npm hydra worm disclosure

Author: Sam Saccone

tl;dr

It is possible for a single malicious npm package to spread itself across most of the npm ecosystem very quickly. This package could enable delivery of a potentially targeted, malicious payload to corporate entities.

Installing code from a package manager has the same level of security as `curl site.com | bash,` however the majority of developers perceive gem install or apt-get install or npm install to be less risky.

The npm package manager, like most other package managers has a concept of lifecycle scripts. These scripts can execute arbitrary commands on your system (ideally doing operations like compiling code). In npm, these lifecycle hooks run as the current user (if run with sudo the command runs as nobody), with the current permissions in a non-sandboxed environment. Simply put, anything that you can do, the scripts can do.
However, the npm client and registry are exploitable due to a combination of the following factors.

Factor 1: semver allows for changing dependencies

The first part of the exploit begins with the simple fact that npm encourages users to use semver. Installed dependencies are not "locked" to a specific version by default and users must manually lock their projects via npm shrinkwrap. It is important to keep in mind that packages have sub dependencies that also may have semver ranges for their dependencies, and so on and so on.

Let's take PhoneGap for example. Currently it has 463 transitive dependencies, and of those dependencies, 276 individual npm accounts can push new versions of these packages. This means that when someone does npm install inside of the project… 463 unknown and unlocked dependencies are installed, each with the ability to execute arbitrary code at install time.

Factor 2: Persistent auth allows any script to publish with user credentials

The second part of the exploit depends on the fact that once a user is logged in to npm on their system they are never logged out until they manually do so. Since npm can run arbitrary scripts on install that means that any user who is currently logged in and types npm install is allowing any module to execute arbitrary publish commands.

Factor 3: Centralized registry serves huge userbase daily

The third part of the exploit is dependent on that fact that a singular npm registry is used by the the large majority of the node.js ecosystem on a daily basis, and typing `npm publish` ships your code to said registry server, to be installed by anyone.

Exploit Steps

Given the above combination of factors, it is a trivial exercise to author the self-replicating exploit. For example:

1. Socially engineer a npm module owner to npm install an infected module on their system.
2. Worm creates a new npm module
3. Worm sets a lifecycle hook on the new npm module to execute the worm on any install
4. Worm publishes the new module to the user's npm account
5. Worm walks all of the user's owned npm modules (with publish permissions) and adds the new module as a dependency in each's `package.json`.
6. Worm publishes new versions to each of the owned modules with a "bugfix" level semver bump. This ensures the majority of dependent modules using the `^` or `~` signifier will include the self-replicating module during the next install.

Exploit Impact

With these steps, the worm would be able to quickly spread across the ecosystem as users installed packages, as most npm users do without thinking. Using PhoneGap as a test bed, it would only take 1 person out of the 276 to install a package that contained the worm to infect the PhoneGap project.

Mitigation strategies:

User mitigation:

- As a user who owns modules you should not stay logged into npm. (Easily enough, `npm logout` and `npm login`)
- Use `npm shrinkwrap` to lock down your dependencies
- Use `npm install *someModule* --ignore-scripts`

npm mitigation:

- Automatically expire login tokens
- Require some form of two factor auth for publish operations
- Tell users that they should logout

Corporate mitigation:

- Run a local mirror of the npm registry
- Prevent installing from the main registry, and instead use a trusted audited registry.

Timeline of disclosures:

- **Jan 1 2016** -- Initial discovery of exploit
- **Jan 4 2016** -- Initial disclosure + proof of concept to npm
- **Jan 5 2016** -- Private disclosure to Facebook
- **Jan 7 2016** -- Response from npm
- **Jan 8 2016** -- Confirmation of works as intended no intention to fix at the moment from npm.
- **Feb 5 2016** -- Shared the disclosure doc

- **Jan 26 2015** -- "forced" reminder of "script runs as user" badness demonstration?
    - https://github.com/joaojeronimo/rimrafall
- **Jan 27 2015** -- Adam Baldwin (node security expert) follow up
    - https://blog.liftsecurity.io/2015/01/27/a-malicious-module-on-npm