

Beyond the Backdoor: How Contagious Interview Is Surgically Tampering with MetaMask Wallets

 sp4rk.medium.com/beyond-the-backdoor-how-contagious-interview-is-surgically-tampering-with-metamask-wallets-0314ae901d85

Seongsu Park

February 14, 2026

Introduction

Contagious Interview is an ongoing cyber threat campaign targeting IT professionals working in cryptocurrency, Web3, and artificial intelligence sectors. The campaign, orchestrated by North Korean threat actors, aims to steal financial information and sensitive data from developers and engineers.

The attackers consistently deploy two primary malware families, BeaverTail and InvisibleFerret, regularly updating these tools with new capabilities. Recent analysis indicates that the threat actors have significantly expanded their data theft capabilities by incorporating manipulation of the MetaMask wallet extension, making the campaign more aggressive and effective in compromising victim systems.

Key Takeaways

- The attackers simplified their first-stage JavaScript code to perform two core functions, confirming successful infection by sending a beacon to their servers, and downloading the next stage of the attack.
- The fetched scripts are two additional JavaScript files and a Python-based malware called InvisibleFerret from the attackers' remote server.
- The downloaded JavaScript files serve distinct roles, one creates a backdoor for remote access and control, while the other searches for and steals sensitive files from the victim's computer.
- Through the backdoor, attackers instruct the infected system to download and install a fake version of the popular MetaMask cryptocurrency wallet extension, complete with a dynamically generated configuration file that makes it appear legitimate.
- Once installed, the compromised MetaMask extension silently captures the victim's wallet unlock password and transmits it to the attackers' command-and-control server, giving them complete access to cryptocurrency funds.

Technical Analysis

The Contagious Interview campaign has relied on script-based malware for years, primarily using JavaScript and Python to compromise victims' systems. Over time, the threat actors have experimented with various delivery methods to distribute their malware. The attackers have

disguised their malware within legitimate-looking chat applications and developer tools, including fake VSCode extensions. However, their most consistent tactic still involves creating fake NPM packages. These poisoned packages are presented to job candidates as part of technical skills assessments during the interview process, making them appear legitimate and work-related. While monitoring these malicious packages, newly updated attack chain was discovered. Most notably, they've introduced new techniques specifically designed to steal cryptocurrency wallets and related sensitive data from victims' computers. Simply, the infection flow goes through the following process.

1. The victim unknowingly executes malicious JavaScript inside a delivered NPM package.
2. After acquiring the next stage C2 address, initial stage JavaScript fetches additional JavaScript that is responsible for downloading additional scripts.
3. The fetched JavaScripts are for light-weight backdoor, file exfiltration, and Python-based backdoor, InvisibleFerret.
4. Using commands sent through the light-weight backdoor, the threat actor deployed an additional script designed to manipulate the victim's MetaMask wallet.

Fig 1: Infection diagram

Initial stage JavaScript

Once the malicious JavaScript within the trojanized NPM package is executed, it established contact with the C2 server with defined endpoint such as `/s/90284f2a56f2`, and the server returned an encoding string(e.g. `ZT3NjcuMjAzLjcuMjA1LFpVMVJJT2s5`). After removing the `ZT3` marker, the script Base64-decodes the content and extracts the information, the next-stage C2 server address(`67.203.7[.]205`) and campaign identifier(`ZU1RIOk9`).

This represents a significant tactical change from earlier versions of the campaign. Previously, the initial script contained more extensive functionality to directly download and execute subsequent payloads. The updated version, however, has been deliberately stripped down to perform only three essential functions: downloading the encoded next-stage C2 server addresses, sending a beacon to confirm successful infection, and retrieving the next stage of the attack. By minimizing the initial script's capabilities, the attackers reduce the likelihood of detection. After obtaining the next-stage C2 address, the script analyzed in this blog follows the steps below:

1. Downloads and saves to: (or home directory if creation fails)
2. Downloads , saves to: .
3. Executes for installing npm dependencies from the downloaded .
4. Executes with the options . This runs the downloaded malicious JavaScript payload silently.

Intermediate Stager

The fetched JavaScript file(*test.js*) performs two primary functions: stealing sensitive files from the victim's system and downloading additional payloads from the remote server. In the latest Contagious Interview campaign, the attackers have divided sensitive data exfiltration into multiple stages, with this script handling the initial theft. The script targets the same file types consistently pursued in previous campaigns, browser credentials, cryptocurrency wallets, and password manager data, across Windows, macOS, and Linux platforms.

This script retrieves two additional JavaScript files from the C2 server identified in the previous stage and downloads InvisibleFerret, the Python-based backdoor that has served as the campaign's primary malware tool for years.

URL	Saved file path	Description
/z/[campaign ID]	~/ .vscode/p.js	File exfiltration JavaScript
/n/[campaign ID]	~/ .vscode/n.js	Light-weight backdoor JavaScript
/client/[campaign ID]	~/ .npl	InvisibleFerret variant

File Exfiltration and Light-weight Backdoor

File Exfiltration JavaScript

One of the additionally delivered JavaScript(*p.js*) components functions as an automated file discovery and exfiltration module. This script has a singular focus, systematically searching the compromised system for files containing sensitive information and transmitting their contents to the C2 server. The module employs pattern-based file discovery with platform-specific search implementations for Windows and macOS systems. The module defines seventeen distinct search patterns targeting cryptocurrency wallets, development environment secrets, password managers, and private keys. The targeted patterns include files matching keywords such as “*env*”, “*config*”, “*wallet*”, “*metamask*”, “*private*”, “*mnemonic*”, “*password*”, “*account*”, “*seed*”, “*keys*”, “*kdbx*” (KeePass databases), “*solana*”, “*hardhat*” (Ethereum development framework), and related variations.

- : find / \(-path “*/[excluded_dir]” -prune \) -o -name “[pattern]” -print
- : dir /b /s [drive]:\[pattern] | findstr /v /i “[exclusions]”

Light-weight Backdoor JavaScript

The second JavaScript file(*n.js*) delivered by the previous stage serves as a light-weight backdoor. This JavaScript backdoor shares similar functionality with InvisibleFerret, the Python-based backdoor the threat actors have relied on for years. The backdoor uses an identical C2 communication pattern to InvisibleFerret, establishing a persistent TCP socket connection to *202.163.147[.]124:1248*. Communication follows a length-prefixed message format: each packet begins with a 4-byte big-endian integer specifying the JSON payload length, followed by the UTF-8 encoded JSON data. Messages contain a “*code*” field indicating the command type and an “*args*” field containing command-specific parameters. Most of the backdoor commands match those previously observed in InvisibleFerret variants. However, one notable addition is the “*ssh_mmc*” command, which fetches an additional script from the remote server.

Code	Function	Description
0x1	ssh_obj	Executes arbitrary shell commands with full output streaming. Special handling for “cd” command to change working directory.
0x2	ssh_cmd	Processes control commands. “delete” command terminates the backdoor and stops the reconnection loop.
0x5	ssh_upload	Multi-mode file collection system supporting directory uploads, single files, and pattern-based discovery.
0x6	ssh_kill	Forcefully terminates Chrome and Brave browser processes (taskkill on Windows, killall on Unix).
0x8	ssh_env	Automated exfiltration of Documents, Downloads, and drive contents. Only uploads files matching sensitive keywords.
0xa	ssh_mmc	Downloads <i>y.js</i> from <i>/c/</i> endpoint and executes it detached with Node.js.

MetaMask extension manipulation

The JavaScript(*y.js*) is fetched by the “*ssh_mmc*” command for a browser extension manipulation attack targeting Chromium-based browsers. The malware specifically targets the MetaMask cryptocurrency wallet extension, modifying browser configuration files to inject attacker-controlled code that intercepts wallet’s key. MetaMask switching process following this order:

1. Scans all profiles across Chrome and Brave browsers to check whether MetaMask is installed by looking for “”.

2. Creates a temporary directory in the system temp folder, fetches a malicious extension ZIP file from the C2 server, extracts the ZIP file to the MetaMask folder, and removes the downloaded ZIP file.
3. Reads the victim's Preferences file and parses the JSON to extract current extension settings.
4. Exfiltrates the victim's profile information to the C2 server, including system ID, extension path, and current extension settings.
5. Overwrites the extension path to point to the malicious extension folder and modifies protection MAC signatures and super MAC values.
6. Enables developer mode to bypass extension signature verification.
7. Removes the "" directory to clear cached extension code and force the browser to load the malicious version.
8. Terminates the Chrome or Brave browser process, forcing it to restart with the compromised extension.

Switching MetaMask and Configuration

To replace the legitimate MetaMask wallet with a malicious version, the script (*y.js*) searches for the MetaMask extension on the victim's system. If the extension is found, the malware constructs a download URL to retrieve the malicious extension package from the C2 server. The URL follows this format:

- URL format:
- Example:

The critical process of manipulation of MetaMask wallet occurs through surgical modification of the browser's Secure Preferences and Preferences JSON files. The malware receives configuration data from the C2 server endpoint at `"hxxp://66.235.28.238:1249/hm"` after transmitting system information including hostname, username, platform, file separator character, and the system identifier. The server responds with extension-specific settings and MAC (Message Authentication Code) values required to bypass Chrome's tamper detection mechanisms.

Example of delivered configuration file:

...

...

...

Chrome employs HMAC-SHA256 signatures to detect unauthorized modifications to extension preferences. Each extension's settings are protected by an individual MAC value stored in the

Secure Preferences file. The browser recomputes these signatures each time preferences are loaded and rejects any configuration where the calculated HMAC does not match the stored value. The configuration file includes three distinct HMAC values that correspond to different

Join Medium for free to get updates from this writer.

validation layers within Chrome's security model.

- The extension-specific MAC validates the MetaMask extension settings object.
- The developer mode MAC authorizes the enabling of developer mode, which permits loading unpacked or unsigned extensions.
- The super MAC provides top-level validation for the entire preferences structure.

When the malware writes these HMAC values alongside the modified extension settings, Chrome's validation routine computes the expected signatures and finds them matching the provided values. If validation succeeds, the browser loads the extension configuration without warning the user, otherwise resetting to defaults. The exact method the threat actors use to dynamically generate valid MAC values from their server remains unclear, as this analysis shows different values generated for each victim. When I tested this attack method using the fetched MetaMask extension and configuration data on the latest version of Chrome, the forced extension replacement failed. Chrome removed the manipulated extension, suggesting either that additional conditions are required for the attack to succeed or that Chrome has already implemented mitigations against this specific technique.

Trojanized MetaMask Wallet

The MetaMask wallet extension is a prominent program containing thousands lines of code. Rather than surgical this comprehensive program many points, the actor only injects minimal lines of code to low detection, but works efficiently. The trojanized MetaMask wallet contains one malicious JavaScript file(*background-2.js*) that follows this procedure to steal credentials:

1. Captures master password when user unlocks wallet.
2. Exfiltrates encrypted vault file containing all seed phrases/private keys.
3. Sends data to C2 server().
4. Attackers decrypt offline to obtain seed phrases.
5. Attackers use seed phrases to drain funds at their leisure.

There are two injected points that do not exist in the legitimate version. The first part is *report()* function to send a critical date to the C2 server.

```
() {
  {
    e = (, {
      :,
      : ({
        : t,
        :
      })
    })
  }
  (e2) {
    .(e2)
  }
}
```

The other modified function is *submitPassword()* function. The legitimate function was modified to include two malicious *this.report()* calls:

```
() {
  { : t } = . . ;
  {
    . . (e),
    . ( )
  } (e1) {
    . ( ), e1
  }
}
```

Once the user successfully unlock the MetaMask wallet, it sends the user entered master password and vault file.

- MM_ok: — Success marker
- \${e} — Plaintext master password
- \${escape(e)} — URL-encoded password (for special characters)
- \${this.store?._state?.KeyringController?.vault} — Entire encrypted keyring vault

The attackers modified only two functions, adding approximately 15 lines of code that captures master passwords and encrypted vault data during wallet unlock operations. This surgical approach ensures the extension maintains full functional compatibility with legitimate MetaMask while exfiltrating credentials sufficient for complete wallet compromise.

Comparison of File Exfiltration

Not only implanting trojanized MetaMask extension, one of the primary purposes of this infection is exfiltrating sensitive files to the C2 server through several scripts. Through infection chain, file exfiltration happened in four stages: intermediate stage script(*test.js*), Light-weight backdoor JavaScript(*n.js*), File exfiltration JavaScript(*p.js*) and InvisibleFerret backdoor. At the initial stage, the threat actor focuses on stealing Browser-based data, subsequently sends keyword-based sensitive files, and finally steals more data depending on the threat actor's commands. It shows how the threat actor aggressively steals critical data from the victim through multiple stages.

Feature	test.js	p.js	n.js	InvisibleFerret
Browser Extensions	O(17)	X	X	O(45+)
Browser Passwords	O	X	X	O
Crypto Wallets	O	X	X	O
Password Managers	O(1Pass)	X	X	O(1Pass, Dashlane, etc.)
Keychain (macOS)	O	X	X	X
Pattern File Search	X	O	O	O
Document Files	X	O	O	O
.env/Config Files	X	O	O	O
Keylogger	X	X	X	O(Win)
Clipboard	X	X	X	O(Win)
Remote Shell	X	X	O	O
File upload/download	X	X	O	O
Encryption	X	X	XOR	XOR+Archive
Exfil method	HTTP	HTTP	FTP	FTP+Telegram
C2 server	67.203.7.205:1244	67.203.7.205:1244	Receives from C2	Receives from C2

Conclusion

The BeaverTail variant used in the Contagious Interview campaign remains one of the most actively deployed malware tools by DPRK-affiliated threat actors targeting financial gain. Since its initial deployment, BeaverTail has undergone continuous updates with evolving infection methods and procedures. This research reveals how the threat actors exfiltrate cryptocurrency-related sensitive data and directly steal cryptocurrency funds from victims' wallets.

Indicators Of Compromise (IOCs)

IOCs related to this research:

800ffb10a79370991c5c918f572fe192, poker_top.tar.gz

8e6db10b5acc15c2cc54592e3dd49bf7, /routes/api/users.js

ddec84f075036f4afee55e708987b05a, test.js

687b235572f3b35c0eb5c6c742862db4, p.js

6d3f1aeed4feca39cb5d53f59bf6d9a5, n.js

6244da9940f50b9f51e3d85766cb1226, y.js

d423bf6b18662aed88ddd69c72b4e116, background-2.js

45.43.11.248:1244

45.43.11.200:1244

67.203.7.205:1244

202.163.147.124:1248

145.59.1.45:1244

66.235.28.238:1249

66.235.168.238:1244

147.124.202.163:1243

Additional IOCs for further research:

d80a29cefae892d26567b14ba9ba21c6, next.config.js

3013d942ee75ee982f66d7d1021a759d, tailwind.config.js

f55560735ae028745cf6c90488b07bd7, next.config.js

b18101a943a149cb1cbb3cac3b4f9f6c, searchFeatures.js

Intermediate Stager:

ddec84f075036f4afee55e708987b05a, test.js

900b95205e414e04eacd0ba5dc4868a5, test.js

Light-weight backdoor:

211d0fad75d20a032803e7cc0d277e09, n.js

6da79a0ddb7c4923f834ba723f8aea6f. skAcMa0

427bb906b72388381ed3d1ef22f0b3ad, reCgNg6

998cc427b2be37bd9dbb109bd1843366, dGVhbTE1

134102aa60f0a97a78a6299b35c30e69, n.js

MetaMask Wallet Switcher:

1b39dfc0ef262baba95b58e3b8d81c8e, background-0.js