CanvasMirror: Secure Integration of Third-Party Libraries in a WebVR Environment

Jiyeon Lee
School of Computing, KAIST
jy.lee@kaist.ac.kr

***Abstract*—Web technology has evolved to offer 360-degree immersive browsing experiences. This new technology, called WebVR, enables virtual reality by rendering a three-dimensional world on an HTML canvas. Unfortunately, there exists no browser-supported way of sharing this canvas between different parties. As a result, third-party library providers with ill intent (e.g., stealing sensitive information from end-users) can easily distort the entire WebVR site. To mitigate the new threats posed in WebVR, we propose *CanvasMirror*, which allows publishers to specify the behaviors of third-party libraries and enforce this specification. We show that CanvasMirror effectively separates the third-party context from the host origin by leveraging the privilege separation technique and safely integrates VR contents on a shared canvas.**

***Index Terms*—Web security, WebVR, third-party sandboxing**

I. Introduction

WebVR [1] is an HTML5 standard technology that enables a virtual reality (VR) in user browsers. It was introduced in 2015 and aims to provide an integrated VR environment for different browser platforms and operating systems. WebVR works in tandem with WebGL and leverages an HTML5 canvas document object model (DOM) to render VR scenes; this canvas becomes a window displaying a VR world.

Inserting third-party libraries is a common practice in the modern mashup Web environment. For example, a standard website often monetizes its content by renting its screen estates for advertising (ad). For this, the website embeds a JavaScript (JS) library from an ad service provider, and this library leverages an iframe element to display ads and confine the execution of their JS. Considering that advertisers seek opportunities to expose their ad campaigns to large audiences, it is natural for them to search for a means to bring promotional content into VR worlds.

Unfortunately, in WebVR environments, there are no iframe-like primitives that serve as an execution container such that the hosting website cannot alter; it shares a portion of the displayed VR scene. As a result, third-party providers with ill intent (e.g., stealing sensitive information from end-users) can easily distort the entire VR app in their fashion. This WebVR limitation stems from the usage of a canvas element to render VR scenes, thus providing no browser-supported method of sharing this canvas between different web origins [1].

To address the problem posed in the new 3D world on Web, we propose a defense system, *CanvasMirror*, which is designed to support secure integration of third-party's VR objects into the host scene. It aims to provide an interface for publishers to restrict malicious behaviors of third-party JS. Therefore, CanvasMirror prevents third-party libraries from altering the host's contents on the VR scene nor placing the ill contents which are not allowed by a publisher.

CanvasMirror leverages the origin separation approach that moves third-party code to another origin and uses a postMessage, an HTML5 standard API for communications between different origins. Therefore, it increases the cost of communication between the host and the guest page depending on the number of shared objects. In our experimental results showed a delay in a page loading time up to 1.4 seconds and an 8.1 frame-per-second (FPS) drop when we mirrored 15 objects.

[1]In this paper, WebVR refers to a part of WebXR, which is designed for both augmented reality (AR) and virtual reality (VR) on the Web.

II. Defense Design

The vulnerabilities facing WebVR stem from the absence of confining the execution of a third-party JS, which should share a canvas element with its host origin, rendering VR content. There exists a vast volume of studies on sandboxing third-party scripts in traditional Web [2]–[4]. These studies share a common security objective, which prevents guests from manipulating the host's DOMs. However, VR contents drawn in a canvas element are not included in DOMs [5], existing methods do not prevent them. Therefore, we need a new defense mechanism for the 3D environment.

We define the security requirements for a new defense:

1) A defense should ensure the Same Origin Policy (SOP) [1] between a host (or a publisher) and a guest (or a third-party JS); thus the third-party JS should not be able to alter the DOMs in the host page.
2) A defense should ensure that the third-party JS is not able to alter the VR scene (e.g., a camera, controllers) if the first party does not permit to do so.

To this end, we chose a privilege separation technique that assigns each embedded third-party code with a separate origin so that SOP forces the confinement of the third-party code. CanvasMirror generates a guest page within an iframe and loads the required resources onto the guest page. It then renders VR objects on the guest page and mirrors these objects on the host page.

To address the second requirement, CanvasMirror offers a security policy language that the publisher establishes to define how specified third-party scripts should interact with host elements. It also offers a set of JS APIs that enable the

programming of VR functionalities while interacting with VR objects in host pages. Therefore, in this solution, a publisher desiring improved security asks a third-party JS provider to implement its VR contents in CanvasMirror APIs.

III. Architecture

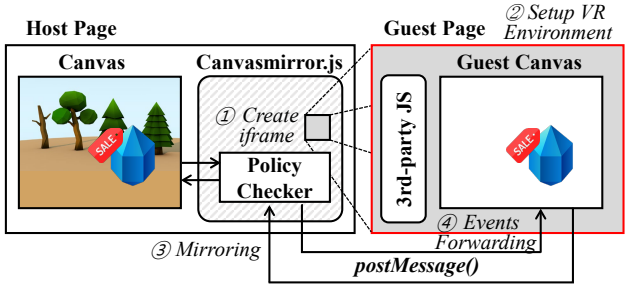


Fig. 1: CanvasMirror overview

The overall architecture of CanvasMirror is demonstrated in Fig. 1. CanvasMirror is a JS library, designed to confine the execution of a third-party script rendering WebVR objects. A publisher furnishes the JS library with a given security policy that specifies how a third-party script should interact with the resources belonging to the first-party origin.

If Canvasmirror.js is successfully loaded on the host page, ① it generates an iframe having a different origin and loads resources needed for a third-party library. ② It then establishes a VR environment and runs a third-party JS code on the guest page. ③ When an addObject() provided by CanvasMirror is called, CanvasMirror mirrors the corresponding object to the host VR scene by using a postMessage() API.④ It listens and passes events (e.g., user interactions) on a mirrored object to execute an event handler that third-party JS defines.

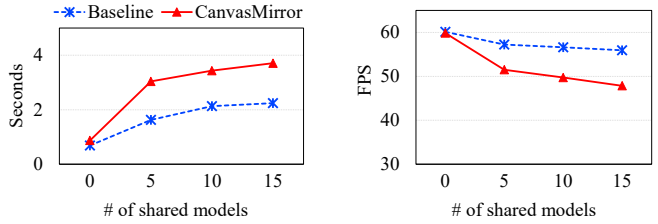
Security Policy. CanvasMirror provides the publisher with an XML format for the policy description. We designed three specific tags; *mirror*, *can-read* and *can-write* for a prototype implementation. A mirror primitive defines a space in which rendering objects in the host scene. A publisher can also specify the objects with can-read and can-write attributes to be accessed by the third-party JS.

Programming Interfaces. CanvasMirror sandboxes a given third-party JS script via providing a limited set of APIs that the third-party code uses to render VR content. Instead of defining a long list of all possible APIs, we focused on defining essential APIs such as *createObject()* and *addObject()* for the prototype. A reference returned from createObject() has a descendant interface called *setAttribute()* which enables to register the properties and event handlers of the created object.

IV. Evaluation

We evaluated the performance overhead of CanvasMirror and compared it with a baseline method. The baseline method is to run a third-party JS without any underlying security defense, thus running it with the same origin as its host.

To understand the overhead of deploying CanvasMirror, we measured variations of page loading time and FPS as we increase the number of shared objects between a host and a guest page. For the host page, we implemented an empty VR world and added a given number of shared 3D models. We measured the page loading time 10 times with the browser cache enabled and reported the average. All experiments were conducted using Firefox 70 on a machine with Intel i9 and GeForce GTX 1070.



(a) Page loading time variations

(b) FPS variations. Note that Firefox caps its FPS to at 60.

Fig. 2: Performance comparison of a testing page while varying the number of shared 3D models.

Fig. 2 shows the experimental results. It demonstrates that the performance overhead of our approach increases with the mirroring of many 3D models. CanvasMirror with 15 objects mirroring showed a 1.4 seconds page loading delay and an 8.1 FPS decrease.

The root cause of this observed overhead is that the privilege separation techniques demand cross-origin communications, which enable the separate origins of third-party code to operate as a single app. Such communications introduce execution latency, thus impeding a stable frame rate.

V. Conclusion

Modern WebVR websites readily provide users unique immersive browsing experiences. However, the usage of canvas elements to render VR content and no browser built-in support of sharing this canvas between different origins leave few options for WebVR developers to run third-party JS libraries. To address this, we proposed CanvasMirror that allows publishers to confine the behaviors of third-party's VR contents according to given security policies.

For a rich user WebVR experience, it is required to improve the performance including the communications costs at runtime. As a future study, we plan to expand the methods [3], [4] that provide sandboxing third-party libraries within the same origin as the hosting page, reducing the communication costs.

References

[1] MDN, Same-Origin Policy (SOP), https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[2] M. T. Louw, K. T. Ganesh, and V. N. Venkatakrishnan, "AdJail: Practical Enforcement of Confidentiality and Integrity Policies on Web Advertisements", Proceedings of the USENIX Security Symposium 2010.

[3] J. G. Politz, S. A. Eliopoulos, A. Guha, and S. Krishnamurthi, "ADsafety: Type-Based Verification of JavaScript Sandboxing", Proceedings of the USENIX Security Symposium 2011.

[4] X. Dong, M. Tran, Z. Liang, and X. Jiang, "AdSentry: Comprehensive and Flexible Confinement of JavaScript-based Advertisements", Proceedings of the Annual Computer Security Applications Conference 2011.

[5] MDN, Document Object Model (DOM), https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model