Shai-Hulud npm supply chain attack: What you need to know | ReversingLabs

By Karlo Zanki, Reverse Engineer at ReversingLabsKarlo Zanki

Published: 2025-10-10 · Archived: 2026-04-05 19:22:33 UTC

ReversingLabs (RL) researchers detected a first of its kind self-replicating worm on the npm open-source registry on September 15. The worm, named “Shai-hulud,” spreads through a cascading compromise of npm accounts that inserts the malicious worm code into legitimate public and private npm packages belonging to the compromised developer.

The name “Shai-hulud” is taken from the open-source repository holding the malicious code. Fans of Frank Herbert's acclaimed "Dune" series will recognize the name of the giant sandworms native to the planet of Arrakis. Much like its namesake, this npm worm is feasting on those that wander the deserts of modern, open-source software development. Tokens, keys, and private repositories all get gobbled up by this malware behemoth.

Once infected by Shai-hulud, npm packages spawn attacks of their own by unknowingly allowing the worm to self-propagate through the packages they maintain. Given the large number of package inter-dependencies in the npm ecosystem, it is difficult to predict who will get compromised next — and how far Shai-hulud could spread. At the time of publishing, RL has identified hundreds of npm packages that have been compromised by the Shai-hulud malware.

This is a ongoing attack (see updated IOCs below), and it is apparent that the impact of the Shai-hulud malware outbreak will be large. RL has identified a number of popular packages that have been compromised, including *ngx-bootstrap (300k weekly downloads)*, *ng2-file-upload* (100k weekly downloads) and *@ctrl/tinycolor* (2.2M weekly downloads). Together, these packages account for millions of combined weekly downloads, making this a high-impact supply chain compromise.

There are also a large number of affected parties. Among those are tech company founders and CTOs; companies providing software development services; developers working for non-profit organizations; tech leads in companies building gambling hardware and software and creating office development suites; developers in AI-first companies; security vendors — including a leading endpoint detection and response (EDR) vendor; student developers; and others that rely on npm each day to build software.

RL has contacted as many affected parties as possible, but the speed with at which the worm spread made it impossible to reach everybody at disclosure time.

While the exact origin of the Shai-hulud malware — and the actors responsible for the malware — have not been identified, RL researchers note striking similarities with the Nx compromise that occurred at the end of August. While the malware used in the two attacks is different, the Nx attack and the Shai-hulud campaign use similar techniques. Both attacks target popular open source packages with millions of weekly downloads. When deployed,

both malicious payloads focused on collecting environment- and other secret information from infected machines and leveraged user-owned GitHub accounts for data exfiltration.

Now, we have a self-replicating worm spreading through the open source supply chain. While there is no way to know what threat vector we might face next, it is clear that the attackers are using every tried and true technique on the new frontier for defenders: the software supply chain.

The Shai-hulud worm itself is a 3MB+ behemoth of JavaScript. However, what it does is pretty straightforward.

After an npm developer account is compromised, the worm looks for other packages the developer maintains. It then creates a new version of each of those packages by injecting itself into them. Each newly created package is modified with a *postinstall* action that will execute the malicious *bundle.js* when an unsuspecting user downloads the compromised package. This is repeated in perpetuity as the worm finds new developers to infect, and then uses them to spread even further.

Figure 1: Postinstall script added to package.json file.

To speed-up the spread of the Shai-hulud malware, the threat actors behind it implemented worm-like functionality that automatically updates the packages published by compromised npm accounts with this same *bundle.js* file. To do that, a function named *updatePackage* adds a *postinstall* script to the *package.json* file that adds the *bundle.js* file to the package archive.

Figure 2: Code responsible for adding malicious functionality to affected npm packages.

We have identified at least one package in which autospreading has occurred, so it can be confirmed that this spreading technique functions properly and probably explains the rapid growth in infected npm packages in the last 24 hours.

The malicious *bundle.js* script is also designed to steal cloud service tokens, primarily targeting npm, GitHub, AWS and GCP tokens. However, the worm also installs TruffleHog, a popular open-source tool that can detect more than 800 different types of secrets. This significantly increases its secret-stealing capabilities.

Figure 3: Part of code showing which service tokens are targeted in the attack.

The collected secrets are then exfiltrated to newly created GitHub repositories using stolen GitHub access tokens. The created repository has the name "Shai-Hulud" and description "Shai-Hulud Repository." Exfiltrated data is double Base64-encoded and uploaded to a file named *data.json* in the newly created repository.

The compromised repositories can be tracked using a GitHub search for the repository description. Keep in mind that search results show only the currently active repositories and we can confirm that there were more victims that removed the exfiltration repositories from their GitHub accounts after becoming aware of the compromise.

Figure 4: GitHub search showing the compromised user repositories.

Another method for GitHub token exfiltration is by creating a new branch named *shai-hulud* in one of the existing repositories. A workflow file named *.github/workflows/shai-hulud-workflow.yml* is uploaded to that branch.

Figure 5: Malicious GitHub workflow file designed to exfiltrate secrets.

The GitHub action has a runnable action triggering on the PUSH event that is designed to exfiltrate the tokens accessible from the workflow environment to the url *hxxps://webhook.site/bb8ca5f6-4175-45d2-b042-fc9ebb8170b7*. This data is also double Base64-encoded. Unfortunately, using GitHub search for branch names is not supported, so it is a bit harder to track the victims using this information.

Another malicious functionality implemented by this malware is the "migration" of private repos belonging to the compromised GitHub account to publicly accessible repositories. The worm tries to create a public copy of all private repositories belonging to the compromised user. This is likely an attempt to gain access to secrets hardcoded in those repositories, and possibly to steal the source code they contain. That stolen code can be analyzed for vulnerabilities that can be used in later attacks on the software.

The newly created repositories get a suffix *-migration* to their original name, and have the description *Shai-Hulud Migration*. This information can again be used to search for and identify victims that have had private repos exposed using GitHub search. At the time of publication, that search yielded close to 700 results on GitHub, giving an indication of the scope of the attack.

Figure 6: GitHub search showing the "migrated" private repositories.

The first npm package for which RL has information about compromise is *rxnt-authentication*. The malicious version of that package, 0.0.3, was published on September 14 at 17:58:50 UTC. As a result, the npm maintainer *techsupportrxnt* can be considered Patient Zero for this campaign.

The exact nature of the initial compromise is not known. However, the past few days saw widespread attacks on open source maintainers that resulted in compromises of leading open source code. Those attacks utilized phishing and social engineering to seize control of maintainers' accounts and modify code. Victims of that campaign included the npm user Qix, a maintainer of some of the most used packages on npm (top 10) — debug and chalk.

According to Wiz, Qix's packages are present in 99% of cloud environments, and 10% of them used the malicious version. RL identified more than 350 other open source packages — many with a lower profile than the Qix packages — that were infected with the same malware.

While the sole nature of the malware used in that incident resulted in minimal impact, it showed how big the exposure surface is. A similar incident occurred in the last week of August, when the authors of another popular package, called Nx, were compromised through a vulnerability in their GitHub actions code. Some security researchers named this attack "s1ngularity" from the name of GitHub repositories used to exfiltrate stolen user secrets. (More about this incident can be found in a recapitulation written by Wiz.)

Key takeaways from the incident are that the packages were compromised by abusing vulnerable GitHub actions, while the malicious code was designed to steal user secrets and web service-tokens. In that attack, data exfiltration was performed by creating new specifically named repositories from the compromised user accounts and private GitHub repositories were exposed.

The design and functional overlap of the Nx campaign with the Shai-hulud worm RL detected is significant. The only remaining question for us is how patient 0 was compromised? While social engineering is a likely tool, the

exploitation of vulnerable GitHub actions cannot be ruled out.

What is even more concerning is the automated spreading of malware to the packages maintained by the compromised npm accounts. Historically, malware worms like SoBig, WannaCry and NotPetya have spread by targeting remotely exploitable vulnerabilities in software or — going even further back — by playing to human weaknesses to trick users into executing malicious code on their own systems. (See the “I Love You” and “Anna Kornikova” viruses.)

Open-source platforms like npm present a new and promising environment for malware to propagate. And, when malware spreads at the speed of continuous integration/continuous delivery (CI/CD) rather than human point-and-click, how far can it spread, and how quickly can it be stopped? Do we need a break-glass function, or a big red button that can temporarily halt all new package publication to platforms like npm? And, if so, who would be tasked with pressing it? Also, how quickly could they act?

The nature of this malware, designed to steal and collect access tokens and quickly reuse them, is making the Shai-hulud outbreak even more important than the Qix developer account compromise that occurred last week. This time around, there’s more at stake than a few stolen crypto-coins. With leading packages compromised — and untold numbers of developer secrets and proprietary code exposed — the keys to the kingdom have been leaked, and there’s no way of telling how they can and will be misused by malicious actors.

To check if your organization is infected, or if your cloud accounts were compromised, review your public GitHub account for suspicious activities like a sudden appearance of repositories you did not publish. Another tell-tale: If any of your repositories suddenly change visibility from private to public.

If you do not have a public GitHub account but are publishing packages on npm, check to see if there are any new versions of your packages that you haven’t authored but that have appeared on npm this week.

You can also check package versions for npm packages you are maintaining using RL’s free Spectra Assure Community website. The RL threat research team is diligently reviewing all published npm packages, and applying the analyst-vetted malware label to confirm that the package has been infected. Here’s is an example of what to look for:

Figure 7: Spectra Assure Community website showcasing an analyst-vetted detection related to this incident.

Indicators of Compromise (IoCs) refer to forensic artifacts or evidence related to a security breach or unauthorized activity on a computer network or system. IOCs play a crucial role in cybersecurity investigations and incident response efforts, helping analysts and security professionals identify and detect potential security incidents.

The following IOCs were collected as part of RL’s investigation of this malicious software supply chain campaign.

While crisis was averted with the recent Shai-hulud worm attack on npm, it proved that a self-propagating malware can automate the compromise of open-source packages. Here's what you need to know about the historic Shai-hulud malware outbreak.
