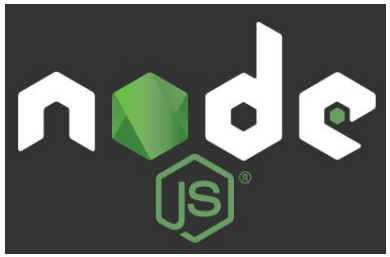
IEEE SecDev 2022

IEEE COMPUTER SOCIETY

How Do Developers Follow Security-Relevant Best Practices When Using NPM Packages?

Md Mahir Asef Kabir, Ying Wang, Danfeng(Daphne) Yao, Na Meng

Background



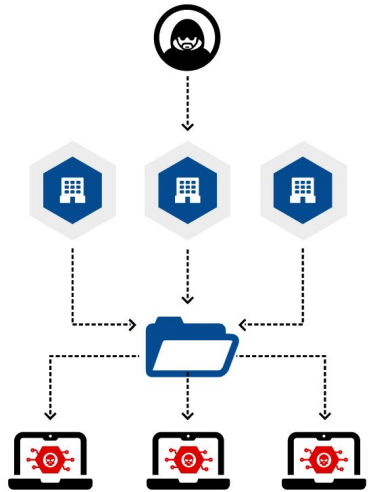
Popular Cross-Platform
JS Runtime Platform

Default Package
Manager of Node.js

Developers Publish & Reuse NPM packages to increase productivity and improve Software Quality

Technical Issue

Usage of NPM packages can expose client applications to security risks. E.g., security experts found -



- Vulnerable packages can cause attacks such as - Software Supply Chain attack [1]
- Vulnerability in library “netmask”, can cause attacks such as - Malware Delivery [2]
  - Can affect more than 278,000 applications

Recommended Mitigation

Follow Best Practices (BP) recommended by domain experts

- BP1: Scan vulnerabilities using “**npm audit**” and remove vulnerabilities with “**npm audit fix**”
- BP2: Scan and/or remove unused and duplicated packages using “**depcheck**” and “**npm dedupe**”
- BP3: Enforce the lock file **package-lock.json** to pin library dependency versions

BP1: Scan & Remove Vulnerable Dependencies

- Projects can contain malicious & vulnerable dependencies that can expose the applications to threats

- “**npm audit**” can scan & show the report of known vulnerabilities

```
$ npm audit
```

- “**npm audit fix**” can install compatible versions of the reported known vulnerabilities

```
$ npm audit fix
```

BP2: Scan & Remove Unused/Duplicate Dependencies

Unused/Duplicate dependencies can cause **attack surface to grow**

```
$ npx depcheck
```

```
Unused dependencies
* @fortawesome/fontawesome-free-webfonts
* @progress/kendo-angular-common
* @progress/kendo-angular-dropdowns
* @progress/kendo-angular-l10n
* @progress/kendo-angular-popup
* @types/googlemaps
* font-awesome
* rxjs-compat
* save
* tslib
```

```
>npm ls --all
```

```
a
+-- b <-- depends on c@1.0.x
|   `-- c@1.0.3
`-- d <-- depends on c@~1.0.9
    `-- c@1.0.10
```

```
npm dedupe
```

```
a
+-- b
+-- d
`-- c@1.0.10
```

BP3: Pin dependency versions using package-lock.json

Using dependency version range can cause 2 problems -

- Non-deterministic package downloads
- Pulling in vulnerable or malicious version from the range

package.json

```
"dependencies": {
  "dependency1":
    "^1.4.0"
}
```

package-lock.json

```
"dependencies": {
  "dependency1": {
    "version": 1.4.0,
    …
  }
}
```

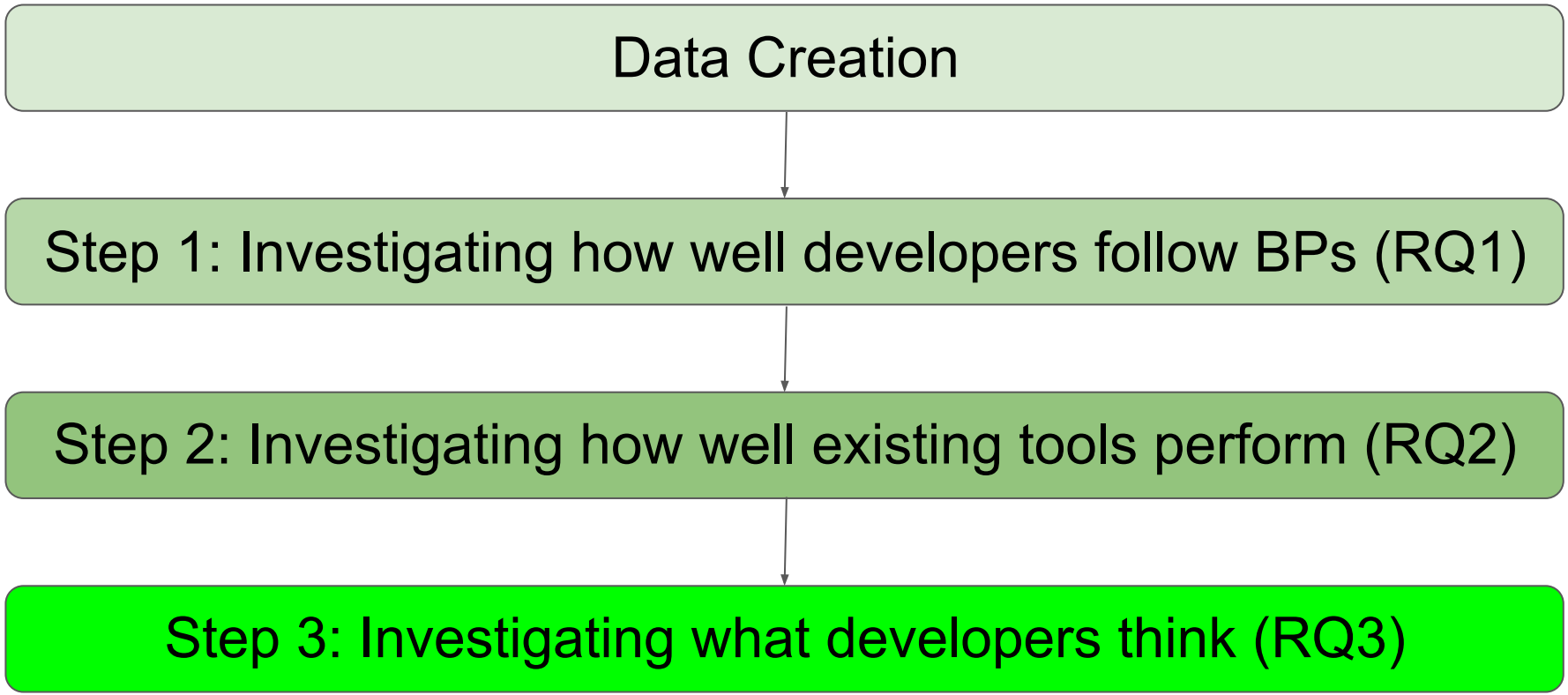
Our Research - Empirical Study

- RQ1: How well did developers follow best practices?
- RQ2: How well can existing tools address developers' violations of best practices?
- RQ3: In the scenarios when developers do not follow best practices, what are the reasons?

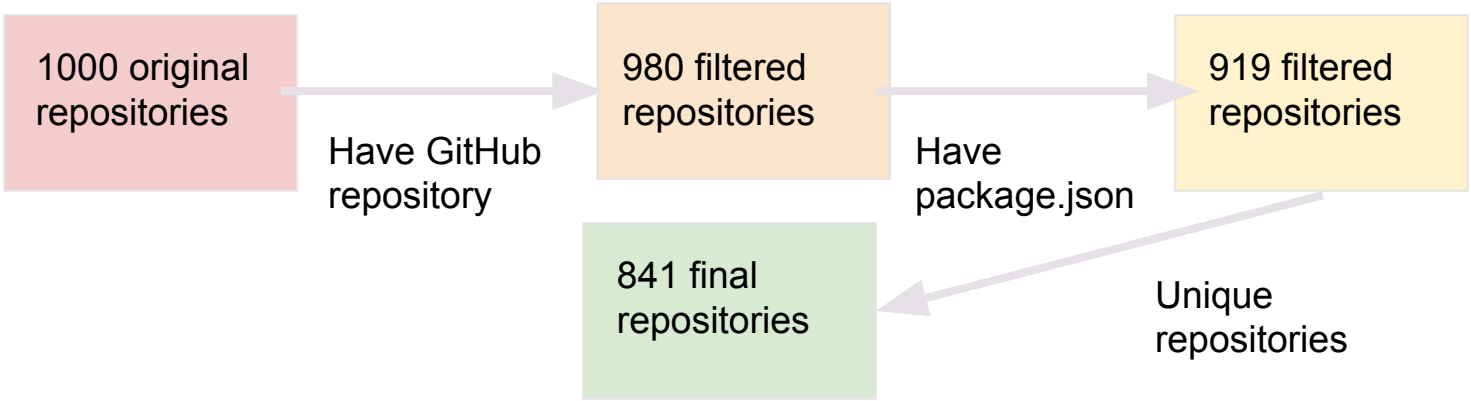
Outline

- Background
- Technical Issue
- Recommended Mitigation
- Our Research
- Methodology
- Experiment Results
- Our Recommendations
- Related Work
- Conclusion

Methodology



Data Creation



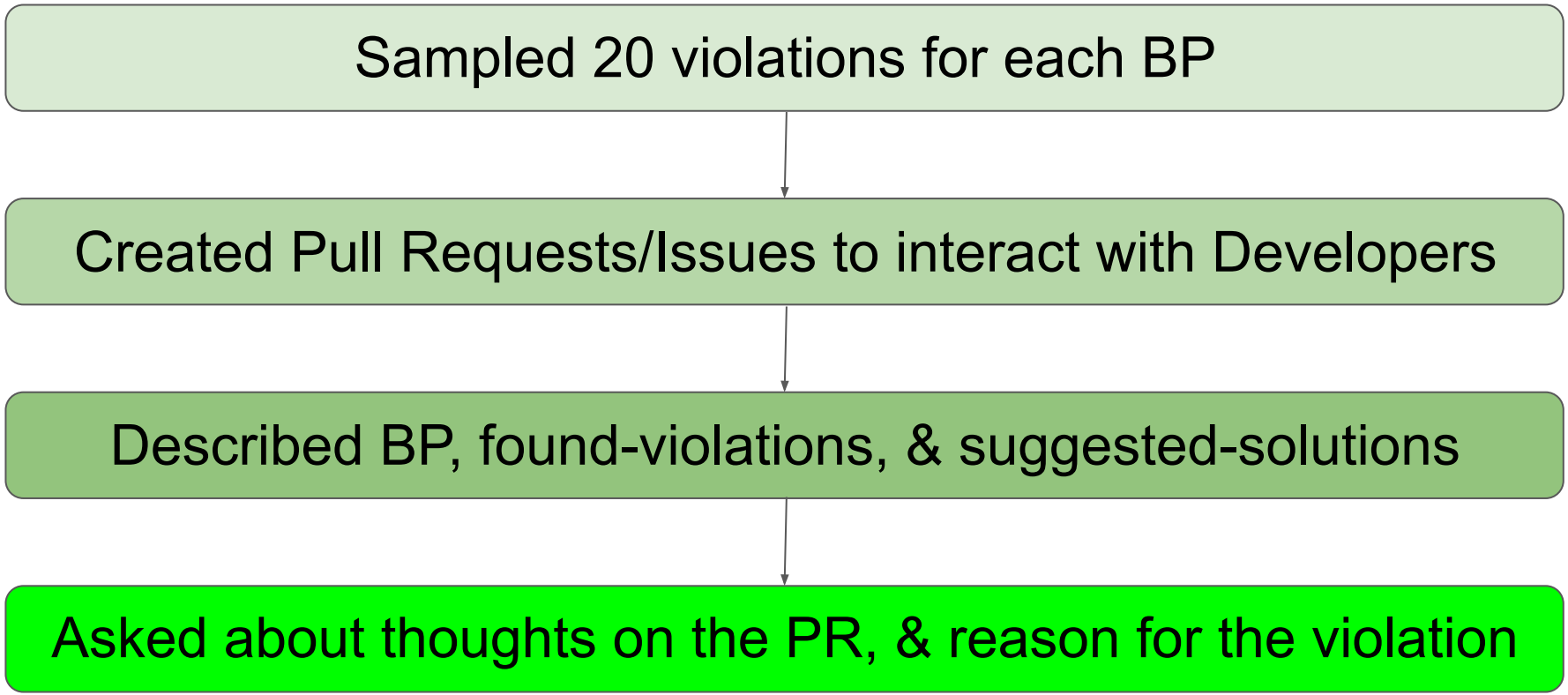
Step 1: Investigating how well developers follow BPs (RQ1)

BP1: Vulnerable Dependencies	`$ npm audit`
BP2: Unused /Duplicates	`$ npx depcheck` `>npm ls --all`
BP3: Lock Files	• Checked version spec in package.json • Checked existence of package-lock.json

Step 2: Investigating how well existing tools perform (RQ2)

BP1: Vulnerable Dependencies	`$ npm audit fix` `$ npm audit`
BP2: Duplicates	`npm dedupe` `>npm ls --all`
BP3: Lock Files	• Lock file is automatically generated when “npm” modifies dependency tree

Step 3: Investigating what developers think (RQ3)



Experiment Result: How well developers follow BPs (RQ1)

460 out of 841 programs had vulnerabilities reported

755 out of 841 programs had unused dependencies

698 out of 841 programs had duplicate dependencies

548 out of 841 programs had not pinned library versions

Most developers did not seem to follow the BPs

Experiment Result: How well existing tools work (RQ2)

"npm audit fix" removed all vulnerabilities in 55 programs

"npm dedupe" removed all duplicates in 10 & partially removed in 467 programs

Existing tools are not sufficient enough to maintain BPs

Experiment Result: What developers think (RQ3)

Some felt npm-audit to be broken for having false positives

Most considered reported unused dependencies to be false positives

Most did not worry about duplicate dependencies

Most did not care about reproducible builds

Most developers are not convinced with the BPs

User Study

Received **22** responses

For **4** projects developers are partially positive

Category	Feedback	# of PRs
Audit	Developers considered the vulnerabilities to be false positives.	4
	Developers accepted the suggested fix	3
Unused	Developers believed the reports to be false positives.	4
	Developers partially agreed, and would like to remove some reported unused packages.	1
	Developers did not worry about unused dependencies.	2
Duplicated	Developers explained the necessity to have duplicated dependencies.	1
	Developers agreed that duplicated dependencies should be removed, but they did not trust npm-dedupe.	1
	Developers assumed the package manager can reduce duplication by default.	1
	Developers did not worry about duplicated dependencies.	1
Lock	Developers did not see the need to pin dependency versions.	4

User Study - False Positives

"npm audit" does not provide exploit. Developers ignored dev/test dependencies

Detected package was used. Can cause no run-time issue

Developers ignored test dependencies. Expected "npm i" to be enough

Developers expected consumers to use lock-file instead of them

Our Recommendations

- For Developers: Generate and commit lock files to avoid hard-to-reproduce bugs
- For Tool Builders: Improve existing tools
- For Researchers: Cautiously use the existing tools in research, as they might not be accurate

Related Work

- Wittern et al. found out that package dependencies increases over time, even for the same core set of packages [4]
- Cogo et al. explored why developers downgrade package dependencies [5]
- Decan et al. [6] and Zerouali et al. [7] studied how soon developers updated their dependencies after the new package releases became available

Conclusion

- This study assesses how developers follow security related best practices when using NPM packages
- Developers recommend to use certain tools to scan or remove vulnerable, unused, & duplicate dependencies, and to add lock files
- The current tools seldom fix all violations, and developers rarely treat tool outputs seriously
- In future, we need to define BPs better, and build better tools

References

[1] “Software supply chain attacks – everything you need to know,” https://portswigger.net/daily-swig/software-supply-chain-attacks-everything-you-need-to-know, 2021.

[2] “Vulnerability in ’netmask’ npm Package Affects 280,000 Projects,” https://www.securityweek.com/vulnerability-netmask-npm-package-affects-280000-projects, 2021.

[3] “npm rank,” https://gist.github.com/anvaka/8e8fa57c7ee1350e3491, 2020.

[4] E. Wittern, P. Suter, and S. Rajagopalan, “A look at the dynamics of the javascript package ecosystem,” in 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, 2016, pp. 351–361.

References (Contd.)

[5] F. R. Cogo, G. A. Oliva, and A. E. Hassan, “An empirical study of dependency downgrades in the npm ecosystem,” IEEE Transactions on Software Engineering, pp. 1–1, 2019.

[6] A. Decan, T. Mens, and E. Constantinou, “On the evolution of technical lag in the npm package dependency network,” in 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Sep. 2018, pp. 404–414.

[7] A. Zerouali, E. Constantinou, T. Mens, G. Robles, and J. Gonz´alez- Barahona, “An empirical analysis of technical lag in npm package dependencies,” in New Opportunities for Software Reuse, R. Capilla, B. Gallina, and C. Cetina, Eds. Cham: Springer International Publishing, 2018, pp. 95–110.

[image-3] https://www.keepersecurity.com/threats/supply-chain-attack.html