V-Achilles: An Interactive Visualization of Transitive Security Vulnerabilities

Vipawan Jarukitpipat, Klinton Chhun, Wachirayana Wanprasert, Chaiyong Ragkhitwetsagul, Morakot Choetkiertikul, Thanwadee Sunetnanta
SERU, Faculty of ICT, Mahidol University
Salaya, Nakhon Pathom, Thailand

Raula Gaikovina Kula, Bodin Chinthanet, Takashi Ishio, Kenichi Matsumoto
Nara Institute of Science and Technology (NAIST)
Nara, Japan

ABSTRACT

A key threat to the usage of third-party dependencies has been the threat of security vulnerabilities, which risks unwanted access to a user application. As part of an ecosystem of dependencies, users of a library are prone to both the direct and transitive dependencies adopted into their applications. Recent work involves tool supports for vulnerable dependency updates, rarely showing the complexity of the transitive updates. In this paper, we introduce our solution to support vulnerability updating in npm. `V-Achilles` is a prototype that shows a visualization (i.e., using dependency graphs) affected by vulnerability attacks. In addition to the tool overview, we highlight three use cases to demonstrate the usefulness and application of our prototype with real-world npm packages. The prototype is available at https://github.com/MUICT-SERU/V-Achilles, with an accompanying video demonstration at https://www.youtube.com/watch?v=tspiZfhMNcs.

KEYWORDS

software libraries, fixing known vulnerabilities

ACM Reference Format:
Vipawan Jarukitpipat, Klinton Chhun, Wachirayana Wanprasert, Chaiyong Ragkhitwetsagul, Morakot Choetkiertikul, Thanwadee Sunetnanta and Raula Gaikovina Kula, Bodin Chinthanet, Takashi Ishio, Kenichi Matsumoto. 2022. V-Achilles: An Interactive Visualization of Transitive Security Vulnerabilities. In *The 37th IEEE/ACM International Conference on Automated Software Engineering*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Third-party dependencies in software applications are now prevalent due to the rise of available software library ecosystems like the Node.js packages (npm). The npm ecosystem hosts over 1.6 million library packages and is relied upon by more than 11 million developers worldwide[1]. Furthermore, the importance of npm packages is evident in the industry, by its purchase by Microsoft's GitHub in 2020[2]. A key threat to the usage of third-party dependencies has been security vulnerabilities, which risk unwanted access to a user application. Due to the transitive nature of the ecosystem of dependencies, applications are prone to any threats not directly adopted [2, 4, 7].

There have been tools to increase the developers' awareness of vulnerable dependencies and their updates. Two tools that have been widely used and the state of the art are Dependabot and `npm audit`. Nonetheless, even with these tools, developers still face challenges when making decisions to update their dependencies. Previous studies find that developers are slow in updating their dependencies [1, 5, 6, 9, 10]. This is due to several factors including compatibility issues, being unaware, and the migration effort outweighing the benefits. Other work [8] similarly finds that most developers were unaware of dependency updates and that the migration effort is a barrier to adopting a dependency update. Although these tools have been successfully adopted in practice, the key problem is understanding the effect of the vulnerability, especially if the vulnerability is detected in a transitive dependency [13].

Hence, in this paper, we introduce our tool that not only considers the direct dependency, which is a library that the application calls, but also considers the *transitive dependencies*. A thread of the direct and transitive dependencies that depend on each other can be called a *chain of dependencies*. For example, in an npm project `P`, the project has a direct dependency called `karma-mocha`. This direct dependency depends on another dependency, i.e., transitive dependency, called `minimist`. This creates a chain of dependencies of `P` → `karma-mocha` → `minimist`. Prior work [14] shows that there are approximately 80 transitive dependencies per each direct dependency installed, and this ratio is increasing over time. These transitive dependencies and their invisibility from the developer's point of view can result in difficulty in locating flaws or vulnerabilities [3, 11].

2 EXISTING TOOL SUPPORT

Nowadays, there are automated tools that can help developers analyze dependency security vulnerabilities in their software projects. The two widely-used tools include GitHub Dependabot and `npm audit`. We explain each of them below.

Dependabot[3] is a bot that analyzes a GitHub repository and automatically creates pull requests to update outdated and vulnerable

[1]https://www.npmjs.com

[2]https://github.blog/2020-03-16-npm-is-joining-github/

[3]https://github.com/dependabot

ASE '22, 10–14 October 2022, Ann Arbor, Michigan, United States
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

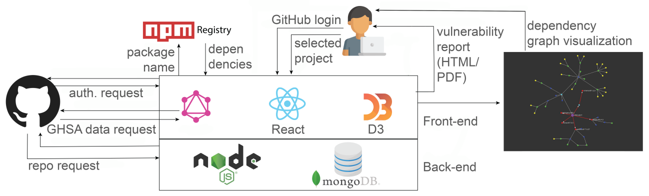


Figure 1: System Architecture of V-Achilles

dependencies. It is acquired by GitHub in 2019. Dependabot currently supports repositories written in 15 programming languages. The process of Dependabot consists of three steps: (i) the bot finds outdated or vulnerable dependencies, (ii) the bot creates a pull request for each dependency, and (iii) the developers check the proposed changes and decide if they want to merge them to the repository.

The second tool is *npm audit.*[4] In 2018, npm introduced a new command for assessing the vulnerability from npm package dependencies called `npm audit`. The command submits the list of dependencies from `package.json` file (i.e., package metadata file) to the npm registry for a vulnerability report. By default, Node.js developers receive the vulnerability report by `npm audit` while installing the dependencies from npm. Developers can also execute the `npm audit` command manually to generate the vulnerability report at any time. `npm audit` can report software vulnerabilities in both direct and transitive dependencies.

Although Dependabot and `npm audit` give an analysis of dependencies and their vulnerabilities, they still have some limitations. Dependabot can detect only vulnerabilities in direct dependencies.[5] For `npm audit`, the tool can analyze transitive dependencies and their vulnerabilities. However, the tool is only available as a command line tool with the vulnerability report in a textual and tabular format. Thus, the developers may not see the relationships of the dependencies and the severity of the vulnerabilities when making their decisions to update the vulnerable packages. **In this paper, we aim to fill this gap by introducing an interactive graph that displays the information needed by developers.**

3 THE V-ACHILLES TOOL

In this section, the overview, interactive features, and the output of V-Achilles.

3.1 Overview and Design

As shown in Figure 1, we create a web-based tool called "V-Achilles"[6] for demonstrating the concept of node-link dependency graph visualization. V-Achilles connects to GitHub and retrieves the user's npm repositories. It then analyzes a chosen repository's dependencies and produces the dependency graph visualization and a security analysis report (as shown in Figure 2 and Figure 3). The tool consists of the front-end and the back-end parts. The front-end part consists of GraphQL, React, and D3.js libraries. The back-end parts consist of Node.js and MongoDB Atlas database. Lastly, V-Achilles queries the vulnerability information of the direct and transitive dependencies from GitHub Advisory Database[7] using GraphQL and creates a dependency graph visualization and report.

[4]https://docs.npmjs.com/cli/v8/commands/npm-audit
[5]https://github.com/Dependabot/Dependabot-core/issues/2640
[6]The name V-Achilles has been chosen as a metaphor for Achilles' heel, which represents a weakness in a software project. V stands for both visualization and vulnerability.

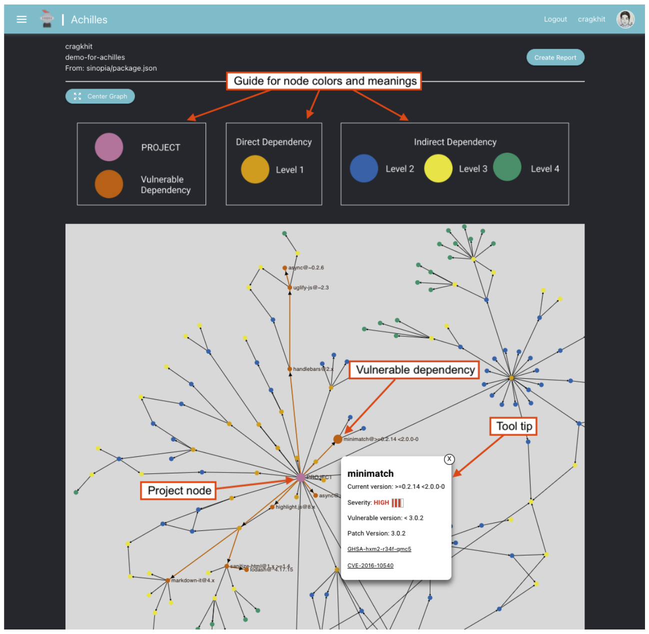


Figure 2: V-Achilles analysis result with dependency graph visualization and a tool tip that shows the dependency's vulnerability information

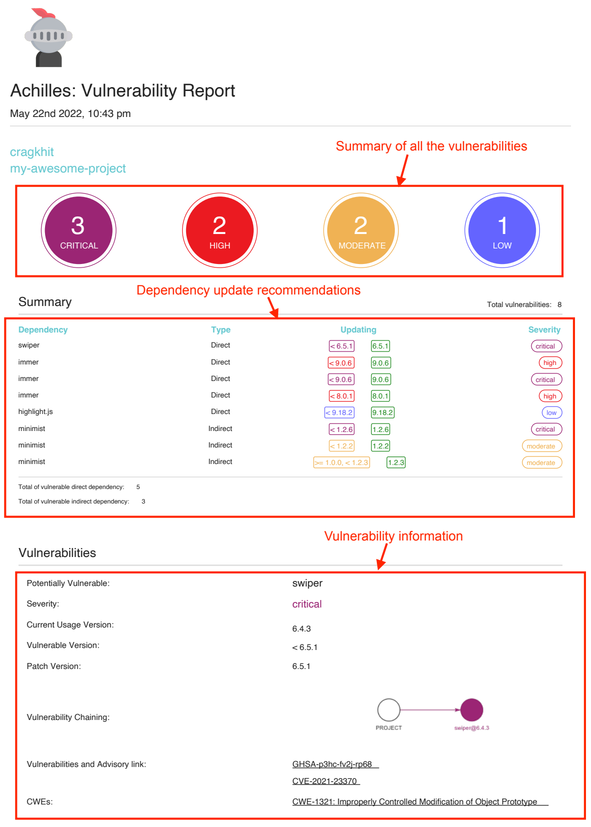
3.2 Interactive Visual Features

V-Achilles also makes use of the interactions of the D3.js visualization library to display an interactive graph that can be zoomed in and out and also navigated around. Moreover, the dependency's vulnerability information is presented as an interactive tooltip. The tooltip information is only displayed when a user hovers the cursor over a node in the graph. This way, V-Achilles only shows relevant security information that the user is interested in.

Figure 2 shows the dependency graph visualization and the design decisions for the tooltip. If the hovered node is vulnerable, V-Achilles shows a tooltip with the vulnerability information including the dependency version, its vulnerability severity (low, medium, high, critical), the range of the versions that are affected by the vulnerability, the patch version of this vulnerability, and the links to the vulnerability's CVE. The user can use this information for their further investigation of the vulnerability and support their decision on making updates. For normal nodes, the tooltip only shows the version of the dependency.

In terms of visual design, the status of each dependency, i.e., node in the graph, is highlighted by a color-blind safe [12] palette. As depicted in Figure 2, the project node is shown in reddish purple, and is slightly bigger than other nodes. The vulnerable nodes are

[7]https://github.com/advisories



Achilles: Vulnerability Report

May 22nd 2022, 10:43 pm

cragkhit
my-awesome-project

Summary of all the vulnerabilities
3 CRITICAL
2 HIGH
2 MODERATE
1 LOW
Dependency update recommendations
Vulnerability information

Summary

Total vulnerabilities: 8

Dependency	Type	Updating		Severity
swiper	Direct	< 6.5.1	6.5.1	critical
immer	Direct	< 9.0.6	9.0.6	high
immer	Direct	< 9.0.6	9.0.6	critical
immer	Direct	< 8.0.1	8.0.1	high
highlight.js	Direct	< 9.18.2	9.18.2	low
minimist	Indirect	< 1.2.6	1.2.6	critical
minimist	Indirect	< 1.2.2	1.2.2	moderate
minimist	Indirect	>= 1.0.0, < 1.2.3	1.2.3	moderate

Total of vulnerable direct dependency: 5

Total of vulnerable indirect dependency: 3

Vulnerabilities

Potentially Vulnerable:	swiper
Severity:	critical
Current Usage Version:	6.4.3
Vulnerable Version:	< 6.5.1
Patch Version:	6.5.1
Vulnerability Chaining:	
Vulnerabilities and Advisory link:	GHSA-p3hc-fv2j-rp68 CVE-2021-23370
CWEs:	CWE-1321: Improperly Controlled Modification of Object Prototype

Figure 3: V-Achilles vulnerability report

shown in vermillion, to make them easily distinguishable from the others. The normal nodes are shown in four different colors as follows. The direct dependencies are shown in orange and the transitive dependencies are shown in blue, yellow, and green according to how many steps they are from the project node. In addition, V-Achilles also shows the edges in a chain of dependencies that contains a vulnerable node in vermillion.

The advantages to having a visualization are that users (i) can differentiate between vulnerabilities from the direct dependencies and transitive dependencies based on the usage of different node colors, (ii) evaluate the complexity of updating a vulnerable dependency by looking at the connections in the graph. Hence, If the vulnerable dependency contains no other packages that depend on it (i.e., having no arrows pointing out from the node), that means the update can be done without impacting other dependencies. On the other hand, if the vulnerable dependency has several packages depending on it, then the update may impact such dependencies and need to be done carefully. (iii) use the tooltip to navigate into its GHSA and CVE records.

3.3 Reporting Mechanism

Figure 3 shows a vulnerability report that can be created after an analysis of the dependency vulnerabilities is finished and the dependency graph is generated. The report shows the summary of all the detected security vulnerabilities in the analyzed npm project. On the top, the report contains the number of vulnerabilities categorized by severity levels. In the middle, it shows the recommendations for updating the vulnerabilities to their patch versions. Lastly, the remaining part of the report contains detailed information about each vulnerability, similar to the information displayed in the tooltip. A graph showing the chain of dependencies of the vulnerability is also included along with the link to the corresponding GHSA entry, the CVE record, and the CWE records of such vulnerability. The report can be saved in PDF format.

4 USAGE SCENARIO

To demonstrate the usefulness, we present three use cases by which we believe the tool can be used alone or complement existing tools.

Case 1: Revealing Insights from Transitive Dependencies. Although recent enhancements of Dependabot include transitive graphs, we believe that V-Achilles is easier to navigate using the interactive graph. Dependabot creates alerts[8] to help developers aware and update their dependency vulnerabilities, however, we find that developers still need to manually fix each vulnerability by themselves by generating a pull request, and trace the code. Different from Dependabot, the developers can use V-Achilles to perform the checking of their npm project in a graph format, and use the tool tip for more information. Although this might be the same information, with V-Achilles, the user can easily use the tool tip information to decide which vulnerability should be addressed first.

Case 2: Just in time Vulnerability Detection. The developers can use V-Achilles to analyze their projects each time a new package is installed. This is to make sure that the newly installed packages (and their dependencies) do not introduce any vulnerabilities into the project. The tool can be used to complement `npm audit` to understand the complexity of the dependencies, i.e., how interconnected the packages within the project are, using the dependency graph visualization.

Case 3: Retrospective Analysis of Project's Security. V-Achilles records all the history of the previous analyses as reports. Thus, if the developers use V-Achilles to analyze their project regularly (e.g., every pull request or every release), the developers can choose to look back and review the analysis information of any npm projects since the beginning to see whether the project is improving in terms of security.

5 APPLICATION

To assess its usability in practice, we used V-Achilles to perform vulnerability analysis on GitHub projects to see if the tool can detect any existing security vulnerabilities. There are two criteria that we chose to sample the repositories. First, the repositories must be developed by using npm. Second, the repositories must have the dependencies in `package.json` or `package.yaml` file.

Then, we choose two sets of projects. First, we retrieved the repositories based on the number of stars. Then, we selected the top 10 projects with the highest number of stars in the study. The information of the 10 projects is shown in Table 1. The project with

[8]https://docs.github.com/en/code-security/dependabot/dependabot-alerts/viewing-and-updating-dependabot-alerts

Table 1: 10-most starred npm projects

Project	Description	Stars
npm	JavaScript package manager	17.3k
np	A better npm publish	6k
sinopia	A private/caching npm repository server	5.4k
nwb	Toolkit for React, Preact, Inferno, & vanilla JS apps	5.3k
concurrently	Command line	4.4k
npm-run-all	A tool running multiple npm-scripts in parallel or sequential	4.1k
node-semver	The semver parser for node	3.6k
cnpmjs.org	Private npm registry and web for Enterprise	3.4k
windows-build-tools	Install C++ Build Tools for Windows using npm	3.2k
npx	Execute npm package binaries	2.6k

the highest number of stars is npm (17.3k) followed by np (6k) and `sinopia` (5.4k). The other projects have a number of stars ranging from 5k to 2.6k. Second, we retrieved the repositories based on the number of dependent packages using the information from the npm registry. This is done using the `all-the-package-names`[9] package in npm registry. Then, we picked the 10 projects with the highest number of dependent packages. The selected projects include `request` (having 15,820 dependent packages), `express` (10,250), `gulp` (2,758), `mocha` (2,074), `grunt` (1,623), `chai` (1,577), `eslint` (916), `should` (612), `sinon` (511), and `istanbul` (391). Nonetheless, after analyzing the projects in this category, we did not find any vulnerabilities. So, we will only discuss the results of the top-10 most starred projects.

As shown in Figure 4, we found that `V-Achilles` discovered the vulnerabilities in 4 most-starred GitHub repositories including `sinopia`, `cnpmjs.org`, `windows-build-tools`, and `npx`. For `sinopia`, `V-Achilless` found 9 direct vulnerabilities (1 low, 1 medium, 6 high, and 1 critical severity based on GHSA), and 3 transitive vulnerabilities (2 low and 1 high). For `cnpmjs.org`, 4 direct vulnerabilities are detected (1 medium and 3 high), and 7 transitive vulnerabilities (4 low, 1 medium, and 2 high). For `windows-build-tools`, the tool found 1 transitive vulnerability (1 low). Laslty, for `npx`, 1 transitive vulnerability is detected (1 low). The full analysis report of the 4 vulnerable projects can be found in the online appendix[10].

6 CONCLUSION

`V-Achilles` allows the developers to explore dependencies in their software intuitively and provides information on vulnerable dependencies to help the developers make decisions on updating such dependencies. The future work includes performing an evaluation of the dependency graph visualization with developers to assess its effectiveness in assisting their decision of dependency updates. We plan to also improve `V-Achilles` to detect vulnerabilities in a longer chain of transitive dependencies (the tool currently supports 4 levels of transitive dependencies). An improved visualization method may be needed due to the complexity of the graph after analyzing all the transitive dependencies. Moreover, visualization like `V-Achilles` can also be included in the code review process to assist the reviewers with the security of the added packages.

[9]https://github.com/nice-registry/all-the-package-names
[10]https://muict-seru.github.io/V-Achilles

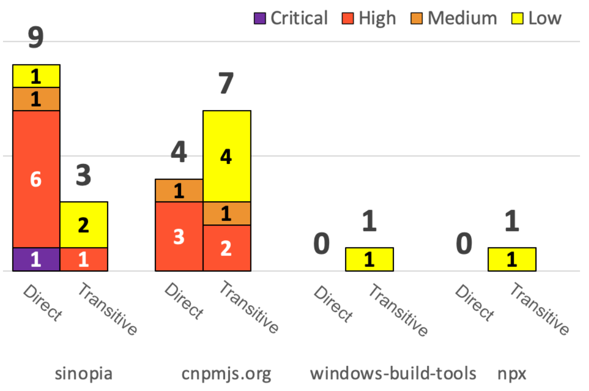


Figure 4: Discovered vulnerable direct and transitive dependencies in 4 GitHub most starred npm projects

ACKNOWLEDGMENTS

This work has been supported by Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Numbers JP20H05706 and JP20K19774.

REFERENCES

[1] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2015. How the Apache Community Upgrades Dependencies: An Evolutionary Study. *Empirical Software Engineering (ESME)* 20, 5 (Oct. 2015), 1275–1317.
[2] Bodin Chinthanet, Raula Gaikovina Kula, Shane McIntosh, Takashi Ishio, Akinori Ihara, and Kenichi Matsumoto. 2021. Lags in the release, adoption, and propagation of npm vulnerability fixes. *Empirical Software Engineering (ESME)* 26, 3 (March 2021).
[3] Russ Cox. 2019. Surviving Software Dependencies. *Queue* 17, 2 (Apr 2019), 24–47.
[4] Alexandre Decan, Tom Mens, and Eleni Constantinou. 2018. On the impact of security vulnerabilities in the npm package dependency network. In *IEEE/ACM Mining Software Repositories Conference (MSR)*. 181–191.
[5] Andre Hora, Romain Robbes, Nicolas Anquetil, Anne Etien, Stephane Ducasse, and Marco Tulio Valente. 2015. How Do Developers React to API Evolution? The Pharo Ecosystem Case. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 251–260.
[6] Akinori Ihara, Daiki Fujibayashi, Hirohiko Suwa, Raula Gaikovina Kula, and Kenichi Matsumoto. 2017. Understanding When to Adopt a Library: A Case Study on ASF Projects. In *13th International Conference on Open Source Systems (OSS)*. 128–138.
[7] Riivo Kikas, Georgios Gousios, Marlon Dumas, and Dietmar Pfahl. 2017. Structure and Evolution of Package Dependency Networks. In *IEEE/ACM Mining Software Repositories Conference (MSR)*. 102–112.
[8] Raula Gaikovina Kula, Daniel M. German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2018. Do Developers Update Their Library Dependencies? *Empirical Software Engineering (ESME)* 23, 1 (Feb. 2018).
[9] Romain Robbes, Mircea Lungu, and David Röthlisberger. 2012. How Do Developers React to API Deprecation?: The Case of a Smalltalk Ecosystem. In *International Symposium on the Foundations of Software Engineering (FSE)*. 56:1–56:11.
[10] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. 2016. On the reaction to deprecation of 25,357 clients of 4+1 popular Java APIs. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 400–410.
[11] Snyk. 2020. *The state of open source security report*. Technical Report. Snyk.
[12] Bang Wong. 2011. Points of view: Color blindness. *Nature Methods* 8, 6 (Jun 2011), 441–441.
[13] Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, and Laurie Williams. 2022. What are Weak Links in the npm Supply Chain?. In *ICSE-SEIP '22*.
[14] Markus Zimmermann, Cristian Alexandru Staicu, Michael Pradel, and Cam Tenny. 2019. Small world with high risks: A study of security threats in the NPM ecosystem. *Proceedings of the 28th USENIX Security Symposium* (2019), 995–1010.