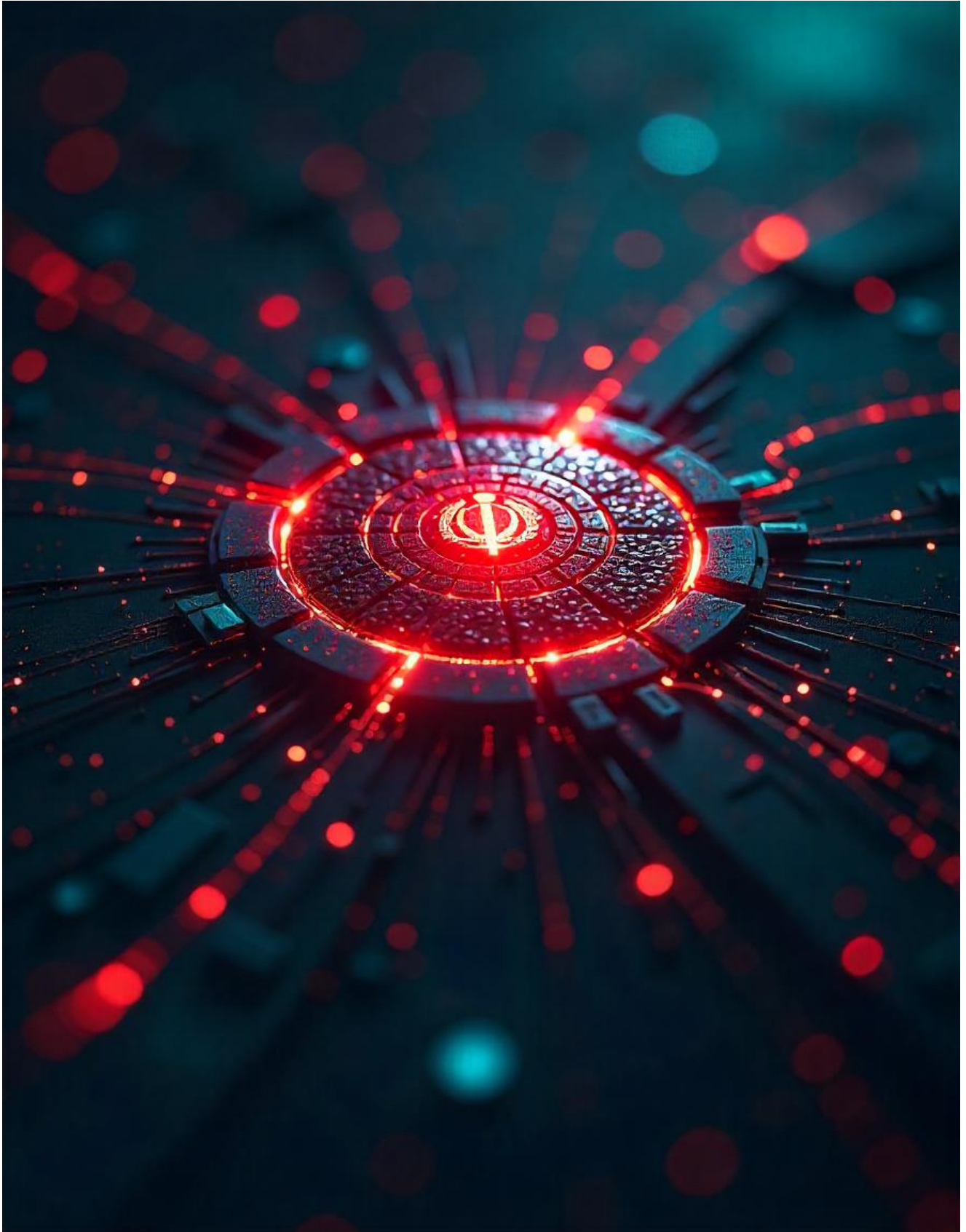


Malware Analysis - NanoCore

 [0xmrماغهzi.github.io/malware analysis/NanoCore/](https://0xmrماغهzi.github.io/malware%20analysis/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

Database Entry


NanoCore


Vendor detections: 20











Intelligence 20	IOCs 1	YARA 18	File information	Comments	Actions
SHA256 hash:	 da551ab6e000732499227a67f2be68d1256b58d95963a903cc316e2730db9d1e				
SHA3-384 hash:	 e1f701be838e81b19efb33828fd0a99fd4d2238004a245bd5c6a0bfaaa57905bcde97ef20568790142e0f54385829cd0				
SHA1 hash:	 4f1038de14e08807f65ca8f240c034469c2479a0				
MD5 hash:	 1d52c927094cc5862349a1b81ddaf10c				
humanhash:	 mountain-georgia-hotel-tennis				
File name:	1d52c927094cc5862349a1b81ddaf10c.exe				
Download:	 download sample				
Signature ⓘ	 NanoCore  Alert				
File size:	1'011'712 bytes				
First seen:	2025-02-23 18:50:17 UTC				
Last seen:	Never				
File type:	 exe				
MIME type:	application/x-dosexec				
imphash ⓘ	 f34d5f2d4577ed6d9ceec516c1f5a744 (47'338 x AgentTesla, 17'542 x Formbook, 10'922 x SnakeKeylogger)				

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.

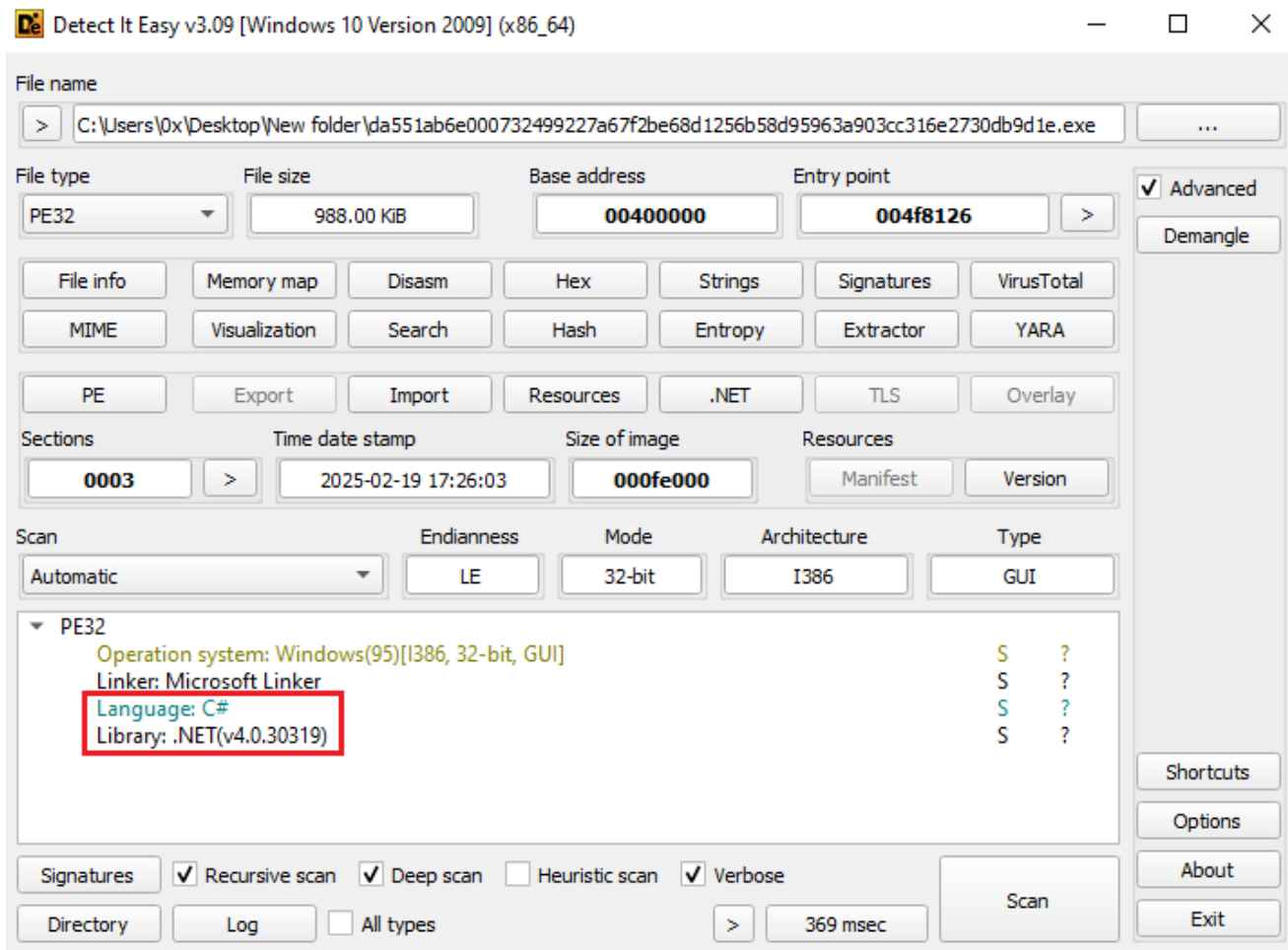


Figure 2: Using Detect It Easy

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 sha1 sha256 analysis os format arch path	1d52c927094cc5862349a1b81ddaf10c 4f1038de14e08807f65ca8f240c034469c2479a0 da551ab6e000732499227a67f2be68d1256b58d95963a903cc316e2730db9d1e static any pe i386 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 access .NET resource (2 matches) check clipboard data (3 matches) check if file exists write file on Windows allocate unmanaged memory in .NET manipulate unmanaged memory in .NET (6 matches) suspend thread (2 matches) unmanaged call (18 matches) compiled to the .NET platform	anti-analysis executable/resource host-interaction/clipboard host-interaction/file-system/exists host-interaction/file-system/write host-interaction/memory host-interaction/memory host-interaction/thread/suspend runtime 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.

c:\users\0x\desktop\new folder\da551ab6e000732499227a67f2be68d1256b58d95963a903cc316e2730db9d1e	indicator (28)	detail	level
indicators (groups > API)	groups > API	file network execution diagnostic security memory resource	+++++
footprints (count > 11)	file > extension > count	16	+++++
virustotal (status > error)	.NET > namespace > flag	System.Security.Permissions System.Security	+++++
dos-header (size > 64 bytes)	string > URL	16.0.0.0	++
dos-stub (size > 64 bytes)	string > URL	http://tempuri.org/dsssl.xsd	++
rich-header (n/a)	string > URL	http://www.w3.org/2001/XMLSchema	++
file-header (executable > 32-bit)	imports > flag	6	++
optional-header (subsystem > GUI)	libraries > p/invoke	b_278_0 ge100	++
directories (count > 6)	imports > p/invoke	27	++
sections (count > 3)	file > entropy	7.497	++
libraries (type > p/invoke)	file > type	executable	+
imports (flag > 2383)	file > cpu	32-bit	+
exports (n/a)	file > signature	Microsoft .NET	+
thread-local-storage (n/a)	file > sha256	DA551AB6E000732499227A67F2BE68D1256B58D95963A903CC316E2730D...	+
.NET (namespace > flag)	file > size	1011712 bytes	+
resources (size > file-ratio)	virustotal > error	The server name or address could not be resolved	+
abc strings (count > 31167)	file > compiler > stamp	Thu Feb 20 01:26:03 2025	+
debug (streams > 3)	file-name > version	liiA.exe	+
manifest (n/a)	debug > streams	3	+
version (LegalTrademarks > MaterialSkin)	debug > format	RSOS	+
certificate (n/a)	debug > file-name	liiA.pdb	+
overlay (n/a)	debug > format	REPRO	+
	debug > format	embedded portable PDB	+
	file > subsystem	GUI	+
	certificate > info	n/a	+
	imphash > md5	F34D5F2D4577ED6D9CEEC516C1F5A744	+
	.NET > module > name	liiA.exe	+
	mitre > technique	T1497 T1055	+
DA551AB6E000732499227A67F2BE68D1256B58D95963A903CC316E2730DB9D1E	cou: 32-bit	file-type: executable	subsystem: GUI
			entry-point: 0x000F8126

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.

```
SUMMARY:

Total scanned:      62
Skipped:            0
-
Hooked:             1
Replaced:           1
Hdrs Modified:      0
IAT Hooks:          0
Implanted:          1
Implanted PE:       1
Implanted shc:       0
Unreachable files:  0
Other:              1
-
Total suspicious:   4
---
```

Figure 6: Extracting Artifacts

Static Analysis - Stage 2

The newly outputted PE was further analyzed using various tools.

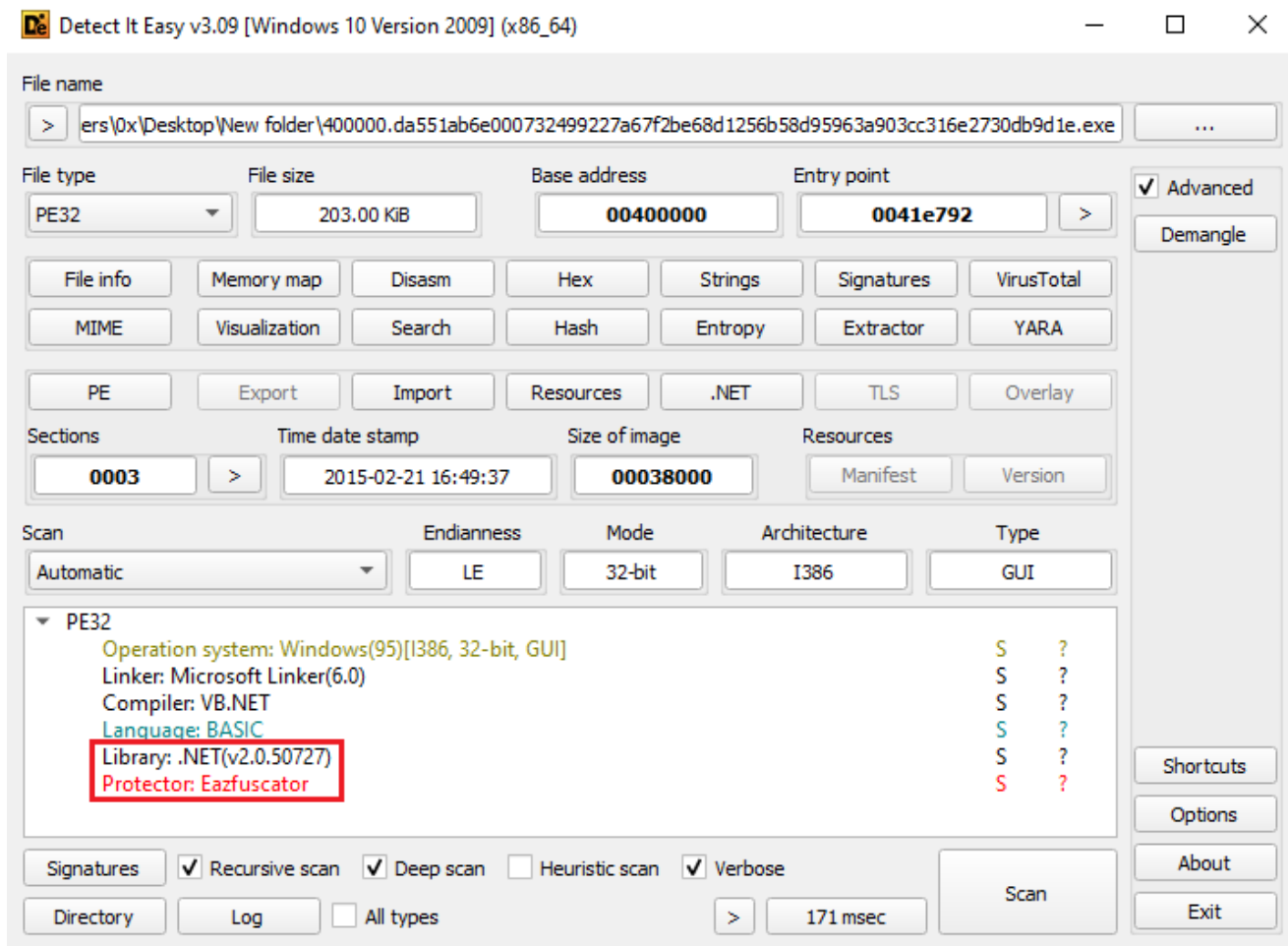


Figure 7: Detect It Easy On 2nd Stage

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]

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

The screenshot displays the Visual Studio IDE with the following components:

- Assembly Explorer (Left):** Shows the project structure for 'NanoCore Client (1.2.2)' under the 'NanoCore Client' namespace. It includes folders for PE, Type References, References, and Resources.
- ClientLoaderForm.cs (Main Window):** Contains the C# code for the 'ClientLoaderForm' class. The code includes:
 - Imports for 'System', 'System.Windows.Forms', and 'System.Threading'.
 - A 'Token' attribute for the RID 461 RVA: 0x000C428 File Offset: 0x000A628.
 - A 'public class ClientLoaderForm : Form' declaration.
 - A 'public ClientLoaderForm()' constructor.
 - Initialization of 'base', 'base.Closing += this', 'base.Shown += this', and 'Application.EnableVisualStyles()'.
 - A 'ShowInTaskbar' property set to 'false'.
 - A 'FormWindowState' property set to 'FormWindowState.Minimized'.
 - A 'Token' attribute for the RID 462 RVA: 0x000C480 File Offset: 0x000A680.
 - A '[STAThread]' attribute and a 'public static void Main()' method.
 - A 'private void' method for the 'Application.Run()' call, which includes a 'FormClosingEventArgs' parameter.
- Locals Window (Bottom):** Displays the current state of variables. The 'Application' variable is highlighted, showing its value as 'System.Windows.Forms.Application' and its type as 'System.Windows.Forms.Application'.

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

```

#qkqs1b2zn8pde7p5cVcU1zAaIrj6t_Hbve7n7O2s3B137PWX4Zr9h8YmyrX : IClientReadOnlyNameObjectCollection
public sealed class #qkqs1b2zn8pde7p5cVcU1zAaIrj6t_Hbve7n7O2s3B137PWX4Zr9h8YmyrX : ICientReadOnlyNameObjectCollection
{
    // Token: 0x0000001F RID: 287 RVA: 0x00000008 File Offset: 0x00000008
    public #qkqs1b2zn8pde7p5cVcU1zAaIrj6t_Hbve7n7O2s3B137PWX4Zr9h8YmyrX(Dictionary<string, object> #qz2Hoyza39rJmsF15qhuYXUQ==)
    {
        if (0 != 0)
        {
            this.#qhv_90q5y9r5PWE1vgk8Fae-- = #qz2Hoyza39rJmsF15qhuYXUQ--;
        }
    }
}

```

100 %

Locals

Name	Value	Type
#qz2Hoyza39rJmsF15qhuYXUQ==	#qkqs1b2zn8pde7p5cVcU1zAaIrj6t_Hbve7n7O2s3B137PWX4Zr9h8YmyrX	System.Collections.Generic.Dictionary<string, object>
[0]	["KeyboardLayout", true]	System.Collections.Generic.Dictionary<string, object>
[1]	["BuildTime", "1/18/2025 10:54:57 AM"]	System.Collections.Generic.Dictionary<string, object>
[2]	["Version", "1.2.2.0"]	System.Collections.Generic.Dictionary<string, object>
[3]	["Machine", "172b2be8d125b6395963a903cc16e7230b8dfe1e9e, 4b2c-4319174209ff0"]	System.Collections.Generic.Dictionary<string, object>
[4]	["DefaultGroup", "JAMAM001"]	System.Collections.Generic.Dictionary<string, object>
[5]	["PrimaryConnectionHost", "tshimjohnson163.airsns.org"]	System.Collections.Generic.Dictionary<string, object>
[6]	["BackupConnectionHost", ""]	System.Collections.Generic.Dictionary<string, object>
[7]	["ConnectionString", "0x596"]	System.Collections.Generic.Dictionary<string, object>
[8]	["RunOnStartup", true]	System.Collections.Generic.Dictionary<string, object>
[9]	["RequestElevation", false]	System.Collections.Generic.Dictionary<string, object>
[10]	["BypassUserAccountControl", true]	System.Collections.Generic.Dictionary<string, object>
[11]	["BypassUserAccountControlData", byte[0x00004F]]	System.Collections.Generic.Dictionary<string, object>
[12]	["ClearAccessControl", true]	System.Collections.Generic.Dictionary<string, object>
[13]	["ClearAccessControl", false]	System.Collections.Generic.Dictionary<string, object>
[14]	["SetCriticalProcess", false]	System.Collections.Generic.Dictionary<string, object>
[15]	["PreventSystemSleep", true]	System.Collections.Generic.Dictionary<string, object>
[16]	["AddOnStartupMode", false]	System.Collections.Generic.Dictionary<string, object>
[17]	["DisableDebugMode", false]	System.Collections.Generic.Dictionary<string, object>
[18]	["RunDelay", 0x00000000]	System.Collections.Generic.Dictionary<string, object>
[19]	["ConnectDelay", 0x00000000]	System.Collections.Generic.Dictionary<string, object>
[20]	["RestartDelay", 0x00001080]	System.Collections.Generic.Dictionary<string, object>
[21]	["TimeoutInterval", 0x00000380]	System.Collections.Generic.Dictionary<string, object>
[22]	["KeepAliveTimeout", 0x00007330]	System.Collections.Generic.Dictionary<string, object>
[23]	["MuteTimeout", 0x00001380]	System.Collections.Generic.Dictionary<string, object>
[24]	["LanTimeout", 0x000009C4]	System.Collections.Generic.Dictionary<string, object>
[25]	["VanTimeout", 0x00001040]	System.Collections.Generic.Dictionary<string, object>
[26]	["BufferSize", 0x0000FFFF]	System.Collections.Generic.Dictionary<string, object>
[27]	["MaxPacketSize", 0x00A00000]	System.Collections.Generic.Dictionary<string, object>
[28]	["GCThreshold", 0x00A00000]	System.Collections.Generic.Dictionary<string, object>
[29]	["CustomOnServer", true]	System.Collections.Generic.Dictionary<string, object>
[30]	["PrimaryOnServer", "3.8.3.8"]	System.Collections.Generic.Dictionary<string, object>
[31]	["BackupOnServer", "3.8.4.4"]	System.Collections.Generic.Dictionary<string, object>

breakpoint(s) : pids=8400/40000, d5a5ab6e00732499227ad7c2be8d125b6395963a903cc16e7230b8dfe1e9e, 40000, d5a5ab6e00732499227ad7c2be8d125b6395963a903cc16e7230b8dfe1e9e

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.

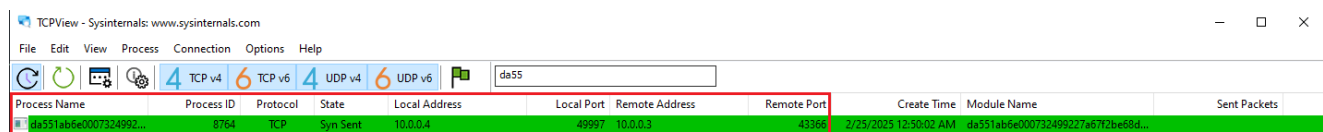


Figure 12: TCPView Trying To Establish Connection

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

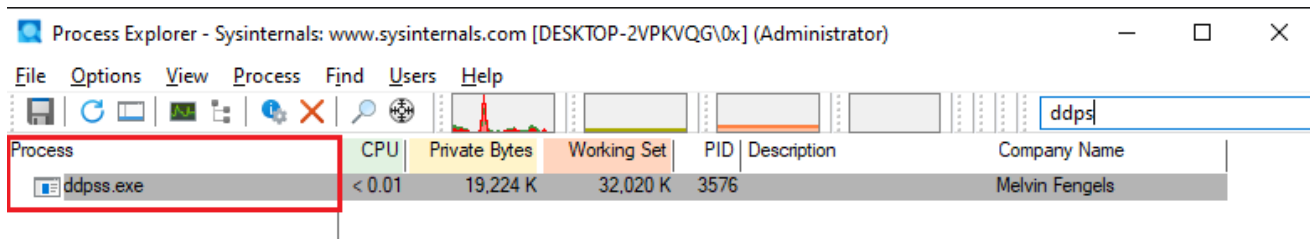


Figure 13: Process Starts Under a New Name

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

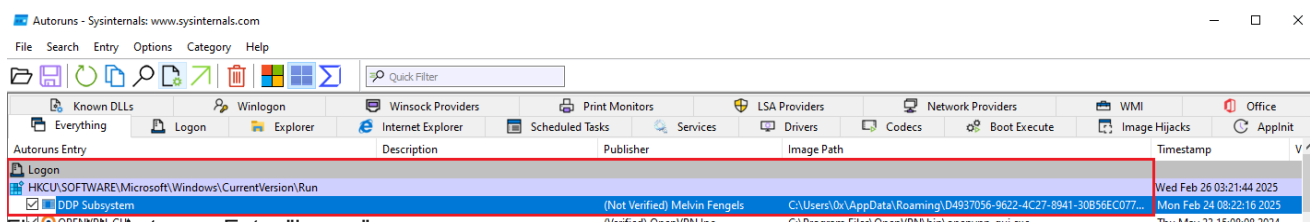


Figure 14: Autoruns Entry "Logon"

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.

89	17.538569198	10.0.0.4	10.0.0.3	DNS	89	Standard query 0x5f74 A lxtihmjohnson163.airdns.org	
90	17.546943941	10.0.0.3	10.0.0.4	DNS	105	Standard query response 0x5f74 A lxtihmjohnson163.airdns.org	A 10.0.0.3

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