Flexible Access Control for JavaScript

Christian Hammer

CISPA, Saarland University
Campus E1.1
66123 Saarbrcken
hammer@cs.uni-saarland.de

Extended Abstract

Many popular Web applications mix content from different sources, such as articles coming from a newspaper, a search bar provided by a search engine, advertisements served by a commercial partner, and included third-party libraries to enrich the user experience. The behavior of such a web site depends on *all* of its parts working, especially so if it is financed by ads. Yet, not all parts are equally trusted. Typically, the main content provider is held to a higher standard than the embedded third-party elements. A number of well publicized attacks have shown that ads and third-party components can introduce vulnerabilities in the overall application. Taxonomies of these attacks are emerging [JJLS10]. Attacks such as *cross site scripting, cookie stealing, location hijacking, clickjacking, history sniffing* and *behavior tracking* are being catalogued, and the field is rich and varied.[1]

This paper proposes a novel security infrastructure for dealing with this threat model. We extend JavaScript objects with *dynamic ownership* annotations and break up a web site's computation at ownership changes, that is to say when code belonging to a different owner is executed, into *delimited histories*. Subcomputations performed on behalf of untrusted code are executed under a special regime in which most operations are recorded into histories. Then, at the next ownership change, or at other well defined points, these histories are made available to user-configurable *security policies* which can, if the history violates some safety rule, issue a *revocation* request. Revocation undoes all the computational effects of the history, reverting the state of the heap to what it was before the computation. Delimiting histories is crucial for our technique to scale to real web sites. While JavaScript pages can generate millions of events, histories are typically short, and fit well within the computation model underlying Web 2.0 applications: once the history of actions of an untrusted code fragment is validated, the history can be discarded. Histories allow policies to reason about the impact of an operation within a scope by giving policies a view on the outcome of a sequence of computational steps. Consider storing a secret into an object's field. This could be safe if the modification was subsequently overwritten and replaced by the field's original value. Traditional access control policies would reject

[1] `www.webappsec.org/projects/threat`, `www.owasp.org/index.php/Category:Attack`.

the first write, but policies in our framework can postpone the decision and observe if this is indeed a leak. While policies of interest could stretch all the way to dynamic information flow tracking, we focus on access control in this talk and present the following contributions [RHZN+13]:

- *A novel security infrastructure:* Access control decisions for untrusted code are based on delimited histories. Revocation can restore the program to a consistent state. The enforceable security policies are a superset of [Sch00] as revocation allows access decisions based on future events.
- *Support of existing JavaScript browser security mechanisms:* All JavaScript objects are owned by a principal. Ownership is integrated with the browser's same origin principle for backwards compatibility with Web 2.0 applications. Code owned by an untrusted principal is executed in a controlled environment, but the code has full access to the containing page. This ensures compatibility with existing code.
- *Browser integration:* Our system was implemented in the WebKit library. We instrument all means to create scripts in the browser at runtime, so if untrusted code creates another script we add its security principal to the new script as well. Additionally, we treat the `eval` function as untrusted and always monitor it.
- *Flexible policies:* Our security policies allow enforcement of semantic properties based on the notion of security principals attached to JavaScript objects, rather than mere syntactic properties like method or variable names that previous approaches generally rely on. Policies can be combined, allowing for both provider-specified security and user-defined security.
- *Empirical Evaluation:* We validated our approach on 50 real web sites and two representative policies. The results suggest that our model is a good fit for securing web ad content and third-party extensions, with less than 10% of sites' major functionality broken. Our policies have successfully prevented dangerous operations performed by third-party code. The observed performance overheads were between 11% and 106% in the interpreter.

References

[JJLS10] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in JavaScript web applications. In *CSS*, pages 270–283. ACM, 2010.

[RHZN+13] Gregor Richards, Christian Hammer, Francesco Zappa Nardelli, Suresh Jagannathan, and Jan Vitek. Flexible Access Control for JavaScript. In *OOPSLA*, pages 305–322. ACM, October 2013.

[Sch00] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3:30–50, February 2000.