25 Million Flows Later - Large-scale Detection of DOM-based XSS

Robert Krzysztof Robert Noparlik

October 30, 2023

1 Summary

The paper's authors developed a fully automated method to find and exploit DOM-based XSS vulnerabilities. Their project consists of a modified Chromium browser, a Chrome extension and a control backend. This browser's V8 engine (JavaScript interpreter) and DOM implementation have been modified with taint tracking to detect exploits. When a piece of tainted data reaches a *sink*, the browser calls a JS function provided by the extension. By knowing the context in which an exploit occurred, the system is able to construct a successful exploit. The Chrome extension scrapes the website for new links, provides a callback for the browser to call in case of an exploit detection and, additionally, splits the links received from the control backend between tabs to parallelize the work. The control backend maintains a queue of websites to check.

Authors detected XSS vulnerabilities for 9.6% of the Alexa Top 5000 sites. The exploit information return by the system of the site's domain, a break out sequence, which is a character sequence to allow the attacker to exit the context the code was run in and code type and code location.

2 Pros

- Modifying the browser frees the authors from the hidden complexities of parsing existing mainstream languages, while making taint-tracking work for the whole language.
- Precise access to the execution context allows the authors to easily determine the break out sequence.
- Compact taint tracking information.

3 Cons

- Requires a using modified Chromium browser.

- While the findings might've been useful in 2013, modern web development rarely consists of calling raw DOM methods and reactive javascript libraries/frameworks automatically take care of sanitization without any input from the programmer.
- Potential for a false alarms: *window.header* alarms did not indicate a XSS vulnerability (but, nethertheless, were potential security issues and programming errors).

4 Meaning of the paper

This project has a few key advantages over other, similar ones. One is that the taint tracking is done by the actual JS engine and DOM implementation. This means, that the taint tracking part is not affected by any syntactical changes and only minimally affected by semantical ones, since the only parts that are modified are objects and strings (to track the tainted parts) and sources/sinks, which taint and report the tainted values respectively. This is a massive advantage, since parsing HTML according to specification has become a Herculean task as the spec became more complex. The other advantage is that exploit generation is fully deterministic, because generating a break-out sequence becomes fairly simple once the execution context is taken into account (and is fully parsed). One minor advantage is that taint information the authors presented is stored in a fairly compact manner, meaning that it can potentially be integrated into a release version of the browser with minimal performance and memory overhead.

5 Discussion and Questions about the paper

Ignoring the optimized macros inside V8, how large are the changes to the JS engine and DOM implemententration? Would it be possible to push the commits downstream from the main repository?