The state of JavaScript frameworks security report 2019

A security review of Angular and React

with a sneak peek into Vue.js, Bootstrap and jQuery

powered by snyk

Table of contents

Introduction

In this report, we investigate the state of security for both the Angular and React ecosystems. This report by no means intends to venture into any rivalries that may exist between the two in terms of whether one or the other is a true framework - we are not comparing them as competitive frameworks at all. Instead, we review them each as viable frontend ecosystem alternatives for building your JavaScript projects, while focusing on security risks and best practices for each and the differences between them.

This report covers:

- the security practices for each of the two different core projects, both Angular and React
- the state of security of each of the two different module ecosystems, based on an in-depth look at the vulnerabilities contained in each of the ecosystems
- the security practices for other common JavaScript frontend framework alternatives such as Vue.js, Bootstrap and jQuery
- the significant security differences between the different alternatives, and particularly between Angular and React

This report reviews the overall security of each framework, their community-powered module ecosystems and the associated security risks with each; based on these insights, this report ultimately provides actionable security advice for Angular and React users by highlighting best security practices employed in the field in order to ensure secure code.

A word about vulnerabilities

In order to investigate the overall security posture of each of the ecosystems included in this report, amongst the factors we discuss are security vulnerabilities identified in the different relevant packages. We review and discuss these vulnerabilities on the landscape of, and sometimes in comparison to, known vulnerabilities. Known vulnerabilities have been assigned an identification number in the list of Common Vulnerabilities and Exposures (CVEs) maintained by the CVE Numbering Authorities (CNAs). CVEs are assigned CVSS scores that provide insight into how severe the listed vulnerabilities are. Learn more about how the severities of vulnerabilities are scored via their CVSS here.

Key takeaways

Angular vs. React core project security

- Angular contains twenty three security vulnerabilities in its legacy AngularJS project (Angular v1.x).
- No security vulnerabilities were identified in the core Angular framework components.
- React has a few security vulnerabilities; vulnerabilities seem to be regularly found in its core libraries and disclosed every couple of years.
- Only one React core project vulnerability has an official CVE assigned. None of the reported Angular vulnerabilities are listed by CVE at all. Together, these prove the need for a vulnerability database that taps into open source community activities, in order to surface relevant security issues.
- Snyk reports twenty six security vulnerabilities across Angular and React core projects, which npm audit falls short of in its reports.

Angular vs. React module ecosystem security

- Both React and Angular module ecosystems exhibit security vulnerabilities in highly popular frontend library components spanning millions of downloads, some of which have no security fix available to date.
- We have witnessed malicious modules impacting both the Angular and the React ecosystems with an attempt to harvest credit cards, passwords and other sensitive information used in frontend web applications.
- The Next.js framework exhibited a great commitment to security by swiftly addressing all five vulnerabilities found throughout the lifetime of their project, offering fixes within just one week.

Angular vs. React security posture

- Angular has visible and attainable security guidelines, a security contact and a responsible disclosure policy, all of which are missing from the React project.
- Angular has broader built-in support for data sanitization and output encoding in different contexts such as URL attributes in HTML anchor (or, link) elements.
- React doesn't have built-in controls for data sanitization, but rather encodes output by default in most cases and leaves it up to developers to address unhandled cases such as refs and URL attributes (the latter of which is addressed in the React v16.9.0 release).
- Angular includes support for Cross-Site Request Forgery (CSRF) vulnerabilities with a built-in security mechanism in its HTTP service. React developers need to address these issues independently.

Frontend ecosystem security

- jQuery was downloaded more than 120 million times in the last 12 months and according to W3Techs, jQuery v1.x is used in 84% of all websites using jQuery, which have four medium severity XSS vulnerabilities affecting it. In fact, if you're not using jQuery v3.4.0 and above, which is true for the majority of jQuery users, then you are using a version that includes security vulnerabilities.
- Bootstrap has been downloaded 79,185,409 times in the past twelve months, all while containing seven Cross-Site Scripting (XSS) vulnerabilities. Three of these were disclosed in 2019. Notable community modules such as bootstrap-markdown have more than 300,000 downloads in the same time frame, despite having no security fix or upgrade path to its XSS vulnerabilities. bootstrap-select features more than two million downloads and has a high severity XSS vulnerability that the Snyk research team surfaced with the help of their proprietary threat intelligence system.
- The Vue.js framework has been downloaded more than 40 million times this past 12 months and records four vulnerabilities in total for Vue.js core, all of which have been fixed.

1 Angular and React core projects: security vulnerabilities

Let's begin this report by exploring the different security vulnerabilities found in the core Angular and React projects. We then review the severity breakdown for each of the vulnerabilities and we inspect the differences between the two. Lastly, for both projects, we review the time gap from when a vulnerability was disclosed until it was fixed, as well as the time gap until the time at which an upgrade was finally published (time-to-fix, time-to-release) for each of the cases.

Core React project: overview

For the purposes of this report, we considered the **react**, **react-dom**, and **prop-types** libraries to be the "core" React modules since, together, they often make up the foundation for web applications built in React.

For these core modules, we found three vulnerabilities in total; two in react and one in react-dom.

All three are Cross-Site Scripting (XSS) vulnerabilities. The two XSS vulnerabilities in the React npm package are quite old and include the 0.5.x versions dated back to 2013, and the versions prior to 0.14 that were disclosed in 2015.

The XSS vulnerability in the react-dom v16.x release branch, on the other hand, is quite recent and was disclosed just over a year ago, in August 2018. This vulnerability, however, only occurs when other pre-conditions exist as well, such as using the react-dom library within a server-side rendering context. Nevertheless, it is always advisable to stay up-to-date with security fixes and to upgrade your open source components as early as possible, in order to avoid any unnecessary security risks.

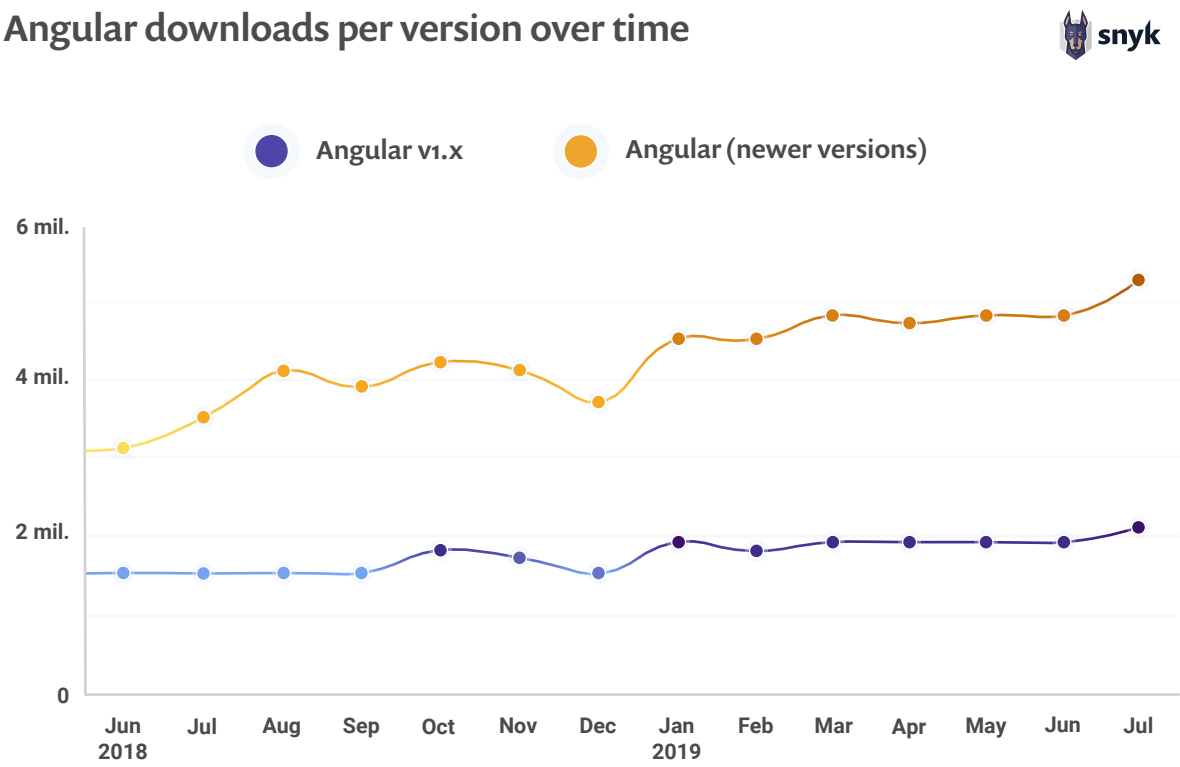
Spotlight: Preact—a React alternative

In addition to these three core Reach project vulnerabilities, we also tracked a medium Deserialization of Untrusted Data security vulnerability in Preact. As many developers prefer Preact over React, for being lightweight and faster, we thought it was worth having a closer look. This medium-severity Preact vulnerability affects the 10.0.0 pre-release branch versions from March and April 2019.

Core Angular project: overview

When we looked at core Angular projects, we specifically investigated security vulnerabilities in the v1.x branch, also referred to as AngularJS. AngularJS is the most widely-used outdated (no longer maintained) version of Angular.

We charted the monthly download counts for the **angular** and **@angular/core** npm packages, which represent Angular v1.x and Angular v2.0 and above, respectively. According to the data we reviewed, we found that Angular v1.x is still very much a considerable player within the Angular market-share, representing 28% of all Angular downloads across all versions. While Angular has reached many more major version releases since 1.x, the reality is that users continue to download this older version millions of times a month.

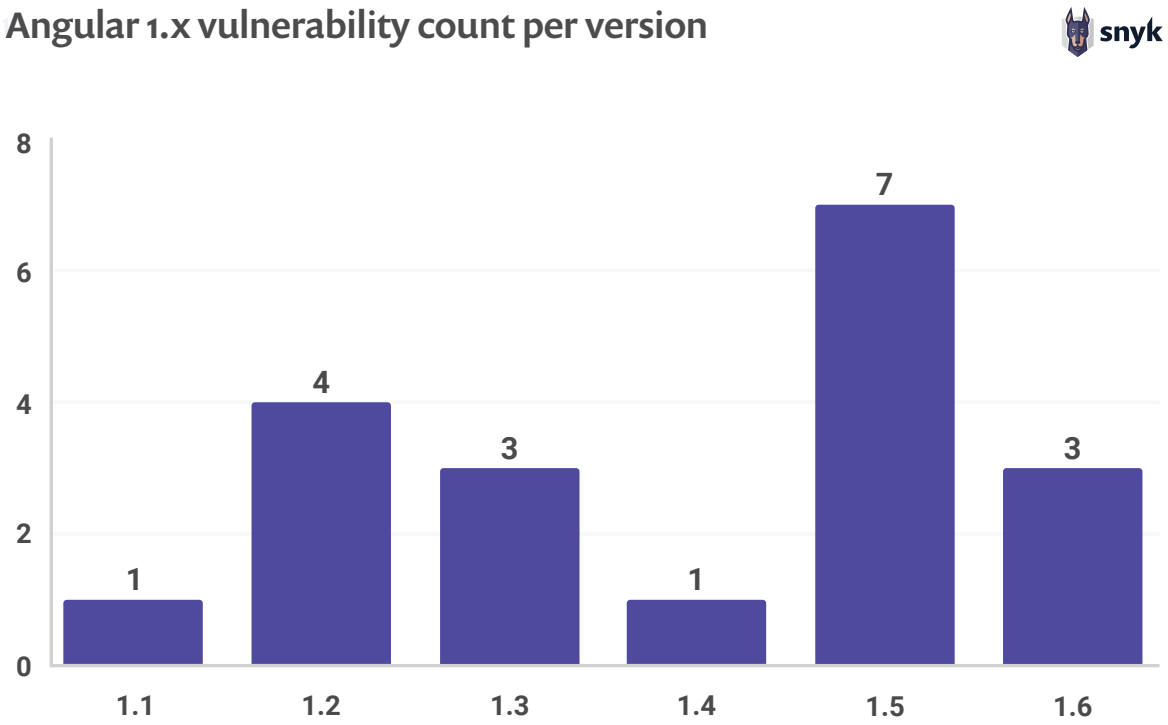


The above graph demonstrates just how popular the Angular 1.x versions are relative to the other Angular options: the Angular 2.0 and above versions are represented ***together*** by the yellow line; we can see from the graph that Angular v1.x alone represents about a third (28% to be exact) of all Angular (new and old) downloads ever.

We can speculate that the continued use of Angular 1.x versions may be due to legacy applications that are still maintained by medium and large enterprises. For these organizations, technology change and migration paths are very costly, but applications still need to be maintained and released regularly.

It is this assumption that emphasizes, even more, the critical need to track security vulnerabilities in open source components such as the Angular framework including for older versions, in order to quickly address any vulnerabilities identified in production-deployed applications and ensure these do not escalate and worsen a company's security posture over time.

With these issues in mind, in total, we found 19 vulnerabilities across six different release branches of Angular v1.x, with the minor version breakdown as itemized in the graph that appears to the right.



Angular and React: good vulnerability databases are key to surface security issues

Though Snyk has tracked **twenty three** Angular v1.x security vulnerabilities, none of them includes a CVE reference because they were not disclosed through any of the officially-recognized CVE programs. This isn't necessarily a failing on the part of Angular, but rather common practice, as CVEs were designed with commercial vendors in mind, requiring substantial time and expertise to file - and this doesn't always scale well for open source.

Without a CVE, vulnerabilities can only be tracked by dedicated analysts who manage and track open-source activity with customized methods; few solutions provide this option.

Tools such as **npm audit** do some tracking, but miss many of the vulnerabilities that lack a CVE as well. For instance, **npm audit**, which is bundled by default with npm client, unfortunately misses all twenty three Angular v1.x vulnerabilities and all React vulnerabilities, and so relying on **npm audit** can provide developers with a false sense of confidence.

```
lirantal@~/angular-1.2.32 $ npm install --save angular@1.2.32

+ angular@1.2.32
updated 1 package and audited 1 package in 1.077s
found 0 vulnerabilities
```

All of this is in contrast to Snyk's vulnerability database for Angular 1.x which, for example, reports eleven security reports eleven security vulnerabilities for Angular v1.2.32.

Version	Published	Licenses	Direct vulnerabilities
angular 1.6.0-rc.1 (pre-release)	21 Nov, 2016	MIT	3 medium
angular 1.6.0-rc.0 (pre-release)	27 Oct, 2016	MIT	3 medium
angular 1.4.14	11 Oct, 2016	MIT	3 high 6 medium
angular 1.2.32	11 Oct, 2016	MIT	3 high 7 medium 1 low

```
✗ High severity vulnerability found in angular
  Description: Cross-site Scripting (XSS)
  Info: https://snyk.io/vuln/npm:angular:20150909
  Introduced through: angular@1.2.32
  From: angular@1.2.32
  Remediation:
    Upgrade direct dependency angular@1.2.32 to angular@1.5.0 (triggers upgrades to angular@1.5.0)

Organization:      snyk-demo-567
Package manager:   npm
Target file:       package-lock.json
Open source:       no
Project path:      /Users/lirantal/angular-1.2.32/
Licenses:          enabled

Tested 1 dependencies for known issues, found 11 issues, 11 vulnerable paths.

Run `snyk wizard` to address these issues.
```

— These vulnerabilities are reported through the Snyk UI or the Snyk CLI tool. What's more, the Snyk UI automatically creates fix PRs to upgrade vulnerable packages for the developers, through integrations with systems such as GitHub.

```
lirantal@~/react-0.5.0 $ npm install --save react-dom@16.4.1 --save react@0.5.0

+ react-dom@16.4.1
+ react@0.5.0
updated 2 packages and audited 29 packages in 1.307s
found 0 vulnerabilities
```

— The situation with React is similar - npm audit misses all three React related vulnerabilities. Out of the three publicly known vulnerabilities (two affecting the **react** core library, and one affecting the **react-dom** core library), only the latter has a CVE assigned that is tracked in the National Vulnerability Database (https://nvd.nist.gov).

Angular vs. React: comparing vulnerability severities

We can gain further insights into the overall risk posed by the security issues that were found for React-based and Angular-based frontend projects by exploring their severity scores.

What is CVSS?

To do this, we should briefly review the scoring system (CVSS). Similarly to the way in which general bugs in source code are associated with a severity such as high or medium, security bugs, which we refer to as security vulnerabilities, are also associated with a severity that contributes to determining the potential risk for an organization.

Security vulnerabilities are assigned severity through the Common Vulnerability Scoring System (CVSS), which is employed as the de-facto standard by the FIRST organization and is widely used to score Common Vulnerabilities and Exposures, often referred to in short as CVEs.

To easily compare vulnerabilities, the CVSS translates its numerical scores into ranges, associating each range to its severity type.

An in-depth explanation of CVSS and its challenges is available at https://snyk.io/blog/scoring-security-vulnerabilities-101-introducing-cvss-for-cve/.

Throughout the report, we refer to CVSS v2 scoring, which is shown in the following table:

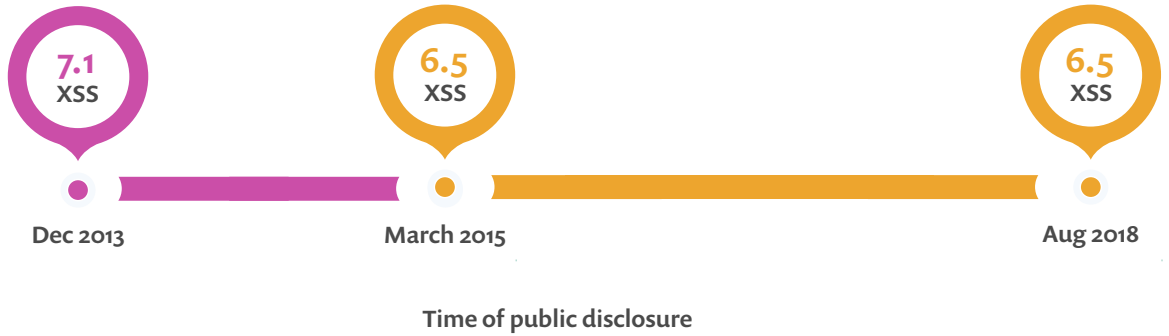
Severity	Score
0.1 - 3.9	low
4.0 - 6.9	medium
7.0 - 10.00	high

Angular and React: CVSS results

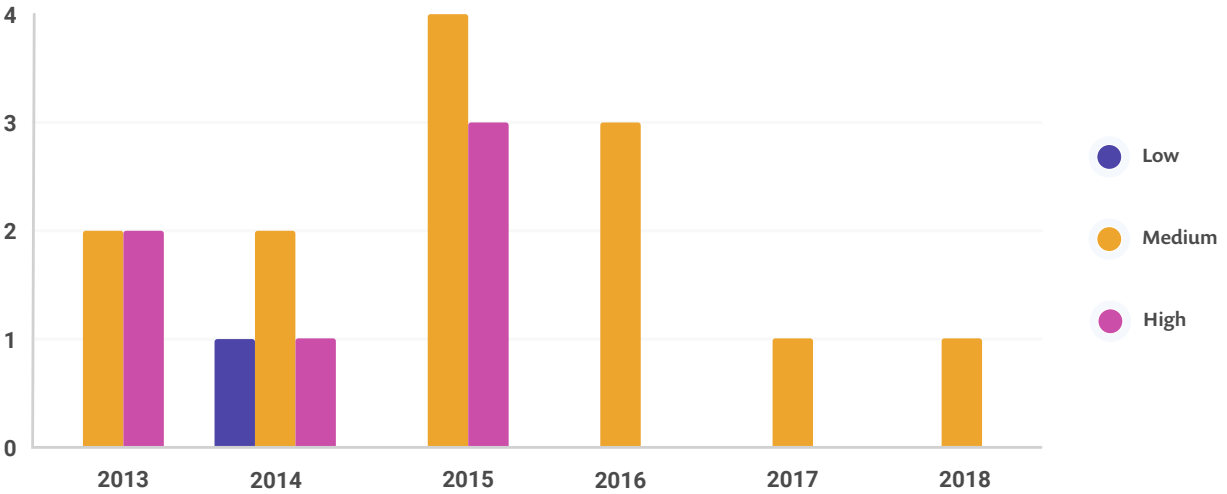
While very few vulnerabilities have been discovered within core React packages, they are all Cross-Site Scripting vulnerabilities and have been steadily disclosed every couple of years. Their CVSS scores range between 6.5 and 7.1—or, in other words, they are all medium to high severity vulnerabilities.

Looking into Angular v1.x security vulnerabilities, we can see that Angular v1.5 exhibits the most vulnerabilities, with seven vulnerabilities in total-three high and four medium. Luckily, the vulnerabilities further decrease as the version matures, in terms of both severity and count. In 2019, we haven't yet seen any newly disclosed vulnerabilities for any Angular versions at all!

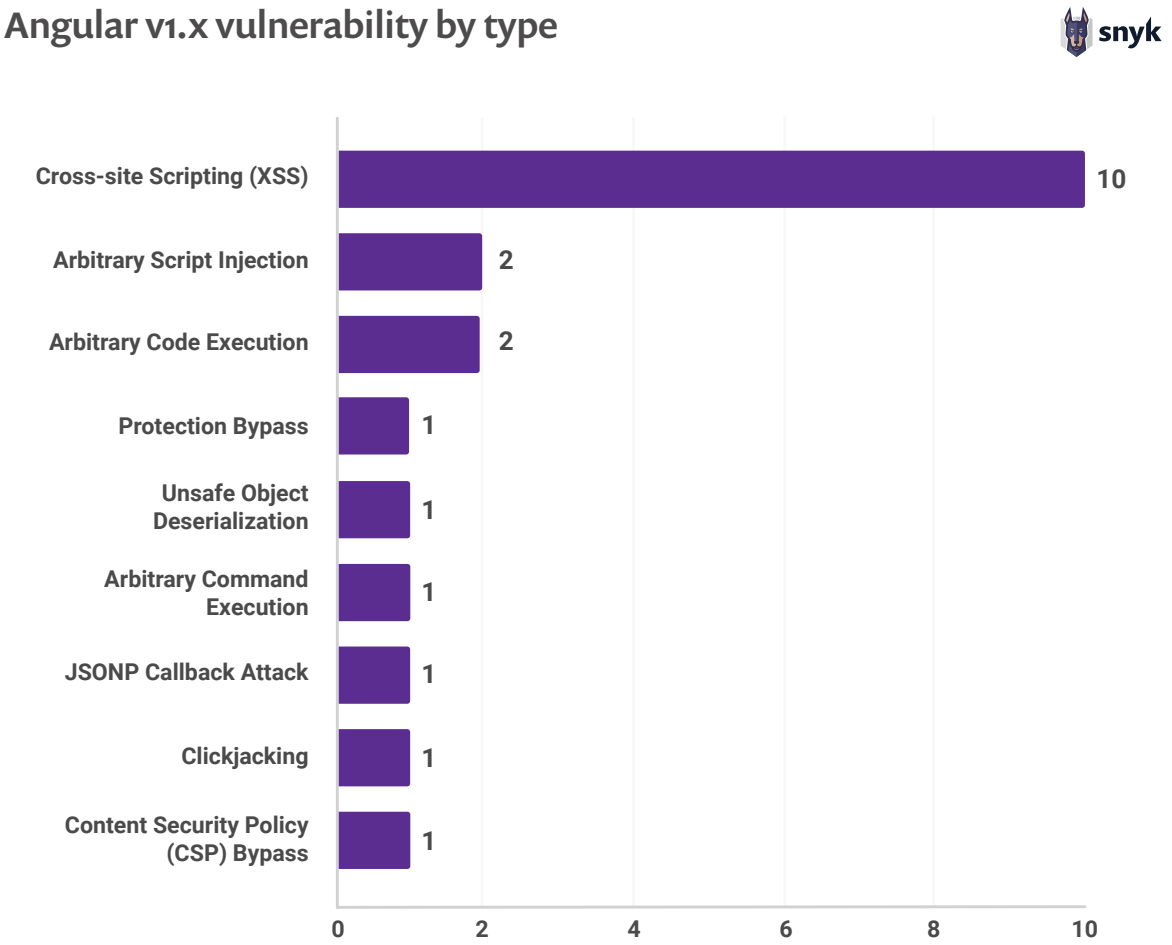
React core project vulnerability severities over time



Angular v1.x vulnerability count per year by severity



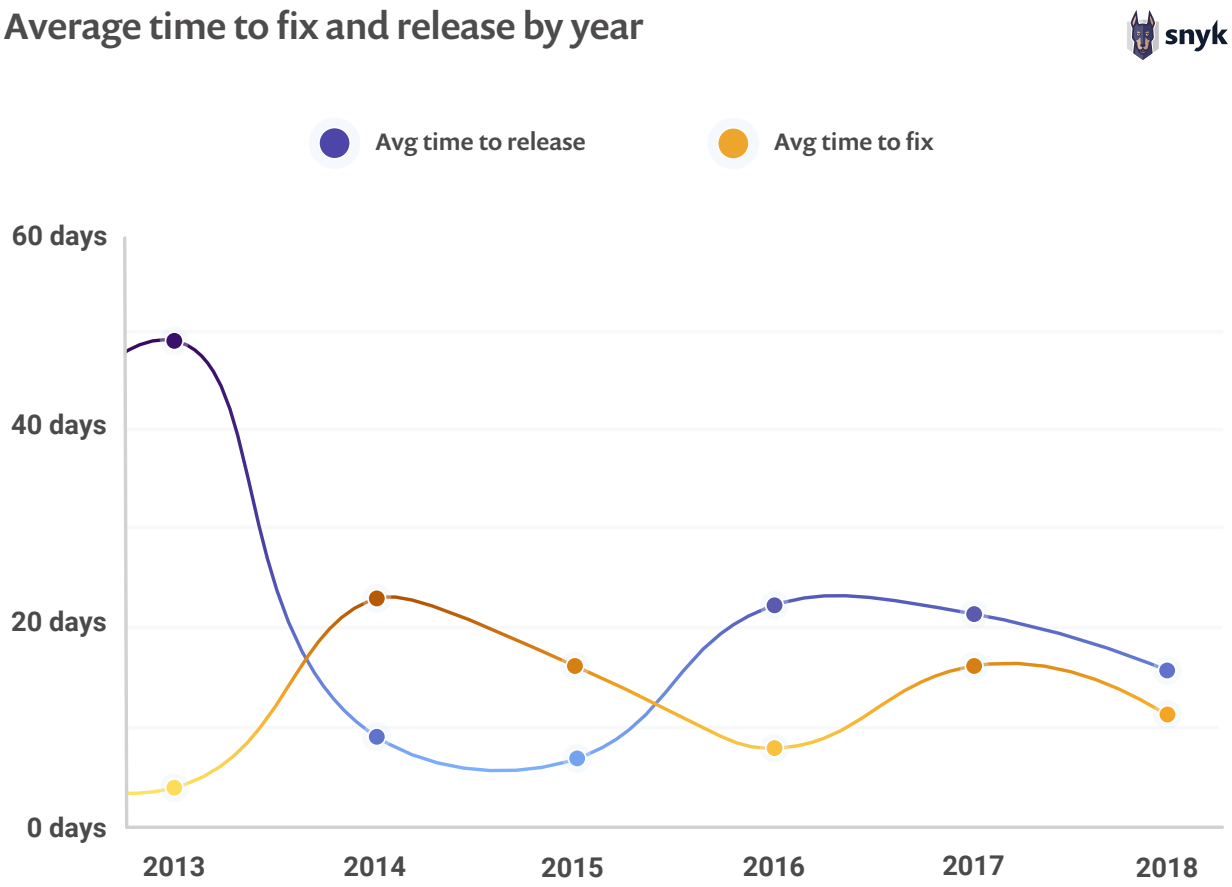
The variety of vulnerability types disclosed across all Angular 1.x versions is hardly a concern but rather, security risks manifest themselves in other aspects. Most notable is the severity of its most common vulnerability type, repeatedly disclosed across versions. In fact, Cross-Site Scripting (XSS) vulnerabilities are a great security concern across the frontend world. And this is no different for Angular. This is reflected in the data we found here, with ten XSS vulnerabilities in total appearing across the Angular 1.x versions.



Time-to-fix, time-to-release

An important factor to weigh for the security posture of open source projects is how quickly maintainers and collaborators are able to respond to security vulnerabilities with timely fixes and to publish releases for their users. We looked at both the Angular and the React core projects for these metrics, tracking the history of known vulnerabilities that have already been handled in each in order to chart this data.

Starting with Angular v1.x, the following chart is sorted by the time period in ascending order, starting from as early as June 2013 for the AngularJS first vulnerability and up until June 2018. The chart shows the number of days it took to fix a vulnerability from when it was reported publicly to the project and until the time it took for the team to release a version that included the fix to which downstream users could upgrade. Low time-to-fix numbers show that the development team could respond to a security report quickly, and low time-to-release numbers show that the team could quickly spin up an official fix for upgrade by users.



As we can see, the first vulnerability reported for Angular received a fix in the code repository within a single day, but it took 74 days for that fix to be published as an official release to which users could upgrade.

With Angular v1.x releases we observed an average of 7.47 days to fix a security vulnerability, and an average of 20.5 days to publish a release that included a security fix.

An exception to the data on the previous page is one JSONP Callback Attack vulnerability that took 570 days to remediate with an actual fix, and 64 days from the fix committed to the source code repository, and until it was officially released.

React has significantly fewer vulnerabilities, with a mere three security vulnerabilities affecting the core project. Two out of the three vulnerabilities were reported and handled internally within the React team and so the time-to-fix appears as 0 days, with only one day to publish an official release. The third vulnerability is a Cross-Site Scripting security vulnerability, dating back to React v0.4 and v0.5, which took a significant amount of time-to-fix (176 days), followed by a 27-day delay to release a security fix.

2 Angular and React module ecosystems: security risks

In this section, we review the security risk of the indirect independencies for both Angular and React, and then we also review the direct dependencies, first for Angular and then for React.

The modules reviewed in this part do not represent a complete list of vulnerable React and Angular modules; some modules may have special naming conventions (such as all modules prefixed ***ng-***, ***angular-***, or ***react-*** for example) that would not appear in the pattern-based search we conducted.

The security risk of indirect dependencies

More often than not, projects based on React or Angular are generated with a scaffolding tool that provides a boilerplate with which to begin developing. With React, the developer go-to practice is to use the **create-react-app** npm package that creates a pre-configured project starting point, such as by implementing the Jest testing framework, CSS processors and other already built-in tooling. In Angular, this is made possible thanks to the **@angular/cli** npm package.

To compare the dependency health and state of the security (which reflect the level of overall security risk) for React and Angular boilerplates, we generated a sample project which resulted in rather good news - both of them include development dependencies with vulnerabilities, but neither contain any production dependency security issues.

Following are the security vulnerabilities that are introduced in your code right from the get-go when starting a project by using the Angular or React boilerplate:

Boilerplate	Vulnerable module	Indirect vulnerability	Indirect vulnerability severity	Yearly module downloads	Fixable?
Angular	jasmine-core	ReDoS	low	94,559,055	✓
Angular	useragent	ReDoS	high	70,181,373	✗
React	lodash	Prototype Pollution	high	1,005,518,049	✓
React	mdn-data	MPL-2.0 License issue	high	89,291,454	✗
React	mixin-deep	Prototype Pollution	high	328,052,052	✓
React	set-value	Prototype Pollution	high	629,781,760	✓

It's worthy to note that Angular relies on 952 dependencies, which contain a total of two vulnerabilities; React relies on 1257 dependencies, containing three vulnerabilities and one potential license compatibility issue.

With regards to licensing, we consider license compliance to be an important factor in overall dependency health, in addition to security issues, and for this reason include license checks in our scans as well. The results we received for licensing were based on the default configurations that were defined for our license policies prior to scanning. Based on those results, we can see that the generated React project has a dependency on the mdn-data package, which in turn makes use of Mozilla's copyleft license MPL-2.0. If you plan to distribute your React application with on-prem installations or other similar setups that include the mdn-data dependency, then you should check licensing requirements to make sure your project complies. Additionally, we advise ensuring your projects are scanned based on the advice you receive from your organization's unique policies, which may or may not raise flags for additional indirect dependencies of React as well.

Remediating vulnerable paths

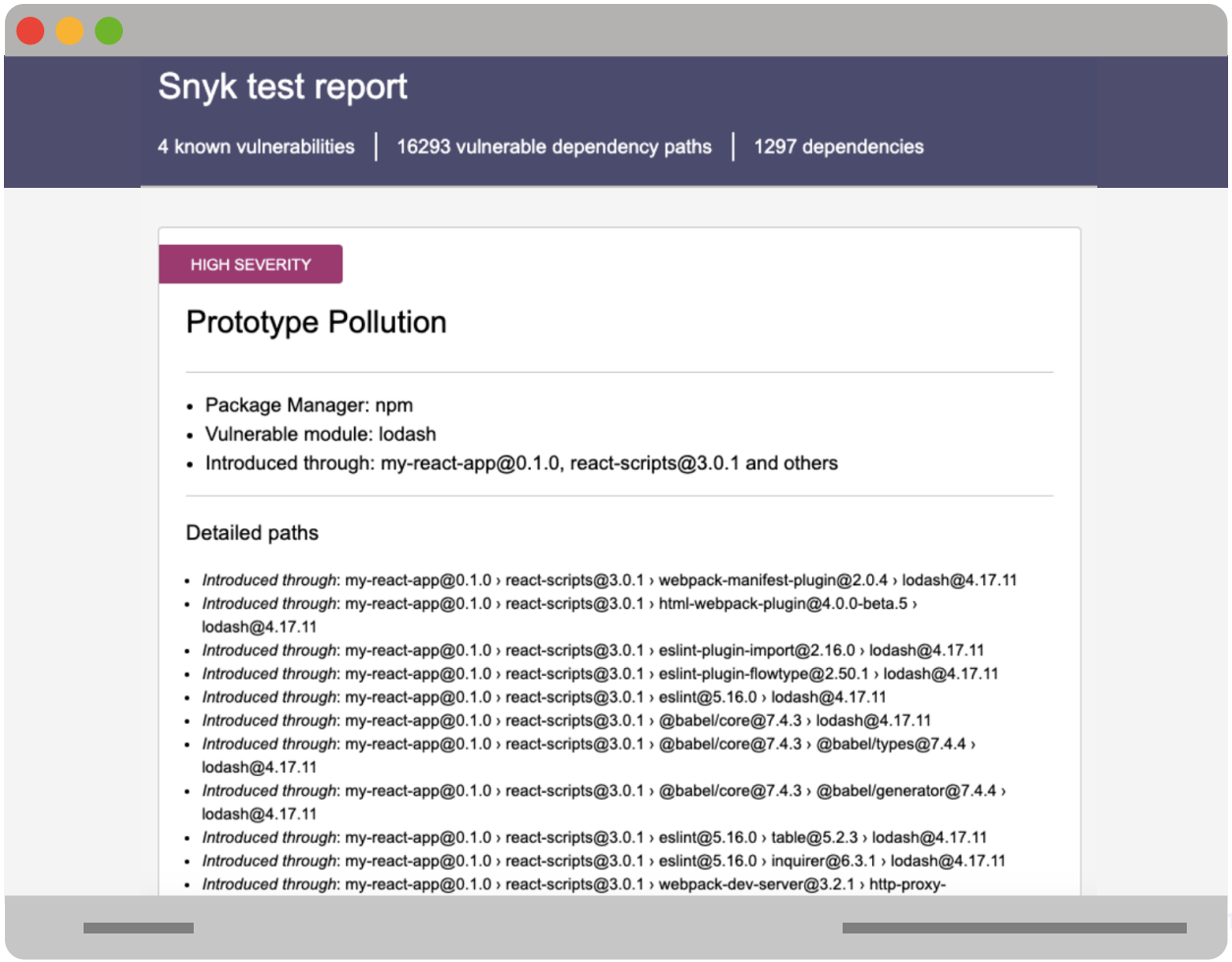
A path describes how an open source dependency is introduced to your project. For instance, let's say you have two direct dependencies called Project A and Project B. Both of these projects introduce dependency, Project C. Project C is now associated with two different paths, because it is installed by both Project A and Project B. If Project C includes vulnerabilities, a developer must consider both of these paths in order to remediate the vulnerabilities.

With React, the three vulnerabilities spread over 16,293 vulnerable paths. Remediating the vulnerability via package upgrades becomes a daunting task with so many packages in the dependency chain that require an upgrade. In contrast, both Angular's vulnerabilities are remediated easily via only two vulnerable paths.

The following image was taken from an August 2019 security scan report for a project generated with React's create-react-app npm package. The report reveals the dependency chain problem to be addressed for a single security vulnerability.

Due to the prominent usage of lodash throughout the ecosystem, its vulnerable version is ultimately used by thousands of dependency paths.

Remediating the vulnerability requires pulling new versions of lodash from every single one of the affected packages in the entire dependency chain.



Vulnerabilities in the Angular module ecosystem

In the Angular ecosystem, module vulnerabilities manifest themselves in three areas:

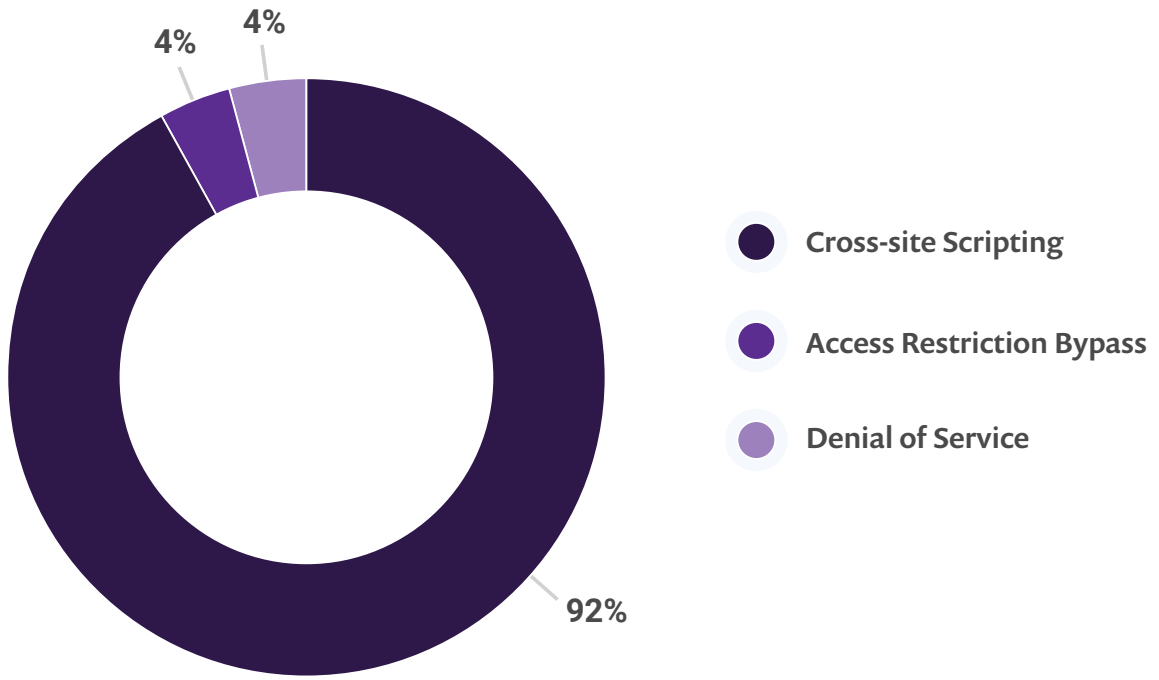
- Angular ecosystem modules
- Malicious versions of modules
- Developer tooling

When we look at the Angular module ecosystem, we can see the following modules stand out most due to their download counts and associated vulnerabilities:

Module name	Vulnerability type	Number of vulnerabilities	Vulnerability severity	Yearly module downloads	Fix exists?
ngx-bootstrap	Cross-Site Scripting (XSS)	1	medium	6,275,854	✗
ag-grid-community	Cross-Site Scripting (XSS)	1	medium	2,710,764	✓
ag-grid	Cross-Site Scripting (XSS)	3	medium	2,203,913	✓
ng-dialog	Denial of Service (DoS)	1	medium	580,674	✗
angular-gettext	Cross-Site Scripting (XSS)	1	medium	514,937	✓
angular-jwt	Access Restriction Bypass	1	medium	514,470	✓
textangular	Cross-Site Scripting (XSS)	2	medium	384,629	✓
angular-froala	Cross-Site Scripting (XSS)	1	medium	104,436	✗
angular-redactor	Cross-Site Scripting (XSS)	1	medium	64,094	✓
i18n-node-angular	Denial of Service (DoS)	1	high	4229	✓

If we line up the vulnerability types based on the number of downloads of the modules that contain them, we can clearly see that XSS vulnerabilities are at the head of the chart, as is also indicated in the *OWASP Top 10 web security risks to watch out for:*

Vulnerability type distribution by module download count



Malicious Angular modules

In total, we were able to track down three malicious versions published for the following angular modules: angular-bmap, ng-ui-library, ngx-pica.

angular-bmap is perhaps the least interesting as can be observed in its dependency health page - it features eight published versions all date back to September 2017. Nevertheless, a 0.0.9 version of angular-bmap has been published that includes malicious code that exfiltrates sensitive information related to password and credit cards from forms and sends them off to the attacker controlled URL of https://js-metrics.com/minjs.php?pl=. This malicious 0.0.9 version has been yanked off of the npm registry.

Unlike the Angular bmap module, ng-ui-library is still maintained and features over 150 versions published, seven of them in 2019 alone. However, ng-ui-library version 1.0.987 specifically has been found to contain the same malicious code that we've seen in angular-bmap. ng-ui-library still gets nearly 400-3000 downloads a month.

Joining the same malicious code that harvests credit card information is a malicious version of ngx-pica, which is an Angular v5 and Angular v7 compatible module to resize images in the browser, featuring about 800 monthly downloads.

Interestingly enough, all of these malicious versions were only found recently. They were all disclosed in June 2019, even though the malicious code was pushed in a month-old release by that time.

Angular developer tooling

As part of the module ecosystem findings, we spotted one module that is used as a general-purpose HTTP server for serving single-page application resources for projects built with Angular, React, Vue and others.

The module angular-http-server was found vulnerable to directory traversal - twice. Both vulnerable versions are a year old and there are already a half of a dozen newer versions published. Even though the module maintainer clearly states that it is not recommended to use this tool as a production-ready service, downloads for it have been ramping up this year with a recorded downloads count of 20,670 in May 2019.

Due to the growing adoption of this Angular HTTP server developer tool we should point out that there's a public exploit for this vulnerability.

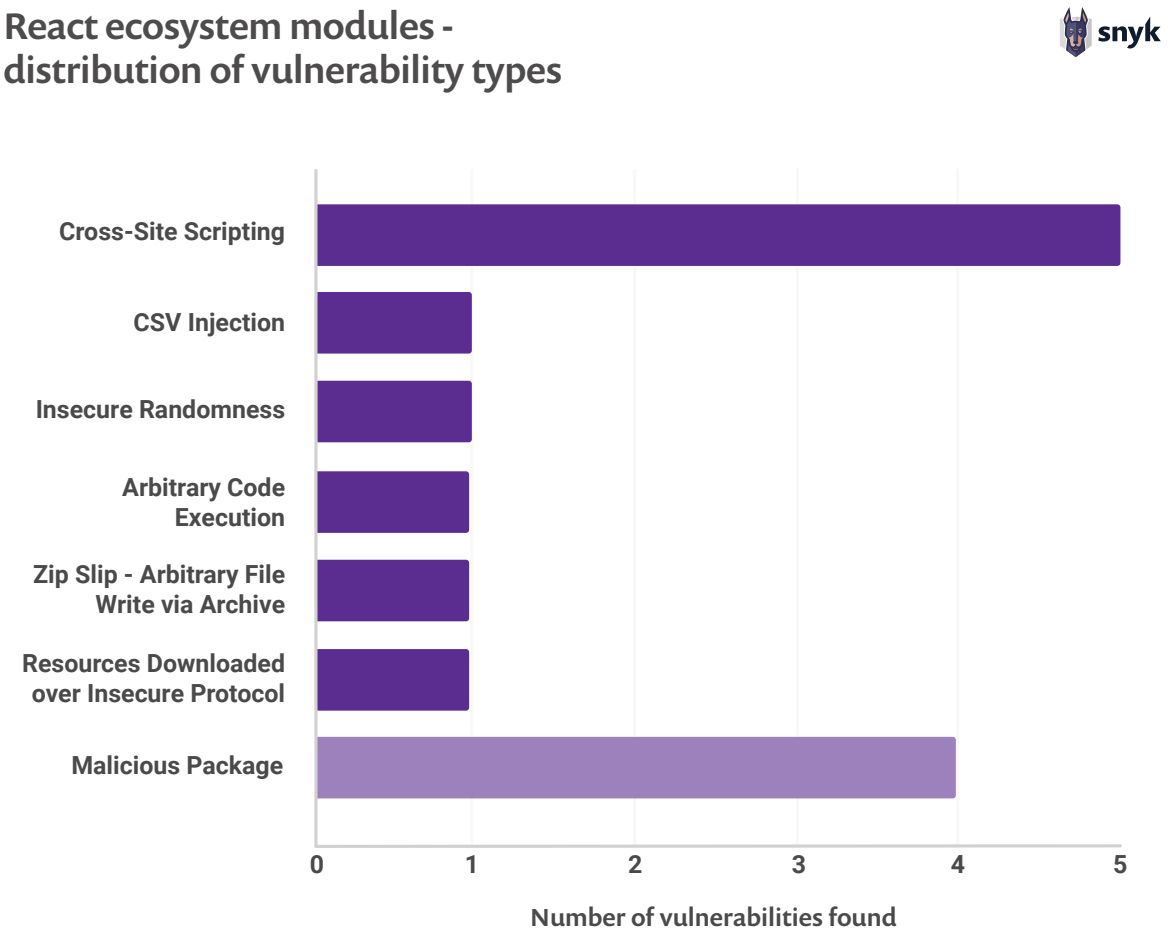
Vulnerabilities in the React module ecosystem

As with Angular, we found that the React ecosystem includes several malicious modules published at some point. The following represents the distribution of security vulnerability types and their counts across all vulnerable modules that we found, highlighting specifically four malicious packages react-datepicker-plus, react-dates-sc, awesome_react_utility, react-server-native.

All four malicious modules have the same malicious code that harvests credit card and other sensitive information; this attack compromised modules on the React ecosystem as well.

This goes further to emphasize that as a maintainer of an open source project it is critical to enable multi-factor authentication such as 2FA support that the npm package registry supports, to avoid putting your users at risk of someone else compromising your account and publishing malicious versions of your package.

If you haven't done so yet, we urge you to enable 2FA on your npmjs.org developer account and follow other npm security best practices.



Notable vulnerable modules that we tracked in React's ecosystem:

- A high severity XSS vulnerability in react-marked-markdown which has no fix available, but this react component wrapper around the marked JavaScript markdown library still gets thousands of downloads, totaling 65,790 in the past 12 months.
- For the preact users among you, the preact-render-to-string library is vulnerable to Cross-Site Scripting in all versions prior to 3.7.2. This library is growing in usage across the last 12 months and totaling in 3,228,049 downloads for this time-frame.
- If you're doing tooltips in your frontend React application you might be one of the users of react-tooltip which received just shy of one million downloads (994,903) in July 2019 alone. This library however is vulnerable to Cross-Site Scripting attacks for all versions prior to 3.8.1 as was disclosed in September 2018.
- If you are working with SVGs a lot, good chances you are using react-svg which features 1,446,442 downloads in the past 12 months. In April 2018 a high severity Cross-Site Scripting vulnerability was disclosed by security researcher Ron Perris affecting all versions prior to 2.2.18.
- A CSV Injection vulnerability in mui-datatables disclosed in March 2019. This react library provides a table data related UI component based on the material ui framework and features more than 350,000 downloads in the past 12 months.

When we track all the vulnerable React modules we found, we count eight security vulnerabilities over the last three years with two in 2017, six in 2018 and two up until August 2019. This calls for responsible usage of open source and making sure you find and fix vulnerabilities as quickly as possible.

Spotlight: Next.js security vulnerabilities

Next.js is the popular React framework delivered from ZEIT, empowering web developers to harness their knowledge of React in order to build SEO-friendly web applications, Server-side rendering applications, Progress Web Applications (PWA) and even Electron-based applications, all based on the Next.js framework.

Next.js continues to gain developer adoption, with 8,414,925 downloads over the past 12 months. As the project continues to grow it becomes increasingly important to take a look at its security status.

We tracked three high Directory Traversal vulnerabilities, and two medium severity Cross-Site Scripting vulnerabilities impacting the Next.js React framework during the course of 2017 through 2018.

We should also point out that the ZEIT Security team swiftly addressed all five security vulnerabilities and provided a fix through an upgrade path for the Next.js framework within a week's time.

Overall, ZEIT employs strong security practices that should be replicated by other open source projects.

Particularly notable includes:

- The team responds quickly to security disclosures by releasing timely security fixes. This translates into a small window during which time there is an actual security risk; ZEIT provides users with an upgrade path so they can quickly mitigate the vulnerability.
- To avoid security regressions the team has written security unit tests to ensure that security mistakes do not repeat themselves.
- Release notes clearly communicate security-related information, its impact and any steps users are required to follow in order to stay up-to-date with a security fix.
- The project maintains a mailing-list dedicated to security reports, a responsible disclosure policy and a dedicated email contact for reporting issues.

ZEIT and their management of the Next.js framework is a great example of good open source security policies; ZEIT takes matters seriously and demonstrates a true commitment to the overall security of their users with policies and actions that should be adopted by others.

3 Angular and React projects: overall security posture

In this section, we explore both the Angular and the React project security postures. This includes secure coding conventions, built-in in secure capabilities, responsible disclosure policies, and dedicated security documentation for the project.

The following table lays out a few of the security components we found to be essential for best-practice maintenance of any open source package, and an indication of how Angular and React manage said components (if at all).

Security policy components

Item	Angular	React
Security page	✅ https://angular.io/guide/security	❌ React's website (https://reactjs.org) does not mention any security guidelines, except for the dangerouslySetInnerHTML function reference in the DOM Elements section of the API Reference documentation.
Security contact	✅ security@angular.io	❌ No security contact
Responsible disclosure policy	✅ Backed by the internal security teams at Google and based on Google security philosophy. Reference: https://www.google.com/about/appsecurity/	❌ No responsible disclosure policy
Examples of vulnerable projects	✅ https://angular.io/generated/live-examples/security/stackblitz.html	❌ No references to any examples of vulnerable projects
Built-in sanitization	✅ DomSanitizer provides a built-in sanitization function for untrusted values. Reference: https://angular.io/api/platform-browser/DomSanitizer#sanitize	❌ Potentially malicious input sanitization is at the users' discretion to be implemented via 3rd-party libraries, such as DOMPurify. Reference: https://github.com/cure53/DOMPurify
Content Security Policy (CSP)	✅ CSP compatibility for Angular v1.x directives. Reference: https://docs.angularjs.org/api/ng/directive/ngCsp	⛔ Not relevant for React
Cross-Site Request Forgery (CSRF)	✅ CSRF built-in support through Angular's HttpClient service. Reference: https://angular.io/guide/http and https://docs.angularjs.org/api/ng/service/$http	⛔ Not relevant for React as a view library. This is up to the developers to handle using custom code or community modules.

Secure coding

Angular secure coding practices

Angular v2 and later, have a completely different architecture than Angular v1, such as unidirectional data binding. What's more, the v2 and later versions have left automatic data interpolation via watchers behind, as well as other techniques that were often the cause for many of the Angular v1 security vulnerabilities.

Ahead of Time (AoT) compilation mitigates issues such as Angular templating expression injection and allows for build-time security instead of run-time security. However, dynamically interpolating templates on the client-side still leaves the door open for security vulnerabilities in the form of Angular code injection. In their own best practices documentation, Angular clearly emphasize that this dynamic interpolating is highly unadvisable. With respect to Angular's documentation, these are highly discouraged as Angular's best practices clearly point out.

To mitigate Cross-Site Scripting vulnerabilities, Angular employs by default context-aware output encoding, or malicious code sanitization. Moreover, method naming conventions are much better understood, in terms of their impact, if a developer consciously chooses to use them, as opposed to earlier Angular versions, namely Angular v1.x.

Methods such as **bypassSecurityTrustHtml(value)** or **bypassSecurityTrustUrl()** implicitly convey the dangers of using them to insert data into the DOM. Moreso, Angular provides a built-in DomSanitizer to explicitly sanitize values.

React secure coding practices

React by default encodes almost all data values when creating DOM elements. To provide users with an escape hatch to insert HTML content into the DOM, React is equipped with the eloquently-named function **dangerouslySetInnerHTML()**, clearly conveying the dangers of using it.

Contexts that are unattended by the React security model and are handled by the users include creating:

- HTML anchor (link) elements with user-provided input as the source for the href attribute value. This mostly applies to versions prior to the recently released React v16.9 which mitigates ***javascript:***-based URLs in ***href*** attribute values and other contexts such as form actions, iFrame sources, and others.
- React components from user-provided input

React's server-side rendering could potentially introduce XSS vulnerabilities if malicious user input is injected as-is to a JavaScript context without being properly encoded or sanitized.

HTTP security

Starting with version 1.2, Angular v1.x release branches have introduced compatibility support for Content Security Policy (CSP) which is necessary due to the use of **eval()** and **Function()** methodology to interpolate expressions.

Cross-Site Request Forgery (CSRF) enables web applications to trust the origin of a request. In newer Angular versions, CSRF support mechanism is built-in to the HTTP client with the **@angular/common/http** module. In Angular v1.x versions similar capability is supported through the **$http** provider.

Unlike Angular, React doesn't include an HTTP client and as such, it is unable to provide CSRF support out-of-the-box. As React aims to be a minimalistic view library, handling this concern is up to the developer, using custom code or community-powered modules.

4

Security vulnerabilities found in other frontend ecosystem projects

After reviewing Angular and React as major JavaScript frameworks, we'll take a brief review of selected JavaScript and CSS frameworks: Vue.js, jQuery and Bootstrap.

Vue.js security

The Vue.js frontend framework attracts no less popularity from web developers than its counterparts React or Angular, and was downloaded 40,054,897 times in the past 12 months and featured as the second most starred project on GitHub with more than 145,000 stars.

We tracked four vulnerabilities in total for Vue.js core project, three medium and one low regular expressions denial of service vulnerability, spanning from December 2017 to August 2018 with a shared Cross-Site Scripting vulnerability that was found in React's server-side rendering with react-dom component.

As for Vue's module ecosystem, we found the following are worth noting:

- bootstrap-vue has 4,620,136 downloads recorded for the past 12 months and includes a high severity Cross-Site Scripting vulnerability that was disclosed in January 2019 and affects all versions prior to <2.0.0-rc.12.
- vue-backbone had a malicious version published, associated with malicious package attempts that we mentioned earlier across Angular and React ecosystem modules. vue-backbone was downloaded 11,658 in the past 12 months.

Bootstrap security

Bootstrap is a component library that leverages CSS and JavaScript to enable developers to build websites and has a strong historical affiliation with jQuery through plugins that enhance the frameowkr's core capabilities.

Bootstrap is the third-most starred project in GitHub with more than 130,000 stars, and 79,185,409 downloads in the past 12 months from the npm package registry. Modern web application frameworks like React have even extended Bootstrap by packaging it for React based web development with projects like reactstrap and react-bootstrap which receive about 20 million downloads each in the past 12 months.

As we look at known security issues for the Bootstrap project, we can track a total of seven Cross-Site Scripting vulnerabilities, three of which were disclosed in 2019 for recent Bootstrap v3 versions, as well as three security vulnerabilities disclosed in 2018, one of which affects the newer 4.x Bootstrap release.

All vulnerabilities have security fixes and provide an upgrade path for users to remediate the risks.

We were also able to track several modules in the Bootstrap ecosystem that are vulnerable. Most notable are:

- bootstrap-markdown with more than 300,000 downloads in the past 12 months despite having an unfixed Cross-Site Scripting vulnerability affecting all versions
- Vue.js developers using bootstrap-vuejs had their usage of this module contributed to 4,620,136 downloads in the past 12 months and worth to note that a recently disclosed high severity Cross-Site Scripting vulnerability affects all versions prior to bootstrap-vue 2.0.0-rc.12 which only a February 2019 release had addressed.
- bootstrap-select featured 2,159,450 downloads in the past 12 months and has a high severity Cross-Site Scripting vulnerability that the Snyk research team surfaced thanks to its threat intelligence system.

jQuery security

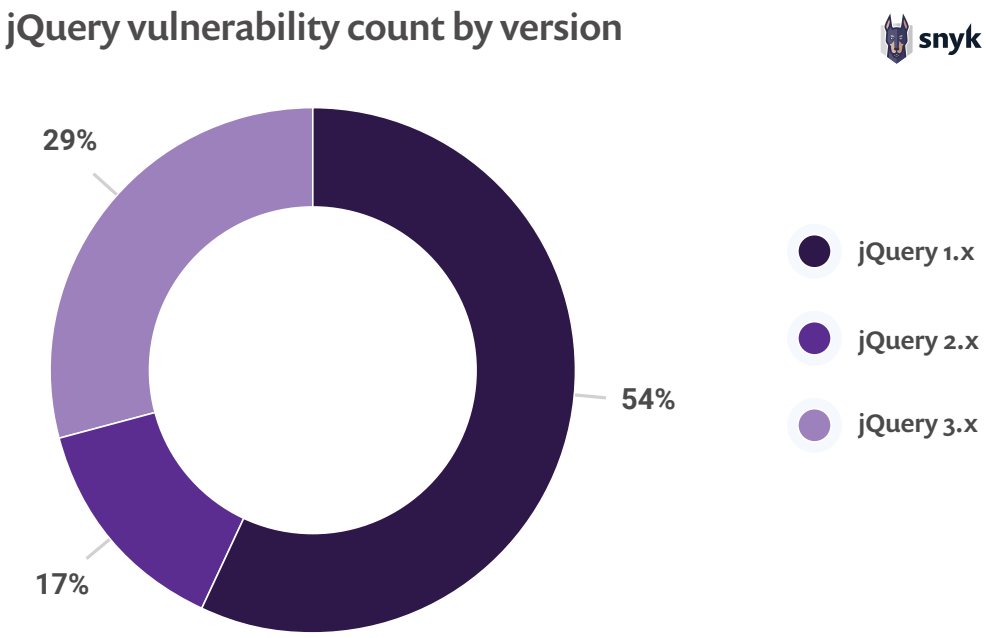
jQuery took web development by storm a decade ago but since then web development have been revolutionized further with single page application technologies such as Angular, and React. That said, according to W3Techs which regularly run surveys and report on web technology usage jQuery is being used within 73% of websites they scanned in August 2019.

A Snyk study from 2017 further amplifies this when it reported that 77% of sites use at least one vulnerable JavaScript library and pointed out jQuery was detected in 79% of the top 5,000 URLs from Alexa. If you're still not convinced, npm's downloads for the jQuery npm module account to 120,641,977 for the last 12 months alone.

In total, we tracked six security vulnerabilities affecting jQuery across all of its releases to date, four of which are medium severity Cross-Site Scripting vulnerabilities, one is a medium severity Prototype Pollution vulnerability, and lastly, one is a low Denial of Service vulnerability. If you're not using jQuery 3.4.0 and above which was released only recently, on 10th of Apr, 2019, then you are using vulnerable jQuery versions.

Since jQuery is usually found in web applications as a legacy component it is important to also understand its version usage patterns and their state of security.

W3Techs reports that of all websites using jQuery, it's 1.x release is dominating with 83.4% of share and version 2 and 3 lag far behind with roughly 8% of all jQuery usage. When looking at the known security vulnerabilities and map them out to jQuery versions we found that four medium severity Cross-Site Scripting vulnerabilities are affecting jQuery v1 which is potentially concerning considering the 83.4% market share for anybody not employing software composition analysis to find and fix vulnerabilities in their open source components.



Many websites and web applications will further make use of jQuery libraries to extend the capabilities of jQuery and will turn to community-powered libraries to do so.

We found 13 vulnerable jQuery libraries as provided in the following table and offer the following observations:

- Three jQuery libraries are malicious versions of open source community modules. As we can't account for the downloads of the actual vulnerable versions since this isn't available from the npm registry, we should call out jquery.js which is a malicious package and accounted for 5,444 downloads in the past 12 months.
- jQuery libraries jquery-mobile, jquery-file-upload and jquery-colorbox account to more than 340,000 downloads in the past 12 months, despite including Arbitrary Code Execution and Cross-Site Scripting security vulnerabilities and not having any upgrade path to remediate them.

jQuery library name	Vulnerability type	Disclosure date	Vulnerability severity	Yearly module downloads	Fix exists?
jquery-airload	Malicious Package	2019-08-06	high	322	n/a
jquery.json-viewer	Cross-Site Scripting	2019-07-03	medium	17,898	✓
github-jquery-widgets	Malicious Package	2019-06-07	high	232	n/a
jquery-mobile	Cross-Site Scripting	2019-05-04	medium	54,991	✗
jquery-file-upload	Arbitrary Code Execution	2018-11-02	low	19,442	✗
jquery.terminal	Cross-Site Scripting	2018-08-19	medium	79,982	✓
jquery.csssr.validation	Regular Expression Denial of Service (ReDoS)	2018-02-13	high	3,069	✓
jquery-colorbox	Cross-Site Scripting	2017-11-14	medium	268,513	✗
jquery.js	Malicious Package	2017-08-02	high	5,444	n/a
jquery-ui	Cross-Site Scripting	2016-07-21	high	8,934,683	✓
jquery-ujs	Cross-Site Request Forgery (CSRF)	2015-06-24	medium	5,763,710	✓
jquery-migrate	Cross-Site Scripting	2013-04-18	medium	1,831,735	✓
jquery-ui	Cross-Site Request Forgery (CSRF)	2012-11-26	medium	8,934,683	✓
jquery-mobile	Cross-Site Scripting	2012-08-01	medium	54,991	✓
jquery-ui	Cross-Site Scripting	2010-09-02	medium	8,934,683	✓

Malicious packages have no fix information.

snyk

Use open source. Stay secure.

Twitter: @snyksec

Web: https://snyk.io

Office info

London

1 Mark Square

London EC2A 4EG

Tel Aviv

40 Yavne st., first floor

Boston

WeWork 9th Floor

501 Boylston St

Boston, MA 02116

Report author: Liran Tal (@liran_tal)

Thank you to our friends, and community leaders, Chris Heilmann (@codepo8), Ron Perris (@ronperris), Daniel Rufde (@danielrufde), Stephen Fluin (@stephenfluin), Sebastian Markbåge (@sebmarkbage) and Rachel Cheyfitz (@spinningrachel) who took the time to provide technical reviews of this report.

Report design: Growth Labs (@GrowthLabsMKTG)