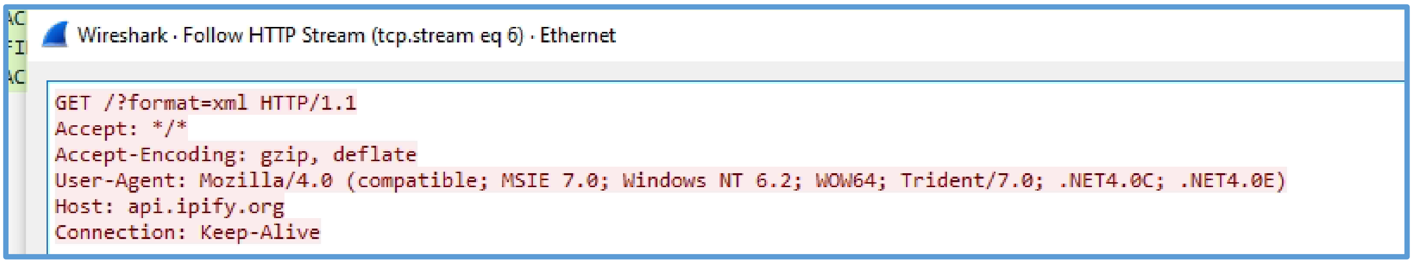
Also by scrolling down, I was able to see that it create some TCP session to my REMnux box, so I was open Wireshark on background to see what it doing.



The by follow the steam I have found the following HTTP GET query, I had another query but they all was to some Microsoft location, so I get this is some issue with my LAB.



Following the all stream lead us to the following information.



Since that domain have not been found on the basic stage then I have search for DNS query about that api.ipify.org and found that this was done after I have ran System Informer, so I have filer out all DNS query and then I have found one of the domain we have seen earlier fatfarts.com.

88	79.735339	10.0.0.4	10.0.0.3	DNS	73 Standard query 0x369b A api.ipify.org
89	79.744626	10.0.0.3	10.0.0.4	DNS	89 Standard query response 0x369b A api.ipify.org A 10.0.0.3
103	80.170765	10.0.0.4	10.0.0.3	DNS	72 Standard query 0x1bcc A fatfarts.com
104	80.179602	10.0.0.3	10.0.0.4	DNS	88 Standard query response 0x1bcc A fatfarts.com A 10.0.0.3
358	711.592042	10.0.0.4	10.0.0.3	DNS	74 Standard query 0x31c0 A ecs.office.com

The checking the order if self, found that after this DNS query some HTTP session was startup.

103	80.170765	10.0.0.4	10.0.0.3	DNS	72 Standard query 0x1bcc A fatfarts.com
104	80.179602	10.0.0.3	10.0.0.4	DNS	88 Standard query response 0x1bcc A fatfarts.com A 10.0.0.3
105	80.180675	10.0.0.4	10.0.0.3	TCP	66 50268 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
106	80.181144	10.0.0.3	10.0.0.4	TCP	66 80 → 50268 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
107	80.181229	10.0.0.4	10.0.0.3	TCP	54 50268 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0

Follow that stream didn't give much, I can't be sure that this steam related to that domain but if yes, it was end right away.

105	80.180675	10.0.0.4	10.0.0.3	TCP	66 50268 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
106	80.181144	10.0.0.3	10.0.0.4	TCP	66 80 → 50268 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
107	80.181229	10.0.0.4	10.0.0.3	TCP	54 50268 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
335	200.194919	10.0.0.3	10.0.0.4	TCP	60 80 → 50268 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
336	200.195005	10.0.0.4	10.0.0.3	TCP	54 50268 → 80 [ACK] Seq=1 Ack=2 Win=2102272 Len=0
496	1124.463594	10.0.0.4	10.0.0.3	TCP	54 50268 → 80 [RST, ACK] Seq=1 Ack=2 Win=0 Len=0

Filter out the information from Procmon again about the TCP session we can see the process ID and the source port which is 50267, so this indication I am not on the right session.

Time of Day	Process Name	PID	Operation	Path
13:10:02.0769348	FickerStealer.exe	2988	TCP Connect	DESKTOP-405B99M:50267 -> www.inetsim.org:http
13:10:02.0959455	FickerStealer.exe	2988	TCP Send	DESKTOP-405B99M:50267 -> www.inetsim.org:http
13:10:02.1097395	FickerStealer.exe	2988	TCP Receive	DESKTOP-405B99M:50267 -> www.inetsim.org:http

This information lead us back to the briviuse session we have found

tcp.port == 50267

No.	Time	Source	Destination	Protocol	Length	Info
90	79.937169	10.0.0.4	10.0.0.3	TCP	66	50267 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
91	79.937662	10.0.0.3	10.0.0.4	TCP	66	80 → 50267 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
92	79.937754	10.0.0.4	10.0.0.3	TCP	54	50267 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
93	79.956324	10.0.0.4	10.0.0.3	HTTP	277	GET /?format=xml HTTP/1.1
94	79.956705	10.0.0.3	10.0.0.4	TCP	60	80 → 50267 [ACK] Seq=1 Ack=224 Win=64128 Len=0

Then on Procmon I have also found the following which means that the GET query to the /?format=xml, was the first query, then the second query was to api.ipify.org, then the second was to fatfarts.com.

After search on google, I have found that api.ipify.org is a legitimate public IP lookup service often used by malware to determine the external IP address of the infected machine. This information can help attackers identify the victim's geolocation or check if the system is running in a sandbox or virtual environment.

13:10:02.2126744	FickerStealer.exe	2988	TCP Disconnect	DESKTOP-405B99M:50267 -> www.inetsim.org:http	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:10:02.3204088	C:\Users\zwerd\Desktop\FickerStealer.exe			DESKTOP-405B99M:50268 -> www.inetsim.org:http	SUCCESS	Length: 0, mss: 1460, sackopt: 1, tsopt: 0,
13:12:02.3341811	FickerStealer.exe	2988	TCP Receive	DESKTOP-405B99M:50268 -> www.inetsim.org:http	SUCCESS	Length: 0, seqnum: 0, connid: 0

Based on the observed behavior, this sample attempts to connect to api.ipify.org and retrieve an XML file via the endpoint /?format=xml, likely to determine the public IP address of the infected host. Following this, it initiates a request to fatfarts.com, which is suspected of serving as a command-and-control (C2) domain.

However, the HTTP session to fatfarts.com does not proceed as expected, suggesting either a failed connection, server-side filtering, or a conditional communication trigger not met during this run.

Since that sample ask for api.ipify.org, we know that it searches for the IP address of the victim, I have set up webserver that contains record of that domain for 10.0.0.1 and that web always return with 200 OK that contain 10.0.0.4.

Source	Destination	Protocol	Lengtl	Info
10.0.0.4	10.0.0.1	TCP	66	50118 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
10.0.0.1	10.0.0.4	TCP	66	80 → 50118 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
10.0.0.4	10.0.0.1	TCP	54	50118 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
10.0.0.4	10.0.0.1	HTTP	277	GET /?format=xml HTTP/1.1
10.0.0.1	10.0.0.4	TCP	60	80 → 50118 [ACK] Seq=1 Ack=224 Win=64128 Len=0
10.0.0.1	10.0.0.4	HTTP	344	HTTP/1.1 200 OK (text/html)
10.0.0.4	10.0.0.1	TCP	54	50118 → 80 [ACK] Seq=224 Ack=291 Win=261632 Len=0
10.0.0.4	10.0.0.3	DNS	72	Standard query 0x0e87 A fatfarts.com
10.0.0.3	10.0.0.4	DNS	88	Standard query response 0x0e87 A fatfarts.com A 10.0.0.3
10.0.0.4	10.0.0.3	TCP	66	50119 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
10.0.0.3	10.0.0.4	TCP	66	80 → 50119 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
10.0.0.4	10.0.0.3	TCP	54	50119 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0

```
 6 0.350565   10.0.0.1   10.0.0.4   HTTP   344 HTTP/1.1 200 OK  (text/html)
 7 0.350656   10.0.0.4   10.0.0.1   TCP     54 50118 → 80 [ACK] Seq=224 Ack=291 Win=261632 Len=0

> Frame 6: 344 bytes on wire (2752 bits), 344 bytes captured (2752 bits) on interface \Device\NPF_{770C416C-C5A1-41E4-B255-E30CD01BA5CA
> Ethernet II, Src: 0a:00:27:00:00:01 (0a:00:27:00:00:01), Dst: PCSSystemtec_52:47:d9 (08:00:27:52:47:d9)
> Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.0.4
> Transmission Control Protocol, Src Port: 80, Dst Port: 50118, Seq: 1, Ack: 224, Len: 290
> Hypertext Transfer Protocol
v Line-based text data: text/html (1 lines)
    10.0.0.4\n
```

But still the behavior is the same, the session ends after trying to get fatfarts.com domain.

No.	Time	Source	Destination	Protocol	Lengtl	Info
10	0.900211	10.0.0.4	10.0.0.3	TCP	66	50119 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
11	0.900474	10.0.0.3	10.0.0.4	TCP	66	80 → 50119 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
12	0.900559	10.0.0.4	10.0.0.3	TCP	54	50119 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
521	120.911496	10.0.0.3	10.0.0.4	TCP	60	80 → 50119 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
522	120.911595	10.0.0.4	10.0.0.3	TCP	54	50119 → 80 [ACK] Seq=1 Ack=2 Win=2102272 Len=0

But please note the time.

12	0.900559	10.0.0.4	10.0.0.3	TCP	54	50119 → 80 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
521	120.911496	10.0.0.3	10.0.0.4	TCP	60	80 → 50119 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0
522	120.911595	10.0.0.4	10.0.0.3	TCP	54	50119 → 80 [ACK] Seq=1 Ack=2 Win=2102272 Len=0

If we change the time format, we can see that this is 2 seconds.

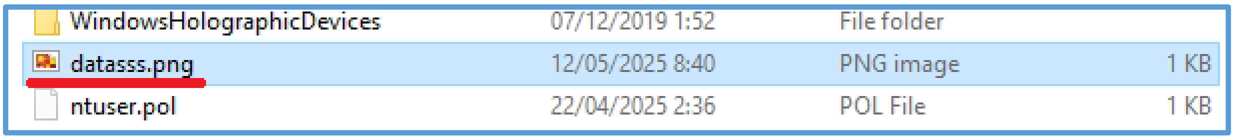
12	2025-05-12 08:40:03.849166	10.0.0.4	10.0.0.3	TCP	54	50119 → 80 [ACK]
521	2025-05-12 08:42:03.860103	10.0.0.3	10.0.0.4	TCP	60	80 → 50119 [FIN,
522	2025-05-12 08:42:03.860202	10.0.0.4	10.0.0.3	TCP	54	50119 → 80 [ACK]

This behavior is typical of any TCP-based application that fails to receive a reply it cleans up the socket to avoid hanging the process.

By searching for another information related to the action of that sample using Procmon, I have found that it left surprise.

8:40:0...	FickerStealer.exe	1168	CreateFile	C:\ProgramData\datass.png	SUCCESS
8:40:0...	FickerStealer.exe	1168	CreateFile	C:\ProgramData\datass.png	SUCCESS

Checking the directory, I have found that this file was created on the local system.



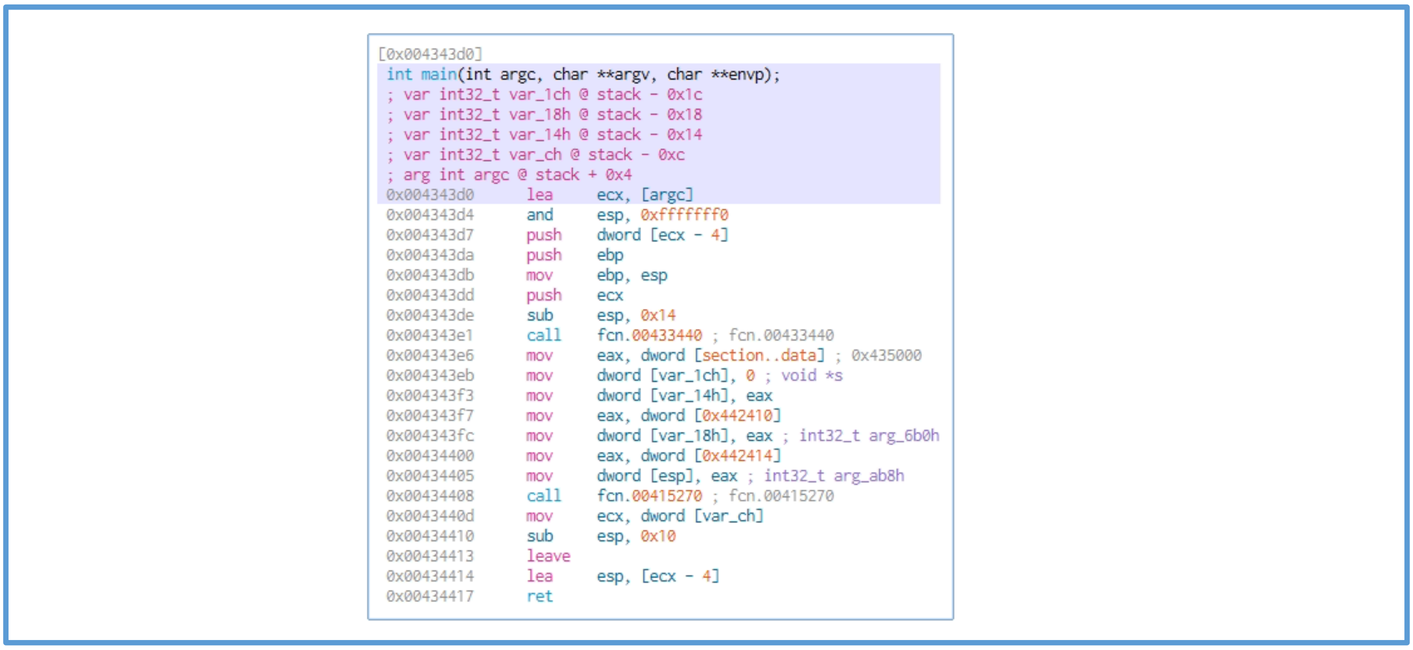
WindowsHolographicDevices	07/12/2019 1:52	File folder	
datass.png	12/05/2025 8:40	PNG image	1 KB
ntuser.pol	22/04/2025 2:36	POL File	1 KB

So far, the dynamic analysis reveals that the **FickerStealer** sample performs environment profiling by retrieving the **victim's public IP address** using the legitimate service **api.ipify.org**, followed by an attempted connection to a suspicious domain (**fatfarts.com**), which may serve as a Command-and-Control (C2) server.

Although the malware didn't manage to connect to its command-and-control (C2) server - possibly because of a sandbox, offline server, or missing conditions - it still showed other activity like accessing the registry and file system. It also created an image file on the local machine. This behavior suggests the malware tries to collect data, may attempt to stay on the system, and is built to talk to an external server.

Advance Static Analysis

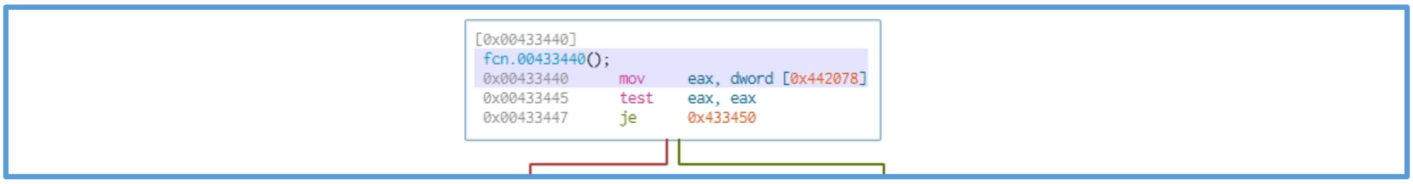
Opening that executable with Cutter leads us to the following main.



```
[0x004343d0]
int main(int argc, char **argv, char **envp);
; var int32_t var_1ch @ stack - 0x1c
; var int32_t var_18h @ stack - 0x18
; var int32_t var_14h @ stack - 0x14
; var int32_t var_ch @ stack - 0xc
; arg int argc @ stack + 0x4
0x004343d0      lea     ecx, [argc]
0x004343d4      and     esp, 0xfffffff0
0x004343d7      push    dword [ecx - 4]
0x004343da      push    ebp
0x004343db      mov     ebp, esp
0x004343dd      push    ecx
0x004343de      sub     esp, 0x14
0x004343e1      call    fcn.00433440 ; fcn.00433440
0x004343e6      mov     eax, dword [section..data] ; 0x435000
0x004343eb      mov     dword [var_1ch], 0 ; void *s
0x004343f3      mov     dword [var_14h], eax
0x004343f7      mov     eax, dword [0x442410]
0x004343fc      mov     dword [var_18h], eax ; int32_t arg_6b0h
0x00434400      mov     eax, dword [0x442414]
0x00434405      mov     dword [esp], eax ; int32_t arg_ab8h
0x00434408      call    fcn.00415270 ; fcn.00415270
0x0043440d      mov     ecx, dword [var_ch]
0x00434410      sub     esp, 0x10
0x00434413      leave
0x00434414      lea     esp, [ecx - 4]
0x00434417      ret
```

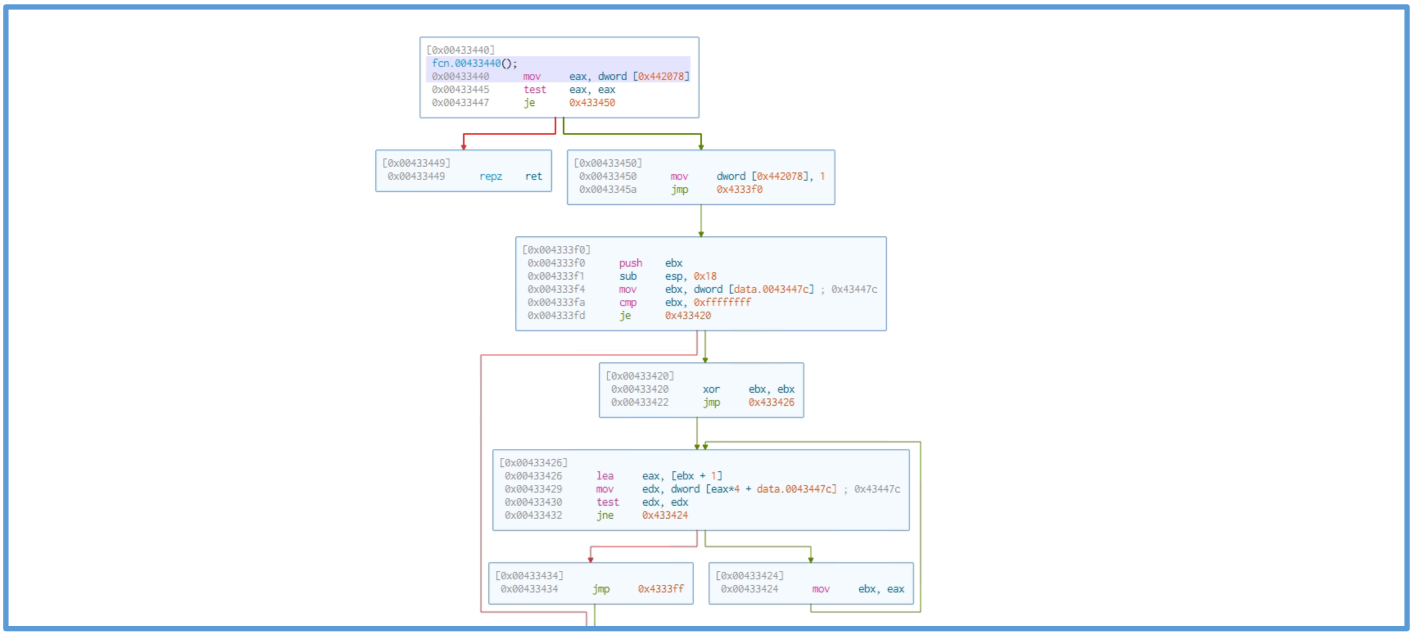
We can see here specifications about variables, then after it down several stuff like checking the values and variables we can see the first call of function named **fcn.00433440**, then we can see another changing that done but right after another call to function named **fcn.00415270**.

The start of the fcn.00433440 looks like it insert value (likely 0 or 1) to **eax** and then test it for getting the ZERO FLAG (**ZF**), if the flag are 0 it jump to the location of **ret** meaning program close.

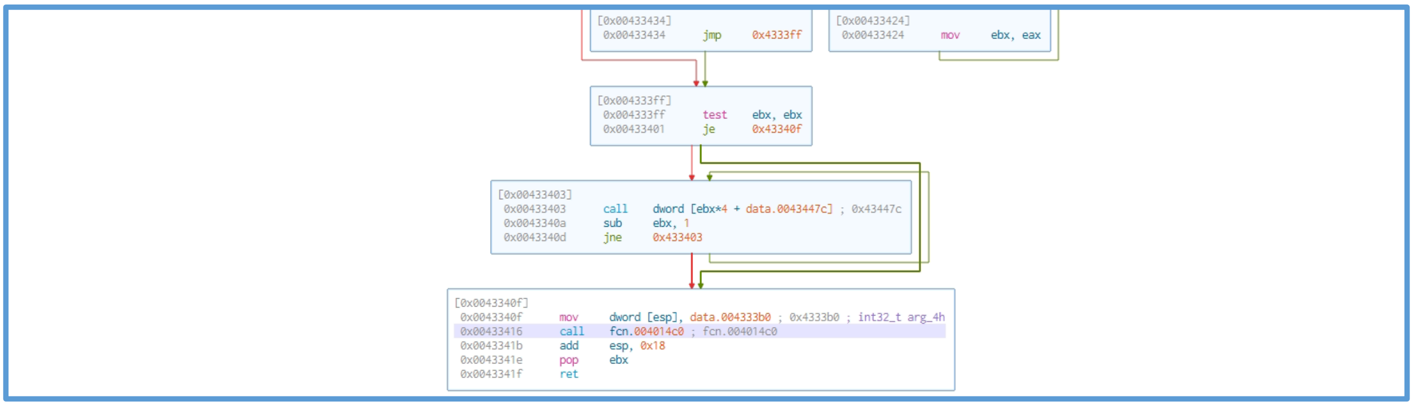


```
[0x00433440]
fcn.00433440();
0x00433440      mov     eax, dword [0x442078]
0x00433445      test    eax, eax
0x00433447      je      0x433450
```

But if the ZF is 1 it will proceed the process, we can see another several operation that move values inside the block ofcode, I can’t tell exactly what they do but I guess that this operation is part of restart the envirument of the main code.



If we look further down, we can see two more function calls. The first one performs some operations and then jumps to another location in the code. The second call leads to a function that doesn't seem to connect to anything recognizable from the basic static or dynamic analysis — in other words, it doesn't reference any strings, APIs, or behaviors we've already observed.



At this stage, we can return to the main function and follow the second function call, which is more likely to contain relevant information about the malware’s behavior and actual functionality. By doing so, we can see that this function gets many arguments, so we need to scroll down to find interesting information.

```
0x00415276    and     esp, 0xfffffff0
0x00415279    mov     eax, 0x1390
0x0041527e    call    fcn.00434370 ; fcn.00434370
0x00415283    lea     eax, [0x42bfaf]
0x00415289    lea     ebx, [arg_ab8h]
```

At first we can see that function call, by looking inside of that we found same, nothing interesting that are reference to the static analysis we have done erliear.

But then we can see the following Windows API call, it is not so interesting but still that is the first time we see some value that can be found and reference on the static stage we have done so far.

```
0x004152ad    add     esp, 0xc
0x004152b0    push    ebx         ; LPCSTR lpLibFileName
0x004152b1    call    sub.KERNEL32.dll_LoadLibraryA ; sub.KERNEL32.dll_LoadLibraryA ; HMODULE L...
0x004152b6    mov     edi, eax
0x004152b8    lea     eax, [0x430816]
```

At this point, we observe that the malware pushes the value of the EBX register as an argument to **LoadLibraryA**. This indicates that EBX likely contains a pointer to a string representing the name of a DLL.

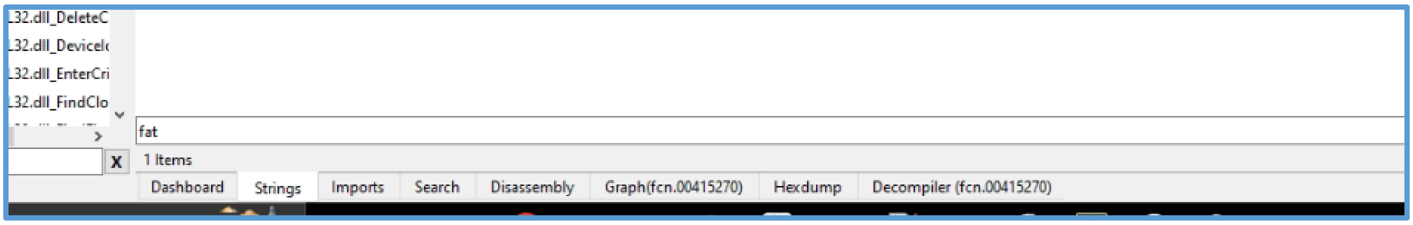
By scrolling through we can see another Windows API's like GetProcAddress, GetUserDefaultLocaleName, GetSystemMetrics,

```
0x0041530b    push    edi         ; HMODULE hModule
0x0041530c    call    sub.KERNEL32.dll_GetProcAddress ; sub.KERNEL32.dll_GetProcAddress ; FARPR...
0x00415311    mov     dword [var_18h], eax
```

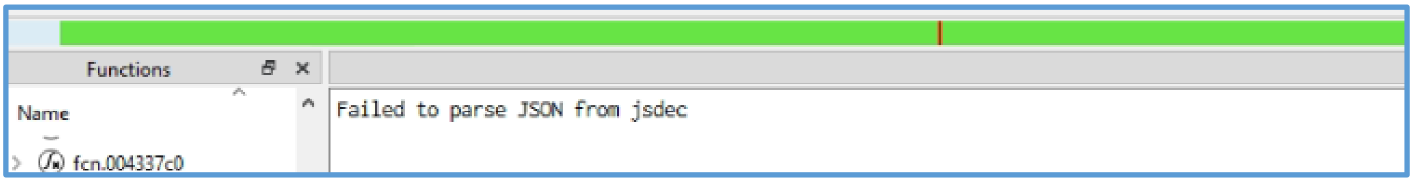
```
0x004153fb    push    0x55        ; 'U' ; 85 ; int cchLocaleName
0x004153fd    push    esi         ; LPWSTR lpLocaleName
0x004153fe    call    sub.KERNEL32.dll_GetUserDefaultLocaleName ; sub.KERNEL32.dll_GetUserDefau...
0x00415403    mov     ecx, esi
0x00415405    call    fcn.00401d05 ; fcn.00401d05
```

```
0x0041e1d7    push    0x4c        ; 'L' ; 76 ; int nIndex
0x0041e1d9    call    sub.USER32.dll_GetSystemMetrics ; sub.USER32.dll_GetSystemMetrics ; int G...
0x0041e1de    mov     dword [dwIndex], eax
0x0041e1e2    push    0x4d        ; 'M' ; 77 ; int nIndex
```

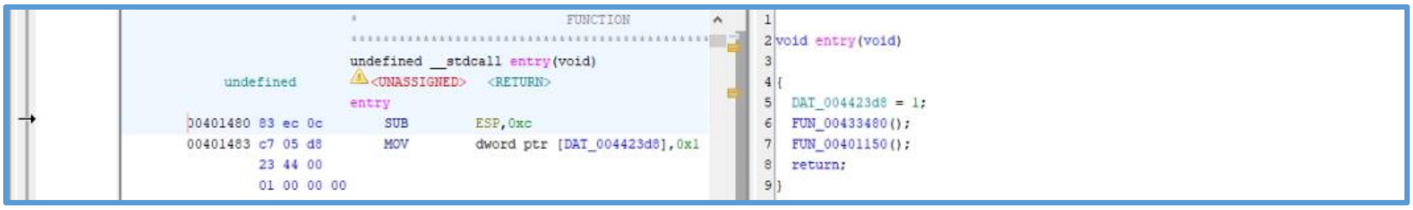
So far we can't find the location of the function that responsible the action of HTTP query to api.ipify.org, and even not the http GET query to fatfart.com, also if we search on string in cutter we can't see them at all, which may tell us that fatfart as example exist on some other function.



What we can do on that case is to use another tool for decompile the code, cutter can do that also but from time to time it just stuck on the following error, this is why I have used ghidra.

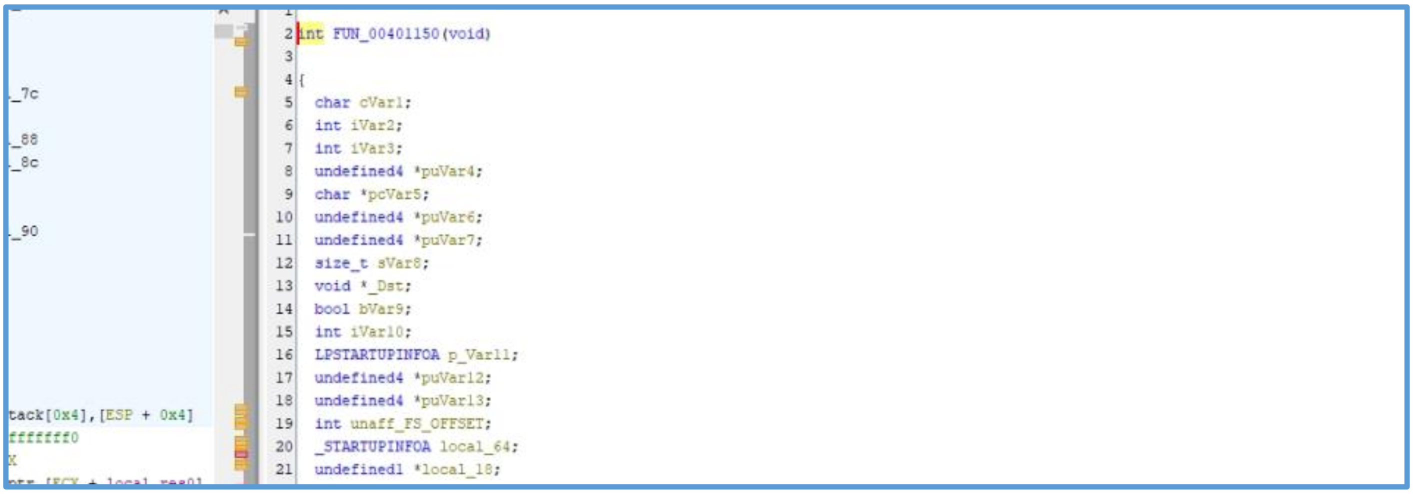


In ghidra, after create new project and import the binary file of FickerSteaer, we can see the following window that show the assembler code and the decompiles code.

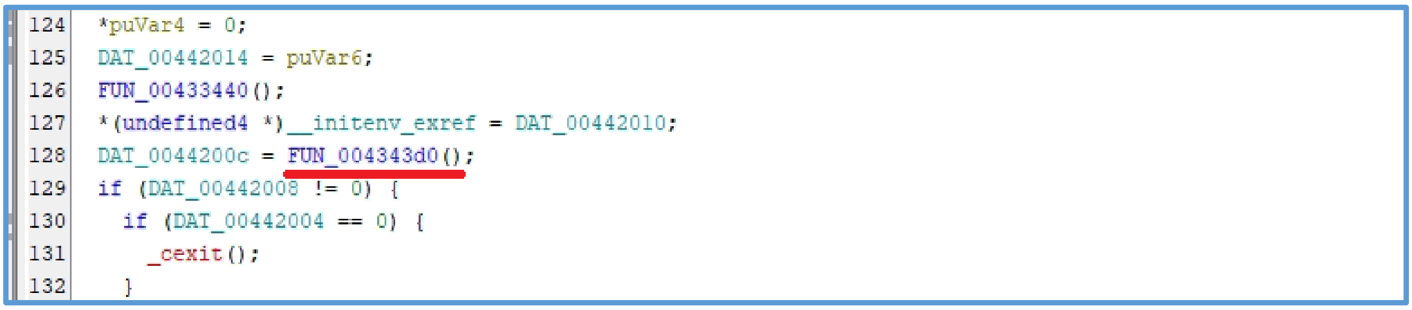


We can see the entry point which is the main function that contain two function, the first we have saw on cutter that responsible for reset stuff related to that malware while the second one in the interesting coder block.

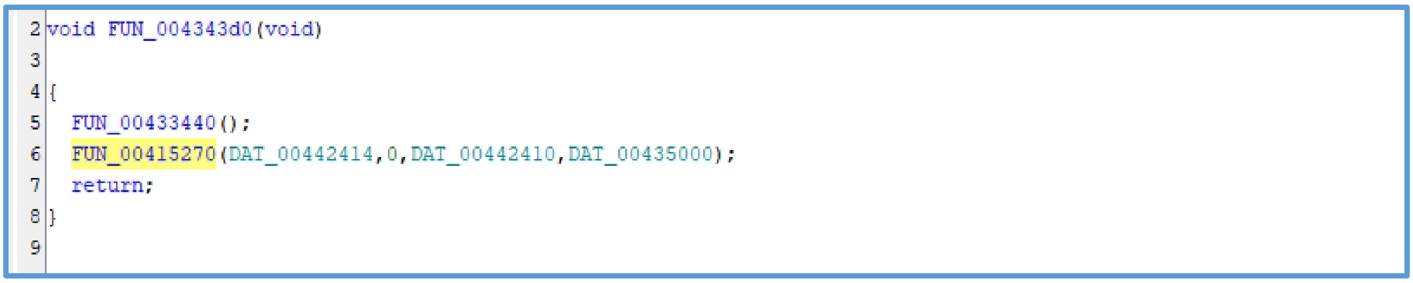
After step into the function FUN00401150(); we can see another block of code which we have seen on cutter.



By digging down deeply I was able to find another function called “call fickerstealer.4343D0”.



And that one was contain FUN_00415270 which have really long block code.



```
undefined4
FUN_00415270(undefined **param_1,undefined1 *param_2,undefined8 *param_3,HKEY param_4,HKEY param_5,
             HKEY param_6,int *param_7,ulonglong param_8,HKEY param_9,undefined *param_10,
             DWORDLONG param_11,ulonglong param_12,DWORDLONG param_13,int *param_14,HKEY param_15,
             int param_16,ulonglong param_17,HKEY param_18,DWORDLONG param_19,int *param_20,
             ulonglong *param_21,int *param_22,int param_23,DWORDLONG param_24,undefined8 *param_25,
             undefined4 param_26,HKEY__ param_27,undefined1 *param_28,undefined8 param_29,
             ulonglong param_30,ulonglong param_31,DWORDLONG param_32,int param_33,int param_34,
             undefined8 *param_35,undefined8 *param_36,undefined8 *param_37,HKEY param_38,
             ulonglong param_39,HKEY param_40,int *param_41,HKEY param_42,HKEY param_43,
             undefined8 *param_44,undefined2 param_45,ulonglong param_46,int param_47,
             undefined4 param_48,ulonglong param_49,undefined8 *param_50,undefined4 param_51,
             HKEY param_52,ulonglong *param_53,DWORD *param_54,undefined1 *param_55,HKEY param_56,
             DWORD *param_57,undefined8 param_58,undefined *param_59,HANDLE param_60,HANDLE param_61,
             HKEY__ param_62,DWORDLONG *param_63,undefined4 param_64,undefined4 param_65,
             undefined *param_66,HKEY__ param_67,undefined4 param_68,undefined8 param_69,
             HKEY param_70,int param_71,HKEY param_72,undefined4 param_73,undefined8 *param_74,
             undefined4 param_75,ulonglong *param_76,int param_77,ulonglong *param_78,
             undefined1 *param_79,DWORDLONG param_80,int param_81,uint param_82,ulonglong param_83,
             ulonglong param_84,DWORDLONG param_85,int *param_86,ulonglong *param_87,int *param_88,
             int *param_89,int param_90,undefined8 param_91,undefined4 param_92,uint param_93,
             int param_94,undefined4 param_95,undefined4 param_96,DWORDLONG param_97,
             undefined8 *param_98,uint param_99,undefined8 param_100,undefined4 param_101)
```

By scrolling down we can see several function that look like used for information container

```
uVar8 = 0x800;
BVar21 = GetComputerNameW((LPWSTR)&stack0x00000a88,(LPDWORD)&stack0x00000268);
if (BVar21 != 0) {
```

```
FUN_00420d93((int *)&param_36,(void *)uVar52,(int)in_stack_00000478 + (int)(void *)uVar52);
FUN_00421063((undefined4 *)&stack0x00000470);
GetSystemInfo((LPSYSTEM_INFO)&stack0x00000208);
FUN_00423ccc((int)&stack0x000006d0,10,0x4367b4);
```

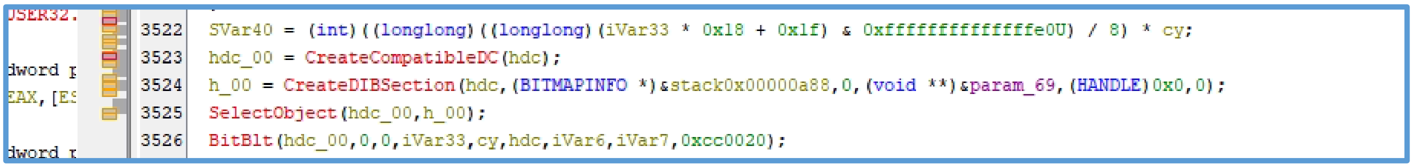
```
uVar74 = 0;
CreateToolhelp32Snapshot();
FUN_00423ccc((int)&stack0x00000270,0xc,0x436880);
FUN_00420d93((int *)&param_36,&stack0x00000270,(int)puVar44);
```

```
LVar22 = RegOpenKeyExW((HKEY)0x80000002,(LPCWSTR)uVar47,0,0x20019,(PHKEY)&stack0x00000238);
if (LVar22 == 0) {
  DVar112 = 0;
  LVar22 = RegQueryInfoKeyW(pHVar90,(LPWSTR)0x0,(LPDWORD)0x0,(LPDWORD)0x0,
                            (LPDWORD)&stack0x000006a0,(LPDWORD)0x0,(LPDWORD)0x0,(LPDWORD)0x0,
                            (LPDWORD)0x0,(LPDWORD)0x0,(LPDWORD)0x0,(PFILETIME)0x0);
```

but part of that at the end contains the following windows API:

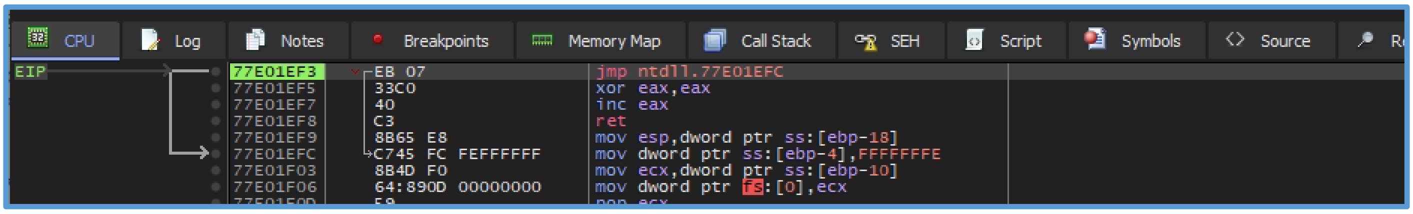
GetProcessHeap
GetSystemMetrics
GetDC
GetCurrentObject
GetObjectW
CreateCompatibleDC
CreateDIBSection
BitBlt

From that all, we can assume that the malware captures a screenshot of the victim's desktop by creating a compatible memory device context, copying the screen content into it, and storing it in memory using a DIB (Device Independent Bitmap) section. This is done through standard Windows API calls like GetDC, CreateCompatibleDC, CreateDIBSection, and BitBlt, allowing the malware to grab an image of the current screen, likely for later exfiltration.

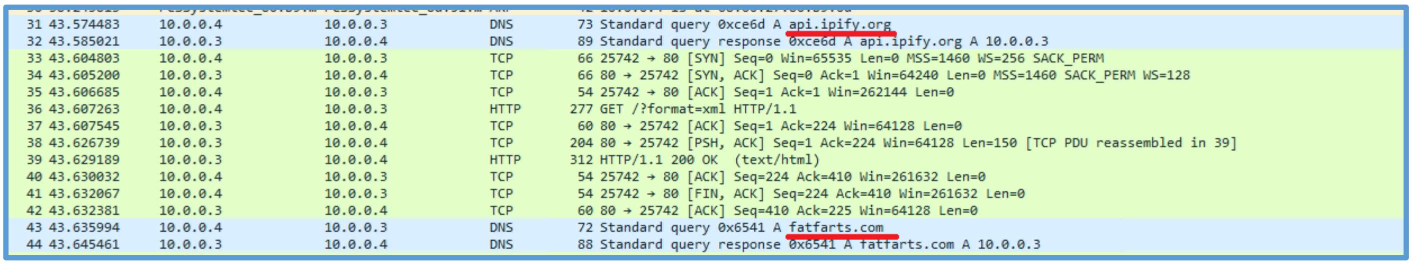


Advance Dynamic Analysis

In that step we can run debugger and see the point where the query about the domain farfart or even the API call was done, in my case I am using x32dbe, the malware start at JMP point, so we go through the flow while Wireshark are open on the background.

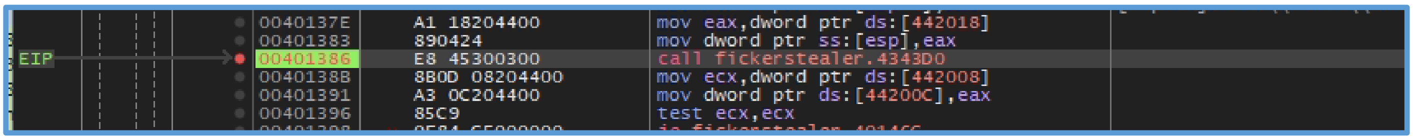


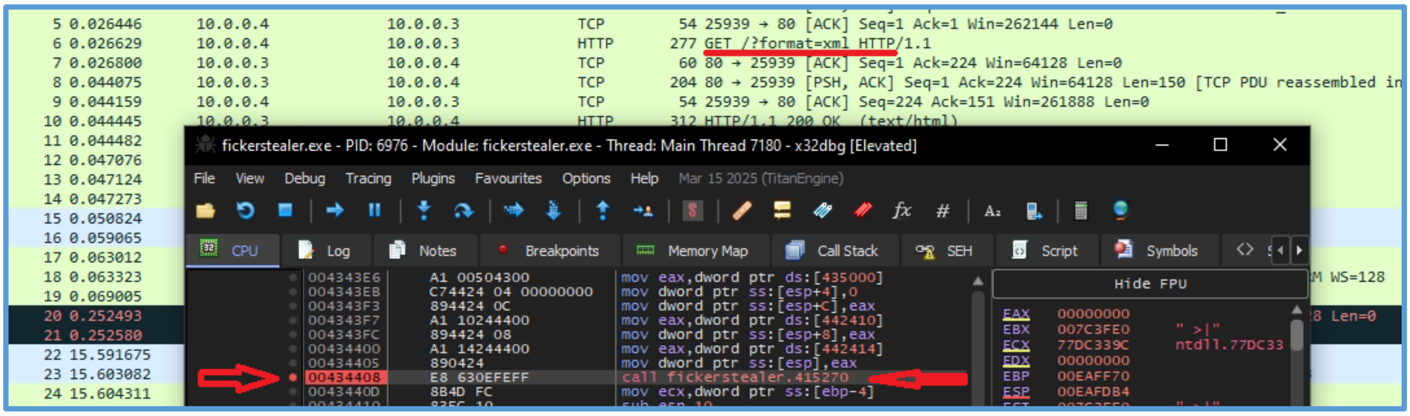
Then after several tests, we can see the following on wireshark appear, so we know the point we land that made that call.



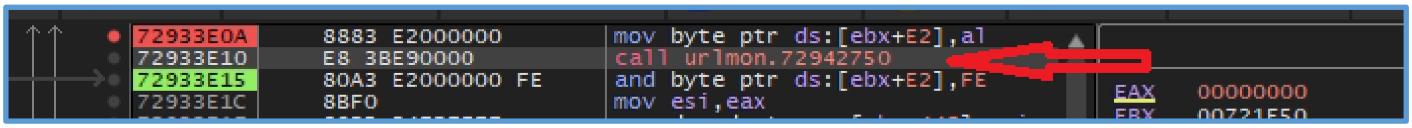
On the debugger we can clearly can see that the following function are the call who made that query in DNS.

call fickerstealer.4343D0

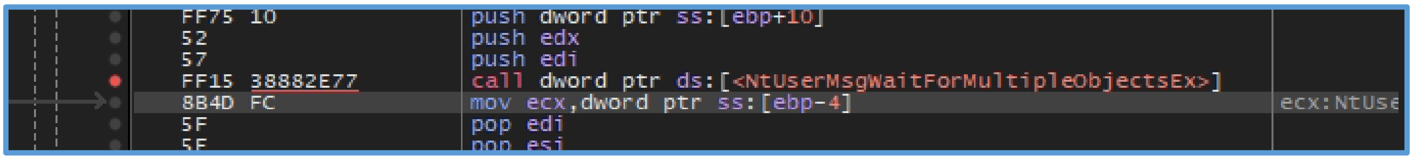


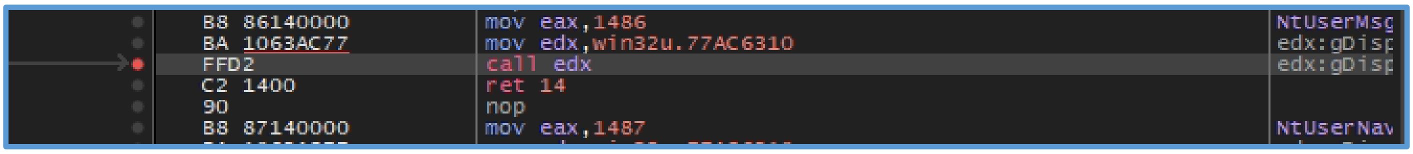


We can see that this is the same point we have found on Ghidra. Then after digging more I was able to find the following function call of urlmon, by execute it, new DNS query for ipify.org is made, so this may be the point of interesting.



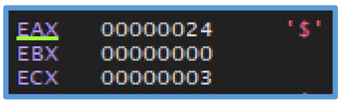
Then by stepping into that call found another call that by execute it the DNS query are made, so I have set another breakpoint on that one, right after I haver found more location of point that need to be looking at.

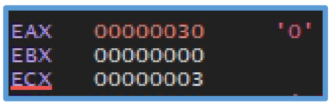


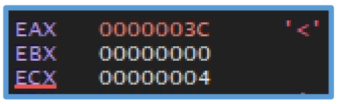


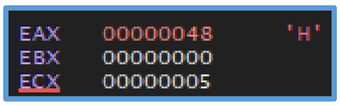
Then I was able to see loop that I think made by **NtUserMsgWaitForMultipleObjectsEx**, the query to the domain we have seen (api.ipify.org) done while that call are made. After reading more about that function, I have found that this function does **not directly perform network communication**, but serves as a **blocking mechanism that waits for multiple event handles** or **Windows messages**. Notably, immediately after this call, a DNS request for api.ipify.org was captured in Wireshark. This suggests that the function is being used to wait for the completion of a background thread or asynchronous event - most likely a network-related operation such as a DNS resolution or an HTTP request. So, it's likely that use of MsgWaitForMultipleObjectsEx in this context indicates that the malware is structured around a multithreaded or event-driven architecture, where synchronization is handled via system-level wait functions to obscure the flow of execution.

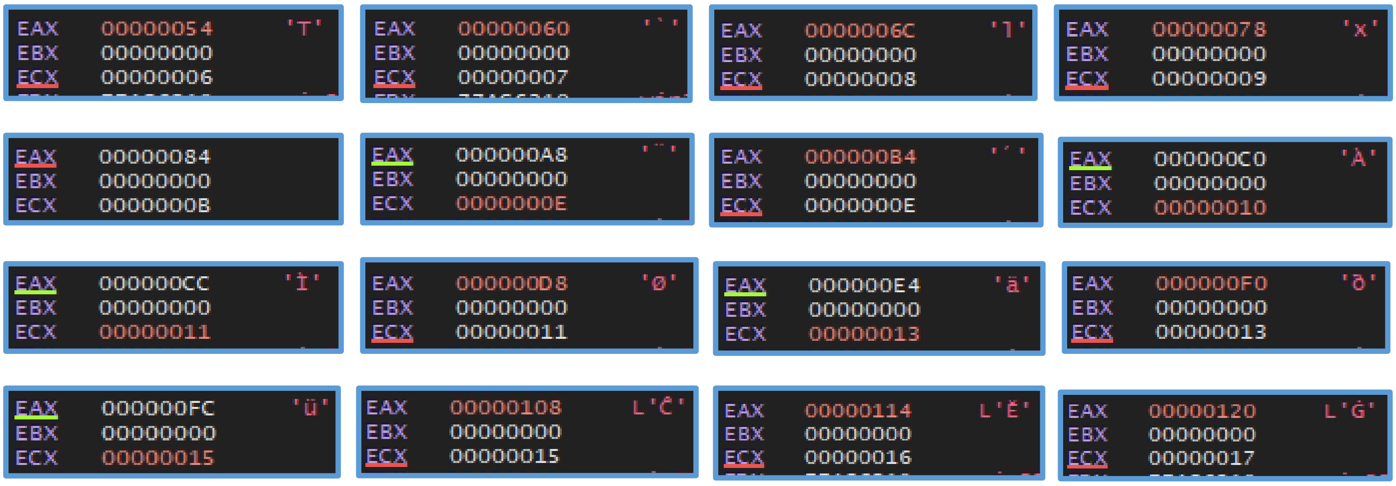
Also by going the loop step by step found in the CPU window the following letters.







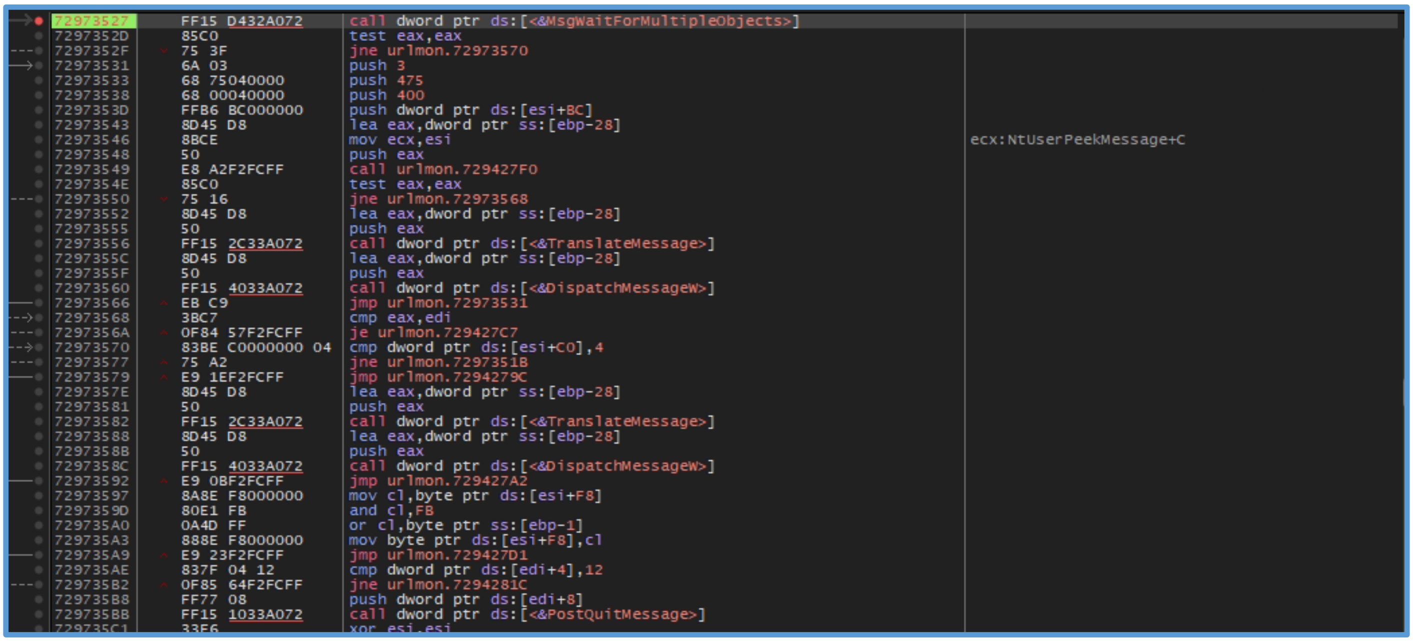




We can clearly see that not all the characters are letters, there are special ones, and also non-Latin letters.

In this case, the string observed ($o<HT\lx "'AIQaq`) appears to be obfuscated or XOR-encoded data, rather than a standard encoding format like Base64. Its use of special characters, non-alphabetic symbols, and non-Latin characters strongly suggests the presence of a custom encoding or encryption scheme.

Additionally, by examining the current assembly code in the debugger, several Windows API functions were identified in use, including **TranslateMessage**, **DispatchMessageW**, and **PostQuitMessage**. These functions are part of the standard Windows message loop used in GUI applications, allowing the program to process window messages and events. Their presence suggests that the malware either maintains a graphical user interface component or emulates a message-driven structure to manage its execution flow or deceive analysis tools. This approach can also serve to delay or obscure malicious activity, especially when combined with waiting functions like MsgWaitForMultipleObjectsEx.



At that point, I realized that continuing with a traditional debugger was becoming inefficient due to the complexity of the control flow and the frequent use of Windows message loop APIs. As a result, I decided to switch to **API Monitor**, a specialized tool that allows real-time monitoring of Windows API calls along with their parameters and return values. This tool can help to observe the malware's behavior more effectively, particularly its use of networking and system APIs, without being obstructed by the obfuscated execution flow.

After play with that found the API that create directory on the summary monitor.

7132	12:06:05.262 AM	1	fickerstealer.exe	CreateDirectoryW ("C:\ProgramData", NULL)
7133	12:06:05.262 AM	1	KERNELBASE.dll	RtlDosPathNameToRelativeNtPathName_U ("C:\ProgramData", 0x0064e9c8, NULL, 0x0064e9b0)
7134	12:06:05.262 AM	1	KERNELBASE.dll	RtlInitUnicodeString (0x0064e96c, "ntdll.dll")
7135	12:06:05.262 AM	1	KERNELBASE.dll	LdrGetDllHandle (NULL, NULL, 0x0064e96c, 0x0064e974)

Then I was able to find that it try to create png file named datasss.png, and even trying to download the external IP address from api.ipigy.org and save it on that datasss.png file.

7150	12:06:05.262 AM	1	fickerstealer.exe	URLDownloadToFileA (NULL, "http://api.ipify.org/?format=xml", "C:\ProgramData\datasss.png", 0, NULL)
7151	12:06:05.262 AM	1	Urlmon.dll	memset (0x0077bc60, 0, 620)
7152	12:06:05.262 AM	1	KERNELBASE.dll	RtlGetCurrentTransaction ( )
7153	12:06:05.262 AM	1	KERNELBASE.dll	RtlSetCurrentTransaction (NULL)

I also notive that he trying to read something from my local cache.

20307	12:06:07.121 AM	1	KERNELBASE.dll	wcsncmp ("C:\Users\Default\AppData\Local\Microsoft\Windows\INetCache", "\\?\", 4)
20308	12:06:07.121 AM	1	KERNELBASE.dll	iswalpha (67)
20309	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("C:\Users\Default\AppData\Local\Microsoft\Windows\INetCache", '\')
20310	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("\Users\Default\AppData\Local\Microsoft\Windows\INetCache", '\')
20311	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("Users\Default\AppData\Local\Microsoft\Windows\INetCache", '\')
20312	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("\Default\AppData\Local\Microsoft\Windows\INetCache", '\')
20313	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("Default\AppData\Local\Microsoft\Windows\INetCache", '\')
20314	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("\AppData\Local\Microsoft\Windows\INetCache", '\')
20315	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("AppData\Local\Microsoft\Windows\INetCache", '\')
20316	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("\Local\Microsoft\Windows\INetCache", '\')
20317	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("Local\Microsoft\Windows\INetCache", '\')
20318	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("\Microsoft\Windows\INetCache", '\')
20319	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("Microsoft\Windows\INetCache", '\')
20320	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("\Windows\INetCache", '\')
20321	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("Windows\INetCache", '\')
20322	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("\INetCache", '\')
20323	12:06:07.121 AM	1	KERNELBASE.dll	wcschr ("INetCache", '\')

20375	12:06:07.121 AM	1	KERNELBASE.dll	NtClose (0x000002c0)
20376	12:06:07.121 AM	1	WININET.dll	_wcsicmp ("C:\Users\Default\AppData\Local\Microsoft\Windows\INetCache", "C:\Users\zwerd\AppData\Local\Microsoft\Windows\INetCache")
20377	12:06:07.121 AM	1	KERNELBASE.dll	RtlRunOnceExecuteOnce (0x732695d8, 0x730f1520, NULL, NULL)

The malware sample uses the **CreateMutexA** API call to create one or more uniquely named mutex objects (e.g., "ktykftykftyktfyk"). This technique is commonly employed by malware to ensure that only a single instance of the malware runs at any given time.

After attempting to create the mutex, the malware may call GetLastError to check whether the mutex already exists. If the error code ERROR_ALREADY_EXISTS is returned, the malware can assume it is already running and terminate itself to avoid redundant execution.

This behavior is a form of self-regulation and can also serve as a basic anti-analysis or anti-sandbox mechanism.

3802	4:25:21.769 AM	1	fickerstealer.exe	GetProcAddress (0x773e0000, "CreateMutexA")
3808	4:25:21.769 AM	1	fickerstealer.exe	GetProcAddress (0x773e0000, "GetLastError")
3814	4:25:21.769 AM	1	fickerstealer.exe	CreateMutexA (NULL, TRUE, "ktykftykftyktfyk")
3821	4:25:21.769 AM	1	fickerstealer.exe	CreateMutexA (NULL, TRUE, "sgsre")
3828	4:25:21.769 AM	1	fickerstealer.exe	CreateMutexA (NULL, TRUE, "segrserg")
3835	4:25:21.769 AM	1	fickerstealer.exe	CreateMutexA (NULL, TRUE, "jrtdtjrjdrtj")

39430	12:06:09.059 AM	1	fickerstealer.exe	CreateFileW ("C:\ProgramData\datasss.png", GENERIC_READ, FILE_SHARE_DELETE | FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL)
39431	12:06:09.059 AM	1	KERNELBASE.dll	RtlInitUnicodeStringEx (0x0064e8e0, "C:\ProgramData\datasss.png")
39432	12:06:09.059 AM	1	KERNELBASE.dll	RtlDosPathNameToRelativeNtPathName_U_WithStatus ("C:\ProgramData\datasss.png", 0x0064e8e0, NULL, 0x0064e918)
39433	12:06:09.059 AM	1	KERNELBASE.dll	NtCreateFile (0x0064e8d4, FILE_READ_ATTRIBUTES | GENERIC_READ | SYNCHRONIZE, 0x0064e900, 0x0064e8d8, NULL, 0, FILE_SHARE_DELETE | FILE_SHARE_READ | FILE_SHARE_WRITE, F
39434	12:06:09.059 AM	1	KERNELBASE.dll	RtlReleaseRelativeName (0x0064e918)
39435	12:06:09.059 AM	1	KERNELBASE.dll	RtlFreeHeap (0x00760000, 0, 0x007d6758)

Then clearly can see the way for interation with fatfarts.com that start by getting the address information of that domain.

39547	12:06:09.059 AM	1	fickerstealer.exe	memcpy (0x007ca200, 0x007ca1b8, 12)
39548	12:06:09.059 AM	1	fickerstealer.exe	getaddrinfo ("fatfarts.com", NULL, 0x0064e9f0, 0x0064ea30)
39549	12:06:12.027 AM	1	WS2_32.dll	RtlIpv6StringToAddressExW ("fatfarts.com", 0x0064e130, 0x0064e140, 0x0064e12a)

39548	12:06:09.059 AM	1	fickerstealer.exe	getaddrinfo ("fatfarts.com", NULL, 0x0064e9f0, 0x0064ea30)
39549	12:06:12.027 AM	1	WS2_32.dll	RtlIpv6StringToAddressExW ("fatfarts.com", 0x0064e130, 0x0064e140, 0x0064e12a)
39550	12:06:12.027 AM	1	WS2_32.dll	RtlIpv4StringToAddressW ("fatfarts.com", TRUE, 0x0064e0fc, 0x0064e114)
39551	12:06:12.027 AM	1	KERNELBASE.dll	RtlRunOnceExecuteOnce (0x766c84e0, 0x7668a7c0, NULL, NULL)
39552	12:06:12.027 AM	1	KERNELBASE.dll	RtlRunOnceExecuteOnce (0x766c84e0, 0x7668a7c0, NULL, NULL)
39553	12:06:12.027 AM	1	KERNELBASE.dll	NtWaitForSingleObject (0x000002dc, FALSE, 0x0064d5f8)

By trying to read that datasss.png, I was able to see that it contain the default page of inetsim.

datasss.png

```html
<html>
  <head>
    <title>INetSim default HTML page</title>
  </head>
  <body>
    <p></p>
    <p align="center">This is the default HTML page for INetSim HTTP server fake mode.</p>
    <p align="center">This file is an HTML document.</p>
  </body>
</html>
```

Based on the observed behavior, the analyzed FickerStealer sample performs only **initial setup actions** and does not appear to reach the stage of **collecting sensitive information** from the system. The malware loads necessary libraries, gathers basic environment data (such as the system locale), and makes an HTTP request to **api.ipify.org to obtain the external IP address**, which is **saved to a file named datasss.png**. It then initiates a connection to the command-and-control (C2) server at **fatfarts.com**. However, there is no evidence of attempts to access files containing credentials (such as browser Login Data, password vaults, or cryptocurrency wallets), nor any calls to system functions like CryptUnprotectData or sqlite3_open. These findings suggest that the information-stealing phase is not triggered in this execution, potentially due to a missing response from the C2 server or reliance on external configuration data to activate the data exfiltration logic.

Indicators of Compromise (IOC's table)

From what we have found so far we have several indication for detect that malware, so I came up with the following table.

Indicator Type	Value
Filename	SecuriteInfo.com.Trojan.Packed2.42600.30573.20195.exe
MD5	9ada122303e6dee1c0f0171bf2e59253
SHA1	b9f2cac95510c1199083504e0ae57fd14bf559d5
SHA256	b3cfbb058c0ecbd7da7f5bdd740fa729f7b0d9cf61f93b32750ce06745abc24c
Mutex	ktykftykftyktfyk
File (Created)	C:\ProgramData\datasss.png
Domain (C2)	fatfarts.com
Domain (IP lookup)	api.ipify.org
File Accessed	C:\ProgramData\datasss.png

Detection Rules & Signatures

So now we could make some YARA rule for detect that IOC's, since we have several indicators, we can used them together.

```
rule FickerStealer_Generic
{
    meta:
        description = "Detects FickerStealer sample based on mutex, filename, and known indicators"
        author = "Zw3rd"
        hash_sha256 = "b3cfbb058c0ecbd7da7f5bdd740fa729f7b0d9cf61f93b32750ce06745abc24c"
        malware_family = "FickerStealer"
        date = "2025-06-08"

    strings:
        $mutex = "ktykftykftyktfyk"
        $domain1 = "fatfarts.com"
        $domain2 = "api.ipify.org"
        $filename = "datasss.png"

    condition:
        uint32(0) == 0x5A4D and
        (all of ($mutex, $domain1, $domain2, $filename))
}
```