Protecting Browsers from Cross-Origin CSS Attacks

Lin-Shung Huang
Carnegie Mellon University
linshung.huang@sv.cmu.edu

Zack Weinberg
Carnegie Mellon University
zack.weinberg@sv.cmu.edu

Chris Evans
Google
cevans@google.com

Collin Jackson
Carnegie Mellon University
collin.jackson@sv.cmu.edu

ABSTRACT

Cross-origin CSS attacks use style sheet import to steal confidential information from a victim website, hijacking a user's existing authenticated session; existing XSS defenses are ineffective. We show how to conduct these attacks with any browser, even if JavaScript is disabled, and propose a client-side defense with little or no impact on the vast majority of web sites. We have implemented and deployed defenses in Firefox, Google Chrome, and Safari. Our defense proposal has also been adopted by Opera.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security

Keywords

CSS, Content Type, Same-Origin Policy

1. INTRODUCTION

The World Wide Web was originally envisioned [5] as a means to collate a wide variety of human-readable, static documents, present them via a unified interface, and facilitate browsing through them by searching or via inter-document references. It has grown into a versatile platform for all kinds of computing tasks, progressively gaining support for data entry, client-side scripting, and application-specific network dialogues. Web-hosted applications have supplanted traditional desktop applications for almost everything that requires network communication, and are becoming competitive in other areas.

The *same-origin policy* [23] is the basic principle used to secure Web applications from each other. An HTML document can include many sorts of content—including images, scripts, videos, and other documents—from any site. However, the document's scripts may not directly examine content loaded from other sites. This policy applies even within what appears to the user to be one unified page; for instance, a script can only inspect the content of a nested document if it came from the same origin as the script itself. Cross-origin content inclusion allows sites to share popular script libraries and store large, rarely-changing content on servers dedicated to that purpose, while preventing malicious sites from reading content that should be visible only to the user.

Cascading style sheets (CSS) are another type of content that a document may include; they define appearance, just as HTML defines content and JavaScript defines behavior. CSS is a relative late-comer to the Web; although the need for a style sheet language was recognized as early as 1993, the first specification of CSS dates to 1996, and the earliest browser to implement enough of CSS to be generally useful was Internet Explorer 6.0, in 2001. [20]

To allow future extensibility, the CSS specification mandates *error-tolerant parsing*. Browsers skip over CSS directives they cannot interpret, while continuing to honor what they do understand. [26] These rules allow web designers to build sites that take advantage of the very latest CSS features but "degrade gracefully"and remain usable with older browsers. Unfortunately, error-tolerant parsing can find valid CSS constructs in an input stream that was not intended to be CSS at all; for instance, in an HTML document.

This leads to a security hole, first described (to our knowledge) in 2002 [13] and rediscovered at least twice since then [11, 22]. If a malicious site can inject chosen strings into a target webpage (whose structure, but not specific contents, are known) and then load that page as a style sheet, it can extract information from the page by examining what the CSS parser makes of this "sheet." The attack works even if the target page cannot be retrieved without presenting login credentials, because the browser will present any credentials (e.g. HTTP cookies) it has stored for the target server when it does the load. To date, all published attacks of this type have required JavaScript, and most have been specific to Internet Explorer.

In this paper, we present a general form of this attack that can be made to work in any browser that supports CSS, even if JavaScript is disabled or unsupported. We do not consider this vulnerability to be merely a bug in the CSS specification, but rather a general problem with allowing an including page to override the content type of a cross-origin resource: browsers should obtain independent confirmation that an included resource is appropriate in context before