

Cross-Browser Security Testing: Running WebKit Regression Tests on Mozilla Firefox

James Ren Adam Barth Collin Jackson Dan Boneh
{james.ren, abarth, collinj, dabo}@cs.stanford.edu

ABSTRACT

WebKit, an open-source web browser engine, contains many automated tests that verify the browser's security model. Such tests are very useful for finding new security problems and for ensuring that newly written code does not cause a regression. Moreover, anyone can write a new test and have it added to the test suite. However, many of these tests rely on the presence of certain objects in the browser window, which are normally added by the WebKit framework; these tests will only run correctly on Safari. We have developed a browser extension for Mozilla Firefox that emulates these objects and allows the tests to run to completion on Firefox, and we used this extension to identify new security issues in the Firefox browser.

1. INTRODUCTION

Web browsers must obey the same-origin policy when working with JavaScript code. That is, with one notable exception, JavaScript code can only manipulate objects that originate from the same source. Mozilla interprets this requirement as meaning that the protocol, port, and host must be the same for all pages for the JavaScript code to manipulate objects across those pages.

The specification of the same-origin policy does allow for a special exception, however. Pages are allowed to set their domains to be a suffix of their current domain. For example, a page at <http://banking.example.org> can set its domain to be <http://example.org> and access any objects in that domain. However, the reverse cannot happen; <http://example.org> cannot set its domain to be <http://banking.example.org>. Similarly, <http://banking.example.org> cannot set its domain to be <http://finance.example.org>.

Violations of the same-origin policy very often lead to exploitable cross-domain scripting holes. In cross-domain scripting, JavaScript in one domain can manipulate and entirely compromise a page located at another domain. For example, a cross-domain script can instruct the browser to send the cookie for the targeted page to an attacker, allowing the attacker to hijack the user's session at the target page. Even worse, an attacker could modify a username/password field so that instead of simply logging the user in, it sends the username/password combination to the attacker first, allowing the attacker to impersonate the victim and defeating any token system designed to prevent cross-site request forgery.

To help defend against flaws in browsers' implementations of the same-origin policy, developers can run regression tests whenever a browser is revised. Regression tests are usually part of an automated test suite, and they test various features of the browser including basic functionality and information security. Of particular interest to us is the WebKit regression test repository, which contains a suite of approximately 300 tests that specifically check the security of the browser.

In order to perform these tests, the WebKit framework exposes certain objects to the browser window. The tests can then access these objects and use them to perform actions that standard JavaScript would not normally be able to perform. For these tests to run on Firefox, much of this background work that WebKit performs before and during the tests would need to be emulated on Firefox.

2. REQUIREMENTS

The most important requirement that needed to be translated is the ability to automatically run all the tests and check for deviations from the expected results. In order for this feature to exist, the browser must be able to produce a text dump of the current page. A script

can then compare the dumped files with a set of expected results and notify the user of any discrepancies. In the WebKit test suite, these text dumps contain the on-screen text on the page, the notifications from any `alerts` that the document called, and any messages that were propagated to the error console. Each test comes with a text file containing the expected dump, which is `diffed` with the actual text dump after every test.

The WebKit framework uses a `DumpRenderTree` utility to create these text dumps. Normally, the utility will produce a text dump of the render tree, which corresponds to a representation of the page that the internal rendering engine uses. These types of dumps are primarily used in the WebKit test suite to verify the consistency of the visual renders – the sizes of text blocks, positioning of rendered objects, and other visual elements. The utility can also produce strictly text-based dumps, without any render information.

Of the objects that WebKit exposes, `window.layoutTestController` is the most important for producing text dumps. Most of the tests rely on certain functionality of this JavaScript object in order to produce the correct dump at the correct time. Some relevant methods of `layoutTestController` are given below.

`dumpAsText`

Takes no arguments and produces no return. Instructs the `layoutTestController` to dump the following page as text rather than as a render tree.

`waitUntilDone`

Takes no arguments and produces no return. The text dump is usually produced when the page signals that it is finished loading. However, there are cases where the text dump should be produced later. This method instructs the `layoutTestController` to dump when it receives the `notifyDone` call rather than after page load.

`notifyDone`

Takes no arguments and produces no return. Instructs the `layoutTestController` to produce the text dump now, if `waitUntilDone` has been called previously.

`windowCount`

Takes no arguments. Returns the number of open browser windows (or tabs). Some security tests use this method to make sure all child windows are closed before it tries to dump the text.

In addition, there is one particularly important property in the `layoutTestController`: the `globalFlag`. This property is persistent across all windows and frames, so frames and windows can set the state of the flag, and if the state changes, they can also observe the change. Because the same-origin policy is supposed to prevent cross-domain frames and windows from communicating with each other, this global flag is one of the few ways that messages can be passed across domains.

If these methods are implemented, the emulated `layoutTestController` can attain enough functionality to produce useful text dumps of most of the tests.

3. IMPLEMENTATION

We emulated the `layoutTestController` in Firefox by means of a browser extension. The extension is registered as a chrome content provider in the browser. The extension should be installed to a separate user profile in Firefox, not to the default profile. The content is split into two JavaScript files. One parses the two command line options for the extension. The second implements the functionality of the `layoutTestController` and adds it to the browser's DOM window object.

Additionally, we made the decision to disable the browser from accessing any remote files while the `layoutTestController` emulator is running. We chose to do this because much of the extension's functionality comes from its being able to perform privileged actions, like creating and writing to files on disk. However, allowing the extension to access the remote documents opens the possibility of security breaches through the extension. This restriction is the primary reason for our suggesting that users of the extension install it to a separate profile.

The security tests require that frames and opened windows be of a different origin, in order to test the same-origin policy. Hence, the tests cannot be loaded into the browser as HTML documents from the local file system. Rather, they must be hosted on an HTTP server in order to function properly. The tests assume that the main document is being accessed through `http://127.0.0.1:8000`, since they create frames and open windows that use `http://localhost:8000`. Note that, while these two locations both refer to the local machine, the same-origin policy considers them to be separate domains.

In addition to creating the `layoutTestController` object in the Firefox DOM window, we also needed to modify some of the WebKit security tests themselves. Most of the modifications served to work around a difference between WebKit's and Firefox's security model. In WebKit, a security error in JavaScript code simply logs an error to the developer console and continues executing as if the call had succeeded. In Firefox, however, the code throws an exception, which usually bubbles up to the error console and halts execution of the JavaScript code. Because of these different security models, some of the tests do not run to completion on Firefox, since the test assumes that any security errors will be logged and subsequently ignored. Therefore, we had to add `try/catch` blocks around each line that is supposed to cause a security violation.

Much of the remaining modifications were necessary to work around a peculiarity related to JavaScript function hoisting in Firefox (See Section 4).

The extension accepts `-close-after-dump` and `-timeout` as options from the command line. Both of these options change the behavior of the extension in order to facilitate the automatic dumping of the tests. The flag `-close-after-dump` instructs the extension to close the browser window after producing a text dump. The flag `-timeout` instructs the extension to close the browser after a specified timeout, which is currently set at five seconds. We chose this particular timeout period because, if a test has not produced a text dump after five seconds, we are reasonably confident that it will not execute to completion in Firefox under the current version of the `layoutTestController` emulator.

These command line options are primarily for use by the shell script that loads every test into the browser. The script takes three arguments. The first is the path to the Firefox executable. The second is a directory containing all the tests. Typically, this points to the directory that the HTTP server (e.g. Apache) is using to host the files. The final argument is the path to the Firefox profile under which the extension is installed.

The script runs through each file in the test directory and loads it in the browser through the HTTP server. It also sets the two command line options above. When a file is finished dumping or has timed out after five seconds, the browser closes and the script loads the next file in the browser. Typically, running all of the approximately 300 security tests takes less than five minutes.

We did not emulate every method of WebKit's `layoutTestController` in the Firefox extension. The primary reason for this decision was that some methods of the `layoutTestController` provide little to no benefit in terms of coverage on the security tests.

Methods such as `testRepaint` were never called in the security tests, nor would there be much of a reason to call these methods in a future security test, so implementing them would neither help us run the existing tests in Firefox, nor would it increase compatibility with future tests.

Additionally, the extension is only capable of producing strictly text-based dumps. That is, the Firefox extension cannot produce any render tree information. There were two reasons for not including this functionality. First, the additional information that the render tree gives us relates to the positions and sizes of the elements on the webpage. Testing the security of the browser, however, requires only that we know the content of the webpage displayed, not the appearance of the page; the security tests do not require render tree information to decide on success or failure. Second, Firefox uses a rendering engine different from WebKit's; Firefox does not use a render tree as its internal representation. Dumping Firefox's analogue to WebKit's render tree would produce completely different results. Hence, implementing this functionality would only produce results that are very difficult to compare to WebKit's expected results for tests that use the render tree dumps, and would not have any effect on testing Firefox's security, since all the security tests instruct the `layoutTestController` to produce strictly text-based dumps.

4. EVALUATION

As a result of running the existing tests, we have discovered two problems in Firefox. The first deals with function hoisting in the browser's implementation of JavaScript. Under normal circumstances, functions defined in the global scope are hoisted to the top, so that defining a function underneath the first call to it does not produce an error. However, Firefox's hoisting behavior on functions defined in blocks – inside at least one set of curly braces – does not seem consistent in our opinion. We believe the most reasonable action here would be to hoist

the function to the top of its defining scope; Internet Explorer, Safari, and Opera all match our expected behavior. However, neither Firefox 2 nor Firefox 3 does any hoisting for functions defined within blocks; they only hoist functions defined in the global scope. This discrepancy is filed as a bug against Mozilla, at https://bugzilla.mozilla.org/show_bug.cgi?id=434912.

The second issue we found with Firefox is a security-related bug. The bug is filed at https://bugzilla.mozilla.org/show_bug.cgi?id=434898. Unfortunately, because the bug is security-sensitive, the details about it are still being kept under restricted access until the issue is addressed.

The rest of the issues that the tests identified were differences in the security model that did not result in functionality differences or security problems. The browsers' behaviors upon encountering a security error were addressed above in Section 3. The extension also identified several differences between Firefox's and WebKit's security model in addition to this one. For example, Firefox allows pages to obtain references to data URLs – those of the form `data://... –` across domains, but throws an exception when the page tries to read or modify any of the properties of that reference; WebKit simply denies access to referencing the data URL across domains.

Similarly, Firefox 2 allows pages to see that cross-domain `document` objects contain a `toString` method and to retrieve a reference to that method. The browser then throws an exception if the page tries to call that method across domains. As before, WebKit simply denies access entirely, even to retrieving the reference. However, Firefox 3 behaves in the same manner as WebKit with regard to `document.toString`: it also denies access to retrieving the reference.

In some cases, Firefox 3 is more restrictive than both Firefox 2 and WebKit. In Firefox 2 and in WebKit, pages are allowed to get the `window.toString` property across domains. Firefox 3, conversely, throws an exception if pages attempt to perform this access.

5. RELATED WORK

Sylvain Pasche is working on a central repository for reporting the results of regression tests on multiple browsers, which he has posted at <http://browsertests.org>. The repository contains the results of running over 8000 tests across five different browser configurations. The results for each test show whether the test passed, failed, or caused the browser to hang or crash. Unfortunately, because of the massive scope of the project, there is little specialization that can be done in terms of increasing the number of non-hanging and non-crashing tests on each browser. Therefore, each test is run as-is on each browser, with no modifications to either the test or the browser.

The scope of our project is considerable narrower than Pasche's. We are running a particular suite of about 300 tests on a single browser, but our goal is to approach as close to 100% coverage as feasible.

6. FUTURE PLANS

The extension still needs a little more work before it reaches a satisfactory level of test coverage. Once it does reach this level, however, we hope to be able to contribute this to Mozilla and add it as part of the automated testing harness for Firefox. Then, whenever there is a new version of the Firefox browser, these tests can be run to verify that the revision does not introduce any new security problems.

We would also like to investigate the possibility of running more cross-browser configurations, rather than just WebKit tests on Firefox. In particular, we want to run Mozilla's tests on WebKit, and to run the WebKit tests on Microsoft Internet Explorer.

7. CONCLUSIONS

For the most part, Firefox does not exhibit any of the security holes for which WebKit's test suite checks. There is only one security-sensitive problem, but we do not think it is exploitable. The JavaScript implementation issue affects only functionality, not security. The other discrepancies are choices made when designing the security model; while WebKit's and Firefox's handling of some security issues differ, these differences do not affect the security of either browser.