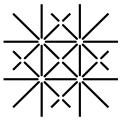

University of Basel

Sandbox Implementation for Enterprise Collaboration Platforms

Scientific Writing Seminar Paper

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science

Examiner: Prof. Dr. Craig Hamilton
Supervisor: Dr. Tanja Schindler

Tri Nguyen
tri.nguyen@unibas.ch
24-065-948

20th December 2024

Abstract

Enterprise collaboration platforms, such as Monday.com[1] and ClickUp[2], have generated billions in revenue by addressing the data silos problem and integrating a broad range of functionalities into a single, unified platform. However, third-party app integrations are often limited due to the absence of a flexible sandbox model. In this paper, we propose a novel sandbox solution based on Secure EcmaScript and the Membrane programming pattern. Our evaluation demonstrates how this sandbox model expands the technical capabilities of third-party applications and improves the developer experience, all while not sacrificing security. We conclude that this sandbox model provides a viable solution for building more effective enterprise collaboration platforms.

[1] https://monday.com/
[2] https://clickup.com/

Table of Contents

1

Introduction

1.1 Background

One of the most pervasive challenges faced by organizations is the existence of data silos—disconnected, fragmented pockets of information that reside in isolated platforms or applications. Data silos present significant challenges for knowledge workers, who often find themselves navigating a maze of isolated data sets, each requiring manual coordination between teams[1].

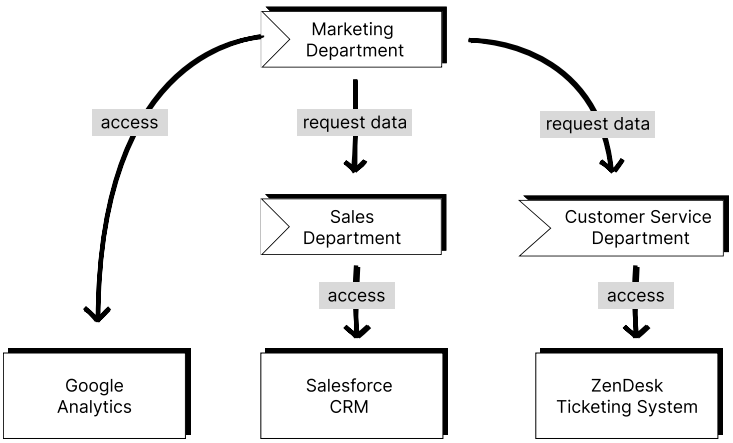

Figure 1: The Marketing Department struggles to combine three different data sources for a new advertising campaign.

To illustrate the impact of data silos, consider a typical scenario within an organization, as shown in Figure 1. The Marketing Department may want to launch a new advertising campaign but only has access to Google Analytics, a tool that provides user behavior data. To obtain a broader understanding of customer profiles and the issues customers are facing, they must contact the Sales and Customer Service departments—each of which stores its data on different platforms. The result is a cumbersome

process of collecting and consolidating data from three different sources—a process fraught with delays and coordination challenges.

This has led to many organizations turning to *collaboration platforms* as a lightweight and user-friendly approach to solving the data silos problem. Collaboration platforms bring together a suite of tools—such as messaging app, project management tools, wiki, meeting scheduler, and CRM[3] software—into a unified space[2]. This integration makes it easier for teams to share and access the data they need without navigating multiple systems or encountering fragmentation.

1.2 Motivation

Many collaboration platforms, in order to address a wide range of business use cases, invite third-party developers to build and integrate their applications into the platform. However, due to high security requirements[3] and the challenges of implementing a robust sandbox system, these platforms often resort to restricting third-party app functionalities. As a result, these apps typically store core logic and data outside the platform, have limited control over the platform's UI[4], and require developers to invest effort in learning niche, platform-specific frameworks. An overview is shown in Table 1. Data silos persist because data is still not centralized, but merely shared through APIs[5], which does not solve the problem of fragmented access.

Thus, these apps neither extend the platform's functionality in a meaningful way nor are they easy to develop. Implementing a sandbox system that satisfies both security and developer experience requirements would enable the platform to offer a more flexible runtime environment for third-party developers, which in turn would enhance its ability to solve the data silos problem.

Platform	Third-Party Data Integration Capability	UI Customization	Platform-agnostic Development Framework
Lark[6]	Low	Limited (pre-made blocks)	No
ClickUp	Moderate	No customization	Yes (via API calls)
Monday.com	Moderate	Limited (pre-made blocks)	Yes (via API calls)
Asana[7]	High (via Work Graph®)	Limited (pre-made blocks)	No
Salesforce[8]	High (via Standard Objects)	Full customization	No

Table 1: Overview of third-party app support across collaboration platforms: Most platforms offer limited UI customization or require a specialized development framework.

[3] Customer Relationship Management
[4] User Interface
[5] Application Programming Interface
[6] https://larksuite.com
[7] https://asana.com
[8] https://salesforce.com

1.3 Objective & Key goals

The primary objective of this work is to develop a secure, browser-based sandbox solution that addresses the challenges of integrating third-party apps within collaboration platforms. The lack of existing solutions presents a significant barrier for platform developers seeking secure and flexible third-party app integration. The sandbox model aims to achieve several key goals:

1. **Protection of critical resources**: Any potentially harmful code is confined within a controlled environment, preventing third-party apps from executing unauthorized actions. Essential platform resources are safeguarded from third-party code, ensuring system integrity.

2. **Ease of third-party app development**: The model supports universal frameworks like React, Angular, or Vue and includes development features such as HMR[9]. It is designed with data access and UI customization capabilities in mind.

3. **Performance**: The sandbox does not degrade platform performance and remains close to the performance of native, unsandboxed code.

[9] Hot Module Reload

2

Related Work

Traditionally, sandbox systems in the browser have been built using native web technologies such as iframes, Web Workers, and Service Workers[4]. While these technologies require minimal setup, they pose unique security and accessibility challenges[5]. They also introduce significant communication overhead[6], as their security model isolates third-party apps in separate threads. The rendering process requires observing mutations in the iframe's DOM[10] and synchronizing its state with the main thread's DOM. This technique, known as remote rendering, remains useful when the app is light and simple, as employed by Shopify to render customizable post-purchase pages for merchants[7]. Unfortunately, communication overhead becomes more noticeable when multiple apps are run simultaneously, as they may interact with each other and share state. There have been attempts to reduce this overhead, notably through the Near Membrane implementation by Salesforce[8].

Recently, WebAssembly has become another popular choice for building sandboxes. While WebAssembly provides an isolated environment for running third-party code, it is not a complete solution. WebAssembly allows for executing a JavaScript runtime[9] within a sandbox, but it lacks direct access to the DOM and the ability to share JavaScript objects[10]. As a result, code running in the sandbox cannot interact directly with the platform. This limitation can be addressed by writing glue code that exposes platform APIs to the sandboxed environment[11]; however, the solution is not scalable and is limited to specific, well-defined use cases.

Another successful approach to sandboxing is language-based analysis and code sanitization, as demonstrated by the Caja compiler[12]. This method offers a low runtime performance penalty, but when used with dynamic or interpreted languages like JavaScript, it may introduce vulnerabilities because code executed at runtime was not originally part of the program's source code. While our

[10]Document Object Model

implementation incorporates elements of this approach, the primary security mechanism is still based on environment isolation.

3

Implementation

3.1 Model Overview

In our model, each third-party app corresponds 1:1 to a dedicated sandbox, making it particularly well-suited for collaboration platforms where users can have multiple apps running simultaneously. Each sandbox is an isolated environment that limits the third-party app to a controlled subset of sensitive resources. These resources can be owned by the browser (e.g., the DOM, web APIs like camera, cookies, and network), the business (e.g., proprietary data), or the platform itself. Access to these resources is governed by the sandbox's permissions, which are granted by the platform, IT administrators, or end users. Resource access is transparent to the platform, which can revoke permissions at any time, and usage can be introspected. The sandbox model is illustrated in Figure 2.

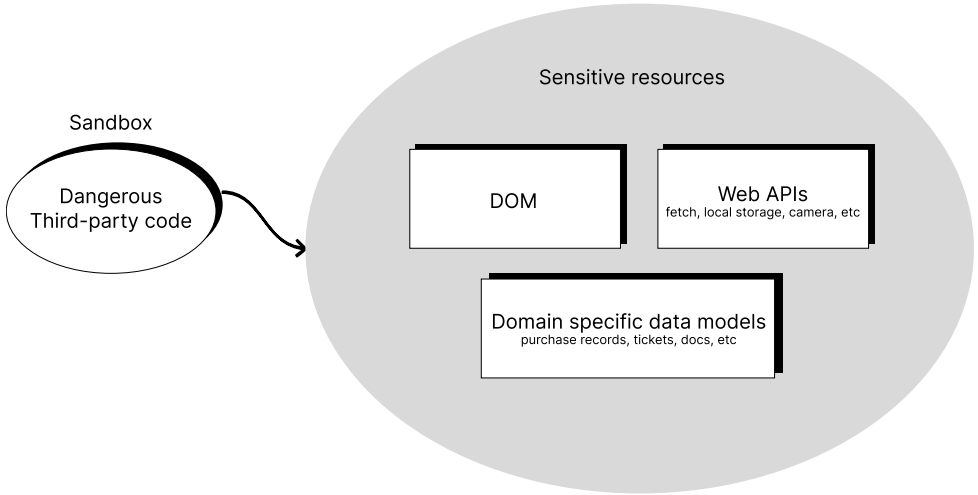

Figure 2: Isolating third-party apps within a sandbox with controlled access to platform resources. Third-party code with potential harm is safely contained within the sandbox.

The sandbox also distorts the functionality of several web APIs to enhance security and improve the development experience through isomorphism. For example, the JavaScript code snippet shown in Snippet 1 appears similar to that of a normal web app, but behaves differently within the sandbox:

```
// fetch is distorted to automatically checks the domain against a whitelist
const response = await fetch('https://example.com/api/reservations')
const data = await response.json()
// localStorage.setItem is distorted to prevent storing the data
// forwarding it instead to a separate data store
// Each app then has a non-conflicting, isolated data store
localStorage.setItem('data', JSON.stringify(data))
// getElementById is distorted to query only elements
// under a specific DOM node granted to the sandbox
 const element = document.getElementById('app')
 // element.innerHTML is distorted to sanitize the HTML before setting
 element.innerHTML = JSON.stringify(data)
```

Snippet 1: Example of how web APIs are distorted within the sandbox to prevent dangerous dynamic content, isolate apps from each other, and prevent UI style leaks.

The list of distortions is not exhaustive and will be expanded as new web APIs emerge. Since automatically introducing an undistorted API could pose a security risk, each version of the sandbox maintains a list of supported APIs and only exposes those to the third-party app.

3.2 Technologies Used

Under the hood, the sandbox treats each app as a collection of HTML, CSS, and JavaScript files, each with its own security requirements. This is illustrated in Figure 3. Several different technologies are combined to achieve the desired sandboxing functionality.

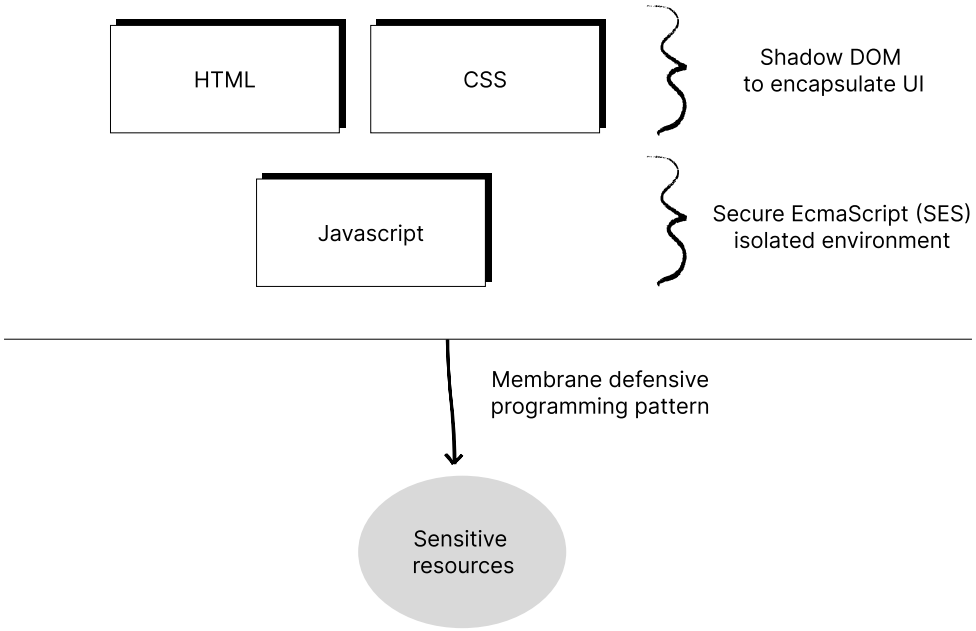

Figure 3: Technologies used for each component of the sandbox model. HTML, CSS, and JavaScript are stored within the sandbox (inside the membrane) and secured using Shadow DOM and Secure ECMAScript. Access to sensitive resources (outside the membrane) is safeguarded using distortion techniques.

3.2.1 For HTML and CSS

For HTML and CSS, Shadow DOM[13] is used to prevent style leakage from the app to the platform. Shadow DOM is a browser-native feature that facilitates the creation of an isolated UI environment for each app, as seen in Web Components[14]. Each sandbox has access to a shadow root node that it fully controls. Necessary distortions are implemented to prevent querying elements outside the shadow root (e.g., via `shadowRoot.ownerDocument`).

3.2.2 For JavaScript

For JavaScript, the sandbox utilizes the Secure ECMAScript (SES) library by EndoJS[15] to secure global prototypes, preventing prototype pollution attacks. SES creates a compartment where access to web APIs is disabled by default, with access only granted through explicit endowments. Unlike iframes or Web Workers, SES does not require a separate thread to execute code. Instead, the code runs within the same main thread as the platform, offering improved efficiency by avoiding communication overhead and enabling shared object address space.

3.2.3 For Resource Access

To enable granting and revoking access to resources, as well as applying distortions, the sandbox employs the Membrane defensive programming pattern[16]. The Membrane pattern is simpler than other capability-based security measures, automatically scales to cover all web APIs via deep annotation, while remains theoretically secure[17]. When implemented in JavaScript, the pattern leverages ES6 proxies[18]: by exposing only the proxy instead of the underlying resource object, the sandbox can distort certain operations on the object through the proxy's handler. Any object returned as a result of these operations is also wrapped in a proxy[19].

3.3 Improved Developer Workflow

We aim to provide a developer experience that closely resembles working with a typical web app. To achieve this, the sandbox takes additional steps to ensure a seamless workflow:

1. Rather than requiring a manifest file to define entry points, the index HTML file is parsed to identify them. Once the entry points are obtained, external files are fetched, while inline CSS and JavaScript are sanitized and evaluated.

2. Dynamic imports and ES modules[18] are supported, as they are the primary mechanisms for development servers (e.g., Vite) to load and replace code, such as through Hot Module Replacement (HMR). Since JavaScript dynamic imports are part of the ECMAScript Language specification[18], their implementation involves analyzing the source code and replacing the `import` statement with a function call[11]. This function fetches the file and returns an ES module object.

As a result, the developer experience remains nearly identical to that of a traditional web application. Developers can use familiar tools and frameworks, without any modification, and can load the application directly into the sandbox from the local environment via localhost ports. All safeguard mechanisms are abstracted from the codebase.

3.4 Workaround for SES Limitations

At the time of writing, there are several key issues with SES that we have encountered and implemented workarounds for. These issues have been reported to the SES repository and are expected to be resolved in future versions of the library. Currently, the versions that we are using are `ses@1.9.1` and `@endo/module-source@1.1.2`. Since SES is still actively evolving and may not be fully stable, this section highlights common pitfalls encountered when implementing a sandbox model with SES, along with the workarounds we've applied to address them downstream.

Our workarounds are integrated into the sandbox's transform and sanitization process, which occurs before the code is executed within the SES compartment. Notably, this may reduce the security guarantees provided by SES, as SES itself does not rely on the parser's correctness. However, this approach allows us to implement a functional sandbox while keeping the attack surface to a minimum.

The issues and corresponding workarounds are as follows:

1. *Assigning to Empty Objects Error*: The creation of an empty object using the syntax `{}` results in an object with the prototype `Object.prototype` instead of `null`. This is considered a security risk as it allows for prototype pollution attacks. By default, SES hardens the Object prototype, so attempting to assign fields to the prototype of objects results in an exception. The official response from the SES team is that this is a known issue[12] and a fundamental flaw in JavaScript itself, not SES. SES advises developers to flag libraries with this issue and work with the library maintainers to make their

[11] The distortion mechanism only works on resources, such as web APIs, and not on language features. Modifying language features requires altering the source code before importing it into the sandbox.

[12] https://github.com/endojs/endo/discussions/1855

code more secure. Since a sandbox aims to be developer-friendly, we could not expect developers to address this issue directly. Therefore, we implemented a workaround to automatically replace all instances of `{}` with `Object.create(null)`.

2. *Babel Transformation Errors*: This issue relates to SES's inability to process default destructuring syntax[13] and `import.meta._`[14]. We implemented a workaround to replace all instances of default destructuring with the equivalent syntax and to replace all instances of `import.meta._` with a function call to obtain the correct value.

3. *HTML Comments*: Due to the similarity between the syntax of HTML comments and the JavaScript syntax `<!--`, SES uses special regular expressions to distinguish them[15]. This feature however is not supported when evaluating ES Module code. Our sandbox relied heavily on loading the modules to support hot module reloading during the third-party app development workflow. As a result, we had to move this component downstream and modify it to meet our needs.

As mentioned, while these workarounds are practical, they do lower the security guarantees. We are mindful of this and are looking forward to the SES team resolving these issues in future releases.

[13] https://github.com/endojs/endo/issues/2633
[14] https://github.com/endojs/endo/issues/2649
[15] https://github.com/endojs/endo/blob/master/packages/ses/error-codes/SES_HTML_COMMENT_REJECTED.md

4

Evaluation

For evaluation, we developed a demo collaboration platform to test the sandbox model's security and performance, with screenshots provided in Figure 4. This platform served as an experimental and controlled environment, allowing us to explore various sandbox configurations and assess their effectiveness in isolating application functionality. The platform hosts four distinct placeholder apps —Chat Hub, Project Management, Notes, and Whiteboard—each running on a different development port. The applications are simple but developed to demonstrate unique requirements of each business case, like drag-and-drop items in the Whiteboard app or typing in the Notes app. This diverse set of third-party apps demonstrates the platform's ability to cover all use cases and efficiently run multiple applications simultaneously using the sandbox model. The platform is hosted at https://thirdcloud.org to replicate the environment of real-world platforms.

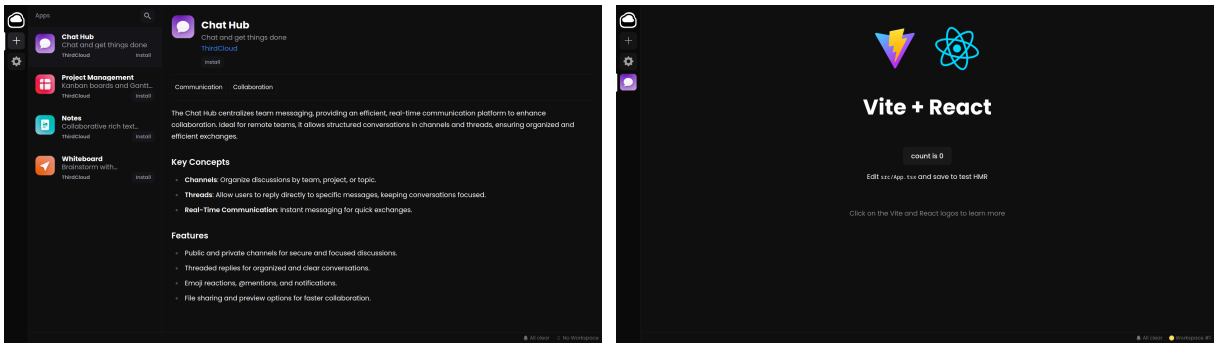

Figure 4: Demonstration of the platform, where the sandbox model effectively isolates third-party applications. The currently displayed application, built using the React + Vite template, is fully interactive, with animations and static assets loading correctly.

By introducing the distortion rules and requesting resources from within the sandbox, we successfully achieved the intended result of catching and preventing all illegal invocations. Testing various interac-

tions—such as button clicks, keyboard inputs, popup open/close actions, and browser alerts—revealed that the safeguarding mechanisms do not interfere with the rendering engine or interactions with the app. Additionally, all static assets, such as image files, work surprisingly well out-of-the-box. This is a free consequence of the development server loading them as ES modules, which the sandbox supports, and the sandbox being able to map the fetch request to the correct port. These results demonstrate the platform's potential to effectively support large-scale, production-ready applications that utilize a diverse set of web APIs.

We have also successfully loaded all four apps to the platform at the same time. The sandbox running on a production domain does not pose any challenge in loading the development ports, except in the case of Brave Browser[16], where the Brave Shield feature has to be disabled to communicate with localhost. The platform supports Hot Module Replacement, with real-time updates during development without full page reload.

Despite these strengths, the platform exhibits a notable limitation in its initial load time. During the first load, the application experiences a delay of approximately 5 to 10 seconds. This delay is primarily caused by the overhead associated with fetching large libraries and converting source code into Secure ECMAScript (SES)-compatible modules. Consequently, the platform becomes totally unresponsive to user interactions, such as mouse clicks or keyboard input, during this period.

In summary, the sandbox model effectively isolates the application's functionality, limiting its access to a predefined subset of the platform's resources. Further research and optimization are required to improve the platform's performance, particularly in terms of load time. This also points to the need for more control over the runtime configuration of third-party applications (e.g., maximum stack size, RAM, CPU allocation).

[16]https://brave.com

5

Conclusion

In conclusion, the sandbox implementation has successfully met its objectives. We have demonstrated the platform's ability to isolate third-party applications while supporting the efficient execution of complex applications. This is all done while providing third-party developers with first-class support for their preferred frameworks.

Nevertheless, several areas present opportunities for improvement. One challenge is the caching of large libraries, such as React, which are shared across multiple applications. Implementing a caching strategy could significantly reduce load times and enhance overall performance. On top of that, we could accelerate the SES module transformation process by offloading it to a separate thread, further improving load time.

Looking ahead, future work on benchmarking is essential. While initial tests using manual interactions show promising results, establishing clear benchmarks—such as frames per second (FPS) for rendering or the number of iterations for data processing—would provide a more objective basis for assessing performance and pinpointing areas for further optimization. We also plan to explore security from a new angle by revisiting the integration of WebAssembly with Membrane. This approach could help address WebAssembly's current limitation regarding direct DOM access. Finally, we aim to expand the scope of our security measures by defining precise data models, role-based access policies, and user consent prompts—allowing users to make well-informed decisions.

As we move forward, the sandbox model will continue to evolve, adapting to the changing needs of both platform & third-party developers.

B

Bibliography

[1] A. Shahrokni and J. Söderberg, "Beyond Information Silos: Challenges in Integrating Industrial Model-based Data," in *BigMDE@STAF*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:17194919

[2] B. Manko, "The adaptability of Monday.com's app-based software: Discover the company building a flexible business model that adapts to individual company needs," *Journal of Information Technology Teaching Cases*, vol. 12, Aug. 2021, doi: 10.1177/20438869211028855.

[3] P. Saa, O. Moscoso-Zea, A. C. Costales, and S. Luján-Mora, "Data security issues in cloud-based Software-as-a-Service ERP," in *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, 2017, pp. 1–7. doi: 10.23919/CISTI.2017.7975779.

[4] J. Terrace, S. R. Beard, and N. P. K. Katta, "JavaScript in JavaScript (js.js): Sandboxing Third-Party Scripts," in *3rd USENIX Conference on Web Application Development (WebApps 12)*, Boston, MA: USENIX Association, Jun. 2012, pp. 95–100. [Online]. Available: https://www.usenix.org/conference/webapps12/technical-sessions/presentation/terrace

[5] D. F. Some, N. Bielova, and T. Rezk, "On the Content Security Policy Violations due to the Same-Origin Policy," in *Proceedings of the 26th International Conference on World Wide Web*, in WWW '17. Perth, Australia: International World Wide Web Conferences Steering Committee, 2017, pp. 877–886. doi: 10.1145/3038912.3052634.

[6] L. Ingram and M. Walfish, "Treehouse: Javascript Sandboxes to Help Web Developers Help Themselves," in *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, Boston, MA: USENIX Association, Jun. 2012, pp. 153–164. [Online]. Available: https://www.usenix.org/conference/atc12/technical-sessions/presentation/ingram

[7] J. Freund, "Remote rendering & UI extensibility." [Online]. Available: https://shopify.engineering/remote-rendering-ui-extensibility

[8] Salesforce, "near-membrane: JavaScript Near Membrane Library that powers Lightning Locker Service." [Online]. Available: https://github.com/salesforce/near-membrane

[9] M. Wu, W. Dong, Q. Zhao, Z. Pan, and B. Hua, "An Empirical Study of Lightweight JavaScript Engines," in *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, 2023, pp. 413–422. doi: 10.1109/QRS-C60940.2023.00103.

[10] P. P. Ray, "An Overview of WebAssembly for IoT: Background, Tools, State-of-the-Art, Challenges, and Future Directions," *Future Internet*, vol. 15, no. 8, 2023, doi: 10.3390/fi15080275.

[11] R. Chen, "How to build a plugin system on the web and also sleep well at night." [Online]. Available: https://www.figma.com/blog/how-we-built-the-figma-plugin-system/

[12] B. L. I. A. Mark S. Miller Mike Samuel and M. Stay, "Caja: Safe active content in sanitized JavaScript," 2008. [Online]. Available: http://google-caja.googlecode.com/files/caja-spec-2008-06-07.pdf

[13] J. Krause, "Shadow DOM," in *Developing Web Components with TypeScript: Native Web Development Using Thin Libraries*, Berkeley, CA: Apress, 2021, pp. 43–52. doi: 10.1007/978-1-4842-6840-7_3.

[14] J. Yang and M. P. Papazoglou, "Web Component: A Substrate for Web Service Reuse and Composition," in *Advanced Information Systems Engineering*, A. B. Pidduck, M. T. Ozsu, J. Mylopoulos, and C. C. Woo, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 21–36. doi: 10.1007/3-540-47961-9_5.

[15] Endo, "Endo: Secure JavaScript sandboxing and capability-based programming." 2024.

[16] Y. Jaradin, F. Spiessens, and P. Van Roy, "Capability confinement by membranes," 2005.

[17] D. Gorla, M. Hennessy, and V. Sassone, "Security Policies as Membranes in Systems for Global Computing," *Logical Methods in Computer Science*, vol. 1, no. 3, p. 2, 2005, doi: 10.2168/lmcs-1(3:2)2005.

[18] E. International, "ECMA-262: ECMAScript® 2020 Language Specification (11th Edition)." Accessed: Dec. 20, 2024. [Online]. Available: https://262.ecma-international.org/11.0/index.html

[19] M. Keil, S. N. Guria, A. Schlegel, M. Geffken, and P. Thiemann, "Transparent Object Proxies for JavaScript." [Online]. Available: https://arxiv.org/abs/1504.08100