



NPM Security

**Security & Stability risks when working
with node package manager(s).**

Why tho?

- prevent build and production failures
- prevent system compromisation
- prevent leaking of secret information



Image courtesy of [Miguel Vasquez](#)



Attack Vectors



Most common attack vectors you'll encounter are ...

- Typosquatting
- Malicious Contributors
- Malicious Packages

Typosquatting A

Typosquatting takes advantage of a developer unwillingly installing a package with a slightly different, misspelled name.

In most cases those packages are similar to the original packages but include one or another malware or malicious code.

Typosquatting Examples

```
npm install -
```

Would install the empty package `_` which still has **26.000 Downloads** a week!

In most cases not harmfull as many packages have been disarmed by npm in the past. Some examples:

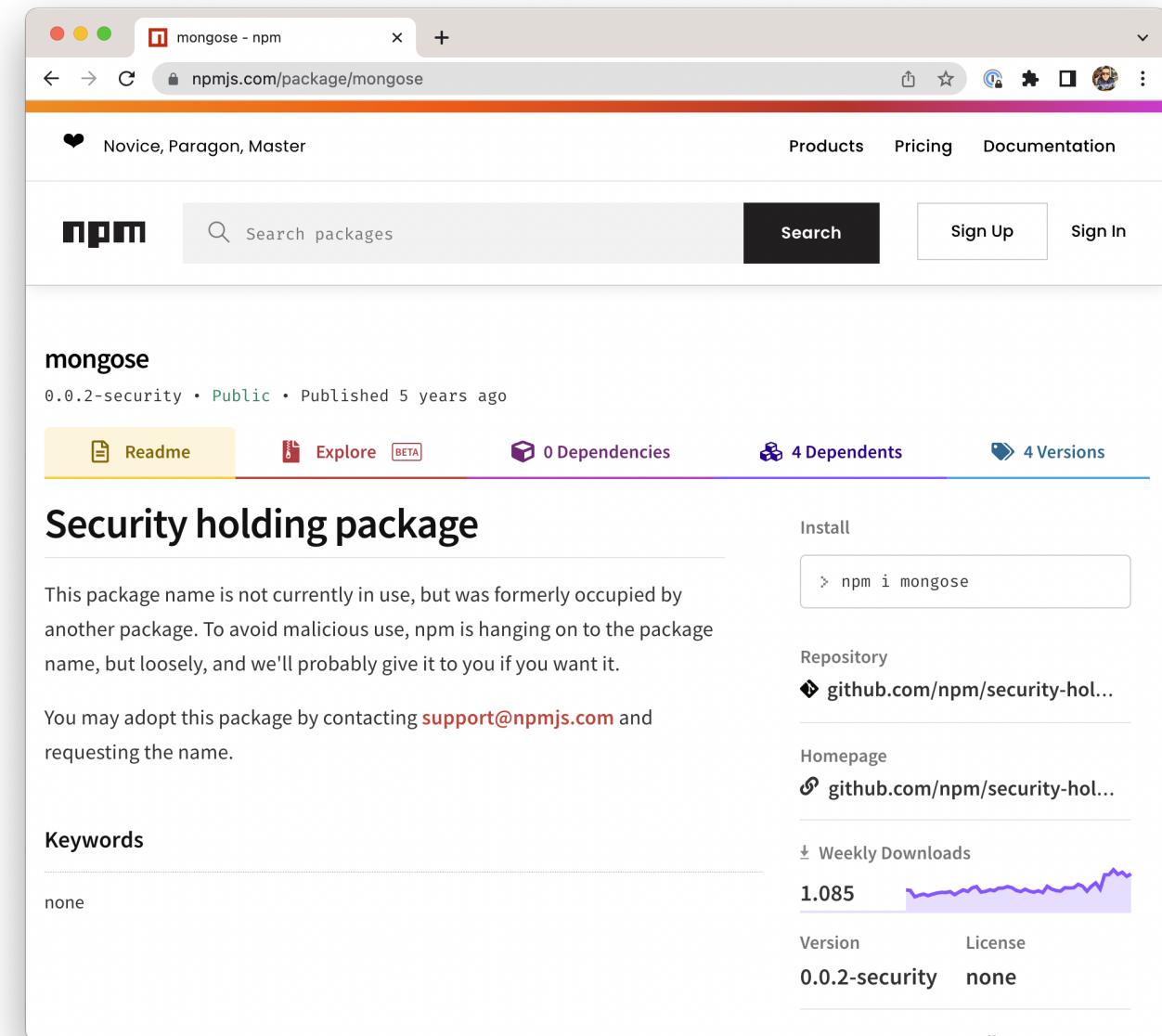
mongoose - mongoose

cross-env - crossenv or crossenv.js

lodash - lodashs

babel-cli - babelcli

... some still have more than 1k downloads a week! (*Though they are empty!*)



Compromised Contributors

Some open source packages have many contributors that have access to publish npm packages. Some of these processes are automated using NPM tokens.

In case those accounts are hacked or tokens are leaked this opens up the possibility to manipulate the packages.

- **Protect your NPM Account with 2FA**
- **Protect your NPM Token in CI secrets**



```
brandon@penguin:~  
brandon@penguin:~$ live-server  
LIBERTY LIBERTY LIBERTY  
LIBERTY LIBERTY LIBERTY  
LIBERTY LIBERTY LIBERTY  
  
H|S|H|S|H  
H| * * * * | H|H|H|H|H  
H|H|H|H|H  
H|H|H|H|H  
H|H|H|H|H
```

Malicious Contributors



A package contributor decides willingly to break a package or inject malicious code.

Example: marak colors and faker package 2022 (full story: [1](#), [2](#))

Malicious Package? 🤔

A Malicious package can use at least two attack vectors injecting and executing code on behalf of the user or system.

Abused Lifecycle Hooks



npm lifecycle hooks (preinstall, postinstall) containing shell scripts or any kind of executable code.

- run in background (use: `--foreground-scripts`)
- executed with the permission of the user that ran `npm`

Everything is possible: Leak environment variables, configuration files, install malware, trojans, manipulate & delete files.

Malicious Source File(s) 😠

Every file that is included in a project can contain malicious parts!

Those parts are not easy to get as the code is in most cases obfuscation, encoded and mangled.

Again – this opens abilities to leak environment variables, configuration files, install malware, trojans, manipulate & delete files.

Demo: Malicious Package



Install a package containing a `preinstall` shell script to show the current username in a OSX notification.

see [examples/node-package-security-risks directory](#)

Countermeasures



We can do something! 💪

- use `npm ci` (even on dev machines!)
- do not install packages for everything
- use packages with a certain age
- never edit `package.json` versions by hand



pin versions

Use exact version (no semver-ranges) to make sure no undesired updated version is installed.

```
npm install --save-exact <package-name>
```

or make it the default behavior

```
npm config set save-exact true
```

not recommended for library type of modules cause of package dublication

ignore scripts



```
npm ci --ignore-scripts  
npm link --ignore-scripts <path>  
npm install --ignore-scripts <package>  
npm update --ignore-scripts <package>
```

or disabling scripts once and for all:

```
npm config set ignore-scripts true
```

Check if your project uses scripts at all with [can-i-ignore-scripts](#)

use npq

- runs several "marshals" before installing
- but still not 100% secure

Try it:

```
npx npq install cross-env.js
```

report packages



Use the Report Malware
Button

cross-env - npm

npmjs.com/package/cross-env

7.0.3 MIT

Unpacked Size Total Files
29.1 kB 10

Issues Pull Requests
1 0

Last publish a year ago

Collaborators

>-Try on RunKit

Report malware

NODE_ENV=production like that. (The exception is **Bash on Windows**, which uses native Bash.) Similarly, there's a difference in how windows and POSIX commands utilize environment variables. With POSIX, you use: \$ENV_VAR and on windows you use %ENV_VAR% .

This solution

cross-env makes it so you can have a single command without worrying about setting or using the environment variable properly for the platform. Just set it like you would if it's running on a POSIX system, and cross-env will take care of setting it properly.

- Installation
- Usage
- cross-env vs cross-env-shell
- Windows Issues
- Inspiration
- Other Solutions
- Contributors
- LICENSE

Installation

This module is distributed via **npm** which is bundled with **node** and should be installed as one of your project's devDependencies :

```
npm install --save-dev cross-env
```

Use package rating websites

Check packages on websites that collect additional information about packages like popularity, ratings, stars, update-frequency, downloads, number of issues and other kpis and metadata:

- <https://snyk.io/advisor/npm-package>
- <https://openbase.com>
- <https://www.npmtrends.com>

More to Read



Most important thing is to be aware of security-related topics and not ignoring them.

- General Security
 - What you need to know when installing packages, Account Takeover
- Typosquatting
 - Typosquatting Explained, Attack undetected for 2 weeks
- Malicious Modules
 - 25 compromisde packages, discord packages, cryptocurrency miner, Azure Developer Attack (jfrog), Malware Civil War (jfrog)



Thanks for
listening! 

Please give feedback & ask
questions!