Malware Analysis, the case of Njrat

Authors

Aurélien Izoulet
aurelien.izoulet@epita.fr

Sahar Tajouri
sahar.tajouri@epita.fr

École pour l'informatique et les techniques avancées
Le Kremlin-Bicêtre
14/16 rue Voltaire
94270 Le Kremlin-Bicêtre

March 19, 2025

Contents

1 Introduction

1.1 Context

NjRAT is a compelling malware that concretely illustrates key concepts from our course, especially persistent system compromise. It demonstrates a deep infection of the target machine and maintains persistence at every system startup. This behavior underscores its significance in understanding modern cyber threats.

1.2 Choice of Malware

NjRAT provides full remote access, allowing an attacker to control the victim's desktop or active window. It collects vital system details including IP address, computer name, username, operating system, installation date, and country. The malware can remotely execute files, manipulate files, and open a command shell. It also manages processes, alters the registry, records audiovisual inputs, logs keystrokes, and steals stored passwords.

1.3 Main Features

Also known as Bladabindi, njRAT is a remote access trojan first identified in June 1987, with some variants dating as recently as December 2020. Developed by the hacker group Sparclyheason, it has been used primarily against targets in the Middle East and spread via phishing and infected media. Classified as severe by Microsoft Malware Protection Center, its activities include notable attack surges and deceptive software distribution campaigns.

2 Methodology

2.1 Analysis Environment

The analysis was conducted in an isolated and secure environment to prevent any accidental contamination. A virtualized Windows 11 machine, deployed using Hyper-V, served as our primary testbed, with the network configured in host-only mode to ensure complete isolation. Additionally, we employed system snapshots to guarantee the reproducibility and consistency of our results throughout the experimentation.

2.2 Tools Used

For this study, the following specialized tools were employed:

- **The Zoo GitHub** – Source of malware for analysis.
- **Pestudio** – Initial static analysis.
- **DiE (Detect It Easy)** – File type identification.
- **Procmon** – System activity monitoring.
- **Procdot** – Analysis and visualization of Procmon data.
- **Jetbrain dotPeek** – Binary decompilation.
- **Wireshark** – Network traffic capture and analysis.
- **Joe Sandbox Cloud** – Automated analysis platform.
- **Any.Run** – Automated analysis platform.

2.3 Analysis Methodology

Our methodological approach was divided into two primary phases. The **static analysis** phase involved examining the malware file without executing it; we scrutinized the PE header, identified embedded strings and imported libraries, and decompiled the binary to assess its source code structure. In contrast, the **dynamic analysis** phase focused on observing the malware's behavior during execution. During this phase, we monitored both parent and child processes, recorded system interactions, and generated detailed reports documenting each step of the attack lifecycle. This comprehensive dual-phase approach enabled a thorough understanding of the malware's operational mechanisms and its potential impact on target systems.

3 Static Analysis

3.1 Analysis with DiE

The Detect It Easy (DiE) tool revealed the fundamental characteristics of the file:

feature	Value
Type of file	PE32 (Portable Executable 32 bits)
File size	8.54 MiB
Base Address	0x00400000
Entry Point	0x00c831ea
Target operating system	Windows (95) [386, 32 bits, GUI]

Table 1: Results of DiE's Analysis

3.2 Analysis with Pestudio

3.2.1 General Information

This section provides basic details about the analyzed file, including its hash, architecture, and compilation information.

- SHA-256 : BFD5FE72651B4EC588BD5FC6A9F17E0972248146
- Type of file: PE32 (Portable Executable)
- Machine Type : Intel-386 (x86, 32-bit)
- Compiler Stamp : 29 octobre 2017
- Compiler Signature : PE00 (format standard Portable Executable)

3.2.2 Antivirus Detection

The file has been flagged as malicious by numerous antivirus engines, indicating a high likelihood of it being malware, commonly classified as a Trojan, spyware.

- Detection rate: 58/72 security vendors
- Common classifications: Trojan.GenericKD, Win32:Malware-gen, TrojanSpy, etc.

3.2.3 Identified Suspicious Strings

Several suspicious strings were detected that links to cloud storage services, which are used for data exfiltration.

- Firewall deactivation commands: `netsh firewall delete allowedprogram`
- Keylogging techniques via keyboard function interception
- Links to cloud storage services: `https://dl.dropbox.com`

3.2.4 Suspicious API Functions

Function/API	Associated Technique
TcpClient, TcpListener, Socket, IPAddress	T1043: Network Exfiltration
memcpy	T1055: Process Injection
DeleteFile	T1485: Data Destruction
QueueUserWorkItem	T1055: Process Injection
Sleep	T1497: Anti-Detection Evasion
memcpy	T1055: Process Injection

Table 2: Identified Suspicious API Functions

3.3 Deep Analysis using JetBrains dotPeek

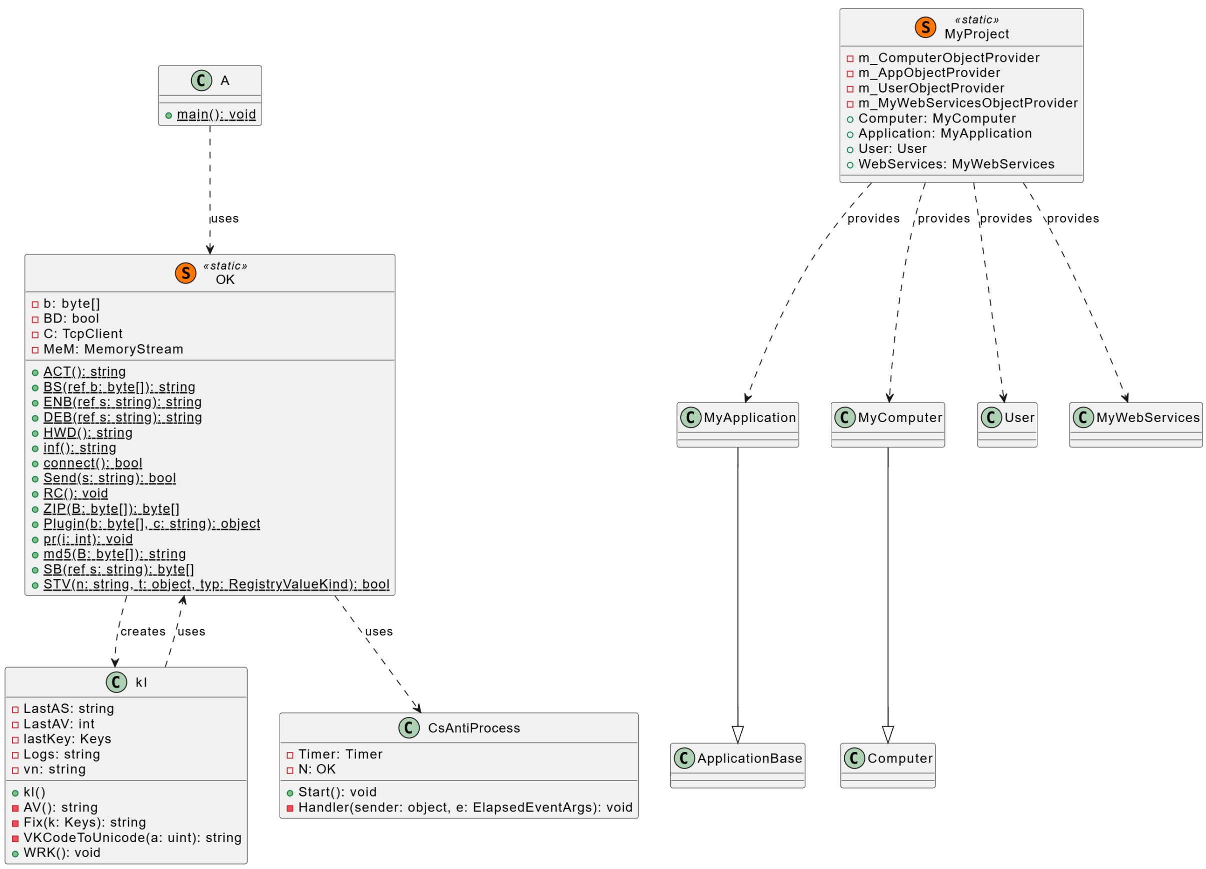

Figure 1: UML Diagram of the Reconstructed Code Structure

3.3.1 Decompiler Steps

JetBrains dotPeek was used to decompile the malware binary and reconstruct the source code. The decompilation revealed the overall structure, including class names, method bodies, and embedded Windows API calls. This process provided insights into control flow, error handling, and obfuscation techniques implemented by the malware.

3.3.2 Classes and Functionalities

The analysis uncovered a modular design with distinct responsibilities:

- **A**: The entry point of the application that calls the core initialization routine.
- **kl**: Implements keylogging functionality by intercepting keystrokes and monitoring active window changes.
- **OK**: A central static class that handles system information gathering, network communication, persistence, dynamic plugin loading, and anti-analysis measures.
- **CsAntiProcess**: Monitors for analysis tools and debugging processes, terminating the application if any are found.
- **MyProject, MyApplication, MyComputer**: Components of the Visual Basic "My" framework that provide system and application-level abstractions.

The accompanying Figure 1 (generated via PlantUML) visually represents these class relationships and functionalities.

3.3.3 Keylogging and User Input Interception

The *kl* class is at the core of the keylogging functionality:

- **Keystroke Capture**: The `WRK()` method continuously iterates over all virtual key codes (0 to 255) using the Windows API function `GetAsyncKeyState()`. When a key press is detected (and Control is not pressed), the method processes the key.
- **Active Window Monitoring**: The `AV()` method retrieves the current foreground window using `GetForegroundWindow()` and fetches the window title with `GetWindowText()`. If the active window changes, it logs the new window's process name and title, providing context to the captured keystrokes.
- **Key Translation**: The `Fix(Keys k)` method converts raw key codes into human-readable strings. It accounts for special keys (e.g., Backspace, Tab, Enter, Space) and determines the proper casing by evaluating the state of modifier keys such as Shift and CapsLock. The helper function `VKCodeToUnicode(uint a)` is used to convert virtual key codes to their corresponding Unicode characters.
- **Log Persistence**: The logged data is periodically saved using the `STV()` function from the *OK* class. This mechanism writes the keylog to persistent storage (for example, the registry), ensuring that keystroke data is retained across sessions.

Overall, the keylogging component is intricately integrated with system APIs to intercept user input and record contextual information, making it both effective and stealthy.

3.3.4 Network Communication

The network communication routines are primarily managed by the *OK* class:

- **Connection Establishment**: The `connect()` method initializes a TCP connection to a remote command-and-control server using a predefined host and port. It sets up socket parameters such as buffer sizes and timeouts, and then transmits initial system information gathered by the `inf()` method.
- **Data Transmission**: The methods `Send(string s)` and `Sendb(byte[] b)` implement a custom protocol. Each message is prefixed with a header that specifies the message length, allowing the receiving end to correctly reconstruct the full message.
- **Receive Loop**: The `RC()` method implements a continuous loop that polls the socket for incoming data. It reads a length header to determine the size of the incoming message, then collects the complete message before handing it off to a command interpreter. This design supports robust and asynchronous communication with the server.
- **Dynamic Plugin Loading**: Through the `Plugin(byte[] b, string c)` method, the malware can load additional code dynamically. This enables the modular extension of functionality without static linkage, making the malware more adaptable to new commands or updates.
- **Resilience and Reconnection**: The network routines include error handling and reconnection logic. If the connection is lost or an error occurs during communication, the system attempts to re-establish the connection, ensuring persistent control.

These network capabilities enable remote command execution, system information exfiltration, and dynamic updates, forming the backbone of the malware's remote control infrastructure.

3.4 CAPA Analysis

CAPA (Capability Analyzer) is a powerful tool used for analyzing executable files to identify malicious capabilities, techniques, and behaviors. It provides insights into how malware operates by mapping its functionalities to frameworks like MITRE ATT&CK, MBC (Malware Behavior Catalog), and MAEC. In this analysis, CAPA detected that Client.exe (a modified NjRat 0.7D variant) exhibits keylogging, screen capture, command-and-control (C2) communication, anti-analysis techniques, and registry modifications, making it a high-risk Remote Access Trojan (RAT) capable of data theft and remote system control. Les tableaux suivants présentent une analyse détaillée des comportements malveillants et des capacités associées à un programme potentiellement nuisible.

4 Dynamic Analysis

4.1 Analysis with ProcDot and Process Monitor

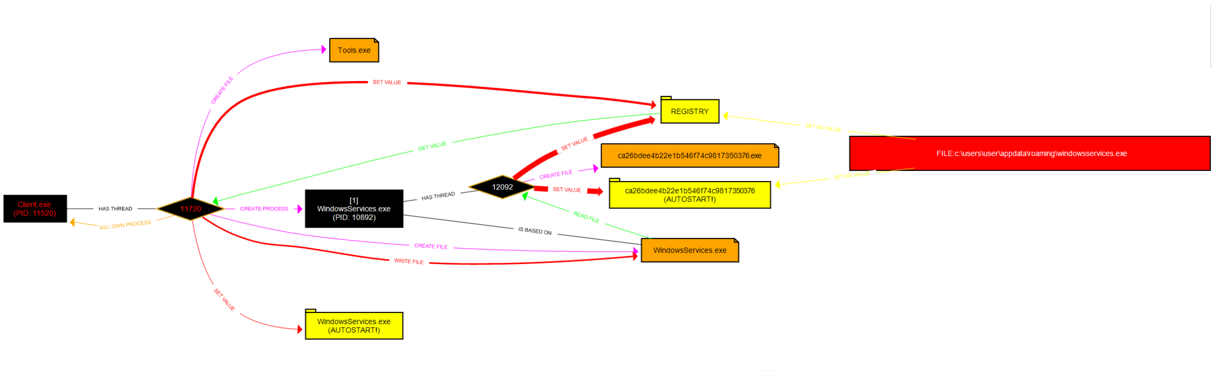

Figure 2: Attack Schema Generated from ProcDot and Process Monitor

4.1.1 Activity during execution

Figure 2 (the schema of the attack) shows the flow of events observed using ProcDot and Process Monitor. The following sequence summarizes the key actions performed by the malware:

1. **File Creation:**
   - `Tools.exe` is created in `C:\Tools.exe`.
   - `WindowsServices.exe` is created in `C:.exe` and written to multiple times.
2. **Registry Modification:**
   - Sets registry key `di` (Data: `!`) and additional keys such as `418A073AA3BC3475` and `data`.
   - Sets autostart registry key `WindowsServices.exe` for persistence.
   - Sets environment variable `SEE_MASK_NOZONECHECKS` (Data: `1`) to disable certain security checks.
   - Modifies or creates a key named `ca26bdee4b22e1b546f74c9817350376` pointing to `"C:\Users\User\AppData\Roaming\WindowsServices.exe"`.
3. **Process Spawning:**
   - The malware spawns `WindowsServices.exe` (PID: 10892) after writing its payload.
   - The original `Client.exe` process terminates itself after these steps, handing off execution to the newly created `WindowsServices.exe`.
4. **Further Registry Changes by `WindowsServices.exe`:**

- Resets the registry key `di` to `!`.
- Again sets the autostart key under `ca26bdee4b22e1b546f74c9817350376`.

The diagram (Figure 2) visually correlates file creation, registry operations, and process launches in chronological order.

4.1.2 Permanent changes to the system

From the sequence above, we note the following persistent changes:

- **Executable Placement:**
  - `Tools.exe` is placed directly in the `C:\` directory.
  - `WindowsServices.exe` is stored in `$AppData\Roaming\`.
  - An additional copy of the executable may appear in the Startup folder under the name `ca26bdee4b22e1b546f74c9817350376.exe`.
- **Registry Autostart Keys:**
  - The malware writes to `Software\Microsoft\Windows\CurrentVersion\Run` to ensure that `WindowsServices.exe` runs automatically on system startup.
  - Various other registry keys (e.g., `418A073AA3BC3475`, `data`, `di`) are set or modified for configuration or tracking.
- **Environment Variable:**
  - `SEE_MASK_NOZONECHECKS` is enabled to bypass certain Windows security prompts or checks.

These changes remain on the system even after the original `Client.exe` process terminates, allowing `WindowsServices.exe` to persist and automatically restart. Administrators should remove these executables and registry entries to fully neutralize the threat.

4.2 Analysis Using Automated Platforms

4.2.1 Joe Sandbox Cloud

The Joe Sandbox Cloud platform analyzed the file and classified it as MALICIOUS, assigning it a score of 60/100 (Figure 2). The observed techniques indicate sophisticated evasion mechanisms and potential system compromise. The malware demonstrates sandbox and virtual machine detection, uses timers to delay malicious actions, dynamically allocates memory, which suggests possible code injection, and actively checks system configurations, including registry values and .NET settings. Additionally, the confirmed MITRE ATT&CK techniques (Figure 4) include DLL Side-Loading for execution, sandbox evasion (T1497), and system discovery, which align with typical behaviors of stealthy malware.

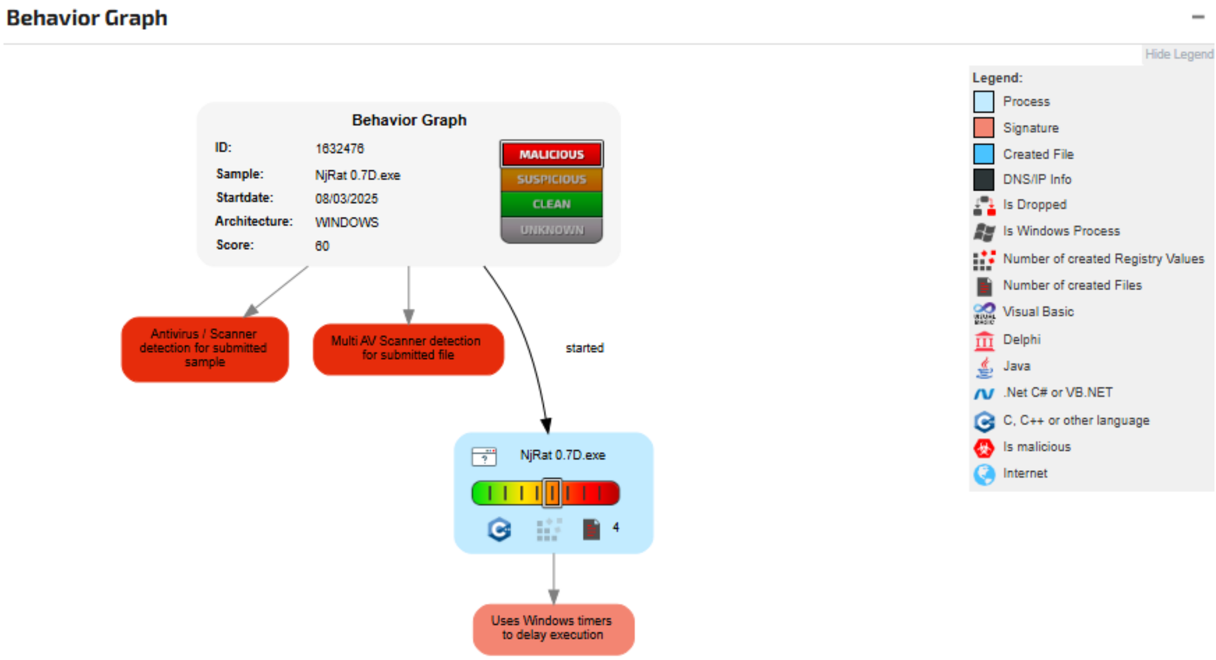

Figure 3: Represent the result of analysis with joesandbox

4.2.2 Analysis on Any.Run

The Any.Run platform revealed significant behavioral indicators of malicious activity. In terms of process behavior, the malware establishes an active TCP server on port 6522 and spawns suspicious processes, such as dw20.exe and WerFault.exe, which are often used in process masquerading. The network activity further raises concerns, as the malware initiates 131 TCP connections, performs 116 DNS requests, and sends multiple HTTP requests (200 OK responses) to servers located in Germany, suggesting potential command-and-control (C2) communication. These findings indicate that the malware is actively engaging in remote communication and system reconnaissance.

Mitre Att&ck Matrix

Reconnaiss...	Resource Development	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command and Control	Exfiltration	Impact
Gather Victim Identity Information	Acquire Infrastructure	Valid Accounts	2 Command and Scripting Interpreter	1 DLL Side-Loading	1 DLL Side-Loading	1 1 Virtualization/Sandbox Evasion	OS Credential Dumping	1 Security Software Discovery	Remote Services	Data from Local System	Data Obfuscation	Exfiltration Over Other Network Medium	Abuse Accessibility Features
Credentials	Domains	Default Accounts	Scheduled Task/Job	Boot or Logon Initialization Scripts	Boot or Logon Initialization Scripts	1 Disable or Modify Tools	LSASS Memory	1 1 Virtualization/Sandbox Evasion	Remote Desktop Protocol	Data from Removable Media	Junk Data	Exfiltration Over Bluetooth	Network Denial of Service
Email Addresses	DNS Server	Domain Accounts	At	Logon Script (Windows)	Logon Script (Windows)	1 DLL Side-Loading	Security Account Manager	1 3 System Information Discovery	SMB/Windows Admin Shares	Data from Network Shared Drive	Steganography	Automated Exfiltration	Data Encrypted for Impact

Figure 4: Representation of Analysis Results with ANY.RUN Using MITRE ATT&CK