Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Trusted Types integration into open source frameworks and libraries

Masters Thesis

2021
Emanuel Tesař, Bc.

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Trusted Types integration into open source frameworks and libraries

Masters Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	FMFI.KAI - Department of Applied Informatics
Supervisor:	RNDr. Peter Borovanský, PhD.

Bratislava, 2021
Emanuel Tesař, Bc.

Acknowledgments:

Abstrakt

Kľúčové slová: Trusted Types, Web APIs

Abstract

Trusted Types is a modern Web API which aims to reduce DOM XSS attack surface in web applications. They give you the tools to write and maintain applications free of DOM XSS vulnerabilities by making the dangerous web API functions secure by default. Currently, they are supported in Chrome, Edge and Opera.

Integrating Trusted Types in web application requires code changes. The problem is when these changes need to be made in third party code which you can't easily modify. Trusted Types support in open source projects is gradually improving and our plan is to analyze these integrations and implement one or more of the challenging ones.

Keywords: Trusted Types, Web APIs

Contents

List of Figures

List of Tables

Concepts and definitions

1. **DOM source and sink** - In the context of XSS, a DOM source is the location from which untrusted data is taken by the application (which can be controlled by user input) and passed on to the sink (e.g. location, cookies). Sinks are the places where untrusted data coming from the sources is actually getting executed resulting in DOM XSS (e.g. eval, element.innerHTML) [1].
2. **Hot reload** - Hot reload just displays the code changes according to new code changes without restarting the app.
3. **DOM XSS** - DOM Based XSS is an XSS attack wherein the attack payload is executed as a result of modifying the DOM environment in the victim's browser used by the original client side script, so that the client side code runs in an "unexpected" manner [11].
4. **CSP** - Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including XSS and data injection attacks [5].

Chapter 1

Introduction

1.1 Trusted Types

Trusted Types is a relatively modern web API designed by Google based on a long history of mitigating XSS [3].

It is a browser security feature that limits access to dangerous DOM APIs to protect against DOM XSS (3). Trusted Types provide type guarantees to all frontend code by enforcing security type checks directly in the web browser. They are delivered through a CSP (4) header and have a report-only mode that does not change application behavior and an enforcement mode that may cause user-observable breakages [7].

When enforced, Trusted Types block dangerous injection sinks (1) from being called with values that have not passed through a Trusted Types policy [7].

There are many other resources which can be used to explore Trusted Types in depth [9].

1.2 Web frameworks and libraries

Web frameworks and libraries is a software that is designed to support development of web applications and services related to web applications. Frameworks try to solve common problems faced in web development, such as building and reusing user interfaces, managing state, performing API requests and enforcing the best security practices.

1.2.1 Next.js

Next.js is one of the most popular frameworks for building web applications. The framework is build upon React library which is the most used library as of 2021 [10]. There are many reasons for why Next.js is among the most popular frameworks. Notably, it is opinionated about how the application backend should work. This design choice allows Next.js to decide the best possible way to build a particular web page.

The important fact to take away is that Next.js is not only about the client side (the DOM), but it can also handle server side rendering, data processing, API requests which open up different means of attacks such as reflected XSS, SQL injections and more.

Next.js is a complex framework when looking at the source code. As of Oct 6, 2021 it consists of more then 200,000 lines of code. Looking at each file or have a complete understanding of the entire codebase is impossible for external contributor.

1.2.2 Create React App

Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration. It is a CLI which creates and configures minimal React application. This is ideal for starting new projects as the configuration is abstracted away from the user.

The source code of this tool is not dependant on React directly. It only focuses on generating the project files from the template and then it installs React and other necessary dependencies.

Chapter 2

Trusted Types integration

Integrating Trusted Types to target applications and libraries is a complex task that requires good knowledge of the target inner workings. There is a small sample of already implemented integrations, which show that the necessary code changes are relatively small [8]. Each integration is different, but on a high level they follow the same steps:

1. Locate all of the DOM sinks (1)

2. Find the most suitable workaround for every sink found

3. Merging the integration and releasing a new version of the target

2.1 Locate all of the DOM sinks

Locating the sinks of the integration target is important for scoping the integration effort and the implementation afterwards. This task is complex since the problem of "locating all of the sinks in a codebase" generally cannot be proven. This is because JavaScript is a dynamic language and it is possible to access and set the sink value dynamically (using the property element access).

Gladly, there are a few methods and tools, which can help to catch most of the sinks:

1. Static search through the codebase

2. Static code analyzers

3. Using the target in an real world application

2.1.1 Static search through the codebase

This method is the simplest one and arguably the least effective. Statically searching for sinks through codebase produces many false positives and it is also possible to miss some. The output of the static search is usually cluttered with less important violations in tests and built tools, which you need to manually exclude from the search. Nevertheless, this method is easy to reason about, fast to iterate and the final search results can provide a good estimate of the integration scope. This works well in practice since there is an assumption that the target source code is not malicious and doesn't actively use code patterns which are trying to hide or cause an XSS vulnerability.

There is not an actively maintained static search tool, but it is relatively easy to build a script, which uses the existing tools such as *grep* with a list of already known sinks [14].

2.1.2 Static code analyzers

A better approach for locating the DOM sinks statically, is to search through the AST of the code. The quality of the output produced depends on the quality of the AST information. When the codebase is using TypeScript, the AST information is generally richer compared to codebases using JavaScript. One can then build and use tools like Tsec [18] which uses the compiler to parse the code and create the AST to ultimately find the sinks in a more reliable manner.

These tools have an advantage that they can be used to maintain the Trusted Types compatibility even after the violations are fixed. For example, they can be run in CI pipeline or their output can be leveraged by linters and other language plugin tools and the errors can be shown to the developers directly in their IDE [19].

2.1.3 Using the target in a real world application

While static search through the codebase is fast and locates most of the sinks, it usually doesn't catch them all. The author of the integration should verify the integration on the applications using the target as a dependency. Also, having good test coverage in the application code greatly increases the chance that the integration is implemented correctly.

That said, finding a suitable application might be challenging and it may not be an ideal way to test all edge cases. For example, when implementing an integration to React, one can find lot of real world applications using React, but most of them are already XSS free so you can only verify if the integration doesn't break these clients. What you want is to also test how the integration works when used with applications that uses the DOM sinks and now has to produce Trusted Types values. Another test vector is simulating an attacker by constructing malicious payloads and expect the site to break due to Trusted Types violations. These are done best by creating an application from scratch by the integration author.

2.2 Merging the integration and releasing a new version of the target

Merging the integration and releasing a new version of the target is undoubtedly an important part of Trusted Types integration. However, this can take long time for multiple reasons, which are described in separate sections:

1. Reasoning about the integration, section 2.2.1
2. Trusted Types compatibility in dependencies, section 2.2.2
3. Knowledge of the integration author, section 2.2.3

2.2.1 Reasoning about the integration

Proving that the integration is correct is often hard or impossible. However, proving implementation correctness in general is often impossible, so this is not that big of an issue. One can look at the integration empirically instead - if the integration is tested on multiple projects and they all seem to work it is safer to assume that the integration is correct. This is especially true when the integration is tested by large scale organizations on lot of their services.

That being said, Trusted Types integration is a breaking change and targets should make this change opt-in or release a new version with breaking changes [4]. If the integration is turned on by default when Trusted Types are available in the browser, the target is risking breaking existing applications [6].

Releasing an opt-in change might be problematic if the target is not configurable. This can result in the integration being put behind a feature flag which hurts the adoption [13].

2.2.2 Trusted Types compatibility in dependencies

Implementing the integration in large targets, which consists of many dependencies can bring a lot of overhead, because to be fully Trusted Types compliant one must ensure that all of the dependencies are Trusted Types complaint as well.

When there is non compliant dependency, one has multiple options:

- Implement the integration for the dependency - This option has all the problems already mentioned and can possible lead to even more integrations.
- Find an alternative dependency - This option is often impossible as a viable alternative might not exist.
- Use Trusted Types default policy - This option is applicable only for applications, is more complex and the application is not fully Trusted Types compliant.

2.2.3 Knowledge of the integration author

It is likely that the integration author is either a security engineer familiar with Trusted Types or a part of the engineering team of the target familiar with the target code. This means that the author might not be fully familiar with both of the technologies, which results in both:

- Increased propability of a bug in implementation
- Harder and longer review, since the reviewers are likely in a similar position

2.3 Recommended setup for libraries and frameworks

The following section describes the recommended setup on how to prepare for the implementation of the integration based on our experience. We also assume, the target code is put under version control, such as *git*.

This section should give the reader a high level overview of how the integration setup is performed, not a definitive guide. We also use this section as a guideline when implementing the integrations. The general structure is very simple:

1. Clone the repository of the library locally
2. Create an application which uses your local version as a dependency

2.3.1 Clone the repository of the library locally

Having the repository cloned locally has an advantage that you can keep your integration up to date with the latest changes. It is also probably required to submit the integration once implemented.

2.3.2 Create an application which uses your local version as a dependency

This is the more interesting part, but it is also very project specific. For web development, the dependencies are usually managed with package managers such as *npm* or *yarn*. Both of these allow you to configure local dependencies. The only thing that the integration author needs to do is to be able to build the target as a dependency for the application.

Afterwards, one can implement the integration and verify assumptions on the application simultaneously, which can greatly speed up the integration. Also, after the integration is implemented, one can have a present the application code as an empiric proof of the integration correctness.

Chapter 3

Trusted Types integration into Next.js

Next.js (described in 1.2.1) was the initial choice for Trusted Types integration, because of a large impact this would have and also because the framework seemed interested and open to the integration of Trusted Types for a longer time [17].

The integration setup followed chapter 2. There were changes needed in both the application and Next.js code and we ended up with a working version of a blank application running in dev mode with Trusted Types aware Next.js as a dependency. However, we decided not to pursue this project no more and shifted focus to a different target.

3.1 Fixing violations reported by Tsec

Tsec found 8 violations [16] inside Next.js sources. Out of these 7 were indeed Trusted Types violations that needed to be fixed. Some of these could be fixed simply on a type system level since they expected a value from the user. The others needed to be allowed explicitely by using a policy. The implementation for this can be found on github [15].

Since our utmost goal was to find the sinks and create a prototype for the integration we wrapped all of these values in Trusted Types objects. This was to be revisited in the future.

3.2 Fixing the dev mode of an example application

Fixing the violations found by Tsec was not enough and the application still wouldn't launch with Trusted Types enforced. The reason for this hasn't been clear, but it was probably a webpack plugin used in one of the Next.js dependencies. The violating code relied on eval, which threw an error when Trusted Types were enforced. The workaround for this was to use a default policy and allow all eval calls.

Apart from this, there was another violation that was triggered when the application wanted to hot reload (2). Tsec wasn't able to catch this, since the violation came from a JavaScript file where the AST information was limited.

Chapter 4

Trusted Types integration into Create React App

Create React App (CRA) (described in 1.2.2) was a second choice for integrating Trusted Types. After taking a look in the implementation of CRA we found that this tool doesn't depend on React directly, but only installs it after creating the project template. Our goal was to make sure the generated project by CRA is Trusted Types compatible.

4.1 Using Trusted Types compatible version of React

To accomplish this, we would need to change the implementation of CRA to install Trusted Types compatible version of React. Unfortunately, such version of React is implemented under a feature flag which needs to be turned on at build time. The published version of React contains the already built files with the feature flag turned off. This means that if you want to use the Trusted Types compatible version of React you have to clone the React repository from sources, turn on the feature flag and then build the framework. You can then use this version of React as a dependency in your project.

Implementing this is non trivial, since it requires knowledge about React. More importantly though, this is harder to maintain for the application authors since you need to keep up with the new releases of React manually.

4.2 Using Trusted Types compliant version of Webpack

CRA internally uses webpack version 5 to provide fast development experience with convenient features, such as hot reload. For production it handles minification and other optimizations. Some of these features, especially hot reload breaks under Trusted Types in enforcement mode. Gladly, webpack can be configured to be Trusted Types compliant by a small configuration change [12].

The problem with CRA is that the webpack configuration is abstracted away from the user. This means that there is not a simple way how to change the configuration. One can find the webpack configuration in the downloaded dependencies and change it there directly, but this won't really work since those changes will be removed by the package manager when modifying the dependencies. In general, one should never modify the contents of dependencies and treat it only as an output of a package manager.

The optimal solution as of now is to use a package which allows you to spy on modules and change what they return. This pattern is often used when mocking or spying in unit tests. However, it is also a suitable workaround for this case [2]. CRA team is reluctant to change this, since the hidden configuration is a good default for the majority of users.

Another option is to use the *eject* command from CRA. This is a one way operation and it essentially unwraps all of the hidden configuration and creates these files in your project. You can then simply edit the webpack configuration which is now part of your project. This is not recommended and it should be used only when all other attempts fail. The generated webpack configuration is pretty complex and updating it in the future might not be trivial.

Bibliography

[1] Anirudh Anand. Xss dom source and sink definition, February 2019. https://blog.0daylabs.com/2019/02/24/learning-DomXSS-with-DomGoat/#source.

[2] ashvin777. Modify webpack configuration in cra (issue comment). https://github.com/facebook/create-react-app/issues/10307#issuecomment-898889701.

[3] Multiple authors. Trusted types design history, September 2019. https://github.com/w3c/webappsec-trusted-types/wiki/design-history/8a57f5cb1a7773cfa512943e9b7560813d61a83f#why-client-side-trusted-types.

[4] cure53. Dompurify 2.0.0 release. https://github.com/cure53/DOMPurify/releases/tag/2.0.0.

[5] MDN Web Docs. Content security policy (csp). https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP.

[6] Uzlopak (DOMPurify issue #361). Trusted types breaking applications using dompurify in chrome 77. https://github.com/cure53/DOMPurify/issues/361.

[7] Krzysztof Kotowicz. Trusted types background, December 2021. https://github.com/w3c/webappsec-trusted-types/wiki/Effects-of-deploying-Trusted-Types-on-browser-extension-developers/c110df6329c46363c65e3c3de4ef24c5fbcecee9#background.

[8] Krzysztof Kotowicz. Trusted types integrations, November 2021. https://github.com/w3c/webappsec-trusted-types/wiki/Integrations/3e5ff5b7613f18c15a1895129fec063d5491ff83.

[9] Krzysztof Kotowicz. Trusted types resources, July 2021. https://github.com/w3c/webappsec-trusted-types/wiki/Resources/22dcd3a4cc55fae2e1706b47f9d26feb2aacaefb.

[10] Stack overflow insights. Web frameworks statistics for 2021, 2021. `https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks`.

[11] OWASP. Dom based xss. `https://owasp.org/www-community/attacks/DOM_Based_XSS`.

[12] Webpack team. Webpack trusted types documentation. `https://webpack.js.org/configuration/output/#outputtrustedtypes`.

[13] Emanuel Tesar. Add trusted types to react on client side pr. `https://github.com/facebook/react/pull/16157`.

[14] Emanuel Tesar. Static xss code scanner. `https://github.com/Siegrift/xss-sink-finder/tree/5c98c3e80e16c15bda2ccc0609288cc536601a83`.

[15] Emanuel Tesar. Fix tsec violations in next.js, October 2021. `https://github.com/Siegrift/next.js/commit/a45df82d5e86f7cbaa9f0e327f1b4ea0e956fbdf`.

[16] Emanuel Tesar. Tsec output for next.js, October 2021. `https://github.com/Siegrift/next.js/blob/a45df82d5e86f7cbaa9f0e327f1b4ea0e956fbdf/tsec-packages-next-output`.

[17] TyMick. Add trusted types policy (next.js pr), 2020. `https://github.com/vercel/next.js/pull/13509`.

[18] Google (unofficial product). Tsec. `https://github.com/google/tsec/commit/a57933da74af5aef5fd2342f207b03c4fe305003`.

[19] Google (unofficial product). Tsec language service plugin documentation. `https://github.com/google/tsec/tree/a57933da74af5aef5fd2342f207b03c4fe305003#language-service-plugin`.