Analyzing Downloads of NPM Packages*

Samuel Steinberg, Jeffrey Dong and Bryan Pace

I. Research Question

Is there an efficient way to gather data on suspicious NPM packages and can we identify them in an orderly fashion. Additionally, can we develop or find systematic methods to find where possibly bad data is hidden in these packages.

II. INTRODUCTION AND MOTIVATION

Node Package Manager or NPM is a package manager for JavaScript, hosting 600,000+ projects. Like any other software ecosystem, NPM packages have a number of dependencies. When a package is installed, all its dependencies are installed as well, unless already cached. This indicates a causal relationship between addition/ deletion of dependencies and downloads for a package. This relationship can be potentially exploited by some package by adding itself as a dependency to a popular package which results in an in increase in the number of downloads for the first package. This can also be exploited by some malicious package that wants to sneak in a vulnerability into a popular package. Therefore wed like to investigate the unusual increases in the number of downloads of different packages.

III. PROCEDURE FOR DISCOVERY

A. PART ONE: MAIN PROJECT

Look up the package in NPM website https://www.npmjs.com/package/package or search in Google. Then check the Graph for weekly downloads and verify the sudden increase of downloads and the shape. Note: the candidate packages should be significantly more popular so that adding them as dependents can cause the rise in the number of downloads of the first package. Next, checking the version history and identifying the actual package that caused the rise (and fall in case of a peak).

B. ADITTIONAL TASKS: AUTOMATION AND MODELING

Propose an automated way for identifying the package that caused the rise. Next, propose a modeling strategy that would be useful for identifying the package from a list of candidate packages

IV. PEAKS AND STEPS

If the shape is like a peak, the dependent that caused the rise has likely moved on (although there could be other causes as mentioned before). So, look at the output of the follower script to find all the dependents (i.e. all packages that ever had the first package as a dependency). The likely suspects are the ones that once had it as a dependency, but not any more. Find the list of these candidate packages and verify the versions that added and removed the first package as a dependency. There could be more complicated scenarios, as illustrated in the htmlnano example. In such cases it is recommended to check the GitHub page of the candidate package and check the commit history to identify the situation. If the shape is like a step, i.e. the rise in the number of downloads wasnt followed by a drop, the dependent was likely never removed. So, it will be easier to find. Just look at the list of dependents of the package, find the popular dependent package(s) as likely candidate(s). Verify when the first package was added as a dependency in each candidate package by looking at the package history from https://replicate.npmjs.com/package. Try to find a match between the time the rise in downloads was seen and the time the first package was added as a dependency to the candidate package. By following this procedure, find the package that caused the increase in the number of downloads for the first package.

*This work was not supported by any organization

A. Abbreviations and Acronyms

NPM is used to describe the Node Package Manager.

B. SPECIAL CASES

Special case: There is a second possibility that some package X added the package under consideration (say package A) as a dependency, and the popularity of X is actually due to it being a dependency of a popular package Y. Now it could be the situation that sometime after X added A as a dependency, Y removed X from its list of dependencies, causing X to lose its popularity, which in turn meant A also lost its popularity, resulting in the peak. It can be easily identified by seeing if a dependent of A lost its popularity around the same time when the number of downloads of A dropped. In such cases, we want the names of packages X and Y with an explanation of the situation.

V. MORE ABOUT NPM PACKAGES IN GENERAL

An NPM is a package manager for Node.js packages or modules. The previously mentioned site www.npmjs.com hosts thousands of free packages. Modules are JavaScript libraries that can be included in a project. Once a package is downloaded and installed, it can be utulized immediately.

A. NPM RISKS

This free, open-source sharing does come with a risk. As Ferenc Hamori[1], Managing Director of Rising Stack puts it, ”Roughly 76/100 of Node shops use vulnerable

packages, some of which are extremely severe; and open source projects regularly grow stale, neglecting to fix security flaws....The more packages you use, the higher the risk of having a vulnerable or malicious package amongst them. This holds true not only for the packages you use directly but also for the indirect dependencies they use." In other words, Node.js packages are extremely useful but there is always a catch. Many times, bad actors will chain dependencies together in long strands, making them very difficult to find.

B. HOW DO PEOPLE BECOME VULNERABLE

Note the strand of packages below. Each — represents a dependency package associated with its link:

yummy—- LICENSE —- README.md —— like-tweet.js —- index.js —- package.json

Here, one could have installed a package that cam with a dependency "yummy", and when the code is checked it looks fine. However, after its "LICENSE" and "README.md" dependencies comes "like-tweet.js". This is what makes identifying NPM vulnerabilities so difficult: They are extremely random. This bad package is sandwhiched in between five perfectly safe packages. Most people do not even know about these dependencies which is why it is so easy to never realize it.

VI. TIMELINE – SUBJECT TO CHANGE

September 26 : Team will have the required preliminary completed and committed to GitHub. Meeting times will also be established, subject to change based off workload, availability, etc.

October 10: Team will be familiarized with the NPM packages website.

October 24: Group members will scour and analyze packages from spreadsheet–Perhaps group members can come together and work to identify strings of packages that could be related.

November 7: By this time, group will have looked for patterns and analyzed code pertaining to the steps and/or peaks, and will continue analyzing code.

November 7th-End: Group will gather data and findings, construct necessary models and bring analysis together. Group will work on and finish final presentation and paper at this time.

VII. MEMBER RESPONSIBILITIES

Group members will be expected to carry their weight and divide work accordingly. Sam, Jeff, and Bryan will be the more hands-on team members dealing with the spreadsheet and attempting to identify potentially harmful patterns or packages. Tapajit will be responsible for guiding the previously said members through the project and be able to assist along the way. Each meeting, a coordinator, checker, and/or recorder role will be assigned to a given group member. Meeting times are to be determined.

VIII. CONCLUSION

The relationship between popular packages and dependencies in an open source world can be beneficial to the user or harmful with malicious intent. This can also be exploited by some package that wants to sneak in a vulnerabilit. Therefore in this project we would like to investigate and potentially exploit the flaws in the number of downloads of different packages. We would also like to POTENTIALLY come up with ideas to combat malicious intent dealing with NPM's. We plan to isolate packages by checking their version history and identifying the actual package that caused the rise (and fall in case of a peak). We will then analyze the code to see if anything nefarious is goingon with the code, or if any code has the potential to put users at risk.

APPENDIX

$https://en.wikipedia.org/wiki/Npm_(software)$
$https://www.w3schools.com/nodejs/nodejs_npm.asp$

ACKNOWLEDGMENT

Tapajit Dey of the University of Tennessee also contributed to this proposal and will have a limited role in this project.

REFERENCES

[1] Ference Hamori, "https://blog.risingstack.com/controlling-node-js-security-risk-npm-dependencies/"