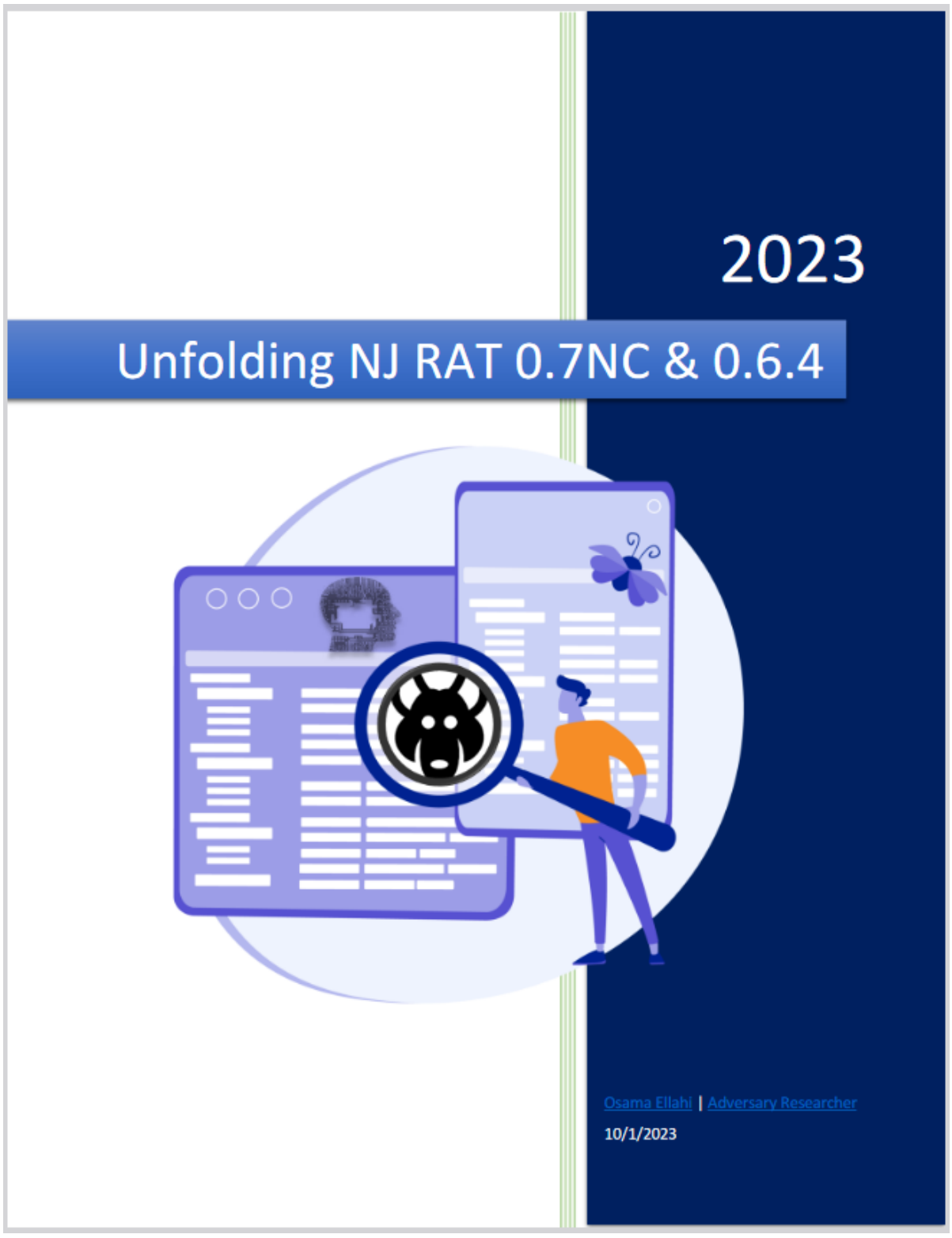
Unfolding NJ RAT 0.7NC & 0.6.4

infosecwriteups.com/unfolding-nj-rat-07nc-and-064d14b875c7cd8-d14b875c7cd8

Osama Ellahi November 21, 2023



| NJRAT Malware analysis

Executive Summary

This version {0.7NC} of **NJRat was first seen on 17 August 2023** with the name utah-Robert-magazine- speaker. It was delivered by email using phishing. Red Packet Security defines NJRat as a type of remote access trojan (RAT). This malicious software can do a range of things, like **recording keystrokes, accessing the victim's camera, stealing saved login information from web browsers**, creating a way for attackers to control the victim's computer from a remote location, transferring files to and from the victim's computer, **seeing what's on the victim's screen, making changes to files, processes, and the Windows registry, and even allowing the attacker to update, remove, restart, close, disconnect, or change the name of their attack campaign**.

This analysis comprises two samples labeled as NJ RAT 0.7NC and 0.6.4. The 0.7NC variant introduces a novel method for evading analysis, while 0.6.4 is responsible for managing all other malicious activities.

High-Level Technical Summary

NJRAT is a sophisticated malware that operates in **two primary stages**. **The initial stage involves phishing and obfuscation tactics.** In August 2023, security experts first encountered malware, which was distributed via email in the form of a malicious and highly obfuscated VBS (Visual Basic Script) file embedded in documents.

Upon execution, **this VBS file performs deobfuscation and reveals a PowerShell script**. Within this script lies a base64-encoded DLL (Dynamic Link Library). Once the script successfully decodes the DLL, it proceeds to invoke the "VAI" method within the DLL. This marks the beginning of malware's further exploitation and malicious activities.

Initial Stage

This stage consists of deobfuscation and decoding of real dll and invoking the binary.2

> SHA256
>
> vbs = 5f66c7336f8469a6ab349a3f0f3f7aca1b483f2f2a8b4ad71af79ff51a8aad6b
>
> dll = 153c9ffe148909981900c59c2ccba8ef66f94688ce7ab5e01e3a541937a31294

.VBS

The initial executable comprises a VBS file containing obfuscated PowerShell code. After modifying the VBS file and revealing the de-obfuscated PowerShell code, we can observe its initial command in the terminal. This command involves **pinging localhost** for a dynamic delay, followed by the **self-copying of the executable to the startup folder**. This technique is employed to achieve persistence, ensuring that the executable runs every time the device starts up.

> This is the command which copy the malicious file in startup folder for future purposes.

> cmd.exe /c ping 127.0.0.1 -n 10 & powershell -command [System.IO.File]::Copy(‘’,’C:\Users\’ +
>
> [Environment]::UserName + ‘\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\.vbs’)

After persistence VBS goes for de-obfuscating the malicious DLL. As you can see in the figure below, there is an obfuscated string, and the script is using the **yWaUTuYlQuUWknat method** to perform a straightforward task: **locating and replacing the string with the specified third parameter.**

After printing the shell code, I get the real DLL in terminal. So far, we don’t know what this code is and how to invoke it or use it.

By looking at the string, it was base64 encoded. So, I decided to decode it with PowerShell and real binary came out from it.

But let’s see how the exploit is using it.

The VBS has its third obfuscated command which was **then de-obfuscated** to execute this DLL.

After patching and de-obfuscation, I rephrase the final command which looks like this following script.

It was **Invoking its VAI method after decoding** the encoded string with base64.

Exploitation

This final exploit has so many malicious functionalities, we will divide them into **persistence, keylogging and c2 communication.**

Initial behavior

This DLL has all the malicious functions, **its VAI starts with adding mutation in system**. If mutation is already there it will not execute. This technique keeps exploit safe for only one time run. It starts with **copying itself to AppData and running that exe within process**. It did not execute malicious function directly because this way it is making it hard for reverse engineers to go through dynamic debugging.

Persistence

It starts Infinite loop and it have multiple cases. Let’s start from case 0.

Case 0: Reading first obfuscated variable and reversing it and de-obfuscating it. It gets command from the server and process it.

Case 1

Reverse the string only

Case 2

It creates a new **guid** id and gives this name to the VBS. After that it searches inside AppData if any VBS present in the AppData, if there is no VBS in AppData then **it runs command in hidden windows style and copy VBS files from current directory to AppData.**

Case 3

After copying the file to AppData it sets persistence registry **SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run.** By adding this, at every startup it executes this file.

Case 11

Case 11 is focused on **persistence** but this time it is happening through creating a LNK file on run time in startup folder.

This LNK file performs specific action in minimized window using PowerShell.

> 1. Sleep for 5 sec
>
> 2. Start VBS which is inside AppData/roaming.

> C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -WindowStyle Hidden C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -WindowStyle Hidden Start-Sleep 5; Start-Process C:\Users\burgo\AppData\Roaming\4df328c8–4a5e-4b9b-8385–1495d13b45dd.vbs

INS () function is totally persistence based which controls the foothold of exploit inside system for future purposes.

It is setting Environment variable **See_MASK_NOZONECHECK** to 1 which allows it to download the files and execute the files without zone identifier.

This code includes a step to add itself to the list **of allowed programs in the Windows Firewall**. By executing this command, you are essentially instructing the Windows Firewall to allow network traffic for a specific program or application**. This action ensures that the program can freely communicate over the network without being hindered by the firewall's restrictions.**

It also checks if **Isu** flag is true which is pre-default true then it sets some registries.

It copies a file from one location to another (possibly into the Startup folder), and then it initializes a File Stream object to open and read the copied file in the Startup folder.

And if server commands to remove, server sends command with "un" and "~" to remove all footprints.

At first it removes registries, removes from firewalls allowed program, deletes file from startup folder and at the end it pings 127.0.0.1 and deletes itself.

Figure 21 Uninstalling from system.

Keylogging

The keylogger in njrat is doing following steps.

> 1· It initializes various properties and objects, including a keyboard listener (this.keyboard), a log file path (this.LogsPath), and other variables.
>
> 2· The keylogger continuously monitors keyboard input using a for loop. Inside the loop, it checks the state of each key using the **GetAsyncKeyState** function, allowing it to **capture key presses and releases asynchronously**.
>
> 3· When a key is pressed, **the Fix method** is called to convert the key code into a **standardized representation**. It considers the Shift and Caps Lock keys, maps function keys and special keys to specific strings **(e.g., "[F1]", "[ENTER]"),** handles whitespace and Enter key presses, and converts other keys to their corresponding Unicode characters.
>
> 4· The keylogger appends the converted key representation to a log **(this.Logs),** which accumulates the logged keystrokes over time. It also includes special entries for Enter and Tab key presses to format the log properly. **{AppData\services64.exe.tmp}**
>
> 5· To prevent the log from growing indefinitely, the keylogger periodically truncates the log to a certain length and updates the log file on disk (**File.WriteAllText).**
>
> 6· The keylogger continues to monitor and log keyboard input indefinitely within the for loop while sleeping briefly between iterations to control the rate of input capture.

There is constructor call of kl () initialize some general variables and prepare all settings for keylogging like clock and path. This keylogger monitors all the key logs and process information.

The constructor is named kl, and it initializes various properties and objects when an instance of the class is created.

The **WRK** method appears to continuously monitor keyboard input, **log the pressed keys** along with additional information, and update the log file.

GetAsyncKeyState is used to monitor and capture keyboard input events asynchronously, allowing the code to track and log key presses as they occur in real-time within the **for** loop. This is typically used for purposes such as keylogging or tracking user input in certain types of applications.

The Fix method in this code handles keyboard input by mapping different keys to specific representations for logging purposes. It considers the state of the Shift and Caps Lock keys, encloses function keys and special keys in square brackets, maps certain keys to empty strings, and converts others using a custom method while maintaining uppercase or lowercase based on the Shift key state. It ultimately returns the resulting string representing the converted keyboard key.

VKCodeToUnicode() this method attempts to convert a virtual key code to its corresponding Unicode character. It uses **the GetKeyboardState, MapVirtualKey, and ToUnicodeEx** functions from user32.dll to perform the conversion.

This is where all logs are stored, you can see the following figure.

Kl () class is only responsible for monitoring and storing all the logs in the file. It is not sending the logs back to server.

To conclude this keylogger these were steps performed by this keylogger.

> 1· The **GetAsyncKeyState** function is called in a loop to check the state of keyboard keys with virtual key codes ranging from 0 to 255. It checks each key one by one. If GetAsyncKeyState returns -32767 for a specific key code, it indicates that the key with that virtual key code is currently pressed down. In other words, it's in the "pressed" state at the time of the function call.
>
> 2· When a pressed key is detected**, it is converted to a Keys Enum value** (k) to represent the specific key.
>
> 3· The Fix(k) method is called to process **the key and convert it into a suitable string representation**, considering factors like special keys, shift, caps lock, etc.
>
> 4· The processed key information is **then logged into the Logs field**, which stores the captured keyboard input.
>
> 5· Finally, the Laskey field is **updated to keep track of the last key** that was pressed.

C2 communication

In the main, it is creating a **thread which is executing the RC method of OK class.**

Execution according to Flags

There is an infinite loop which gets command from server, and it **calls Ind (byte [])** which then handles all the commands and controls.

Proc

In **Ind ()** first it checks for "proc" if it exists in the array which is converted to string. Ok.Y = "|'|'|";

~

If the flag is **"~"** then it gets current process id **using GetCurrentProcess()** and sends it to the server.

Ok.Y = "|'|'|";

After this it gets the length of **processes using GetProcesses()**

Ok.Y = "|'|'|";

Then, it gets file descriptions of all files and processes which are running. **File name, file description, processID using GetProcesses().**

k

After completing **“~”** it checks for **“k”** flag in string, this flag is implemented to **kill the process from process id.**

If it could not kill it will send exception to server. Ok.Y = “|’|’|”;

kd

And then it goes **for kd** flag which not only kill the process it also deletes the file. Before deleting the file and after killing the process it sends “proc |’|’| RM |’|’| process-id”.

After **deleting file from system**, it sends “proc |’|’| ER |’|’| Deleted process-id”. If any error occur it will send “proc |’|’| ER |’|’| error-exepction”

re

Then, it checks for **“re” flag**, if it is true it sends “proc |’|’| RM |’|’| process-id”. It kills this running process. And sends “proc |’|’| ER|’|’| process-file-path” to server.

In case of error, it sends “proc |’|’| ER|’|’| error-exception.”

rss

The “rss command” handles all the commands running which come from the server. It sends to server “rss”. This code sets up a Process object to run the Windows Command Prompt (cmd.exe) with various configurations, allows interaction with its standard input, output, and error streams, and attaches event handlers to process the output and errors produced by the command prompt. It then sends a “rss” command to the command prompt and starts the process, enabling asynchronous reading of its output and error streams.

Rs and rsc

The “rs” flag shows if the command needed to be executed hidden and “rsc” will kill these processes.

kl

The “kl” flag reads the keylogging logs from AppData and sends them to server.

> act|’|’|IA==
> [endof]act|’|’|UHJvY2VzcyBNb25pdG9yIC0gU3lzaW50ZXJuYWxzOiB3d3cuc3lzaW50ZXJuYWxzLmNvbQ==[endof]act|’|’|RmlsZSBFeHBsb3Jlcg==[endof]
>
> act|’|’|[endof]
>
> act|’|’|UHJvY2VzcyBNb25pdG9yIC0gU3lzaW50ZXJuYWxzOiB3d3cuc3lzaW50ZXJuYWxzLmNvbQ==[endof
>
>]
>
> act|’|’|[endof]
>
> act|’|’|ZG5TcHkgdjYuMS44ICgzMi1iaXQsIC5ORVQp[endof]

This is decoded base64 and it shows every new running process.

> Process Monitor — Sysinternals: www.sysinternals.com File Explorer
>
> dnSpy v6.1.8 (32-bit, .NET)

inf

The inf flag gets the **system drive number and returns with** “encoded bas64, **44gang44.duckdns.org,2222**, |’|’|”. The server **44gang44.duckdns.org is used for c2 and port 2222 i**s used with content encoded base64 of system drive number.

The value in the result variable will be a hexadecimal representation of the volume serial number of the system drive. Depending on the specific system and drive, this value will vary and typically be a combination of letters and numbers.

pof

The **“pof”** flag is then used **to handle all registry functions. It is responsible for both set registry and delete registry.**

~

The **STV** function is used to **set registry**, further if command contains [! Or **~**], it sets the reg. if! It will respond back also.

!

The **DLV** is responsible for removing registry if command contains **@ it remove the registry.**

cap

CAP flag does the following main tasks: **capturing the screen, handling the cursor, checking for changes in the captured image, and preparing the data for sending using encoding and then sending the data.**

It sends the data with CAP flag to the server which indicates that this is screenshot of victim screen.

p

The P flag is just to acknowledge the response is coming.

un

The un fag is coded to uninstall the malware but if there is @ it start itself again.

If there is **~** with **un,** it starts **removing the footprints from the system.**

> *Case1*. it is deleting the registry of persistence from current user and from local machine.
>
> *Case2*. It is deleting this application malware from the allowed program of firewall.
>
> *Case3*. It is deleting itself from the startup folder.
>
> *Case4*. It is removing the software mutation registry.
>
> *Case5*. It deletes itself from the system after pinging localhost. This pinging on localhost creates a little delay and after this delay it deletes itself.

RG

It also checks **for Registry modifications if RG flag is present like registry get values,** checking its permissions, adding new registries, and removing registries.

rn

Flag **"rn"** handles the new zip file downloads. It downloads the zip file from specified URL and it place it in the device for further execution, it uses http web client to download this file.

Indicators of Compromise

The full list of IOCs can be found in the Appendices.

Network Indicators

Callback URLs

> **njnjnjs.duckdns.org :**35888
>
> **44gang44.duckdns.org :** 2222

Decompiled binary

References

dnSpy: https://github.com/dnSpy/dnSpy

https://www.virustotal.com/gui/file/5f66c7336f8469a6ab349a3f0f3f7aca1b483f2f2a8b4ad71af79ff51a8aad6b

https://www.joesandbox.com/analysis/1292688/0/html

https://cybergeeks.tech/just-another-analysis-of-the-njrat-malware-a-step-by-step-approach/