On Dependency Analysis of NPM

Data Collection & Case Study

Cui Zihan

I2EC, ICS, NJU

November 10, 2020

Sections

1. Review
2. Data Collection
3. Case Study
4. Summary & Future Work

Sections

1 Review

2 Data Collection

3 Case Study

4 Summary & Future Work

Vulnerability Propagation Problem

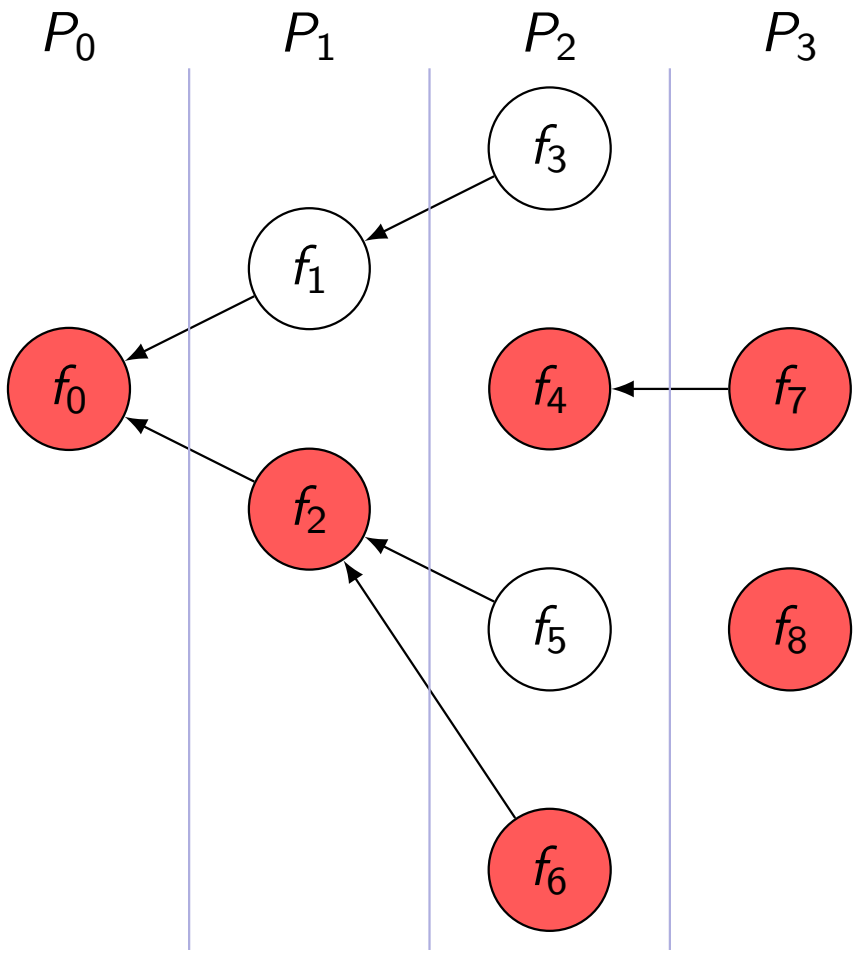


Figure: Graphical description of a vulnerability propagation problem.

Review Problem Formulation

Problem Formulation

Suppose we have $G =< V, E >$ and $V = \bigcup_{i=0}^{m} P_i$. Where P_0 denotes the entry package of the analysed project. Denote $D = \{f_{k1}, f_{k2} \cdots f_{kn}\}$ where f_{ki} refers to a vulnerable function.
Find if "$\exists f_i \in P_0, f_j \in D, f_i$ *is reachable from* f_j. "holds.

Review Research Questions

RQ 1

How to judge if $< f_i, f_j >$ exists?

RQ 2

How to tell if $f_i \in D$?

- Resolve functions and construct the function call graph from packages;
- Judge if a function is vulnerable; To solve this problem, we must collect data related to the vulnerabilities.

RQ 3

How to construct the function call graph within reasonable time?

1 Review

2 Data Collection

3 Case Study

4 Summary & Future Work

Data Source

We researched the databases of vulnerability records in NPM ecosystem. We find 3 relatively comprehensive data sources, and they have their own advantages and disadvantages:

- CVE(Common Vulnerabilities and Exposures): Supervised by MITRE Corporation; Highly recognized; Lack of a detailed description of the vulnerability; Open source;
- Snyk.io: Provide the most detailed description; Maintained by Snyk, one of the leading companies involved in analysing software vulnerabilities; Commercial;
- NPM security advisory: Maintained by NPM; Open source;

Data Source

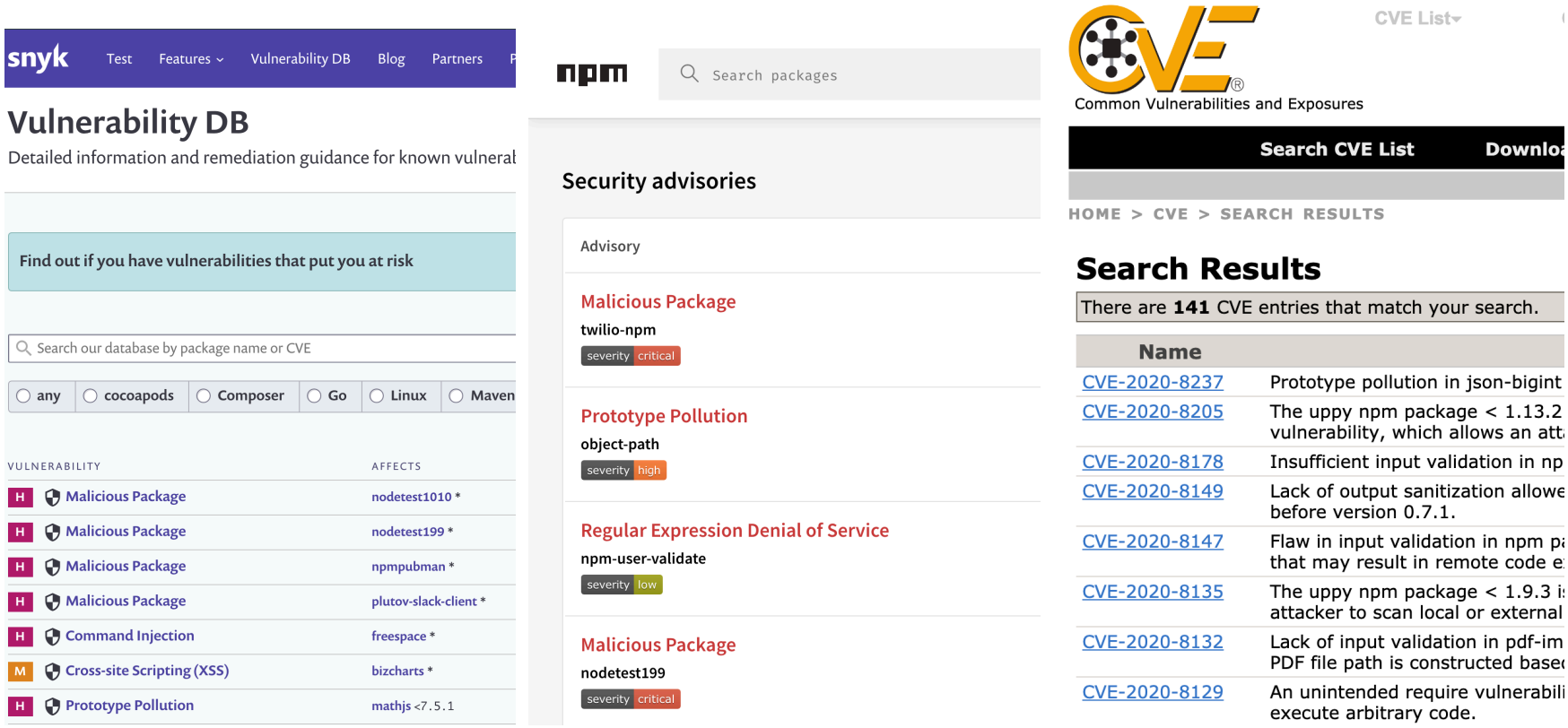


Figure: Snyk.io, npm advisory and CVE.

Data Source

NPM security advisory collects more than 1400 vulnerability reports since 2015. And each report provides information about the affected versions, vulnerability overview, vulnerability detail, remediation, severity and the origin source.

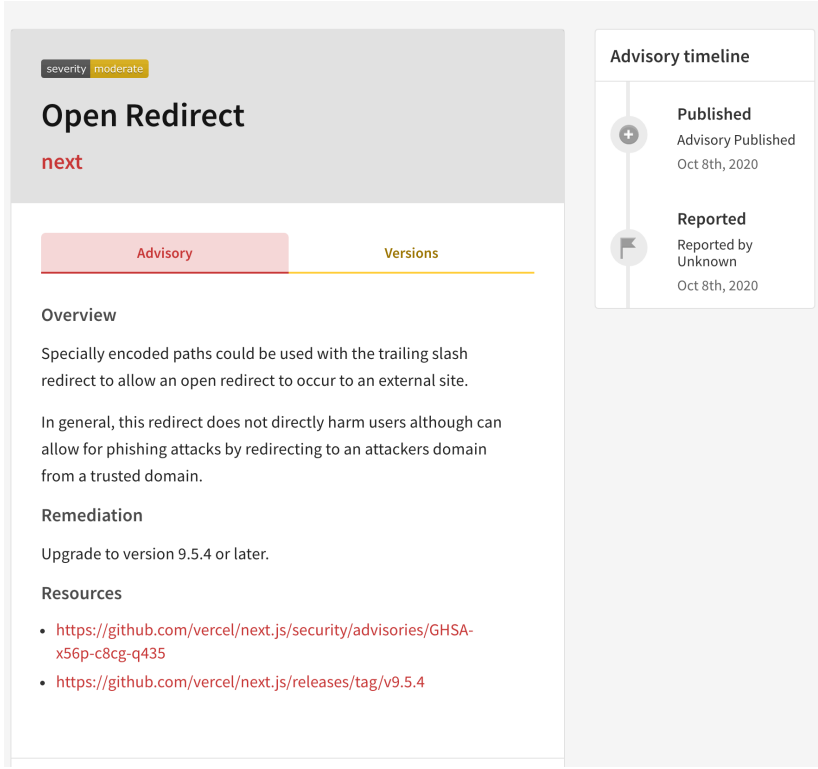


Figure: A vulnerability report example.

Data Collection

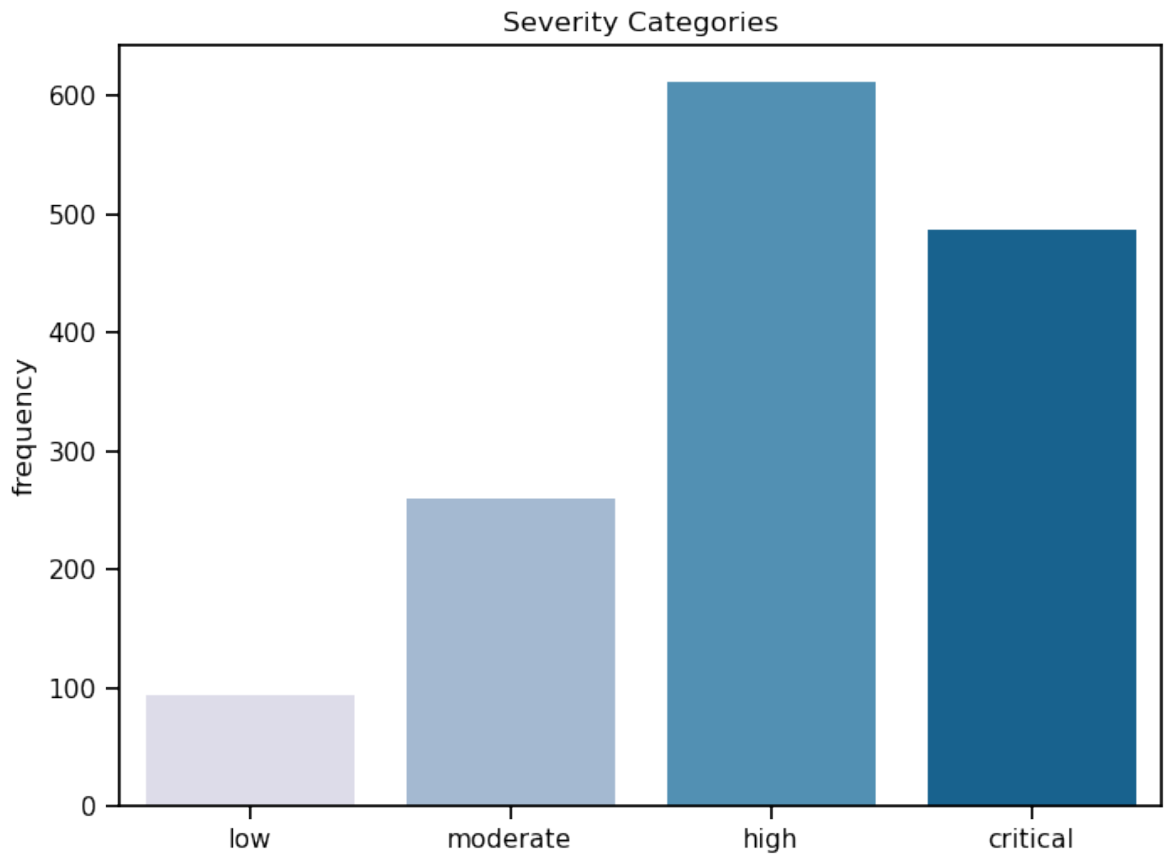
From the npm security advisory website, we collect the vulnerability reports. The output is formatted data for easy use.

title	module_name	vulnerable_versions
Prototype Pollution	object-path	< 0.11.5
Regular Expression Denial of Service	npm-user-validate	< 1.0.1
Malicious Package	nodetest199	>=0.0.0
Malicious Package	nodetest1010	>=0.0.0
Malicious Package	plutov-slack-client	>=0.0.0
Malicious Package	npmpubman	>=0.0.0
Sensitive data exposure in NATS	nats	>= 2.0.0-201 <= 2.0.(
Open Redirect	next	>=9.5.0 <9.5.4
File restriction bypass in socket.io-file	socket.io-file	>=0.0.0
Malicious Package	loadyaml	>=0.0.0
Malicious Package	electorn	>=0.0.0
Prototype Pollution in node-forge	node-forge	< 0.10.0
Universal XSS in Android WebView	react-native-webview	>= 0.0.0
Malicious Package	nagibabel	>=0.0.0
Sensitive Data Exposure	renovate	>=19.180.0 <23.25.1
Authorization Bypass	lemonldap-ng-handler	<0.5.2
Denial of Service	node-fetch	< 2.6.1 || >= 3.0.0-b
Remote Memory Exposure	bl	<1.2.3 || >2.0.0 < 2.
Command Injection	bestzip	<2.1.7
Inadequate Encryption Strength	bcrypt	<5.0.0
Malicious Package	fallguys	>= 0.0.0
DOM-based XSS	auth0-lock	<=11.25.1
Regular Expression Denial of Service	url-regex	>=0.0.0

Figure: Data collection output

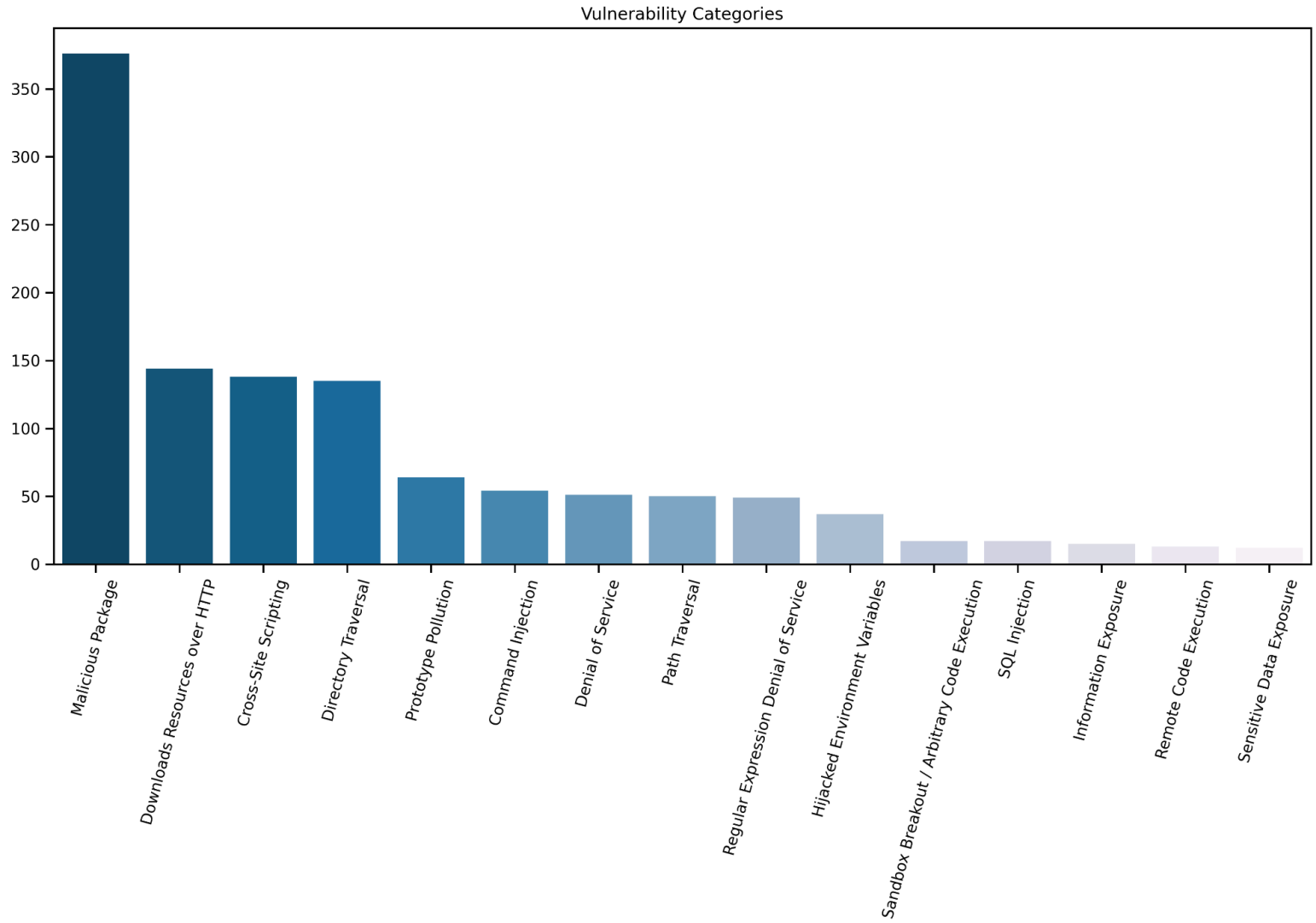
Severity Analysis

Contrary to our expectation, more than 75% vulnerabilities are labeled as high or cirtical severity.



Vulnerabilities Analysis

We have conducted statistics on the root causes of vulnerabilities.



Typical Vulnerability Class

Malicious Package

Malicious packages are packages deliberately uploaded to the npm repository by the attacker and contain malicious code. They are usually disguised as normal packages. Once reported, malicious packages will be removed from the npm repository.

Example:

Typical Vulnerability Class

Code Injectoin

Code injection is the exploitation of a vulnerability that is caused by processing invalid data. Attacker use injection to introduce malicious code into a vulnerable program.
Example: the package "freespace" is a library that tells user how much free disk space in a specific path. However did not check the legitimacy of the path.

```
1 const freespace = require('freespace');
2
3 freespace.check('/ ; touch /tmp/semicolon_file')
4 .then(bytes => {
5         console.log(bytes);
6 });
```

Typical Vulnerability Class

Prototype Pollution

Prototype Pollution is about polluting the prototype of a base object which can sometimes lead to arbitrary code execution.
Example: A frequently used package "lodash" which downloaded more than 30 million times a week was found to be attacked by Prototype Pollution. The function "defaultDeep" could be tricked into adding or modifying properties of "Object.prototype".

```
1 const mergeFn = require('lodash').defaultsDeep;
2 const payload = '{"constructor": {"prototype": {"a0": true
      }}}'
3 function check() {
4     mergeFn({}, JSON.parse(payload));
5     if ({})[`a0`] === true) {
6         console.log(`Vulnerable to Prototype Pollution via
              ${payload}`);
7     }
8   }
```

1 Review

2 Data Collection

3 Case Study

4 Summary & Future Work

Automatically Extract Dependencies

We developed a tool that can be used to extract a package's dependencies automatically.

```
        {
            "name": "ajv",
            "version": "6.6.2",
            "parent": "antd-theme-generator@1.2.8"
        },
        {
            "name": "ansi-styles",
            "version": "3.2.1",
            "parent": "antd-theme-generator@1.2.8"
        },
```

The output is a list that each entry of the list contains the package name, the package version and its dependent.

Vulnerabilities Match

We use this tool to build a tool chain that automatically detects potential vulnerabilities in packages.

```
Find a low severity vulnerability:braces@1.8.5 match the rule braces:<2.3.1, "Regular Expression Denial of Service"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_braces@1.8.5@braces
Find a high severity vulnerability:bl@1.2.1 match the rule bl:<1.2.3 || >2.0.0 < 2.2.1 || >=3.0.0 <3.0.1 || >= 4.0.0 <4.0.3, "Remote Memory
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/bl
Find a high severity vulnerability:cryptiles@2.0.5 match the rule cryptiles:<4.1.2, "Insufficient Entropy"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/cryptiles
Find a low severity vulnerability:debug@2.6.8 match the rule debug:<= 2.6.8 || >= 3.0.0 <= 3.0.1, "Regular Expression Denial of Service"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/debug
Find a moderate severity vulnerability:extend@3.0.1 match the rule extend:<2.0.2 || >=3.0.0 <3.0.2, "Prototype Pollution"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/extend
Find a moderate severity vulnerability:hoek@2.16.3 match the rule hoek:<= 4.2.0 || >= 5.0.0 < 5.0.3, "Prototype Pollution"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/hoek
Find a low severity vulnerability:lodash@4.17.4 match the rule lodash:<4.17.19, "Prototype Pollution"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/lodash
Find a high severity vulnerability:lodash@4.17.4 match the rule lodash:<4.17.12, "Prototype Pollution"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/lodash
Find a high severity vulnerability:lodash@4.17.4 match the rule lodash:<4.17.11, "Prototype Pollution"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/lodash
Find a low severity vulnerability:lodash@4.17.4 match the rule lodash:<4.17.5, "Prototype Pollution"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/lodash
Find a low severity vulnerability:minimist@0.0.8 match the rule minimist:<0.2.1 || >=1.0.0 <1.2.3, "Prototype Pollution"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/minimist
Find a low severity vulnerability:node-fetch@1.7.2 match the rule node-fetch:< 2.6.1 || >= 3.0.0-beta.1 < 3.0.0-beta.9, "Denial of Service"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/node-fetch
Find a high severity vulnerability:sshpk@1.13.1 match the rule sshpk:<1.13.2 || >=1.14.0 <1.14.1, "Regular Expression Denial of Service"
 at path:/Users/cui/WorkSpace/StayInStyle/dirty-package/ant-design-4.6.5/node_modules/_cxs@6.2.0@cxs/benchmarks/node_modules/sshpk
Find a moderate severity vulnerability:stringstream@0.0.5 match the rule stringstream:<=0.0.5, "Out-of-bounds Read"
```

Figure: Partial result of detecting AntDesign package.

Result

	antd	axios	date-fns	umi	electron	express	acorn	pm2
critical	0	1	1	0	0	0	0	0
high	6	5	1	1	0	0	0	0
moderate	4	0	2	0	0	0	0	0
low	8	8	8	3	1	3	0	0
all	18	14	12	3	1	3	0	0

Table: Vulnerabilities in popular javascript packages.

Case: Axios

We found that the critical vulnerability in Axios is caused by "open@0.0.5".

Vulnerability Description

Urls are not properly escaped before concatenating them into the command that is opened using `exec()`.

Steps To Reproduce:

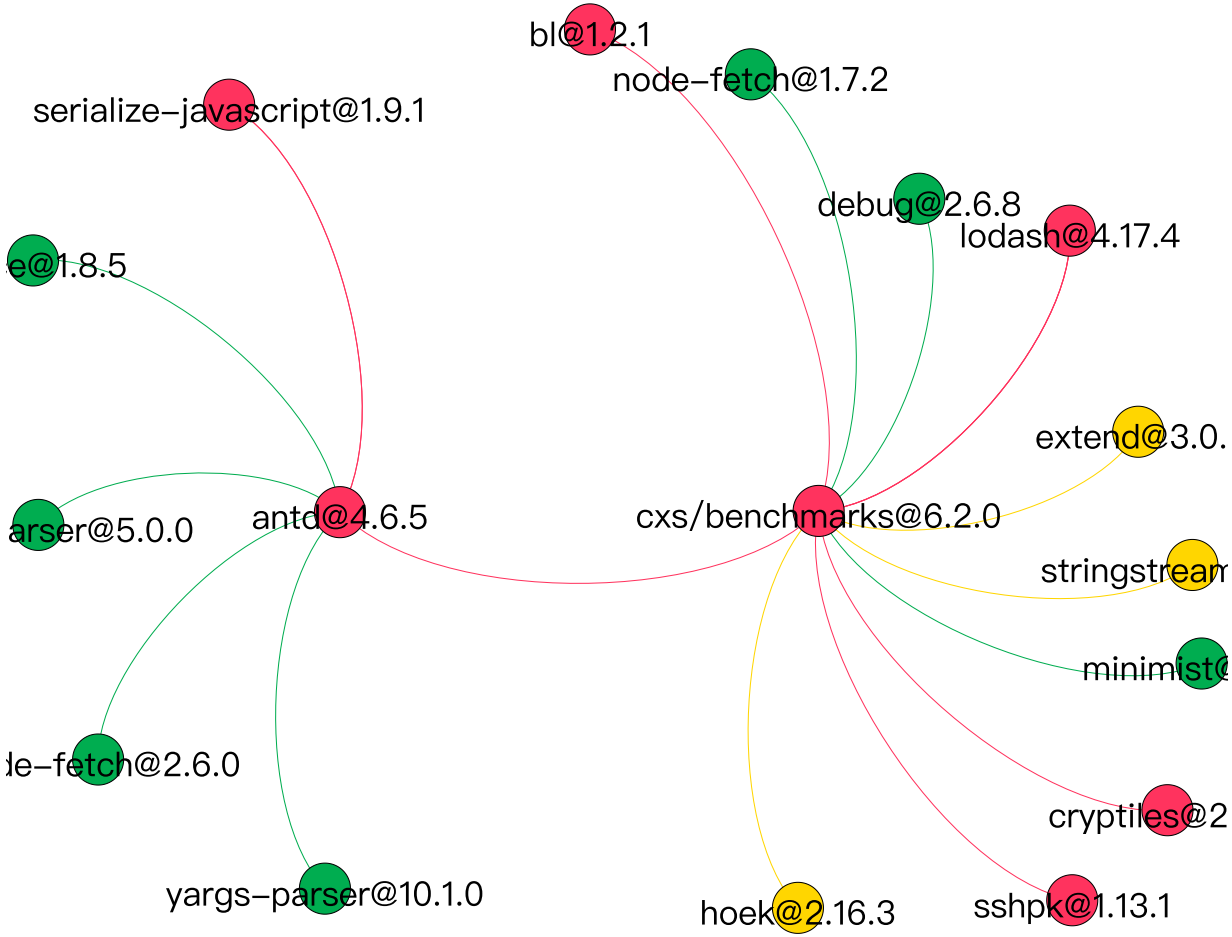
```
require("open")("http://example.com/`touch /tmp/tada`");
```

Observe `/tmp/tada/` file created.

According to Snyk.io's vulnerability report, the older versions(<6.0.0) of "open" are vulnerable. Upgrading "open" to the last verison will prevent this vulnerability but is also likely to have unpredictable effects since it now has a very different API.

Case: AntDesign

AntDesign is a popular front-end framework. It has a complicated dependency graph. We found that if we consider its direct dependencies only, more than half of the vulnerabilities will not be detected.



Study Scope

Threats to Validity

- We mainly focus on Javascript packages that use Node.js runtime.
- We have not analyzed on the function level yet. We mainly want to verify that the vulnerability propagation problem does exist.
- We select popular packages from github for analysis. These packages are mainly frameworks or function libraries. We do not analyze applications.

1 Review

2 Data Collection

3 Case Study

4 Summary & Future Work

Summary

Work Summary

- We built a tool chain that automatically detects vulnerabilities in npm packages.
- We analyzed some popular npm packages.

Preliminary Conclusion

We found high-risk vulnerabilites in some of the popular packages we detected which is concrete evidence to support the existence of this problem.

Future Work

How to get down to the function level?

A Naive Idea Using Approximate Analysis:
Because the lack of basic functions in JavaScript's standard library, there are numerous micro-packages with very few lines of code in npm repository[1]. These small-scale packages contain few functions. Thus if we find a package is vulnerable, we simply mark all of its functions as vulnerable.

- Good news: no false-negative;
- Bad news: not precise enough;

1

[1] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, & Michael Pradel (2019). Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In 28th USENIX Security Symposium (USENIX Security 19) (pp. 995–1010). USENIX Association.

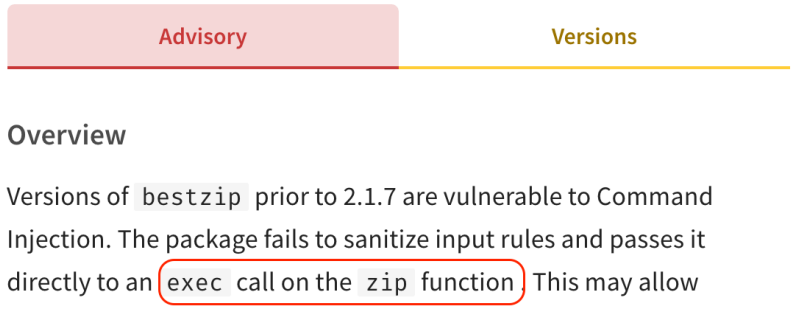
Future Work

How to get down to the function level?

Mining Text Information:

In the process of data collection, we find that some of the vulnerabilites reports mentioned the specifics vulnerable functions at the vulnerability overview.

- Good news: According to our research, quite a few vulnerabilities report have related overview pointing out the location of the vulnerability.
- Bad news: How to acquire the information we need from raw text?



Future Work

How to get down to the function level?

Comparing Different Versions:
Since we already know the vulnerable versions, we can compare the last vulnerable verison package and the first patched version package and the unchanged functions is clear.
Bad news:

- Can not rule out some performance optimization changes.
- What if there is no patched version?

Thanks. Q&A