Trusted Types - mid 2021 report

Krzysztof Kotowicz, 2021-07-06

Takeaways

- DOM-based Cross-Site Scripting (DOM XSS) is one of the **most prevalent web application security vulnerabilities**. Trusted Types (TT) introduced a browser API which aims to comprehensively prevent it and make it impossible to create DOM XSS issues in client-side code.
- Trusted Types complement static analysis by providing runtime guarantees about the absence of uncontrolled data flows in client-side code. Our analysis of the vulnerabilities reported to Google VRP shows that Trusted Types could **effectively prevent at least 61% of DOM XSS-es** missed by our static analysis pipeline.
- Trusted Types **simplify security code reviews** by making the risky injection sink functions secure by default, and reducing the review surface from the entire web application and over 60 different injection sinks[1] to a single function.
- We've successfully migrated **130 services** at Google to Trusted Types. We've helped enforce Trusted Types in Visual Studio Code, one of the largest TypeScript codebases. Other deployments of Trusted Types in major web applications are currently in progress (Facebook).
- Trusted Types are **supported in several popular frameworks and libraries** including Angular, React (with a feature flag), Lit, Karma, and Webpack. Enforcing Trusted Types in applications built on top of these frameworks is now relatively simple; in some cases no application-level code changes are required.
- Trusted Types **improve the security** of applications. To date, we have observed zero DOM XSS in Google applications migrated to Trusted Types. Around ¾ of code locations incompatible with Trusted Types can be addressed by mechanical changes to rewrite the code to be inherently safe and not use a DOM XSS sink. Future web APIs (e.g. the HTML Sanitizer API) may tip the scale even further. Some web applications may be rewritten to not use any DOM XSS sinks at all.
- Based on the feedback from the early adopters, we **simplified the API** and open sourced tools and libraries that facilitate migration.
- To date, Trusted Types migration is still **largely a manual task**. Active work is needed on the migration tooling, libraries and guides, especially for the open source JS ecosystem.

[1] DOM XSS can be caused by passing attacker-controlled data to one of around 60 separate sink functions. The exact number depends on the browser engine and version.

DOM XSS

Cross Site Scripting (XSS) remains one of the top web application vulnerabilities. OWASP estimates it's the second most prevalent issue in OWASP Top 10. HackerOne notes the XSS has grown 23% YoY in 2020. This is recognized by the web platform, which offers Content Security Policy (CSP) to help mitigate XSS exploitation.

With the rise of Single Page Applications and the push towards rich client-side code, DOM XSS is becoming a comparatively larger source of bugs than traditional root causes of XSS - server-side injections. We see this confirmed in external research and in our data (DOM XSS consistently accounts for >50% XSSes reported to Google VRP).

Unfortunately, while CSP is relatively effective in combating server-side injections, it is not well-suited to address DOM XSS, where the data flow that ends up vulnerable is often intended by the application owner or one of the application's dependencies. For example, research shows that dynamic code evaluation (`eval`) cannot be practically eradicated. In the tested web application cohort, 70% web applications used `eval` in first-party code, and 78% through third-party code. As such, CSP's `unsafe-eval` becomes the norm, and flows into `eval` continue to cause DOM XSS vulnerabilities.

Trusted Types is a new approach aimed at preventing DOM XSS. It makes risky injection sinks **secure by default**, and encourages refactoring applications to avoid using injection sinks altogether. In cases where avoiding the sinks is not possible, Trusted Types force processing the *data* that passes to the sinks with specific security rules - Trusted Type policies. These policies require developers to consider *why* a given value should reach a sink, and how to assert that it's safe. Policies then become the only DOM-XSS-relevant part of the application codebase, significantly **reducing the attack surface** and facilitating security reviews to ensure the application is free from DOM XSS. The Trusted Types API was adopted by the Web Application Security WG in 2019, and has since launched in Chrome 83 and other Chromium-based engines.

Trusted Types in applications

Google applications

- To date, over 130 services built using Google's main web application framework fully enforce Trusted Types for HTML responses. This covers over 80% of the most sensitive services built using this framework. In total, 110 Google domains serve TT-enforcing headers; 67 of those domains serve it for over 50% of HTML documents and 29 serve it for all HTML responses. This includes relatively complex web applications like Google Photos, Google Contacts or Google My Activity.
- Following pilot integrations, which resolved the most common patterns incompatible with Trusted Types, we were able to rollout TT to >100 services in parallel, which suggests

that TT can be successfully deployed at scale after the initial time investment to address blocking issues.
- The effort to adopt Trusted Types was able to make use of past work to roll out strict CSP in the targeted applications.
- The approach leverages Google's existing Safe HTML Types libraries designed to centrally control how XSS-prone code can be introduced (background). To prevent functional breakage, we used CSP's report-only mode, and gradual rollouts (both internal, and external). Notably, rollouts at Google did not use the Trusted Types default policy which can facilitate deployment.
- For now, we only control sink assignments (using the `require-trusted-types-for` CSP directive) without enforcing restrictions on Trusted Types policy creation (which could be achieved with the `trusted-types` directive). This ensures full coverage of Trusted Types in the application, but doesn't address the "supply chain" problem where a new dependency may create an insecure policy and use it to unsafely interact with the DOM. Currently, we prevent this problem with static analysis.

Violation sources

Common violation sources identified in our applications were:

- Module loaders, especially in debug mode, that rely on `eval()`
- Non-standard ways to build & deploy code - not covered by XSS prevention static analysis
- Uncommon sinks not covered by static analysis - e.g. `Worker` and `DOMParser`
- Code in legacy libraries and frameworks (AngularJS, Polymer, d3)
- Easy to refactor code, e.g. clearing an element with `innerHTML = ''`

These occurred in:

- 1st party library code (53 occurrences)
- 1st party application code (20 occurrences)
- 3rd party code (15 occurrences)

Our experience based on this shows that:

- For complex web applications, the Trusted Types violations **root causes are present mostly in first party code** (though it might be a byproduct of our development practices and the target application selection)
- **Most of the fixes occur in library code** and unblock future application migrations. After the initial investment, future TT migrations are substantially easier.

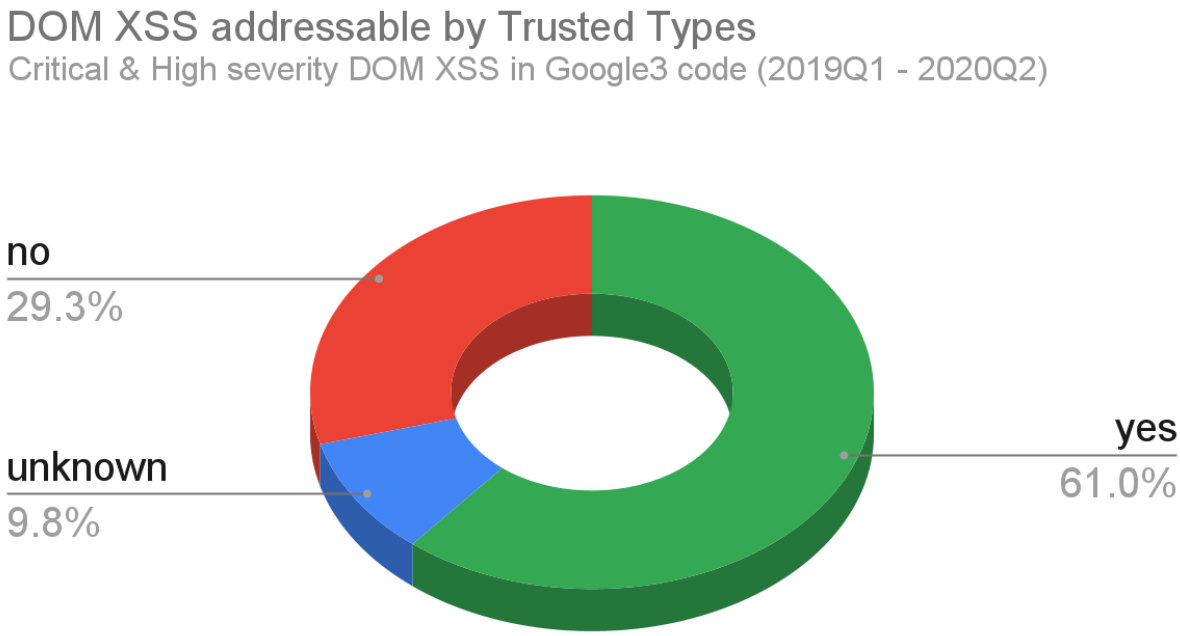
Outcome

Trusted Types migration to date uncovered many **gaps in our JavaScript static analysis** pipeline and surfaced several areas where the code was not sufficiently prevented from

introducing security bugs. It **triggered refactorings** that fix potential DOM XSS issues, **surfaced security risks** to product owners (e.g. hidden risks due to insecure library usage), and helped establish a **security baseline** that ascertains certain classes of vulnerabilities are simply not present.

So far, we've observed **0 DOM XSS issues** in applications migrated to Trusted Types. That said, our rollout strategy occasionally annotates risky and old code areas as "legacy conversions", and allows those to create Trusted Types too. While XSS is possible there, we believe the value of enforcing TT by default for all newly written code is still a substantial security improvement. So far we have not received any reports of vulnerabilities in legacy conversions introduced as part of the migration. Legacy conversions are a **measurable, outstanding risk known to the application owners** and its reduction is tracked separately.

To extrapolate, we performed a deep dive of confirmed DOM XSS reported to Google VRP to understand whether Trusted Types would be able to surface and prevent them. A conservative estimate indicated this to be the case for at least 61% of DOM XSS. To put it differently, **>60% of DOM XSS would be fixed with a savvy adoption of Trusted Types** that invariantly surfaces every risky sink usage.



The remaining bugs we believe would still remain were due to vulnerabilities in HTML sanitizers[2], inherent template mixing issues inside AngularJS[3], and the conversions mentioned above.

Adoption outside Google

Visual Studio Code

Visual Studio Code (VSCode) has migrated to Trusted Types (issue). For that Electron-based application, DOM XSS is the only type of XSS possible, and the risks of XSS exploitation are much higher than for a regular web application. VSCode is one of the largest projects written in Typescript (nearly **1M LOC**).

- **108 violations** were detected by static tooling (tsec). **75% were trivially fixable** without needing to create a Trusted Types policy.
- A few additional issues were discovered dynamically (most surfaced missing checks in static tooling, e.g. 1, 2, fixed now).
- Trusted Types policies were needed for more complex transformations, e.g.
    - bootstrap-window.js
    - markdownRenderer.ts

There are several layers that ascertain the project doesn't regress on DOM XSS protections:

- TT are enforced via the `require-trusted-types-for` header.
- New policy creation is guarded by the CSP `trusted-types` directive.
- Static checks exist, also to prevent new policy creation.

Outcome

VS Code enforces Trusted Types since January 2021. The migration **removed DOM XSS sinks from upstream projects**[4], e.g:

- https://github.com/xtermjs/xterm.js/issues/3189
- https://github.com/electron/electron/issues/27211

It also resulted in downstream changes - VSCode's editor component, **Monaco Editor** **is now Trusted Types compliant** and will no longer block other projects' migration to Trusted Types.

---

[2] In the analyzed period, more than half of the DOM XSS root causes were due to bugs in HTML sanitizers

[3] While TT could prevent DOM XSS in AngularJS applications, it would also break the whole framework. The proper fix is to move away from AngularJS, as TT cannot be used as a tool to make it secure with its inherent design issues.

[4] Notably, VSCode used TT default policy to unblock the migration while waiting for the upstream fix.

The migration exemplifies how TT encourages a **ripple effect** of hardening not just in a single software package, but also in related libraries.

Facebook

Facebook is in progress of rolling out Trusted Types enforcement, beginning with smaller properties. Early testing of TT enforcement in the main Facebook application revealed a long tail of violations. We've received positive feedback about the usefulness of the Chrome DevTools debugger, and the CSP reporting functionality.

Edge / Chromium

Jun Kokatsu (Microsoft) rolled out **Perfect Types**[5] to Chromium WebUI pages (writeup, documentation). This was possible, and relatively straightforward, due to applications on those pages using React. **Many React apps can enforce Trusted Types with no code changes**.

The Trusted Types default policy is used for special cases when loading remote scripts in WebUI.

In Jun's own words:

> [..] While we invest a lot in fuzzing, static analysis, and other automation to scale bug findings, sometimes moving to secure coding practices like Trusted Types is the right answer. While we had to contribute to a few libraries (such as react virtualized), converting code to be compatible with Trusted Types was much better than writing a tool to detect potential XSS (which is difficult 😊). Going forward, we will need to choose a third-party library that supports Trusted Types (or we will contribute to it). And I hope that Trusted Types support for libraries/frameworks will be important for every site, as deployment of Trusted Types grows.

This work has been leveraged in Edge. The integration between Edge and MS applications now requires Trusted Types to simplify code reviews (that bridge is particularly sensitive, as it executes in a privileged origin). The migration used eslint-plugin-no-unsanitized to identify Trusted Types violations statically.

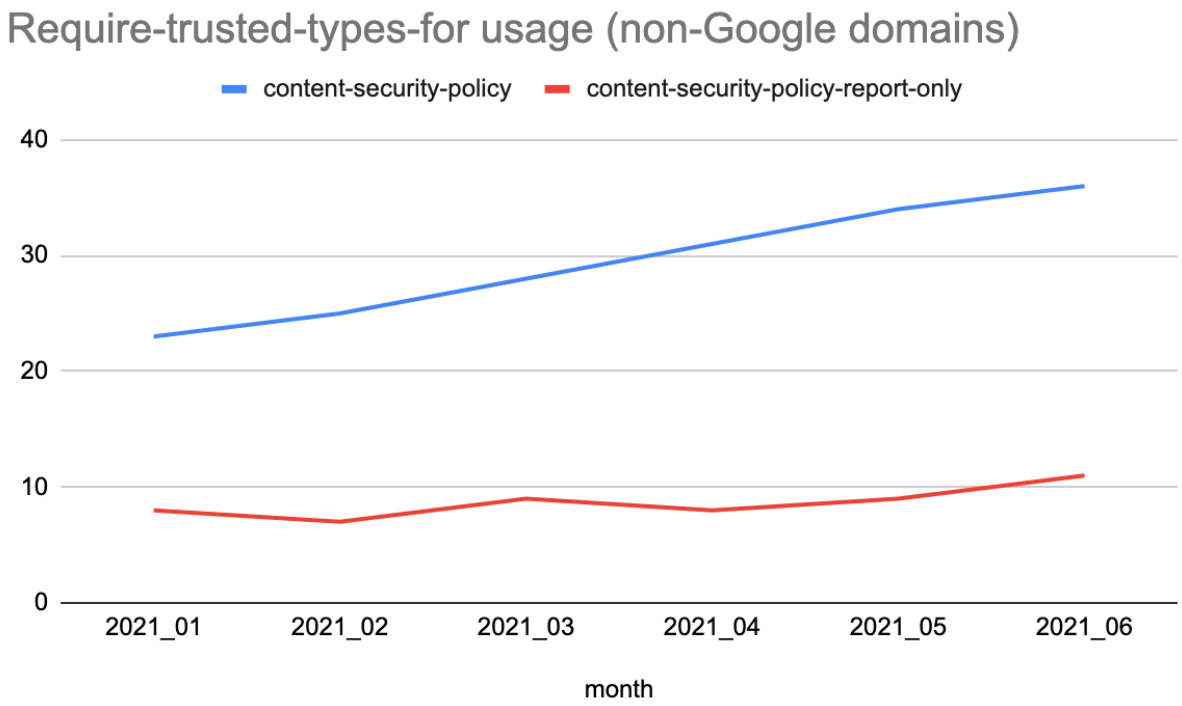
HTTP Archive

HTTP archive crawler[6] identified a small number of non-Google domains using `require-trusted-types-for` directive. We're seeing a slow increase month-to-month.

---

[5] *Perfect Types* is a header setting (Content-Security-Policy: require-trusted-types-for 'script'; trusted-types 'none';) that enforces no DOM XSS sinks are used in the application. In other words, Perfect Types require Trusted Types without allowing any TT policy to be created, thus enforcing that no XSS-prone string-to-DOM flows are being used in the application at all.

[6] HTTP Archive has a significant limitation in that the authenticated content is not crawled - e.g. if Trusted Types are used only in the authenticated part of the application, they would not be included in the dataset.



There are a few complex applications with user authentication in that group, e.g. https://auth.heroku.com, https://secure.loanbeam.com, https://my.sqreen.com, https://dashboard.paymentrails.com, https://gpanel.promevo.com/. https://cloudapis.hilti.com seems to require Trusted Types on the OAuth endpoint.

The rest are mostly smaller sites e.g. https://www.monster.ba/, https://lexmark.cloud-connect.co/ (based on Angular), https://www.kuntokeskusenergy.com/, https://www.gillmanandsoame.co.uk/.

Trusted Types in libraries and frameworks

Trusted Types violations are commonly caused by a relatively small amount of code (for example, ~100 violation root causes in 1M LOC in VSCode), very commonly in library or framework code that:

- Dynamically loads scripts
- Renders application components in the DOM
- Creates Web and Service Workers
- Adds debug-only or test-only functionality (e.g. such code commonly uses `eval()`, see Hardening Firefox against Injection Attacks)

To date, we've helped migrate several common libraries and frameworks to Trusted Types (wiki). The approach there is to:

1. Stop using DOM XSS sinks (a preferred approach that introduces no dependency on Trusted Types policies API).
2. Make the library **produce Trusted Types values** where we can be sure it's safe.

   Examples:
   - Webpack (script URLs come from configuration)
   - DOMPurify (HTML is sanitized)
   - Angular (templates are implicitly trusted; built-in HTML sanitizer makes interpolated data safe)
   - Lit-html
   - Karma runners & reporters (it's test-only code)

3. Refactor libraries that write (presumed safe) application-provided data to the DOM **not to stringify values passed to DOM XSS sinks**. This makes the library "Trusted Types agnostic" and surfaces to the application that the values need to be explicitly annotated as safe — by creating a Trusted Type instance.

   This is relatively rare in modern frameworks, as they tend to "own" security choices for the application, and discourage such patterns.

   Examples:
   - React

Safevalues

To facilitate migration to Trusted Types, we open sourced safevalues, a Typescript **library with a safe abstraction over Trusted Types.** The package exposes safe builders for values to be passed to DOM XSS sinks - for environments with, or without Trusted Types. It solves common problems encountered during TT migrations - e.g. blessing known-safe static values, allowing for limited interpolation into script URLs, or annotating code as legacy and security-reviewed.

We've also introduced Trusted Types directives support for Django, to help rolling out Trusted Types on the server side.

Outcome

Several popular libraries have been migrated to Trusted Types, either by default (**Angular, Lit**) or by using a configuration switch (e.g. **React, Webpack, DOMPurify**). Applications built on top of those libraries, if built within the guardrails of their respective security recommendations, are also Trusted Types compliant - they can be "secure by default" and ensure the absence of DOM XSS issues.

Trusted Types enforcement for those applications functions as a litmus test, able to detect if the application uses a risky coding pattern - either directly, or through a dependency. When the litmus test passes (e.g. Chrome Web UI pages built on top of React), the application may start enforcing TT to prevent future regressions.

Migrating the libraries to TT also uncovered existing security issues (e.g. Angular sanitizer did not cover i18n attributes).

API changes

Since inception, we've made **several changes to the Trusted Types API** to facilitate application migration without regressing on security:

- We removed `TrustedURL` type and replaced it with `javascript:` URL blocking
- We split enforcement controls into two CSP directives - one to control sink assignments, the other one to control policy creation.
- We added a keyword to allow for duplicate policy names.
- We added CSP reporting. Violation samples include the sink and its payload, greatly helping cluster and identify the root causes.

Together, these changes were crucial in simplifying Trusted Types rollouts.

Trusted Types tooling

To date, several Trusted Types related tools have been created:

- Various versions of the polyfill to emulate the API in the browser, or in NodeJS.
- Tsec - Typescript compiler wrapper with rules tailored to migrate code to Trusted Types.
- Tsec-validation - dynamic TT violation finder (injects CSP header, collects violations, deduplicates, understands sourcemaps).
- Debugging support for Trusted Types violations in Chrome DevTools -
- A Chrome extension to find DOM XSS with Trusted Types https://www.youtube.com/watch?v=CNNCCgDkt5k

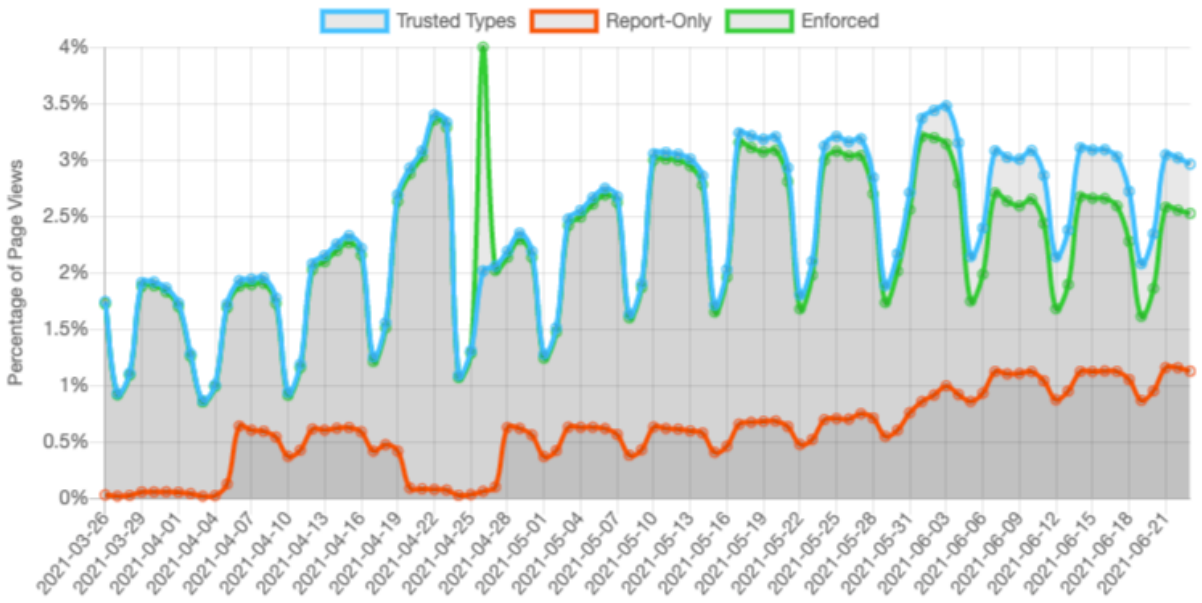
Summary

We believe Trusted Types are **necessary to obliterate DOM XSS**, one of the most prevalent web application vulnerabilities. Being an in-browser guard over all injection sinks, they are free from static tooling false negatives, which still contribute to bugs (in our analysis, > 60% of DOM XSSes were attributable to gaps in static analysis coverage). That said, they don't *solve* the DOM XSS alone, and more effort is needed to make Trusted Types adoption **practical**.

The Trusted Types API is built in a way to **discourage DOM XSS sink usage**, and thus incentivize removing the vulnerability-prone code**.** To a large extent code can comply with TT restrictions simply by avoiding the use of dangerous DOM sinks (see VSCode). In some cases, this results in Perfect Types, where TT restrictions can be enforced without needing to create a single policy (see Edge / Chromium). For cases where that's not possible[7], TT policies allow the applications to make **security rules explicit in the code** and significantly **reduce and track the DOM XSS attack surface**.

In our experience, while many 3rd party dependencies don't interact with XSS sinks and don't affect the migration to Trusted Types, the vast majority of the necessary code changes for TT compliance are in the **library / framework code**. With buy-in from popular library maintainers, this significantly reduces the burden in individual application owners. To date, **Angular, Lit, React, Webpack** and many other libraries support Trusted Types. We've also authored packages and tools that help with Trusted Types rollouts.

Trusted Types were built in a way that facilitates **gradual code migration**. The JS API to create Trusted Types can be used even if the web application doesn't enforce TT. That works in practice - currently nearly 60% of Chrome's page views create Trusted Types policies (likely through one of Google dependencies in a subframe), whereas only around 3% page views set the respective CSP directive. Used correctly, Trusted Types **don't break applications** and may prevent DOM XSS even in browsers that don't support Trusted Types.



Percentage of Chrome page loads that require Trusted Types at sinks in either enforcing or reporting mode.
Source: https://mitigation.supply

[7] For example, when the application loads scripts dynamically

We believe some of the new web platform primitives will make Trusted Types migrations even simpler. For example, the HTML Sanitizer API allows applications to securely create DOM nodes from user-controlled content. We also think it's possible to securely allow constant string assignment to DOM XSS sinks. Together with Trusted Types and the efforts described in this document we believe we finally have a chance of obliterating DOM XSS through web platform mechanisms.