Site	False positives	Prevented
Amazon	0	N/A
CNN	0	N/A
Craigslist	0	N/A
Facebook	0	N/A
Google	0	N/A
Top 100 sites	0 (14)	N/A
DOM-based XSS attacks	N/A	10/10

Table 2: The taint tracking policy prevents all of the attacks found and has zero false positives on our test set and the top 100 sites reported by Alexa. Fourteen warnings are generated indicating that tainted data was written to the web page but not executed by the JS engine.

Site	Reported	Prevented	Percent
Amazon	4	3	75%
CNN	11	8	73%
Craigslist	20	20	100%
Facebook	11	9	82%
Google	9	8	89%

Table 3: The automatic DSI policy prevents many of the XSS vulnerabilities reported for the sites shown. The reported XSS column is the total XSS attacks examined and the prevented column is the number prevented by automatic DSI policy.

teractive browsing sessions for these pages since `archive.org` does not run the server-side components to the web applications. We found that all of the automatic DSI policies we generated introduce no more changes than we found on current pages (Section 5.1).

5.2 Security analysis

DOM-based XSS prevention. We have tested our taint-tracking policy against publicly disclosed attacks and examples of DOM-based XSS attacks. Table 2 shows the results of testing this policy. We test against five documented XSS attacks disclosed for popular services. We also generate five synthetic attacks based on examples used to demonstrate DOM-based XSS attacks. For both sets of attacks our policy successfully prohibits the execution of the injected script.

Automatic DSI. We have evaluated our automatically created document structure policies against a number of previously documented XSS attacks by XSSed.org [27]. Table 3 presents the results of our evaluation. For each site we examine the most recent reported XSS vulnerabilities and test the vulnerable page using our automatically generated policies for each site. A few reported attacks for each site failed to demonstrate an exploit even on an unmodified browser and those have been removed from the data set. There were no archived attacks for `en.wikipedia.org` on XSSed.org so we have omitted it from Table 3. Additionally, the XSS vulnerabilities reported for Craigslist all exploited the same bug but at different subdomains.

Our policies can prevent many forms of XSS and inclusion attacks though there are still possibilities for attackers succeed. Our automatic DSI policy does not inspect the contents of script elements and an attacker can insert malicious content that mimics the same document structure as the benign pages. One of the Facebook XSS attacks that was not prevented, injected an inline <`script`> into the header, a location where the policy allowed script elements.

5.3 Performance

To evaluate the performance of Alhambra, we isolate the effects of the taint-tracking modifications and structural policy enforcement. To support taint-tracking policies the only overhead added is the manipulation of the extra field to propagate taint in JavaScript objects or DOM nodes. Our performance tests indicate that there are no measurable latencies added to support taint tracking.

Enforcing document structure policies has two different points where latency can be introduced. The first is during a change to the document causing the structural policy to be checked. The second is during JavaScript accesses to DOM methods and properties. We add no measurable overhead for the applications we have tested except for Facebook, which adds around 2x overhead during the parsing stage and none during execution. Most of the overhead incurred by enforcing policies is seen during the parsing phase when the document is built because for each addition to the document tree, the HTML engine checks to ensure that the addition does not cause a policy violation. In our current implementation we check the entire document each time. A simple optimization would be to check if *only* the added element violates our security policy, reducing the overhead significantly.

6. RELATED WORK

The closest work to ours are other systems that use testing and replication to detect problems with security policy. Doppleganger [20] is one such system that applies fine-grain policies to cookies and uses a parallel browser for backup when a cookie policy is unknown. When Doppleganger does not have a cookie policy, two browsing sessions are maintained, one with a restrictive cookie policy and one without. If differences in the page are detected the user is prompted to choose the desired functionality. Doppleganger uses a simple replay system when not mirroring the session to generate the page. In contrast, our reply system removes server and client-side non-determinism and uses replay to detect modifications made by security policies.

XSS prevention. There has been extensive research in preventing and detecting XSS attacks. In a recent work, Nadji et al. use document structure information provided by the server and client side taint-tracking mechanisms to enforce the provided document structure integrity properties during page execution [16]. Similarly, we use concept of document structure to mitigate attacks though we are able to do so by determining the structure automatically and without added server help. Our techniques also require no changes to the contents of pages.

NoScript [13] is a Firefox extension that includes limited XSS prevention and other useful security features. The current version of the extension supports automatic prevention of many reflected XSS attacks by inspecting and sanitizing parameters in the URL. In addition to reflected XSS attacks, NoScript has the potential to block almost any XSS attack since it can selectively enable and disable JavaScript for parts of a page though this must be configured manually. Like NoScript, the IE 8 XSS filter performs heuristic matching on URL contents to prevent reflected XSS attacks.

Different approaches for client side prevention include filtering, taint tracking and content blocking. Noxes [11] is a client-side proxy that filters outgoing network requests based on a dynamic white list assembled for each page. Noxes primarily targets information leakage attacks and could experience high incompatibility due to the aggressive denial of dynamically created network requests. Vogt et al. use client-side taint-tracking to identify requests that leak sensitive information across domain boundaries [24]. Their technique works inside of the browser and pro-