Unifying Data Policies across the Server and Client

Jonathan Burket, Jenny Cha, Austin DeVinney, Casey Mihaloew, Yuchen Zhou, David Evans
University of Virginia

Introduction

Web application frameworks have greatly simplified web application development. Securing these applications, however, can still be difficult. A single mistake in a single line of code is often enough to leave a site vulnerable to attack. Earlier this year, we published GuardRails, a tool that helps secure Ruby on Rails applications by attaching security policies directly to sensitive data. These policies are currently enforced only on the server-side of the application, and while they successfully protect against a variety of access control and injection attack vulnerabilities, they do not easily integrate with the user interface and offer no client-side security. Our current work focuses on bridging the gap between the server and client side, such that a single set of security policies are enforced uniformly throughout the application. In this poster, we will demonstrate how this integration of policies on the server and client can be used to easily control what content can be accessed by untrusted scripts.

GuardRails

GuardRails is a source-to-source transformation tool for Ruby on Rails that automatically enforces security policies based on developer-provided annotations. These policies are attached directly to the data, allowing for automatic enforcement of access controls and protection against SQL injection and cross-site scripting with dynamic taint tracking. By requiring the developer to specify security policies in simple annotations in a single location, GuardRails simplifies the usually tedious and error-prone process of adding security checks throughout the application.

Reaching the Client

GuardRails is currently limited to enforcing policies on the server side of the application. In many cases, however, it is valuable to have security policies on the client as well. Consider the case where a web developer wants to include script from an outside source (an ad, for example) in a page sent to the user. The developer can either run the script with full access to the DOM, which might expose private user information contained on the page, or isolate the script in an *iframe*, which blocks the script from accessing the DOM altogether. Since the script is untrusted, it is risky to give it access to the entire DOM, but it is often the case that the script should be allowed to see or modify specific portions of the page, as with ads personalized to the page the user is looking at.

We offered a way of balancing these two interests in paper to be presented at *ESORICS 2011* [2]. The approach is using a modified Chromium browser that reads and uses specially marked HTML tags. These tags have attributes that partition scripts into certain worlds, and other tags as readable or writable by these worlds. This approach leads to the very fine-grained ability to declare sensitive data as private to some scripts, but public to others.

Knowing the syntax needed by the modified browser, we can extend GuardRails to automatically use its annotations and policies to generate the modified HTML. From this, the developer has now ensured the security of the data throughout the entire data flow by using the combination of GuardRails annotations paired with a modified client side tool.

The Poster

In the poster we will be showcasing the GuardRails framework and elaborate on its security annotations. In addition we will be including information about the modified Chromium browser and how it processes the altered HTML. We hope to give the viewers an understanding of their interaction and an idea of the direction of future research.

1. Yuchen Zhou, David Evans. “Protecting Private Web Content from Embedded Scripts,” in *ESORICS 2011*.
2. Jonathan Burket, Patrick Mutchler, Michael Weaver, Muzzammil Zaveri, David Evans. “GuardRails: A Data-Centric We Application Security Framework,” in *WebApps 2011*.