

Weaponized WinRAR Exploitation and Stealth Deployment of Fileless .NET RAT

 cyfirma.com/research/weaponized-winrar-exploitation-and-stealth-deployment-of-fileless-net-rat

Published On : 2026-01-23



EXECUTIVE SUMMARY

At CYFIRMA, we continuously monitor emerging threat techniques that abuse trusted software and routine user behavior to achieve stealthy system compromise. This report analyzes a multi-stage malware infection chain that exploits a WinRAR path validation vulnerability (CVE-2025-8088) to establish persistent, fileless remote access on Windows systems.

The attack begins with a weaponized RAR archive masquerading as a legitimate collection of OSINT tools. Although the archive appears harmless when viewed in WinRAR, extraction triggers malformed path traversal logic that allows a malicious Windows Batch script to be silently written into the Windows Startup

directory. This behavior ensures automatic execution at user logon without requiring administrative privileges or explicit user consent.

Once executed, the Batch script performs multiple malicious actions, including persistence establishment, defense evasion through the removal of Mark-of-the-Web, and extraction of an embedded PowerShell loader. The PowerShell loader decrypts and injects Donut-generated shellcode into a legitimate Windows process, primarily explorer.exe, with svchost.exe serving as a fallback target. The injected shellcode initializes the .NET Common Language Runtime (CLR) in memory and executes a .NET Remote Access Trojan classified as Quasar RAT. The entire execution chain emphasizes stealth, resilience, and minimal disk artifacts.

INTRODUCTION

Threat actors increasingly exploit vulnerabilities in widely trusted consumer and enterprise software to gain initial access while minimizing user suspicion and security alerts. File archiving utilities such as WinRAR are particularly attractive targets due to their extensive adoption, routine handling of downloaded content, and inherent trust in enterprise and personal environments.

This report examines a malware campaign that abuses WinRAR CVE-2025-8088 to deliver a multi-stage payload without relying on exploit kits, macros, or elevated privileges. By weaponizing archive extraction behavior, the attacker is able to place malicious content directly into auto-execution locations, bypassing traditional security prompts and endpoint protection mechanisms.

The observed attack chain demonstrates a deliberate focus on stealth, persistence, and in-memory execution. The use of open-source tooling such as Donut and Quasar RAT reflects a broader trend in modern adversary tradecraft, where legitimate frameworks are repurposed to deliver scalable, low-noise compromises.

KEY FINDINGS

- Threat actors exploited a WinRAR path validation vulnerability (CVE-2025-8088) to abuse archive extraction behavior and write malicious files outside the intended extraction directory.
- The weaponized RAR archive was disguised as a legitimate OSINT tools package, containing only benign-looking content when viewed in WinRAR, effectively reducing user suspicion.
- Exploitation of the vulnerability enabled the silent placement of a malicious Batch script into the Windows Startup directory, establishing persistence without administrative privileges.
- Persistence was reinforced through redundant mechanisms, including Startup folder execution and registry Run key creation under the current user context.
- The Batch script employed heavy obfuscation, randomized variable names, and loop-based logic to evade static detection and hinder analysis.
- Defense evasion was achieved by removing the Mark-of-the-Web (Zone.Identifier), suppressing Windows SmartScreen warnings and security prompts.
- An embedded PowerShell loader was extracted and executed with ExecutionPolicy Bypass and a hidden window, enabling fileless execution.
- The PowerShell loader decrypted Donut-generated shellcode and injected it into a trusted Windows process (explorer.exe), with svchost.exe used as a fallback target.

- A watchdog mechanism continuously monitored the injected process and reinjected the payload upon termination, ensuring self-healing execution and increased resilience.
- The injected Donut shellcode initialized the .NET Common Language Runtime (CLR) in memory and executed the final payload without writing it to disk.
- The final payload was identified as Quasar RAT, providing full remote access capabilities including command execution, file manipulation, keylogging, and system surveillance.
- The overall attack chain demonstrated a low-noise, high-persistence strategy that prioritizes stealth, trust abuse, and in-memory execution over exploit complexity.

VULNERABILITY ANALYSIS – WinRAR CVE-2025-8088

What is CVE-2025-8088?

CVE-2025-8088 is a vulnerability in WinRAR arising from improper validation of file paths embedded within RAR archive metadata during extraction. The flaw allows a specially crafted archive to specify extraction paths that escape the user-selected destination directory, enabling files to be written to unintended locations on the filesystem.

By abusing relative path traversal sequences within archive headers, attackers can target user-level auto-execution directories, such as the Windows Startup folder, without requiring administrative privileges or triggering meaningful security warnings. This vulnerability is particularly impactful because it exploits a trusted application workflow, enabling malware delivery without exploit kits, macros, or executable files.

Exploit Construction and Path Targeting Logic

The exploit logic explicitly defines a relative drop path pointing to the Windows Startup directory under the user's AppData path. The configuration demonstrates that the attacker's intent is not generic file overwrite, but reliable persistence via auto-execution locations. By targeting a user-writable directory, the exploit avoids the need for elevated privileges while still guaranteeing execution at user login.

```
NUM Depths = 10 # Number of different traversal depths to create
RELATIVE_DROP_PATH = "AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\payload.bat"
PAYLOAD = "@echo off\\necho Payload executed!\\npause\\n"
PLACEHOLDER_LEN = 200
OUT_RAR = "exploit.rar"
```

Figure 1: Exploit configuration defining the Windows Startup directory as the malicious payload drop location.

Directory Traversal Abuse via Archive Structure

The archive embeds multiple directory traversal sequences composed of repeated “../” patterns at increasing depths. These traversal variants are used to ensure that, regardless of the initial extraction directory depth, at least one crafted path successfully resolves to the target auto-execution directory. This technique significantly increases exploitation reliability across different system configurations and user environments.

Steps to Reproduce – CVE-2025-8088 (WinRAR Path Validation Vulnerability)

Preconditions:

- A Windows system with a vulnerable version of WinRAR installed.
- User operating under standard (non-administrator) privileges.
- Access to a specially crafted RAR archive containing manipulated internal file paths.

Step 1: Prepare a Malicious RAR Archive

Create a RAR archive that includes files with crafted relative path traversal sequences (e.g., multiple `..\` components) embedded in the archive's internal file metadata.

These paths are designed to escape the user-selected extraction directory and resolve to a user-writable auto-execution location, such as the Windows Startup folder.

Step 2: Deliver the Archive to the Victim

Distribute the crafted RAR archive to the target user using a social-engineering lure (e.g., a benign-looking document or toolset). The archive should appear harmless when viewed in WinRAR, containing only non-executable or decoy content to reduce suspicion.

Step 3: User Extracts the Archive

The user opens the archive in WinRAR and extracts it to any directory of their choice using the normal extraction workflow.

- During extraction, WinRAR may display warnings about invalid paths or filenames.
- Despite these warnings, WinRAR proceeds to process the archive contents.

Step 4: Improper Path Validation Occurs

Due to insufficient normalization and validation of embedded file paths, WinRAR resolves one or more traversal paths outside the intended extraction directory.

As a result, one or more files are written to unintended file system locations, such as:

`%APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\`

Step 5: Persistence Is Established Automatically

Any file written into the Startup directory is automatically executed at the next user logon or system restart.

- No administrative privileges are required.
- No explicit execution by the user is necessary.
- Persistence is achieved at extraction time, before any visible malware execution.

Step 6: Malicious Code Executes on Logon

Upon the next user logon or reboot, the dropped file executes automatically, enabling follow-on actions such as:

- Script execution
- In-memory payload loading
- Further persistence mechanisms

Result

A remote attacker can leverage archive extraction alone to:

- Write files outside the intended directory
- Establish persistent execution
- Bypass common security prompts, all by abusing trusted application behavior.

Security Impact

- Attack Vector: User-assisted (archive extraction)
- Privileges Required: None (user-level)
- User Interaction: Required (extract archive)
- Impact: Arbitrary file write, persistence, potential remote code execution

ANALYSIS

Phase 1: Initial Access – Weaponized RAR Archive

Archive Name:

Free 2026 Osint Tools.rar

Initial inspection of the archive using WinRAR reveals what appears to be a harmless OSINT resource package. The archive displays a single text file, `Osint tools.txt`, with no visible executables, scripts, or suspicious file types. This presentation is deliberately crafted to reduce user suspicion and encourage extraction. The OSINT theme aligns with common research and security workflows, further legitimizing the archive and increasing the likelihood of user interaction.

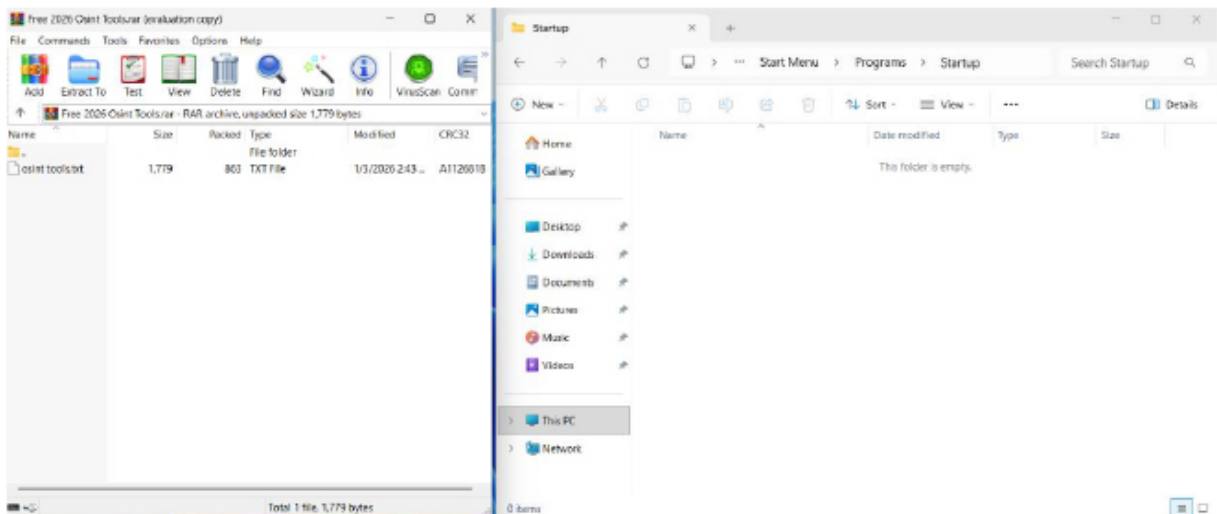


Figure 4: WinRAR displaying the contents of the weaponized archive, which appears to contain only a benign OSINT-related text file.

During extraction, WinRAR generates diagnostic warnings indicating that it cannot create certain files due to invalid filename or directory syntax. These warnings reveal the presence of nested directory traversal sequences embedded within the archive's internal file paths. The paths contain multiple relative traversal

elements (..\)) designed to escape the user-selected extraction directory. Although WinRAR reports an error, the extraction logic still evaluates the crafted paths, confirming improper path normalization and validation.

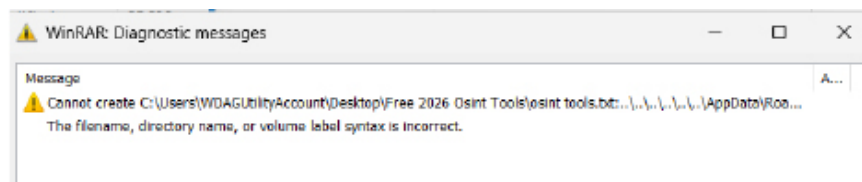


Figure 5: Comparison of the Windows Startup directory before and after archive extraction, showing the appearance of a malicious Batch file.

Any file placed in this directory is automatically executed at user logon or system reboot. This behavior enables persistence without requiring administrative privileges. The file placement occurs during archive extraction itself, meaning persistence is established before any explicit malware execution takes place.

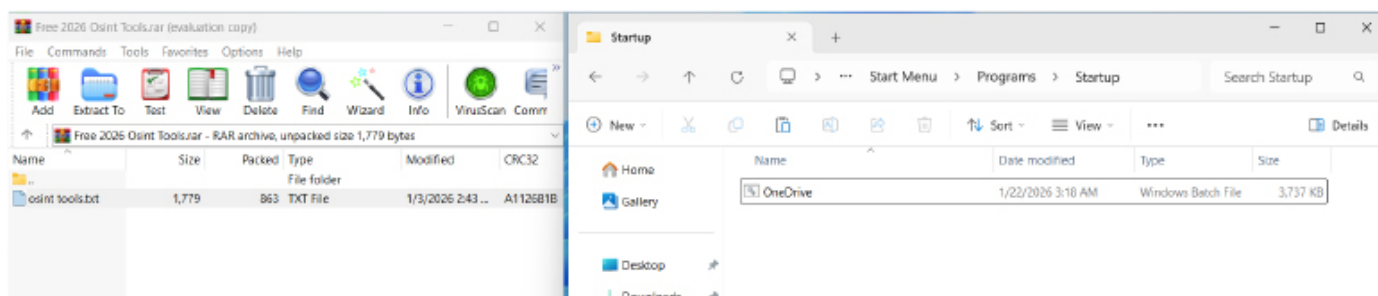


Figure 6: Comparison of the Windows Startup directory before and after archive extraction, showing the appearance of a malicious Batch file.

The embedded decoy file contains a curated list of open-source intelligence tools and links, including email harvesting platforms, domain intelligence services, metadata analysis utilities, and social media investigation resources. The content is realistic, relevant, and contextually appropriate for the lure theme. Its primary function is to distract the victim and reinforce the perceived legitimacy of the archive while the malicious Batch script is silently deployed in the background during extraction.

```
osinttools.txt
1 OSINT Tools:
2
3 Here is a list of free and useful OSINT tools.
4
5 Social Media Account Search
6 theHarvester - https://github.com/laramies/theHarvester
7 tool for collecting emails, subdomains, hostnames, etc.
8
9 Maltego - https://www.maltego.com/
10 Link analysis and OSINT platform
11 Spiderfoot - https://www.spiderfoot.net/
12 Automated OSINT scanning and data collection
13
14 Email address search
15 Hunter.io - https://hunter.io/
16 business email address search
17 Email Dossier - https://emaildossier.com/
18 Email address verification and analysis
19 Phonebook.cz - https://phonebook.cz/
20 Email address information search
21
22 Spies - https://spies.com/
23 Have Important information about email addresses
24
25 Phone number lookup
26 Whatsapp tech -
27 Add the number to your contacts then go to WhatsApp and view its profile
28 Telegram Bot - https://t.me/TrueCaller_Bot
29 With the number, get the Telegram, WhatsApp, operator, and first and last name
30 Num lookup - https://www.numlookup.com/
31 Get information about a number
32
33 Domain and IP address lookup
34 DNSDumpster - https://dnsdumpster.com/
35 DNS and domain information lookup
36 Shodan - https://www.shodan.io/
37 Search engine for internet-connected devices
38 WHOIS - https://whois.domaintools.com/
39 Domain information lookup
40
41 Metadata analysis
42 exiftool - https://exiftool.org/
43 Extraction and analysis of file metadata
44 Metagoofil - https://www.kali.org/tools/metagoofil
45 Extract metadata from public documents
46 Exif Viewer - https://www.exifviewer.org/
47 view image metadata
48
49 Search web content
50 Wayback Machine - https://archive.org/web
51 View archived versions of web pages
52 BuiltWith - https://builtwith.com/
53 Analyze website technologies
54 Wappalizer - https://www.wappalizer.com/
```

Figure 7: Contents of the decoy OSINT tools.txt file used to legitimize the archive and distract the victim.

Phase 2: Installation & Persistence – Obfuscated Batch Script

Obfuscation

The Batch script is heavily obfuscated using:

- Extensive variable indirection (%randomVar%)
- Randomized variable naming
- Loop-based execution flow

This design significantly complicates static analysis.

Hiding & Persistence

The script performs the following actions:

- Creates a hidden directory:
- %APPDATA%\Microsoft\D1HvyuJFBR7g
- Copies itself into this directory as:
- njzyslsx38rhxc5o.bat
- Sets the Hidden attribute on the directory and file

```
set HIDDENFOLDER=%APPDATA%\Microsoft\D1HvyuJFBR7g
set InkptQwc6V=!HIDDENFOLDER!\nJZySLsX38RhxC5o.bat

if not exist "!HIDDENFOLDER!" (
    mkdir "!HIDDENFOLDER!" >nul 2>&1
    attrib +h "!HIDDENFOLDER!" >nul 2>&1
)
```

Figure 8: Batch script creating a hidden directory and copying itself for persistence.

Defense Evasion

To suppress Windows security warnings, the script removes the Mark-of-the-Web (Zone.Identifier) using PowerShell:

Figure 9: Removing the Zone.Identifier (Mark-of-the-Web) to bypass security warnings.

Registry Persistence

The script establishes registry-based persistence:

Figure 10: Adding a Registry Run key to ensure the malware executes on system startup.

Data: cmd.exe /c " %APPDATA%\microsoft\d1hvyujfbr7g\njzyslsx38rhxc5o.bat"

This ensures execution at every user logon.

Phase 3: Payload Extraction & PowerShell Execution

The Batch script reads its own file contents and locates a marker-delimited Base64 payload:

```
powershell -NoProfile -ExecutionPolicy Bypass -Command "try { $content = Get-Content -LiteralPath '%~f0' -Raw -ErrorAction Stop; if ($content -match '9eC031IH8TKn3TLI:([A-Za-z0-9+/=]+)' ) { [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($matches[1])) | Out-File -FilePath '!TEMPPS!' -Encoding UTF8 } } catch {}" >nul 2>&1
```

[illegible]

Figure 11: PowerShell command extracting the embedded Base64 payload from the Batch file itself

Decoding & Execution

- The Base64 data is decoded to:
- %TEMP%\ps_eDxDA4lm.ps1
- The PowerShell script is executed with:
 - -ExecutionPolicy Bypass
 - -WindowStyle Hidden
 - -NoProfile

After execution, the Batch script deletes itself and exits, reducing on-disk forensic artifacts.

Figure 12: Execution of the extracted PowerShell payload in a hidden window.

Phase 4: Memory Injection – PowerShell Loader

Synchronization

The loader enforces single-instance execution using a named mutex:

```
$mutexName = "1Yu_zclrGMj_Mtx"
$mutex = $null
try {
    $mutex = [System.Threading.Mutex]::OpenExisting($mutexName)
    exit

    $mutex = New-Object System.Threading.Mutex($False, $mutexName)
```

Figure 13: Mutex check in the PowerShell loader to prevent multiple instances.

If the mutex exists, execution terminates.

Process Enumeration (Target Discovery)

Before injection, the loader explicitly enumerates running processes to locate explorer.exe, which serves as the primary injection target due to its stability, trust level, and persistence within the user session.

```
$process = Get-Process -Name explorer -ErrorAction SilentlyContinue
if (-not $process) {
    Start-Sleep -Seconds 6
    $process = Get-Process -Name explorer -ErrorAction SilentlyContinue
```

Figure 14: Loader searching for explorer.exe as the target for process injection.

Decryption

An embedded byte array is decrypted using an XOR operation with key: 87

Figure 15: Shellcode byte array definition and the XOR key 87 used for decryption.

Injection Mechanism

The loader defines direct imports of critical Windows APIs from kernel32.dll, including OpenProcess, VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread. This approach provides fine-grained control over remote process manipulation and is a well-established technique for reliable code injection on Windows systems.

The use of OpenProcess allows the loader to obtain a handle to a target process with sufficient access rights for memory allocation and thread creation. VirtualAllocEx is then used to allocate executable memory inside the remote process address space. The decrypted shellcode is written into this allocated memory using WriteProcessMemory, after which CreateRemoteThread is invoked to execute the injected payload within the context of the target process.

```

using System;
using System.Runtime.InteropServices;
public class zA8k5wM {
    [DllImport("kernel32")] public static extern IntPtr OpenProcess(uint dwDesiredAccess, bool bInheritHandle, int dwProcessId);
    [DllImport("kernel32")] public static extern IntPtr VirtualAllocEx(IntPtr hProcess, IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
    [DllImport("kernel32")] public static extern bool WriteProcessMemory(IntPtr hProcess, IntPtr lpBaseAddress, byte[] buffer, uint size, IntPtr lpNumberOfBytesWritten);
    [DllImport("kernel32")] public static extern IntPtr CreateRemoteThread(IntPtr hProcess, IntPtr lpThreadAttributes, uint dwStackSize, IntPtr lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr lpThreadId);
}

```

Figure 16: Native Windows API imports used by the loader to perform remote process injection.

Target Process Selection, Enumeration, and Watchdog-Based Reinjection

Before initiating injection, the loader enumerates running processes to identify a stable and trusted execution target. The primary target selected is explorer.exe, which is consistently present in interactive user sessions and operates with a high trust level within the Windows operating system. Injecting into explorer.exe allows the malware to blend into normal system activity while inheriting the stability and privileges of the user's desktop environment. To ensure execution continuity, the loader implements a fallback mechanism in which svchost.exe is selected as an alternative injection target if explorer.exe is unavailable or terminates unexpectedly.

To maintain persistence, the loader employs a continuous watchdog mechanism that monitors the health of the injected process. Operating within a timed loop, the watchdog periodically verifies whether the target process remains active. If termination is detected, the loader immediately attempts to reinject the payload into a newly spawned explorer.exe instance, falling back to svchost.exe if necessary. This self-healing reinjection strategy significantly increases resilience, as simply terminating the infected process does not remove the threat, thereby complicating incident response and prolonging attacker access.

```
$watchdogProcess = Get-Process -Name svchost -ErrorAction SilentlyContinue | Select-Object -First 1
```

```

if ($watchdogProcess) {
    $wp = $watchdogProcess.Id
    while ($true) {
        Start-Sleep -Seconds 8
        try {
            $checkProcess = Get-Process -Id $p -ErrorAction SilentlyContinue
            if (-not $checkProcess -or $checkProcess.HasExited) {
                $newProcess = Get-Process -Name explorer -ErrorAction SilentlyContinue
                if ($newProcess) {
                    $p = $newProcess[0].Id
                    if (-not $p -or $newProcess[0].HasExited) {
                        continue
                    }
                    $hp = [zA8k5wM]::OpenProcess($pf, $false, $p)
                    if ($hp -eq [IntPtr]::Zero) {
                        continue
                    }
                    $ad = [zA8k5wM]::VirtualAllocEx($hp, [IntPtr]::Zero, $1Yu_VNbnMwef.Length, $mc, $prw)
                    if ($ad -eq [IntPtr]::Zero) {
                        continue
                    }
                    $sc = [zA8k5wM]::WriteProcessMemory($hp, $ad, $1Yu_VNbnMwef, $1Yu_VNbnMwef.Length, [IntPtr]::Zero)
                    if (-not $sc) {
                        continue
                    }
                    $th = [zA8k5wM]::CreateRemoteThread($hp, [IntPtr]::Zero, 0, $ad, [IntPtr]::Zero, 0, [IntPtr]::Zero)
                    if ($th -eq [IntPtr]::Zero) {
                        continue
                    }
                    Start-Sleep -Milliseconds 520
                }
            } catch {
                continue
            }
        }
    }
}

```

Figure 17: PowerShell logic enumerating running processes to identify explorer.exe as the primary injection target, with a watchdog mechanism that reinjects the payload into svchost.exe if the original process terminates.

Phase 5: Donut Shellcode

Analysis of the decrypted payload confirms that it is Donut-generated shellcode, a widely abused framework for executing .NET assemblies entirely in memory. The shellcode is position-independent and architecture-agnostic, supporting both x86 and AMD64 environments, which allows it to operate reliably across a wide range of Windows systems. Rather than dropping an executable to disk, Donut initializes the .NET Common Language Runtime (CLR) directly within the address space of the injected process.

Once the CLR is initialized, the embedded .NET assembly is loaded and executed in memory without creating a corresponding file on disk. This fileless execution model significantly reduces forensic artifacts and bypasses traditional signature-based detection mechanisms. By executing within a trusted process context, the shellcode further blends malicious activity into normal system behavior, complicating both detection and incident response.

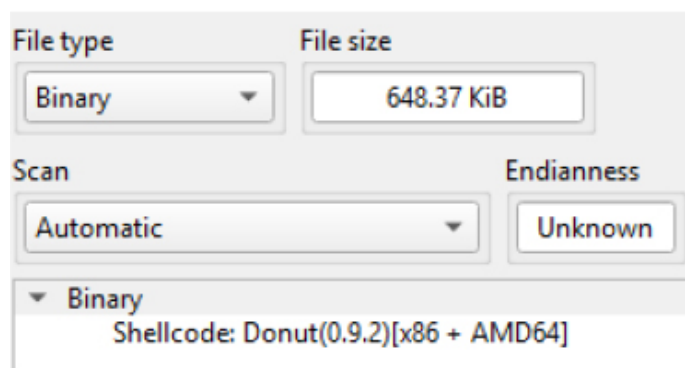


Figure 18: Identification of Donut-generated shellcode supporting both x86 and AMD64 architectures.

Phase 6: Quasar RAT – Final Payload

Extraction and inspection of the .NET assembly loaded by the Donut shellcode confirms the final payload as Quasar RAT Client, a well-known open-source remote administration tool frequently repurposed for malicious operations. Threat intelligence analysis shows the payload is detected by multiple security vendors and categorized as an MSIL-based trojan, with family labels consistently identifying it as Quasar.

Quasar RAT is written in C# (.NET) and provides comprehensive remote access functionality. Its capabilities include remote desktop viewing and control, file upload and download, keylogging and clipboard monitoring, credential and system information harvesting, arbitrary command execution, and registry manipulation for persistence management. When executed via Donut shellcode, these capabilities are delivered entirely from memory, enabling long-term, covert access while minimizing disk-based indicators of compromise.

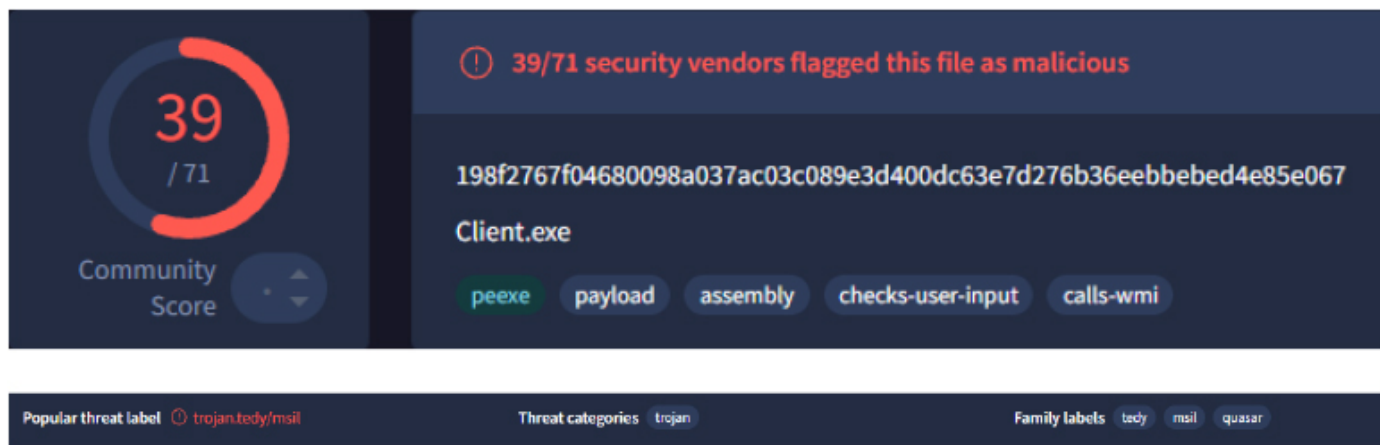


Figure 19: Malware family attribution and threat labeling, identifying the payload as Quasar RAT.

During dynamic analysis, active outbound network communication was observed from the infected host to the following external endpoint.

This connection was established post-in-memory execution of the Donut-deployed .NET payload, confirming that the Quasar RAT client successfully initialized and began C2 beaconing. The use of a direct IP address and a non-standard TCP port (4783) aligns with known Quasar RAT deployments, where hardcoded infrastructure is commonly leveraged to maintain reliable command-and-control while minimizing dependency on DNS-based indicators.

Figure 20: Dynamic analysis confirming active Quasar RAT C2 beaconing to 91.92.243.10:4783

EXTERNAL THREAT LANDSCAPE MANAGEMENT

The activity observed in this campaign reflects a broader shift in the threat landscape toward trust-abuse–based initial access and low-noise, fileless post-exploitation techniques. Rather than relying on exploit kits, macros, or privilege escalation vulnerabilities, threat actors increasingly exploit weaknesses in widely trusted applications and routine user workflows. Archive-based delivery, particularly through formats such as RAR, has become an effective vector as it often bypasses email and web gateway controls while exploiting normal user behavior. Vulnerabilities like WinRAR CVE-2025-8088 enable attackers to establish persistence at extraction time, before traditional security mechanisms can meaningfully intervene.

Following initial access, the use of in-memory execution, process injection into trusted Windows processes, and open-source tooling such as Donut and Quasar RAT aligns this campaign with techniques widely observed across commodity malware, access broker activity, and early-stage intrusions. These approaches significantly reduce disk artifacts and forensic visibility, complicating detection and response. Organizations lacking proactive vulnerability management, behavioral monitoring, and memory-level detection capabilities remain particularly exposed as adversaries continue to favor stealth, resilience, and operational efficiency over exploit complexity.

CONCLUSION

The exploitation of WinRAR CVE-2025-8088 documented in this report demonstrates how modern threat actors combine vulnerability abuse, social engineering, and living-off-the-land techniques to achieve persistent and covert compromise. By leveraging a trusted archive utility, the attacker is able to silently

place a malicious script into an auto-execution directory without executables, macros, or elevated privileges. This initial access method enables reliable persistence while minimizing user awareness and security alerts.

Subsequent stages of the attack chain, including obfuscated scripting, PowerShell-based loaders, Donut shellcode, and process injection into legitimate Windows processes, allow the final payload—identified as Quasar RAT—to execute entirely in memory. The inclusion of watchdog-based reinjection logic further enhances resilience, ensuring continued execution even if malicious processes are terminated. This campaign highlights the limitations of signature-based defenses and underscores the need for timely patching, behavioral detection, and memory-focused monitoring to effectively defend against stealthy, multi-stage threats operating below the detection threshold of conventional security solutions.

YARA Rules:

```
rule QuasarRAT_Attack_Chain_Unified {
  meta:
    description = "Detects Quasar RAT Dropper (Batch) and Loader (PowerShell) components"
    author = "CYFIRMA Research"
    hash = "d722f6b162a8cc49867856f94961ed7827ba4975a20632f6cdc6eba1096e3263"
    severity = "Critical"
    campaign = "Quasar RAT Exploitation"

  strings:
    $batch_marker = "::9eCO31IH8TKn3TLI::" ascii
    $batch_reg_key = "8l05CtD5UP" ascii
    $batch_motw = ":Zone.Identifier" ascii
    $batch_ps_exec = "powershell -NoProfile -ExecutionPolicy Bypass" ascii
    $batch_garbage = /SET [a-zA-Z0-9]+=([a-zA-Z0-9]+)/
    $ps_mutex = "1Yu_zcLrGMj_Mtx" wide ascii
    $ps_xor_key = "$1Yu_WqBWkPw = 87" ascii
    $ps_api_alloc = "VirtualAllocEx" ascii
    $ps_api_write = "WriteProcessMemory" ascii
    $ps_api_thread = "CreateRemoteThread" ascii
    $ps_watchdog_proc = "Get-Process -Name explorer" ascii

  condition:
    ($batch_marker) or
    (all of ($batch_reg_key, $batch_motw, $batch_ps_exec) and $batch_garbage)

  or
    ($ps_mutex) or
    ($ps_xor_key and 2 of ($ps_api_*)) or
    (3 of ($ps_api_*) and $ps_watchdog_proc)
}
```

RECOMMENDATIONS

Strategic Recommendations:

- Organizations should prioritize timely patch management for widely deployed consumer and enterprise software, including file archiving utilities, such as WinRAR. Vulnerabilities in trusted applications represent a high-risk attack surface, as exploitation often requires minimal user interaction and bypasses traditional perimeter defenses. Regular vulnerability scanning and asset inventory validation should be enforced to ensure unsupported or outdated software versions are promptly identified and remediated.
- Security teams should also implement content inspection and behavioral analysis for archive-based deliveries at email and web gateways. Given the increasing abuse of archive formats for initial access, reliance solely on file extension filtering or signature-based scanning is insufficient. Archive extraction behavior, including anomalous path traversal attempts and writes to auto-execution directories, should be treated as high-confidence indicators of compromise.

Tactical Recommendations:

- Endpoint detection controls should be configured to monitor and alert on unauthorized modifications to user-level auto-execution locations, particularly the Windows Startup directory and registry Run keys under HKCU. The silent placement of scripts into these locations, especially during archive extraction events, should be considered suspicious and investigated immediately.
- PowerShell activity should be tightly controlled and monitored. Alerts should be generated for PowerShell executions using ExecutionPolicy Bypass, hidden windows, and Base64-decoded script content. Additionally, security tools should flag behaviors indicative of process injection, such as the sequential invocation of OpenProcess, VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread, particularly when targeting trusted processes like explorer.exe or svchost.exe.
- Given the use of Donut-generated shellcode, organizations should deploy memory-level detection capabilities capable of identifying anomalous CLR initialization within non-.NET parent processes. Monitoring for in-memory .NET assemblies loaded without corresponding files on disk is critical for detecting fileless malware such as Quasar RAT.

Management Recommendations:

- Security awareness programs should be updated to address the growing threat of archive-based social engineering attacks. Users should be educated that archives containing only benign-looking documents can still pose a significant risk, particularly when extracted using vulnerable software. Emphasis should be placed on cautious handling of unsolicited archives, even when no executables are present.
- From an operational perspective, organizations should conduct regular incident response simulations that include archive-based initial access scenarios and fileless malware execution. These exercises help validate detection coverage, response workflows, and cross-team coordination. Investment in advanced endpoint detection and response (EDR) solutions with strong behavioral and memory analysis capabilities should be considered essential to counter modern low-noise and high-persistence threats such as the one documented in this report.

APPENDIX1

MITRE Mapping

Tactic	Technique	ID	Description
Initial Access	Exploit Client Software	T1203	The attacker exploits improper path validation in WinRAR (CVE-2025-8088) to write malicious files outside the intended extraction directory during archive extraction.
Initial Access	User Execution	T1204.002	The attack relies on the user manually opening and extracting a weaponized RAR archive disguised as a legitimate OSINT tools collection.
Execution	Command and Scripting Interpreter: Windows Command Shell	T1059.003	A malicious Batch script (OneDrive.bat) is executed automatically from the Windows Startup directory at user logon.
Execution	Command and Scripting Interpreter: PowerShell	T1059.001	PowerShell is used to decode and execute embedded Base64 payloads, remove security indicators, and launch the in-memory loader.
Persistence	Boot or Logon Autostart Execution: Startup Folder	T1547.001	The malicious Batch script is silently placed into the Windows Startup directory to ensure execution at user logon.
Persistence	Boot or Logon Autostart Execution: Registry Run Keys	T1547.001	A registry Run key is created under the current user to execute the hidden Batch script at every logon, providing redundant persistence.
Defense Evasion	Obfuscated Files or Information	T1027	The Batch script and PowerShell loader employ variable obfuscation, randomized names, and encoded payloads to evade static detection.
Defense Evasion	Remove Mark-of-the-Web	T1553.005	The malware removes the Zone.Identifier alternate data stream to suppress SmartScreen warnings and security prompts.
Defense Evasion	Hide Artifacts: Hidden Files and Directories	T1564.001	The malware creates hidden directories and files within the user's AppData path to conceal persistent components.
Privilege Escalation	Process Injection	T1055	The PowerShell loader injects Donut-generated shellcode into trusted processes using OpenProcess, VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread.
Defense Evasion	Process Injection	T1055	Injecting shellcode into explorer.exe and svchost.exe allows the malware to evade detection by blending into trusted processes.
Execution	Native API	T1106	Direct invocation of Windows native APIs from kernel32.dll is used to perform memory allocation, writing, and thread creation.

Persistence	Event Triggered Execution: Watchdog Timers	T1546	A watchdog loop continuously monitors the injected process and reinjects the payload if the target process terminates, ensuring self-healing persistence.
Defense Evasion	Fileless Malware	T1055.012	Donut shellcode executes the final .NET payload entirely in memory without writing the RAT binary to disk.
Command and Control	Application Layer Protocol	T1071	Quasar RAT communicates with its command-and-control server using standard application-layer network protocols.
Collection	Input Capture: Keylogging	T1056.001	Quasar RAT provides keylogging and clipboard monitoring capabilities for credential and data theft.
Collection	Screen Capture	T1113	The RAT enables remote desktop viewing and control of the victim system.
Discovery	Process Discovery	T1057	The PowerShell loader enumerates running processes to identify suitable injection targets such as explorer.exe.
Impact	Remote Services	T1021	The attacker maintains long-term remote access to the compromised system via Quasar RAT.

IOCs

No	Indicators of Compromise (IOCs)	Type	Remarks
1	198f2767f04680098a037ac03c089e3d400dc63e7d276b36eebbebed4e85e067	SHA256	RAT
2	bf12a13ed0fac0de3dba45f65f14e67b0b62a30cde88ae18785328c8b34b5995	SHA256	Donut Shell
3	4b82f29b17b977a42f08d4f47a505740a31be273ecd76f9655643ed5ce44bb86	SHA256	RAR File
4	d722f6b162a8cc49867856f94961ed7827ba4975a20632f6cdc6eba1096e3263	SHA256	BAT
5	91.92.243.10	IP	C2

Copyright CYFIRMA. All rights reserved.