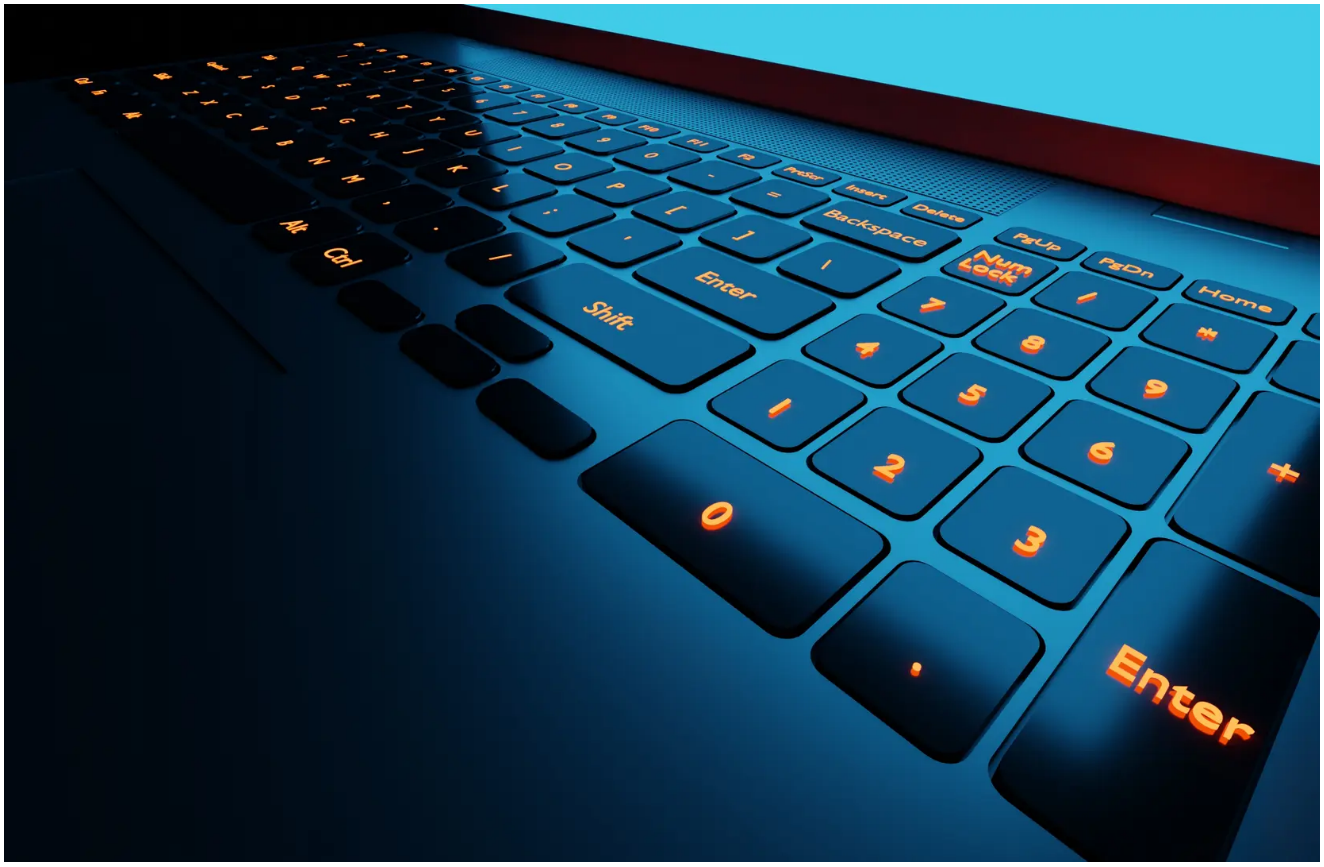
UAParser.js npm Package Supply Chain Attack: Impact and Response

truesec.com/hub/blog/uaparser-js-npm-package-supply-chain-attack-impact-and-response



Note: This is a developing story.

What Has Happened?

Malware was added to a very popular project on npm called *ua-parser-js* (> 7 million weekly downloads). Three malicious versions were published. Those were later superseded by benign versions and finally unpublished. The project is used directly or indirectly by many web projects. Most users are likely not even aware of using it.

Developers that have downloaded or installed npm packages during the time period may have received the malware. Figuring out if that has happened can be tricky. Other changes to the dependency tree or clean installs could have triggered the malicious package. At the very least, check package caches on developer machines for malicious versions and check for running malware.

The most dangerous time period seems to have been Friday 22 between ~12:15 and 16:27 UTC when the malware was tagged as the latest versions. It was available for explicit download a while after before being unpublished.

Why Is This a Notable Problem?

Malware embedded into package managers can endanger many parts of an environment. It can typically execute and affect the developer machine, the build environment (CI/CD), and likely even the production environment. A modern developer computer or CI/CD environment has access to many keys and secrets related to sensitive systems and can of course be used to leverage further attacks on customers as well.

The project is **very common** in modern web projects. It is a direct dependency of common packages such as "fbjs" and "karma". At the time of writing, *ua-parser-js* has 5.9k stars and 949 forks on GitHub. It is listed as having 1,215 direct (public) dependents on npmjs.com and can be expected to have many times as many indirect public and private dependents (deps.dev lists 53 208 publicly known indirect dependencies).
Since the malware likely steals credentials, the secondary effects may not be visible for a long time. Make sure to reimage machines and rotate secrets if affected.

Who Was Affected?

Broadly speaking, anyone that matches the following could have received the malware. If:

- You added the malicious versions to your package.json or otherwise installed the package (0.7.29, 0.8.0, 1.0.0).
- Had a direct or indirect dependency on the package in question, without explicitly locking down versions. For example, a dependency on the package with version set as "**^0.7.28**"

This can affect **both developer machines and CI/CD environments**! Recommendations with regards to rotating secrets and reinstalling will typically extend to servers using this package as well.

What Are Some Mitigating Factors?

- Not running npm/yarn install while the malicious package was available.
- Only running with frozen/locked dependencies. Note, however, that if you added some other package during the timeframe that might have led to the dependencies being reevaluated.
- If "ignore-scripts" or similar was set to true
- MacOS was likely unaffected

How Do I Know if I Was Affected?

As this is a developing story, it is hard to say what all indicators are. Truesec's Fabio Viggiani has posted notes on indicators of compromise and malware behavior on Twitter, and several other teams are working on incidents as well as analysis of the malware.

If you are running Windows with AV enabled it is likely that Defender and others have noted the malware.

Beyond looking for indicators of the malware running on systems, we can also look for the package on the machine:

- Look for the versions 0.7.29, 0.8.0 and 1.0.0 of ua-parser-js in *node_modules* or other cache directories
- Look for the malicious versions in any package-lock.json, yarn.lock, or similar package lock files.
- If a central dependency repository/cache such as Nexus or Artifactory is used: See if the malicious versions have been proxied/cached by any user.

It is **not sufficient to only look for the versions in lockfiles** as those could have been reevaluated on developer machines, ci/cd, or pr branches without being pushed to main.

Don't forget to check CI/CD. Pull requests for other package updates could have triggered the malware.

How Could This Happen?

Somehow the threat actor got access to the publisher's keys/identity and managed to publish malicious versions. It has not been publicly stated how the threat actor got access to the publisher's identity.

How Can We Avoid This in the Future?

The npm ecosystem is not well. Composability is admirable, but we can't trust hundreds of publishers. Locking versions, having vulnerability feeds, and leveraging quarantine can mitigate. We can propose MFA for publishers all we want, we still can't trust that many. I elaborate a bit on this in my blog post about supply chain threats and mitigations: https://www.truesec.com/hub/blog/secure-your-software-supply-chain-threats-and-mitigations.

Why Three Versions?

The project didn't have a 0.8 or 1.0 track previously, but the TA still pushed 0.8.0 and 1.0.0. This increased the reach, as those versions would be the latest minor and major versions (similar to how using later version works for dependency confusion

(https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610)

More

Keep up to date with the GitHub advisory (https://github.com/advisories/GHSA-pjwm-rvh2-c87w) and ua-parser-js issue (https://github.com/faisalman/ua-parser-js/issues/536).