Sandbox Implementation for Enterprise Collaboration Platforms

Scientific Writing Seminar Final Paper

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

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

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

20th December 2024

Abstract

Collaboration platforms, such as Monday.com and ClickUp, have revolutionized enterprise workflow management by integrating diverse functionalities into a single, unified platform. These platforms have generated billions in revenue, but their third-party app integrations are often constrained due to security concerns and the lack of a flexible sandbox model.

In this paper, we present a novel sandbox solution that enhances functionality while maintaining robust security. Our evaluation demonstrates that this sandbox model enables granular access control, expands the technical capabilities of third-party applications, and significantly improves the developer experience.

Table of Contents

1

Introduction

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].

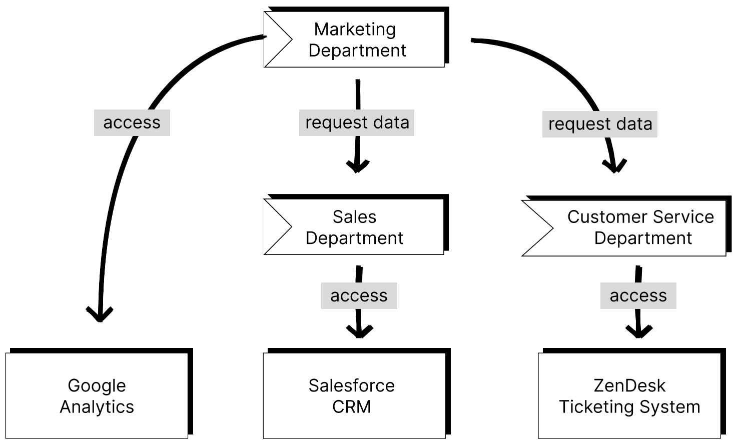


Figure 1: The Marketing Department struggles to combine three different data sources for a new advertising campaign.

To illustrate the impact of data silos, consider a typical scenario within an organization. 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.

1.1 Background

Collaboration platforms bring together a suite of tools—such as messaging apps, project management tools, wikis, meeting schedulers, and CRMs—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.

Many collaboration platforms, in order to cover 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 difficulty of implementing a robust sandbox system, these platforms often resort to restricting third-party app functionalities instead. As a result, these apps typically store core logic and data outside the platform, have limited control over the platform's user interface (UI), and require developers to learn niche, platform-specific frameworks.

Thus, these apps neither extend the platform's functionalities in a meaningful way nor are they easy to develop.

Platform	Third-Party Data Integration Capability	UI Customization	Platform-agnostic Development Framework
Lark	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	High (via Work Graph®)	Limited (pre-made blocks)	No
Salesforce	High (via Standard Objects)	Full customization	No

Table 1: Overview of third-party app support across collaboration platforms.

1.2 Objectives

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 creates a significant barrier for platform developers seeking secure and flexible third-party app integration.

The sandbox model aims to achieve several key goals:

- **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 protected from third-party code, ensuring system integrity.

- **Ease of third-party app development**: Supports universial frameworks like React, Angular, or Vue with development features like Hot Module Reload (HMR). The model is designed with data access and UI customization capabilities in mind.
- **Performance**: The sandbox does not degrade the platform and remains close to the performance of native, unsandboxed code.

2 Related Works

Traditionally, sandbox systems in the browser have been built using native web technologies such as iframes, Web Workers, and Service Workers [4]. While requiring minimal setup, these technologies 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 and synchronizing its state with the main thread's DOM. This technique, known as remote rendering, is still useful when the app is light and simple. It is employed by Shopify to render customizable post-purchase pages for merchants [7]. 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 the communication overhead, notably through the Near Membrane implementation by Salesforce [8].

Another attempt at sandboxing leverages WebAssembly (WASM). While WASM provides an isolated environment for running third-party code, it is not a complete solution. WASM 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 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 due to code executed at runtime was not originally part of the program's source code. Although our implementation incorporates elements of this approach, the main security mechanism is still based on environment isolation.

3 Implementation

3.1 Model Overview

Each sandbox is an isolated environment that limits third-party apps to a controlled subset of sensitive resources. These resources are 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 usage is transparent to the platform, which can introspect or revoke access at any time.

A platform can run multiple sandboxes simultaneously, each with its own set of resources and policies. This model is particularly well-suited for collaboration platforms, where each instance of a third-party app operates within its own sandbox.

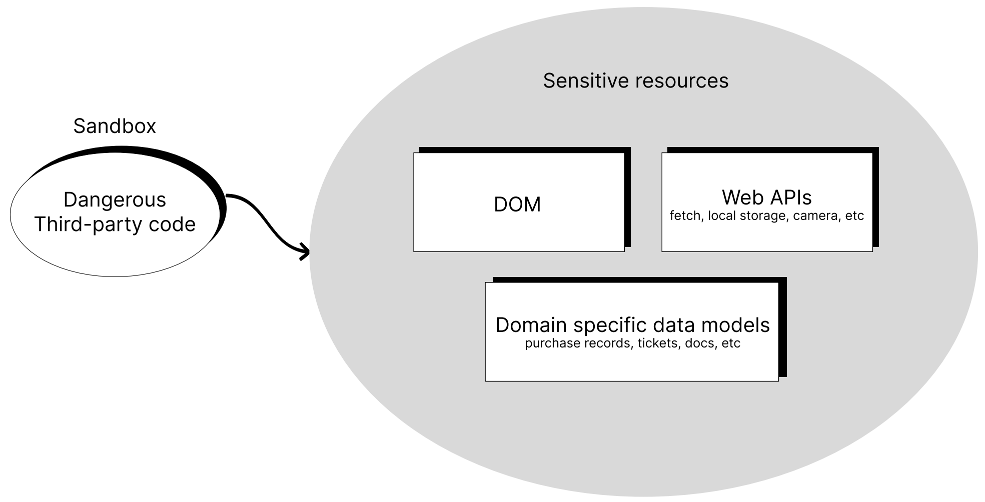

Sandbox
Dangerous Third-party code
Sensitive resources
DOM
Web APIs
fetch, local storage, camera, etc
Domain specific data models
purchase records, tickets, docs, etc

Figure 2: Isolating third-party apps within a sandbox, with controlled access to platform resources.

The sandbox also distorts the functionality of several web APIs to enhance security and improve the development experience through isomorphism. For example, the following JavaScript code snippet 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)
```

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 Technolgies Used

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

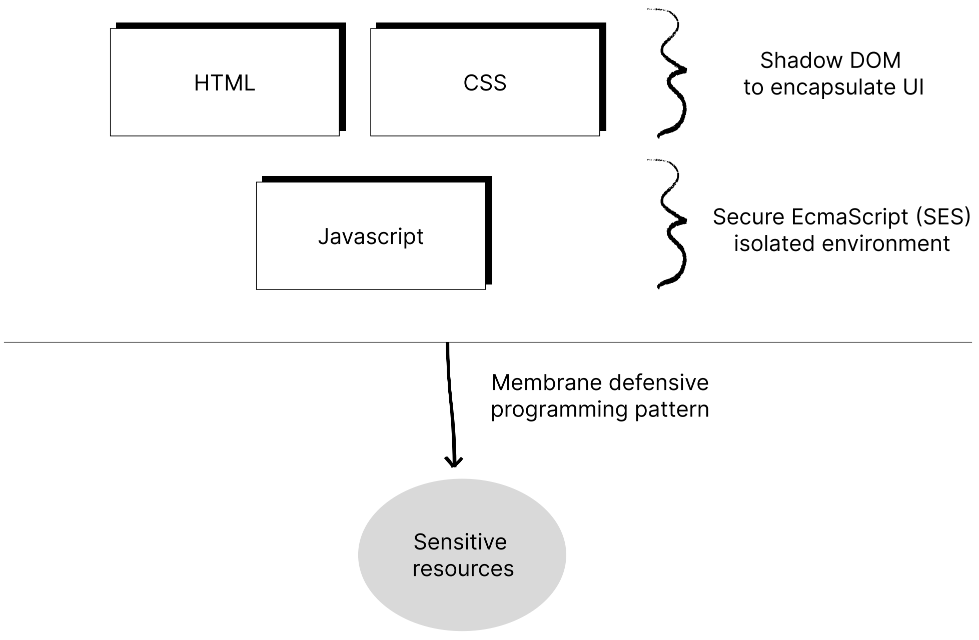

HTML
CSS
Shadow DOM to encapsulate UI
Javascript
Secure EcmaScript (SES) isolated environment
Membrane defensive programming pattern
Sensitive resources

Figure 3: Technologies corresponding to each part of the sandbox model.

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 allows the creation of an isolated UI environment for each app, as also used by Web Components [14]. Each sandbox has access to a shadow root node that it fully controls. Necessary distortions are implemented to stop querying elements outside the shadow root (e.g., via `shadowRoot.ownerDocument`).

3.2.2 For JavaScript

For JavaScript, the sandbox uses the Secure ECMAScript (SES) library by EndoJS [15] to lock down global prototypes (preventing prototype pollution attacks) and create a compartment where access to web APIs is disabled by default and can only be enabled via endowments. Unlike iframes or Web Workers, SES does not require a separate thread to run the code. Instead, the code runs in the same main thread as the platform, making it more efficient and avoiding communication overhead while allowing 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 uses the Membrane defensive programming pattern [16]. The Membrane pattern is simpler than other capability-based security measures, automatically covers all web APIs via deep annotation, and remains theoretically secure [17]. When implemented in JavaScript, the pattern leverages ES6 proxies: 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 [18].

3.3 Developer Experience

We aim to deliver a developer experience that closely resembles a typical web app. To this end, the sandbox takes further steps to ensure a seamless workflow:

- Instead of requiring a manifest file to define which entry points to load, the index HTML file is parsed to find entry points. This means external files are fetched, while inline CSS and JS are sanitized and evaluated.

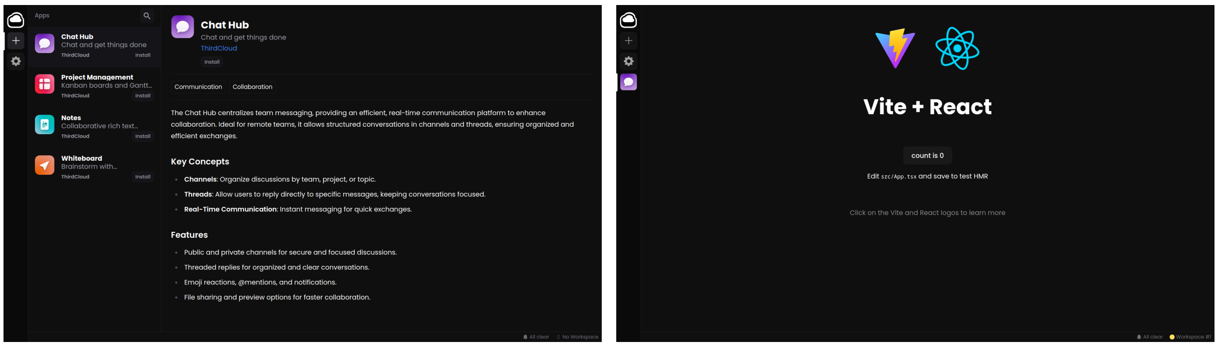
- Dynamic imports and ES modules are supported, as they are the primary mechanism for development servers (e.g., Vite) to load and replace code (e.g., hot module replacement, or HMR). Since JavaScript dynamic imports are part of the JavaScript specification (not a web API), this is implemented by analyzing the source code and replacing the import statement with a function call. This function fetches the file and returns an ES module object.

As a result, the developer experience is virtually identical to that of a normal web app. Developers can utilize familiar tools and frameworks without modification and can seamlessly open a localhost development port, enabling the platform to load the application directly from the local environment.

4 Evaluation

We developed a demonstration collaboration platform to evaluate the performance and security of the sandbox model. This platform served as an experimental environment, enabling us to test various sandbox configurations and assess their effectiveness in isolating application functionality within a controlled and secure framework.

The platform successfully supports loading large-scale applications that utilize a diverse set of web APIs. The sandbox model ensures effective isolation of the application's functionality, limiting its access to a predefined subset of the platform's resources.



The Chat Hub centralizes team messaging, providing an efficient, real-time communication platform to enhance collaboration. Ideal for remote teams, it allows structured conversations in channels and threads, ensuring organized and efficient exchanges.

Key Concepts

- **Channels**: Organize discussions by team, project, or topic.
- **Threads**: Allow users to reply directly to specific messages, keeping conversations focused.
- **Real-Time Communication**: Instant messaging for quick exchanges.

Features

- Public and private channels for secure and focused discussions.
- Threaded replies for organized and clear conversations.
- Emoji reactions, @mentions, and notifications.
- File sharing and preview options for faster collaboration.

Edit `src/App.tsx` and save to test HMR

Click on the Vite and React logos to learn more

Figure 4: Demo platform, with the sandbox model successfully isolates third-party applications

The platform, hosted at https://thirdcloud.org, includes the following features:

- **Development Port Communication**: The platform sucessfully communicates with the development server running on localhost:3000.
- **Universal Framework Compatibility**: It is fully compatible with modern frontend development tools, such as Vite + React template.

- **Hot Module Replacement (HMR)**: The platform supports Hot Module Replacement, allowing for real-time updates during development without requiring full page reloads.
- **Multi-Application Support**: The platform enables the simultaneous execution of multiple applications within distinct sandboxed environments.
- **Security Safeguards**: The platform successfully secures dangerous browser APIs. All APIs tested were successfully distorted.

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 unresponsive to user interactions, such as mouse clicks or keyboard input, during this period.

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).

5 Discussion

The demonstration platform successfully isolates third-party applications, but there are key areas for improvement.

Firstly, one challenge is caching heavy libraries, such as React, shared among multiple apps. Implementing a caching strategy would reduce load times and improve overall performance. Additionally, the SES module transformation is currently slow. Offloading this task to a separate thread could speed up the process and reduce delays during the initial load.

Secondly, we plan to revisit integrating WebAssembly (WASM) with Membrane, which could help overcome WASM's limitation of direct access to the DOM, enhancing its usability for more complex applications.

Thirdly, we aim to define precise data models and access policies. These policies will specify which resources third-party applications can access, with granular permissions based on the application's role or the user's consent. The platform will include UI prompts with visual indicators of the data being requested, enabling users to make informed decisions. This approach will ensure that only necessary resources are accessed while preventing unauthorized interactions or data leaks.

In summary, while the sandbox implementation is effective and meets the defined objectives, there remain opportunities to further enhance its performance, flexibility, and security.

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.

[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] 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