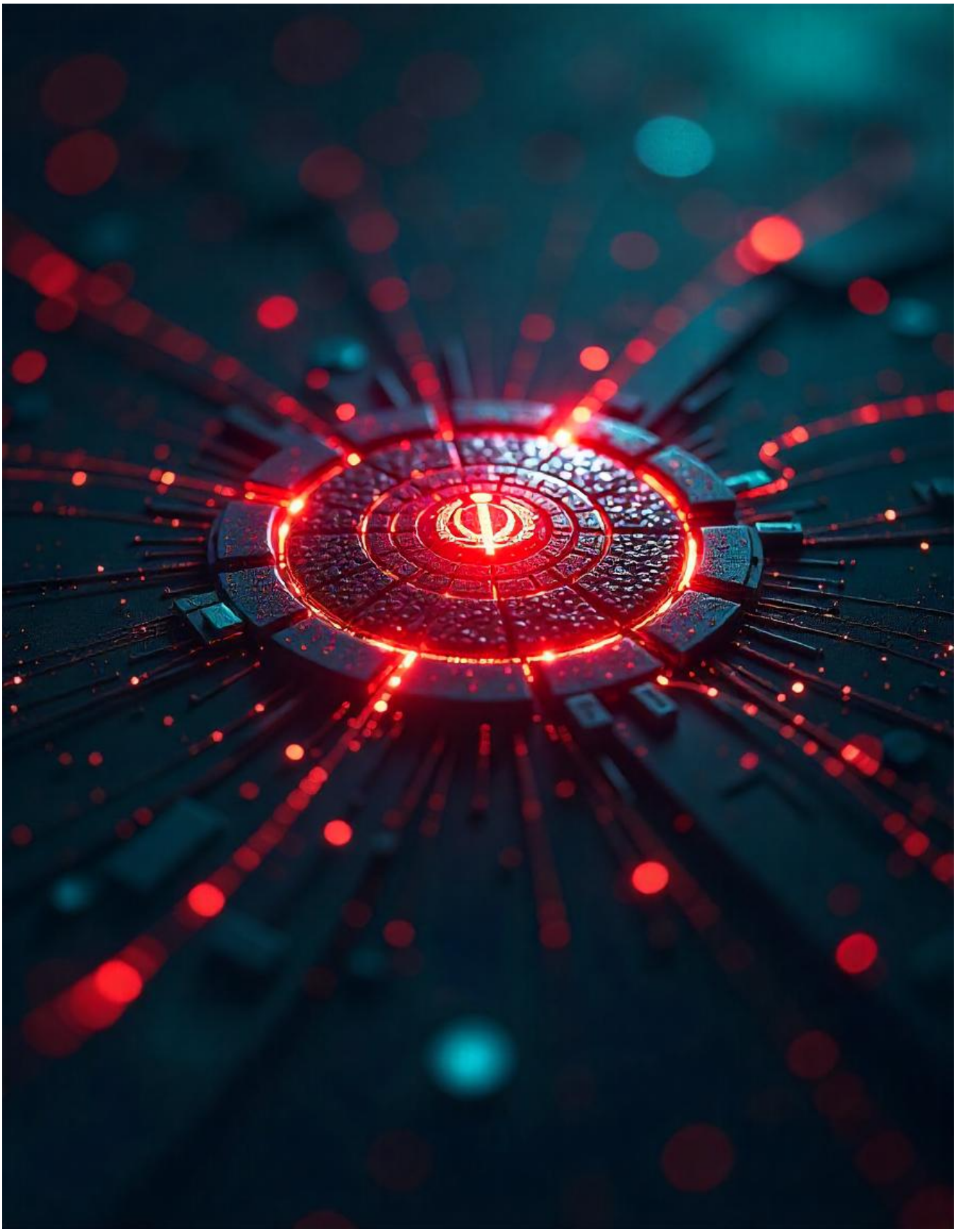
Malware Analysis - NanoCore

0xmrmagnezi.github.io/malware analysis/NanoCore/

February 27, 2025



4 minute read

Sample:

1d52c927094cc5862349a1b81ddaf10c

Background

NanoCore is a modular remote access tool developed in .NET that can be used to spy on victims and steal information. It has been used for a while by numerous criminal actors, as well as by nation-state threat actors such as the Iranian group APT33.

Static Analysis - Stage 1

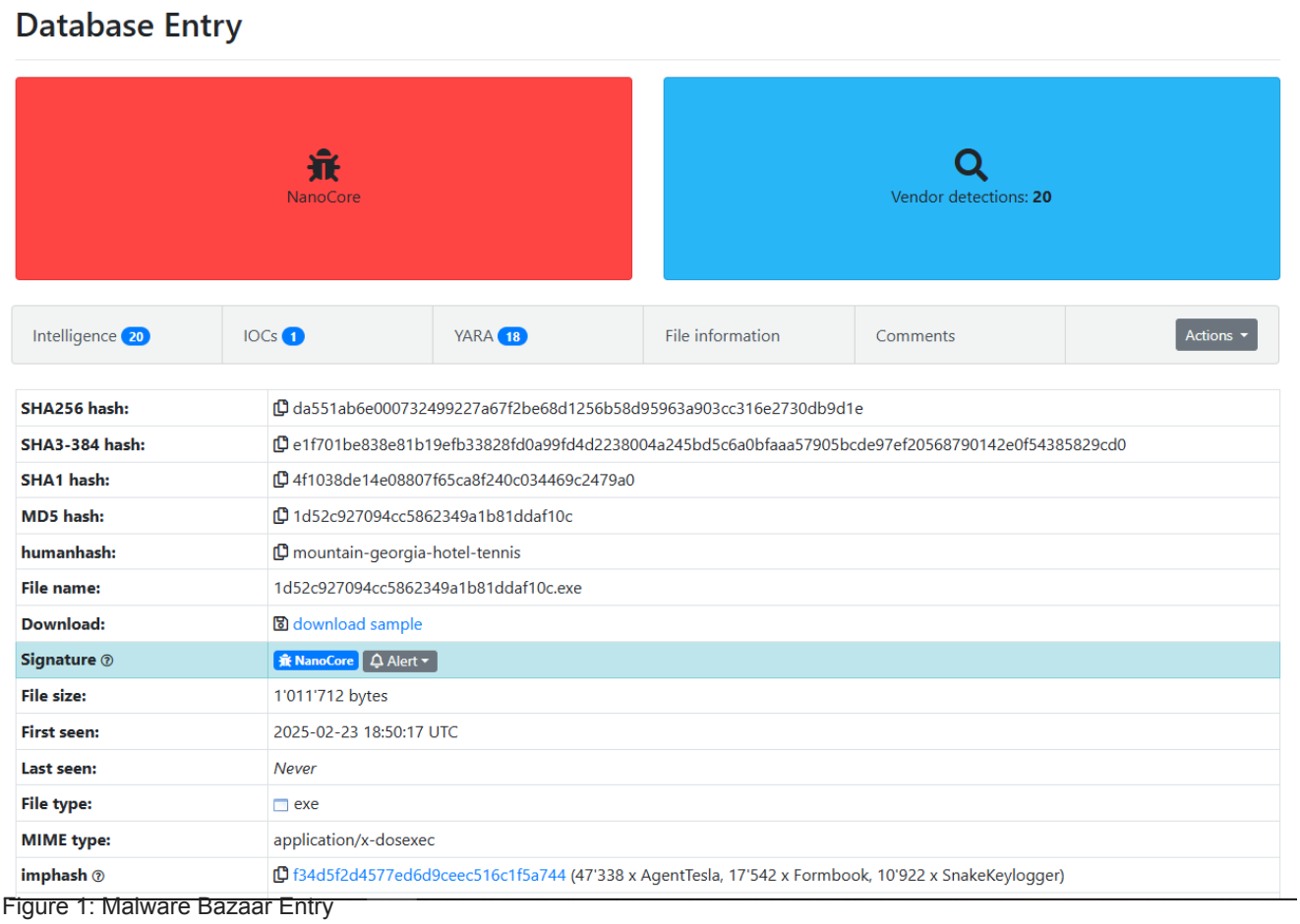


Figure 1: Malware Bazaar Entry

This sample is detected by 20 vendors and contains multiple stages, with the analysis revealing key details, including the extraction of the malware's configuration.

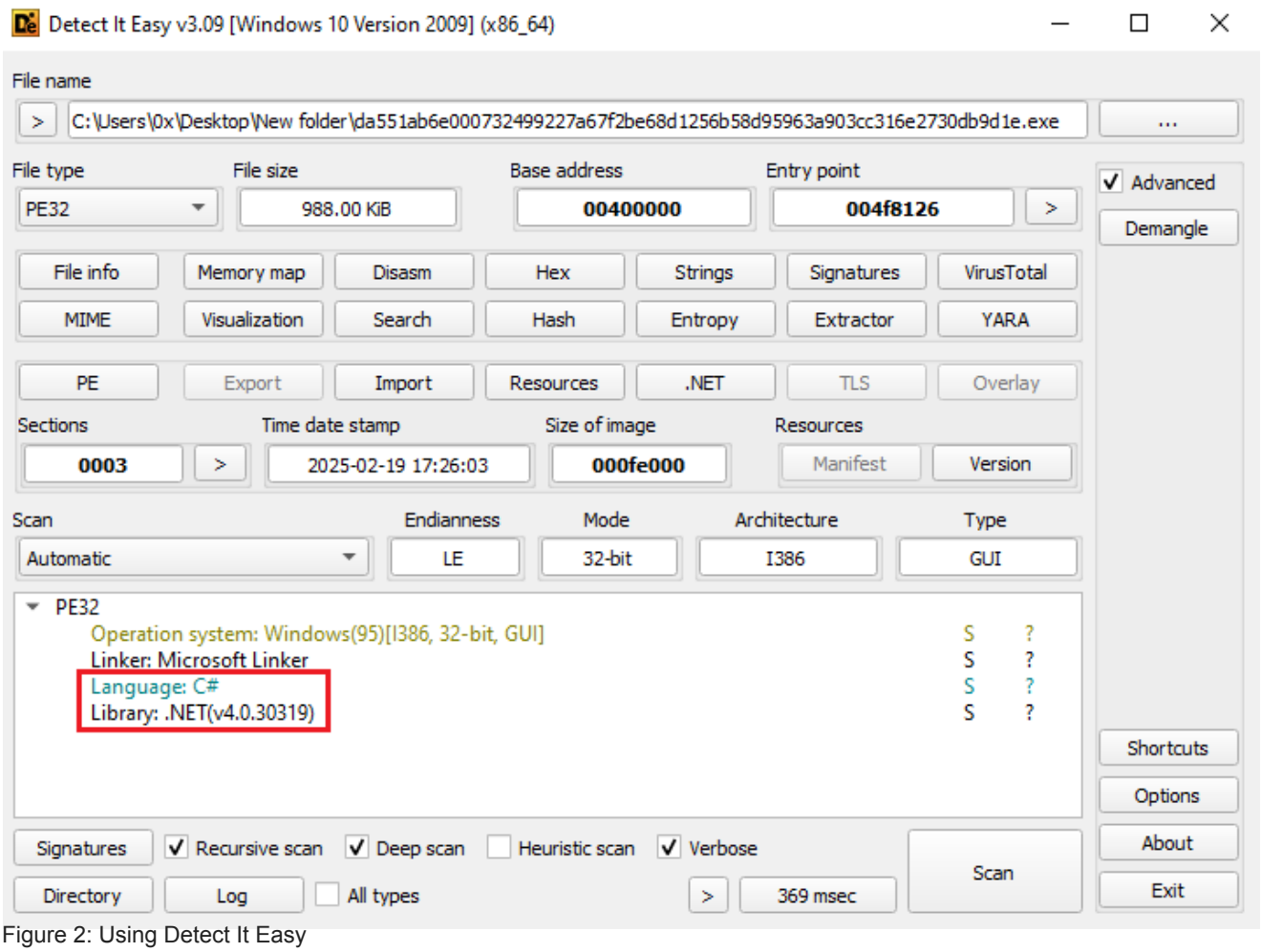


Figure 2: Using Detect It Easy

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.

md5	1d52c927094cc5862349a1b81ddaf10c
sha1	4f1038de14e08807f65ca8f240c034469c2479a0
sha256	da551ab6e000732499227a67f2be68d1256b58d95963a903cc316e2730db9d1e
analysis	static
os	any
format	pe
arch	i386
path	C:/Users/0x/Desktop/New folder/da551ab6e000732499227a67f2be68d1256b58d95963a903cc316e2730db9d1e.exe

ATT&CK Tactic	ATT&CK Technique
COLLECTION	Clipboard Data T1115
DISCOVERY	File and Directory Discovery T1083

MBC Objective	MBC Behavior
DISCOVERY	Analysis Tool Discovery::Process detection [B0013.001] File and Directory Discovery [E1083]
FILE SYSTEM	Writes File [C0052]
PROCESS	Suspend Thread [C0055]

Capability	Namespace
reference analysis tools strings	anti-analysis
access .NET resource (2 matches)	executable/resource
check clipboard data (3 matches)	host-interaction/clipboard
check if file exists	host-interaction/file-system/exists
write file on Windows	host-interaction/file-system/write
allocate unmanaged memory in .NET	host-interaction/memory
manipulate unmanaged memory in .NET (6 matches)	host-interaction/memory
suspend thread (2 matches)	host-interaction/thread/suspend
unmanaged call (18 matches)	runtime
compiled to the .NET platform	runtime/dotnet

Figure 3: Using CAPA

Based on the CAPA output, I speculate that this is likely only the first stage, with additional stages potentially following. Furthermore, the output suggests the presence of anti-analysis techniques.

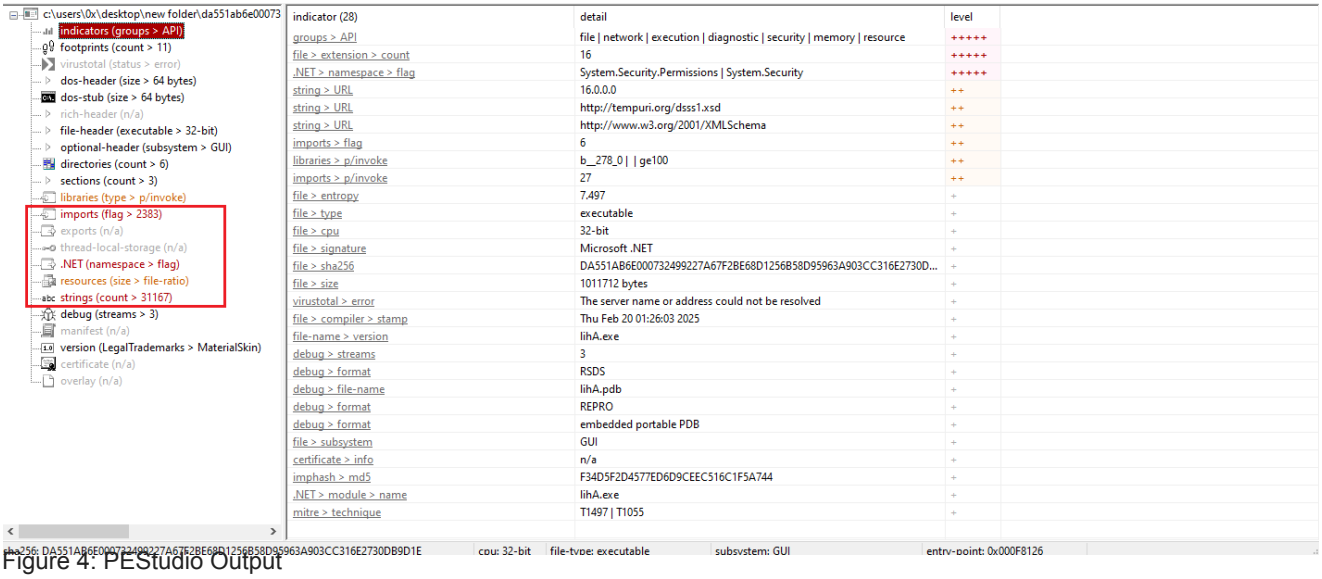


Figure 4: PEStudio Output

As shown in Figure 4, multiple strings and indicators are flagged by PeStudio, providing a better understanding of the malware's functionality. It is most likely packed and contains Stage 2.

This malware includes anti-debugging techniques, making it more challenging to statically extract the unpacked malware. As a result, I decided to take a different approach. The second stage was dynamically extracted from memory after the malware was executed.

Dynamic Analysis - Stage 1

The behavior of the malware was as follows:

A process for the first executed program was created. After a few seconds, the process was terminated, and a new process was created under the same name as the first process.

Figure 5: New Process Creation

From this process, a tool was executed to extract any suspicious artifacts, such as implemented PE, as shown in Figure 6.

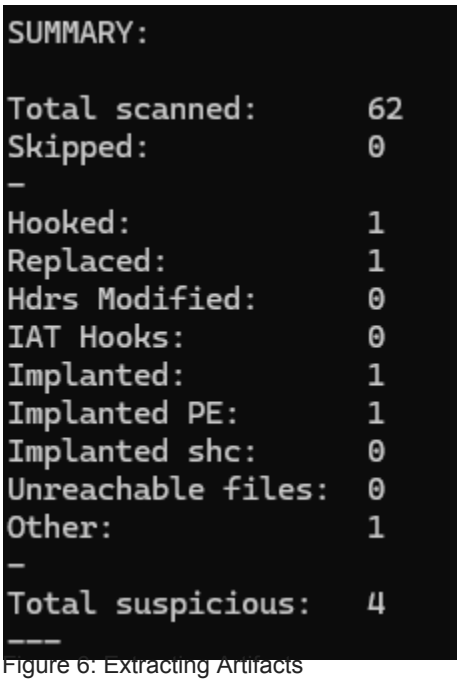


Figure 6: Extracting Artifacts

Static Analysis - Stage 2

The newly outputted PE was further analyzed using various tools.

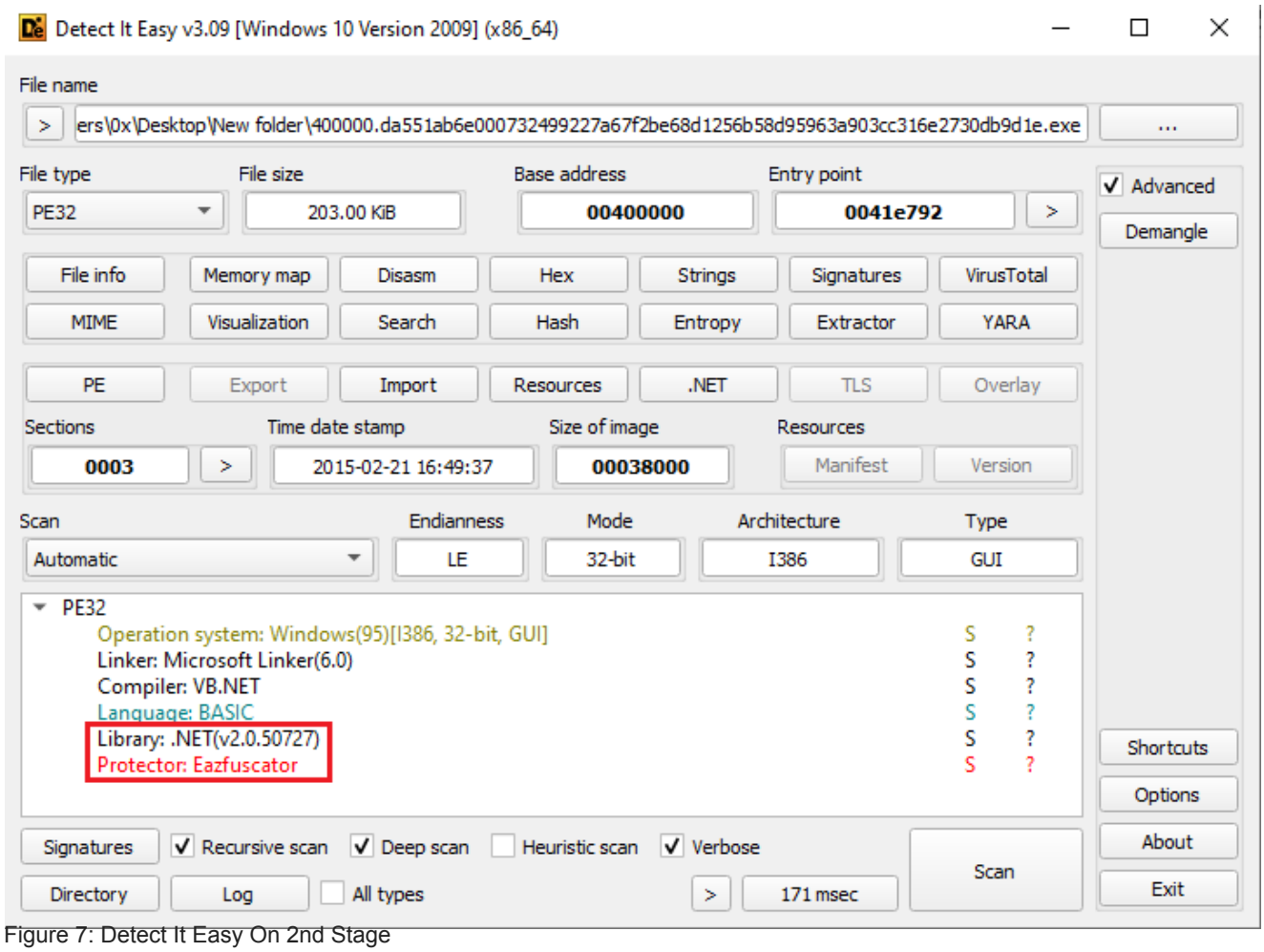


Figure 7: Detect It Easy On 2nd Stage

From the output of DIE, it was observed that the malware was written in .NET and protected with Eazfuscator, a tool designed to obfuscate .NET code to prevent reverse engineering and tampering.

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	Modify Registry T1112 Reflective Code Loading T1620
DISCOVERY	Account Discovery T1087 File and Directory Discovery T1083 Query Registry T1012 System Information Discovery T1082 System Owner/User Discovery T1033

MBC Objective	MBC Behavior
COMMAND AND CONTROL	C2 Communication::Receive Data [B0030.002] C2 Communication::Send Data [B0030.001]
COMMUNICATION	DNS Communication::Resolve [C0011.001] Socket Communication::Create TCP Socket [C0001.011] Socket Communication::Create UDP Socket [C0001.010] Socket Communication::Receive Data [C0001.006] Socket Communication::Send Data [C0001.007]
CRYPTOGRAPHY	Cryptographic Hash::MD5 [C0029.001] Generate Pseudo-random Sequence::Use API [C0021.003]
DISCOVERY	Code Discovery::Inspect Section Memory Permissions [B0046.002] File and Directory Discovery [E1083] System Information Discovery [E1082]
FILE SYSTEM	Copy File [C0045] Create Directory [C0046] Delete Directory [C0048] Delete File [C0047] Read File [C0051] Writes File [C0052]
OPERATING SYSTEM	Console [C0033] Registry::Delete Registry Value [C0036.007] Registry::Query Registry Key [C0036.005] Registry::Query Registry Value [C0036.006] Registry::Set Registry Key [C0036.001]
PROCESS	Create Mutex [C0042] Create Process [C0017] Suspend Thread [C0055] Terminate Process [C0018]

Figure 8: CAPA Output

From the CAPA output, we can observe significantly more details than in the previous analysis, as this is the unpacked version, revealing many more techniques and behaviors.

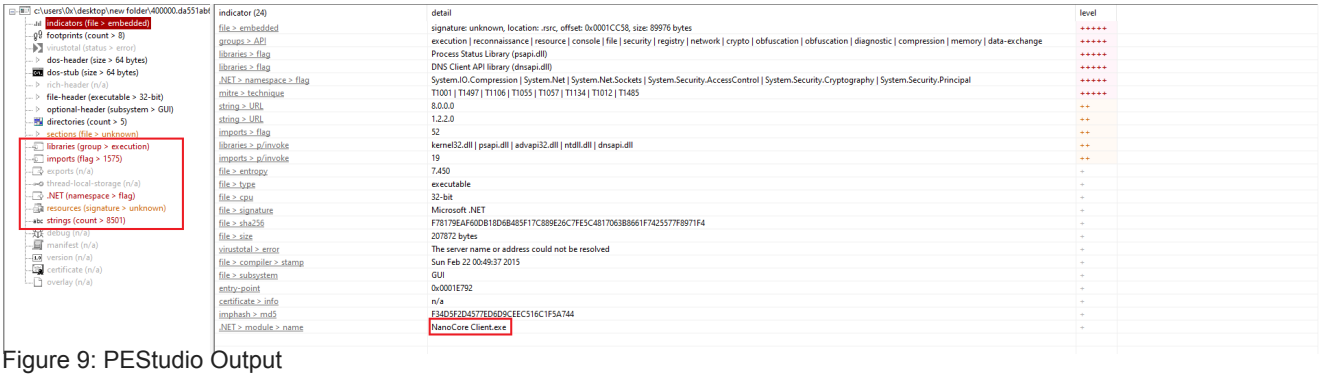


Figure 9: PEStudio Output

This second stage was analyzed in dnSpy, a popular tool for decompiling and inspecting .NET assemblies, allowing for a deeper examination of the code and its behavior. As shown in Figure 10, this is the entry point of the malware.

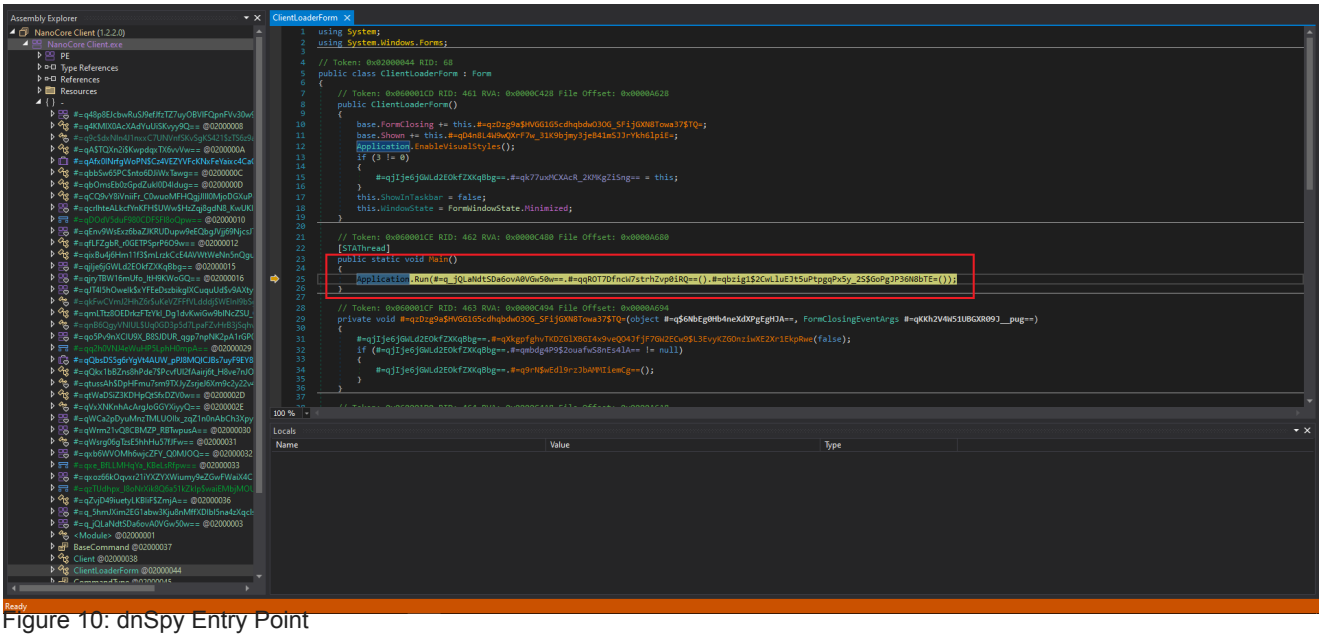


Figure 10: dnSpy Entry Point

After some time spent debugging, I was able to locate and extract the malware's configuration, as shown in Figure 11.

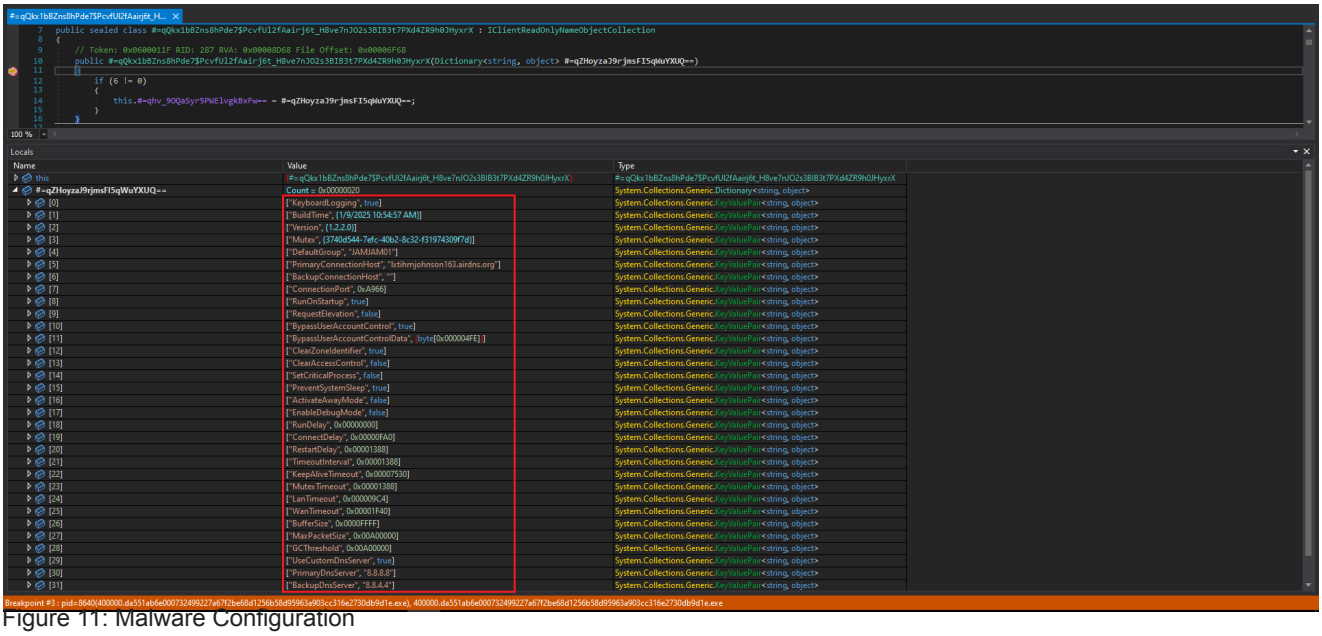


Figure 11: Malware Configuration

Details such as the C2 domain, port, run-on startup, and mutex were observed. A mutex (short for mutual exclusion) is a synchronization object used to prevent multiple processes from accessing shared resources simultaneously, often used by malware to ensure a single instance of itself runs on the system.

Decoded Malware Configuration:

```
+               [0]     ["KeyboardLogging", true]
+               [1]     ["BuildTime", {1/9/2025 10:54:57 AM}]
+               [2]     ["Version", {1.2.2.0}]
+               [3]     ["Mutex", {3740d544-7efc-40b2-8c32-f31974309f7d}]
+               [4]     ["DefaultGroup", "JAMJAM01"]
+               [5]     ["PrimaryConnectionHost", "lxtihmjohnson163[.]airdns[.]org"]
+               [6]     ["BackupConnectionHost", ""]
+               [7]     ["ConnectionPort", 43366]
+               [8]     ["RunOnStartup", true]
+               [9]     ["RequestElevation", false]
+               [10]    ["BypassUserAccountControl", true]
+               [11]    ["BypassUserAccountControlData", {byte[0x000004FE]}]
+               [12]    ["ClearZoneIdentifier", true]
+               [13]    ["ClearAccessControl", false]
+               [14]    ["SetCriticalProcess", false]
+               [15]    ["PreventSystemSleep", true]
+               [16]    ["ActivateAwayMode", false]
+               [17]    ["EnableDebugMode", false]
+               [18]    ["RunDelay", 0]
+               [19]    ["ConnectDelay", 4000]
+               [20]    ["RestartDelay", 5000]
+               [21]    ["TimeoutInterval", 5000]
+               [22]    ["KeepAliveTimeout", 30000]
+               [23]    ["MutexTimeout", 5000]
+               [24]    ["LanTimeout", 2500]
+               [25]    ["WanTimeout", 8000]
+               [26]    ["BufferSize", 65535]
+               [27]    ["MaxPacketSize", 10485760]
+               [28]    ["GCThreshold", 10485760]
+               [29]    ["UseCustomDnsServer", true]
+               [30]    ["PrimaryDnsServer", "8.8.8.8"]
+               [31]    ["BackupDnsServer", "8.8.4.4"]
```

Dynamic Analysis - Stage 2

After running the malware, more information was revealed, such as registry manipulation, changes to file locations, access to the camera, and keylogging techniques.

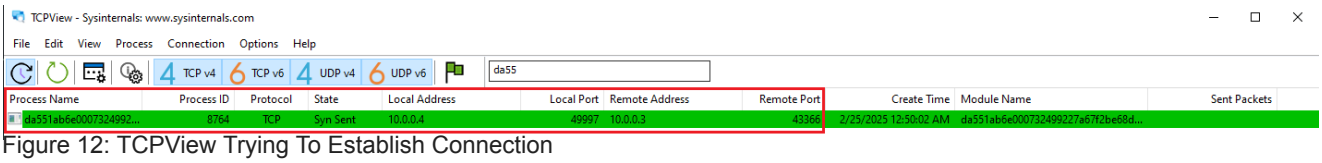


Figure 12: TCPView Trying To Establish Connection

After a restart, the malware starts from a new location under the name “ddpss”, attempting to impersonate a legitimate process.

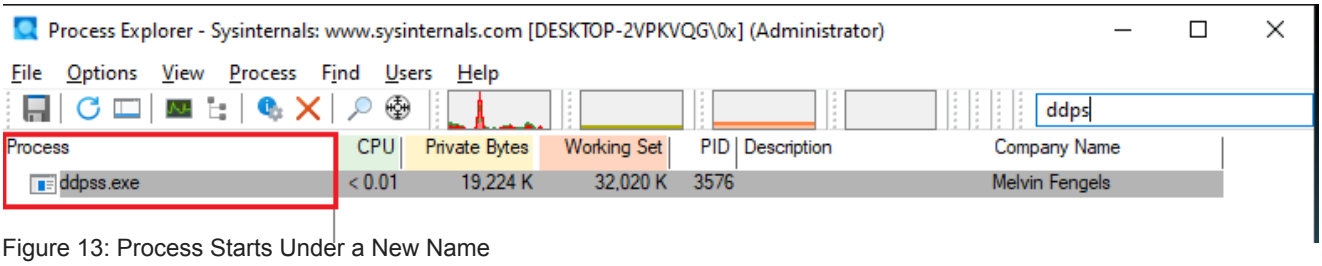


Figure 13: Process Starts Under a New Name

In Autoruns, it was observed that a new entry was added under ‘Logon,’ indicating that this process will start after the computer boots up.

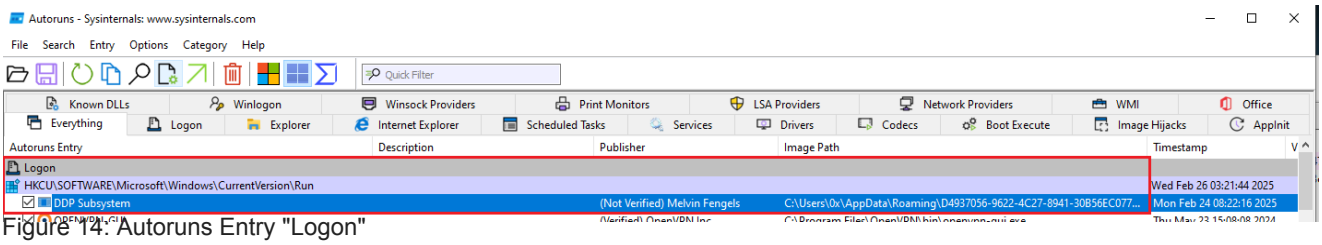


Figure 14: Autoruns Entry "Logon"

Network Analysis

Using Wireshark, a C2 domain was discovered, which matched the domain found in the malware’s configuration, confirming that this is the real configuration for the malware.

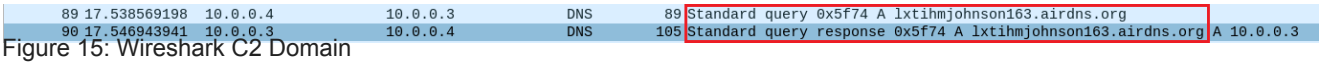


Figure 15: Wireshark C2 Domain

Summary

NanoCore is a remote access Trojan (RAT) linked to Iranian threat actor APT33. It features multiple stages, anti-analysis techniques, and obfuscation. During analysis, I extracted its configuration, which revealed C2 domains, mutexes, bypass UAC, and other key details. The malware ensures persistence across reboots by impersonating legitimate processes and manipulating the registry.

IOCs

- Hash:

  ```
  1d52c927094cc5862349a1b81ddaf10c
  6a6a79c0c2208774bfb564576ee1c25c
  ```

- Domain:

  ```
  lxtihmjohnson163[.]airdns[.]org
  tunhost[.]duckdns[.]org
  ```

- IP:

  `213[.]152[.]161[.]114`