On the impact of security vulnerabilities in the npm package dependency network

Alexandre Decan, Eleni Constantinou and Tom Mens
Software Engineering Lab, University of Mons
Belgium
first.last@umons.ac.be

Abstract

Security vulnerabilities are among the most pressing problems in open source software package libraries. It may take a long time to discover and fix vulnerabilities in packages. In addition, vulnerabilities may propagate to dependent packages, making them vulnerable too. This paper presents an empirical study of nearly 400 security reports over a 6-year period in the npm dependency network containing over 610k JavaScript packages.

1 Introduction

Security vulnerabilities are among the most pressing problems in our software-intensive society, given the increasing reliance on software libraries and the potentially harmful impact security vulnerabilities they may have [Tho03]. A recent report by Snyk, one of the leading companies analysing software vulnerabilities in Node.js and Ruby packages, summarises the current state of open source security [sny17]. Based on an analysis of 430k websites, they report that no less than 77% of them run at least one front-end library with a known security vulnerability.

A well-known example of a security vulnerability is the so-called Heartbleed Bug [DLK+14]. It represented a serious security leak in the OpenSSL cryptography library, allowing anyone on the Internet to read the memory of the software systems relying on the vulnerable versions of the OpenSSL software. The vulnerability was introduced in 2012 and remained lingering until it was discovered and traced in April 2014. Upon its discovery, half a million of servers certified by trusted authorities were believed to be affected by the security vulnerability, connected to a simple programming mistake.

Little is known about, and even less automated support is available for, assessing the ecosystem-wide impact of security vulnerabilities in package dependency networks. Vulnerabilities, as well as their fixes, may spread over the network through dependencies between software packages. Analysing the propagation of security vulnerabilities and their fixes is technically challenging and computationally intensive, due to the intricate interactions of the mechanisms of semantic versioning and dependency constraints.

This paper provides an empirical study of the propagation of security vulnerabilities and their fixes in the package release history of the npm distribution of Node.js packages.

2 Dataset

Snyk.io provides a continuous monitoring service aiming to help developers and organizations identify vulnerable packages on which they depend. For npm it includes information about the vulnerability, the name of the affected package, the vulnerability constraint specifying which releases of the package are concerned by the security report, the discovery date indicating when the vulnerability was discovered, and the date of *public* exposure of the vulnerability which is by definition greater than or equal to the discovery date. The large majority of vulnerabilities in the dataset have a publication date that is strictly later than the discovery date. To each vulnerability a *severity* label (*low*, *medium* or *high*) is assigned. This severity is assigned manually based on the impact of the vulnerability and

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

how easy it is to exploit it[1] and seems to be determined primarily by the vulnerability's CVSS score[2].

To identify the security vulnerabilities that affect npm packages, we manually gathered from Snyk.io all 700 security reports that were published before 2017-11-09. For each package identified in a security report, we retrieved the list of its releases from the open source discovery service libraries.io [NN17]. Their dataset, available under a CC-BY-SA 4.0 licence, contains metadata from the manifest of each package, based on the list of packages provided by the official registry of the npm package manager.

Based on this list of releases, we identified which ones were affected by the vulnerability. We filtered out those vulnerabilities for which the affected packages no longer exist in npm or for which none of the releases satisfy the vulnerability constraint (i.e., they have been deleted from npm). We also ignored vulnerability reports indicating a universal vulnerability constraint (i.e., "*") and removed vulnerabilities of type "Malicious Package" since they correspond to typosquatting packages. The filtered dataset contains 399 vulnerabilities affecting 269 distinct packages. These packages account for 14,931 distinct releases of which 6,752 are affected by a vulnerability.

As a vulnerable package can affect packages that make use of it, for each package affected by a vulnerability, we considered the packages that have direct dependencies towards the vulnerable ones. To do so, we analyzed the direct dependencies of all packages that were available on 2017-11-02 on libraries.io. These 610K packages account for more than 4M releases and for more than 20M runtime dependencies. Among these packages, we identified 133,602 packages that directly depend on a vulnerable package and 52% of these packages, i.e., 72,470 packages, have at least one release that relies on an affected release of a vulnerable package. Table 1 summarizes the descriptive statistics of the dataset.

Table 1: Descriptive summary of the npm dataset

610,097	npm packages
4,202,099	releases of npm packages
20,240,402	runtime dependencies
399	security vulnerability reports
269	packages affected by the vulnerability
14,931	releases of such vulnerable packages
6,752	releases affected by the vulnerability
133,602	packages depending on a vulnerable package
72,470	dependent packages affected by the vulnerability

[1]See `https://support.snyk.io/frequently-asked-questions/finding-vulnerabilities/how-do-you-determine-the-severity-of-a-vulnerability`

[2]See `https://nvd.nist.gov/vuln-metrics`

3 RQ_0: How many packages are known to be affected by vulnerabilities?

RQ_0 aims to provide some initial understanding about how many npm packages suffer from security vulnerabilities, and how this number increases over time.

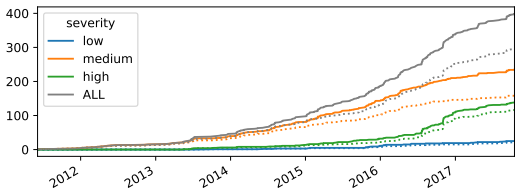

Figure 1: Evolution of the number of discovered vulnerabilities (straight lines) and corresponding distinct packages (dotted lines) per severity.

Figure 1 shows the number of discovered vulnerabilities in the considered dataset. Straight and dotted lines correspond respectively to the number of discovered vulnerabilities and the number of corresponding packages. The results suggest an increasing number of new vulnerabilities and of vulnerable packages over time. We observe that the large majority of vulnerabilities have a medium or high severity (respectively 235 and 139 out of 399), while there is a much lower number of low severity vulnerabilities (25 out of 399). This might be due to the fact that vulnerabilities that have a very limited impact in terms of security are considered as not worth having a vulnerability report by maintainers.

4 RQ_1: How long do packages remain vulnerable?

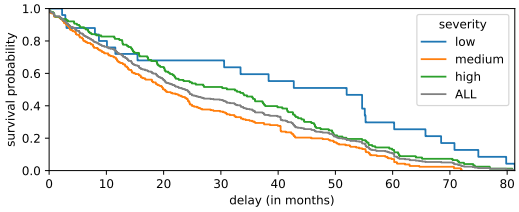

Figure 2: Survival probability for event "vulnerability is fixed" w.r.t. the date of first affected release.

Figure 2 shows Kaplan-Meier survival curves for the event "vulnerability is fixed" w.r.t. the date of the first affected release. Regardless of the severity of the vulnerability, it takes a surprisingly long time before the vulnerability is fixed. One can observe that high severity vulnerabilities take longer to fix than medium severity ones, probably because it is more difficult to fix them. Low severity vulnerabilities take even longer to fix, perhaps because developers consider them as

low priority. For example, it takes about 52 months before 50% of all low severity vulnerabilities get fixed, 20 months for 50% of all medium severity vulnerabilities, and 32 months for 50% of all high severity vulnerabilities.

We carried out log-rank tests to compare whether statistically significant differences could be found between the survival curves of time-to-fix by severity. The differences were statistically confirmed at $\alpha = 0.95$, i.e., the null hypotheses H_0 assuming that the survival curves are the same were rejected with p-values < 0.05. This confirms that there is a statistically significant difference in the time required to fix a vulnerability with respect to its severity.

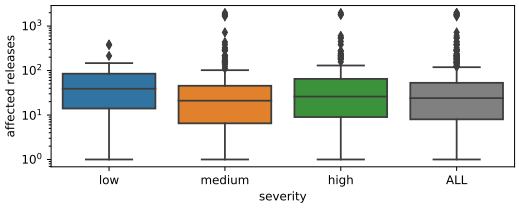

Figure 3: Distribution of the number of affected releases of vulnerable packages by severity.

Figure 3 illustrates how many releases of a vulnerable package are affected by each vulnerability, among all releases of the package that were available when the vulnerability was discovered. One can observe that the number of affected releases per vulnerable package is quite high. The affected releases represent a large proportion of all the releases that were available for the vulnerable packages at discovery time. Regardless of the severity, 75% of all vulnerable packages have more than 90% of their releases affected (94% for low, 80% for medium and 93% for high severity vulnerabilities).

5 RQ_2: When are vulnerabilities discovered?

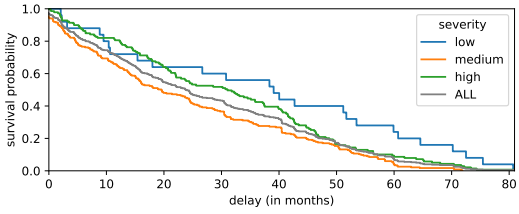

Figure 4: Survival probability for event "vulnerability is discovered" w.r.t. the date of first affected release.

Figure 4 presents the Kaplan-Meier survival curves for event "vulnerability is discovered" w.r.t. the date of first affected release. Although vulnerabilities should be discovered early, this does not seem to hold true in practice. It takes more than 24 months to discover 50% of all vulnerabilities. The survival curves of Figure 4 also reveal that it takes more time to discover low severity vulnerabilities than medium or high severity vulnerabilities. For instance, it takes 39, 20 and 31 months to discover 50% of all vulnerabilities respectively for low, medium and high severity. We used log-rank tests to compare the time to discover vulnerabilities depending on their severity, and found a statistically significant difference ($p < 0.05$) when comparing low with medium severity, and medium with high severity vulnerabilities.

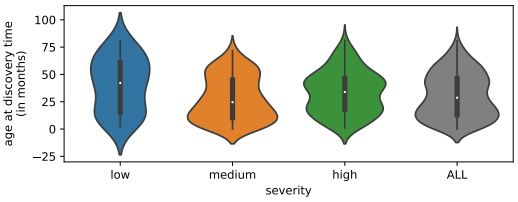

Figure 5: Violin plots of package age at discovery time by vulnerability severity.

Given the long time required to discover vulnerabilities, we expect most of them to be found in old packages. Figure 5 shows the distribution of package age at discovery time by vulnerability severity, and confirms that most vulnerabilities, regardless of their severity, are discovered in old packages. For instance, 75%, 50% and 25% of all vulnerabilities are discovered in packages respectively older than 13, 28 and 46 months. One possible explanation is that younger packages have not yet had time to reach a wide audience and to receive as much security-related attention as older packages.

6 RQ_3: When are vulnerabilities fixed?

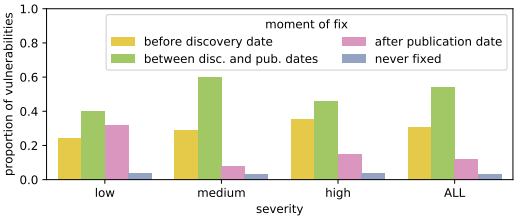

Figure 6: Proportion of vulnerabilities by severity according to the moment of fixing the vulnerability.

Figure 6 shows the proportion of vulnerabilities, by severity, according to their moment of fix: before the vulnerability has been discovered ("before discovery date" in Figure 6), between discovery and publication date, after the vulnerability has been made public ("after publication date"), or "never fixed". We observe that the vast majority of vulnerabilities are eventually

fixed, regardless of their severity. Indeed, only 3.5% of the vulnerabilities from our dataset have not (yet) been fixed (resp. 4%, 3.4% and 3.6% for low, medium and high severity).

Surprisingly, 30.9% of all vulnerabilities (resp. 24%, 28.9% and 35.3% for low, medium and high severity) were already fixed at discovery time. One possible explanation is that the higher the severity of a vulnerability is, the less likely the maintainers of an affected package are willing to disclose the vulnerability and to report its discovery while working on a fix. Finally, of the 65.7% remaining vulnerabilities (i.e., those that were fixed after the discovery date), a large majority (82% of them) are fixed before being publicly announced. About 12% of all vulnerabilities are fixed only after their official publication date (resp. 32.0%, 7.7% and 15.1% for low, medium and high severity). This makes them particularly vulnerable for being exploited by malevolent users. These results indicate that maintainers seek to fix vulnerabilities before their publication, thus minimising the chances of their exploitation.

Let us focus now on only those vulnerabilities that get fixed after their discovery, in order to understand how long it takes to fix discovered vulnerabilities. To achieve this, we ignored all vulnerabilities that were fixed *before* the reported discovery date (31% of all vulnerabilities, as shown in Figure 6). For the remaining vulnerabilities, Figure 7 presents the survival curves of the probability for event "vulnerability is fixed" w.r.t. their discovery date.

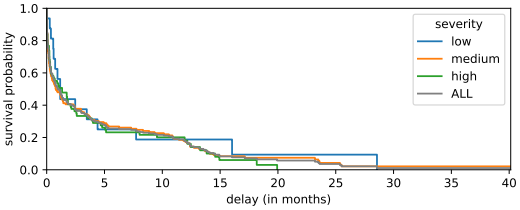

Figure 7: Survival probability for event "vulnerability is fixed" w.r.t. vulnerability discovery time.

We observe that most vulnerabilities, regardless of their severity, are fixed within a few months after their discovery. The probability that a vulnerability is fixed within the month following its discovery is 50%, while it is 74% after only 6 months. There is, however, a non-negligible proportion of vulnerabilities that take a long time to be fixed after their discovery. For instance, the probability that a discovered vulnerability is not fixed after 12 months is still 17.4%.

7 RQ_4: When are vulnerabilities fixed in dependent packages?

While our dataset contains 399 security vulnerabilities for 269 distinct npm packages, we identified 133,602 packages that directly depended on these potentially vulnerable packages, and more than half of them (72,470 i.e., 54%) have at least one release that relies on an affected release of a vulnerable package. These 72,470 vulnerable dependents account for 920,661 releases, of which 411,169 are affected by a vulnerability because of one of their dependencies.

Dependent packages can explicitly fix vulnerabilities in three different ways: (1) by rolling back to an earlier version of the dependency that is not affected by the vulnerability; (2) by updating to a more recent version of the dependency that has fixed the vulnerability; or (3) by removing the dependency to the vulnerable package.

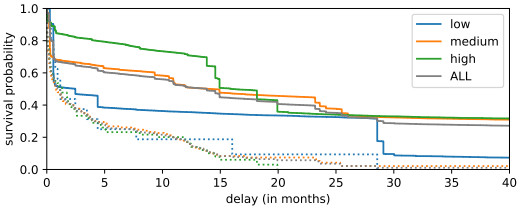

Figure 8: Survival probability for event "package is fixed" w.r.t. vulnerability discovery time. Dependent packages are shown as straight lines and upstream packages as dotted lines.

Figure 8 presents the Kaplan-Meier survival curves of the probability for event "package is fixed", w.r.t. discovery date of the vulnerability. Straight and dotted lines correspond to the time it takes for a package to be fixed for the dependent and upstream packages, respectively. Dotted lines thus correspond to the survival curves of Figure 7.

Dependent packages need considerably more time to be freed from vulnerabilities than their upstream packages. We confirmed this using log-rank tests, and found a statistically significant difference between dependent packages and their upstream packages in the time required to fix a vulnerability for medium and high severity vulnerabilities. While 50% of the upstream packages are fixed within the month, only 33.1% of the dependent packages are fixed within this time frame, and it takes nearly 14 months to fix 50% of them. These results illustrate the importance of having continuous monitoring of dependencies, so that maintainers of dependent packages can be notified quickly of the presence of a vulnerability in one of their dependencies, as well as the presence of a fix.

We analysed in more depth the moment that af-

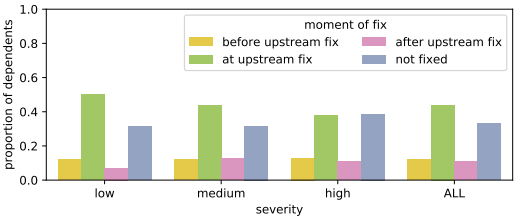

Figure 9: Moment of fix for affected dependents.

fected dependents are freed from a vulnerability in their upstream packages. Figure 9 presents the proportion of affected dependents that are fixed before, simultaneously with, or after the fix of the upstream package, or that are never fixed.

The results indicate that only a small fraction (12.2%) of all dependents are fixed before the upstream fix, either by removing the dependency to the vulnerable package or by rolling back to a non-affected release. Most dependents (44%) are fixed at the same time as the upstream fix; the dependent package automatically benefits from the fix, because the specified dependency constraint allows to update to releases containing the fix. We also observe that 10.8% of dependents are fixed after the upstream fix, suggesting that "manually" managing dependencies and changing dependency constraints requires effort and leads to delays in benefiting from vulnerability fixes.

More importantly, Figure 9 reveals that more than 33% of all dependents affected by an upstream vulnerability are not (yet) fixed. Zooming in on those dependent packages that have not (yet) been fixed from a vulnerability in an upstream package, Figure 10 presents the proportion of these affected dependents according to the status of the upstream fix.

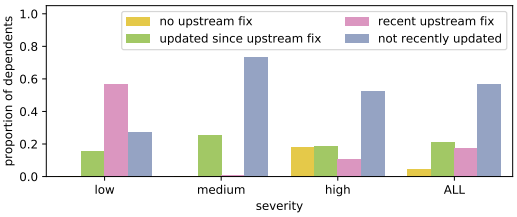

Figure 10: Status of affected dependents that are not fixed.

We observe that only 4.8% of them are not fixed because the vulnerable upstream package has not yet published a fix ("no upstream fix" in Figure 10). For affected dependent packages for which an upstream fix is available, we consider their latest update date. The packages that had some more recent update yet did not integrate the upstream fix ("updated since upstream fix") represent 21.3% of the dependents that are not fixed. For dependent packages that were not updated since an upstream fix was available we consider two cases, to address the fact that recent upstream fixes may not have had sufficient time to be taken into account in affected dependent packages. Affected dependent packages for which the upstream fix was published during the last 6 months ("recent upstream fix") represent 17.2% of the unfixed dependents. Affected dependent packages that have not been updated during the last 6 months ("not recently updated") represent 56.6% of the unfixed dependents. The reason why such a large majority of affected dependents continue to remain vulnerable even after the availability of a fix appears to be that these dependent packages are no longer actively maintained.

8 Conclusion

We studied 399 security reports, affecting 269 distinct packages and 6,752 releases of these packages. Considering package dependencies and taking into account dependency constraints, 72,470 other packages are affected by these vulnerable releases. Our findings scream for a higher awareness among npm package maintainers of the risks incurred by security vulnerabilities, not only at the level of individual packages, but also at a wider ecosystem level by considering package dependencies. Package maintainers should also rely on better use of policies and automated tools to detect and fix vulnerabilities faster, and to reduce the impact of vulnerabilities on dependent packages.

8.0.1 Acknowledgements

This research was carried out in the context of FRQ-FNRS collaborative research project R.60.04.18.F "SECOHealth", FNRS Research Credit J.0023.16 "Analysis of Software Project Survival" and Excellence of Science project 30446992 SECO-Assist financed by FWO - Vlaanderen and F.R.S.-FNRS.

References

[DLK+14] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The matter of heartbleed. In *Internet Meas. Conf.*, pages 475–488, 2014.

[NN17] A. Nesbitt and B. Nickolls. Libraries.io open source repository and dependency metadata, June 2017.

[sny17] snyk. The state of open source security. https://snyk.io/stateofossecurity/, 2017.

[Tho03] H. H. Thompson. Why security testing is hard. *IEEE Security Privacy*, 1(4):83–86, July 2003.