Precise client-side protection against DOM-based Cross-Site Scripting

USENIX Security 2014, San Diego
Ben Stock (University of Erlangen-Nuremberg)
Sebastian Lekies, Tobias Müller, Patrick Spiegel, Martin Johns (SAP AG)

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

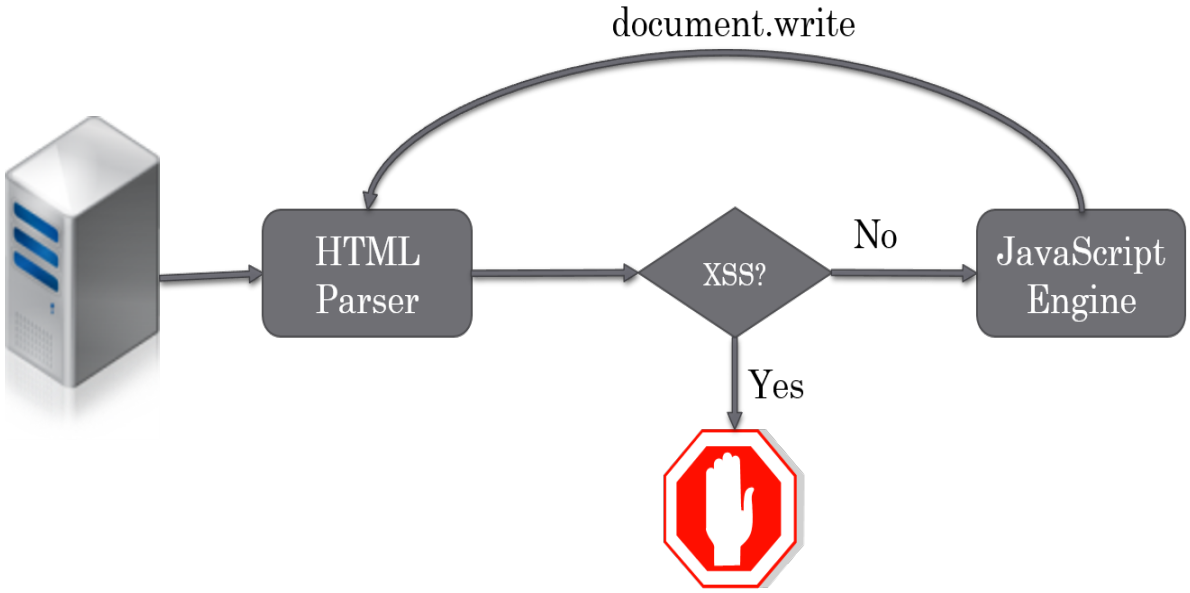
DOM-based Cross-Site Scripting

- All kinds of XSS vulnerabilities that are purely inside client-side code
  - both "reflected" (e.g. extracting part of the URL)
  - ... and stored (e.g. localStorage)

Source: http://blogs.sfweekly.com/thesnitch/cookie_monster.jpg

SotA in XSS filtering: XSSAuditor

- Deployed in all WebKit/Blink-based browsers
- Located inside the HTML parser
  - whenever dangerous element/attribute is found, search for "payload" in request



DOM-based XSS in the wild and effectiveness of countermeasures

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Finding DOMXSS at scale (CCS 2013)

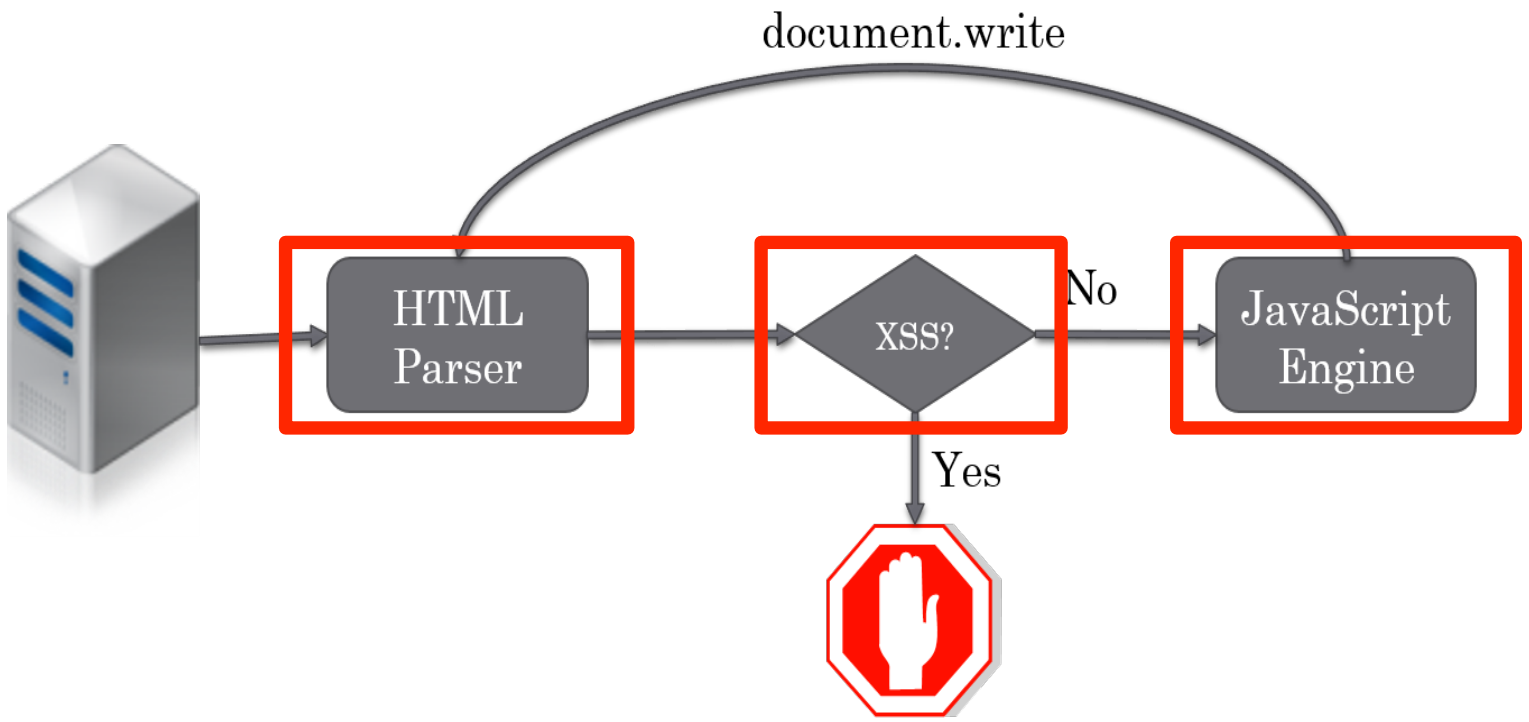
- using byte-level taint tracking in Chromium
  - precise source information for every character
  - patched sinks (e.g. document.write or eval)
- Chrome extension to crawl given set of Web sites
  - and act as interface between taint engine and backend
- and an exploit generator
  - using precise taint information
  - and HTML and JavaScript syntax rules
  - to generate exploits fully automatic

DOMXSS in the wild

- CCS 2013
  - Alexa Top5k, one level down from homepage
  - ➔ **480** domains vulnerable

- This talk (moar crawling power)
  - Alexa Top10k, two levels down from homepage
  - ➔ **958** domains with **1,602** unique vulnerabilities
  - with disabled XSSAuditor

Bypassing the XSSAuditor



Bypassable exploits

- **776** out of **958** domains bypassable
- **1,169** out of **1,602** vulnerabilities bypassable

➔ State of the Art XSS filter cannot protect against DOM-based XSS*

* was not necessarily designed that way, though

Our proposed solution

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

The hard life of a reflected XSS filter

- XSS abstracted: user-provided **data** ends up being interpreted as **code**
  - same for SQLi, CMDi, ..
- XSS filter's problem: find this code among all the other code
  - string matching to approximate **data flow**

Our proposal

- Approximation unnecessary imprecise for local flows
  - we can use taint tracking
- XSS boils down to being **JavaScript** execution
  - build filter into JavaScript engine
- XSS means that **data** ends up being interpreted as **code**
  - allow user-provided data only to generate Literals (Numeric, String, Boolean)
  - **never** anything else

Our proposal exemplified

```
var userinput = location.hash.slice(1)
eval("var a='" + userinput + "';")
```

Userinput: `userdata`

```
Declaration
   Identifier: a
   StringLiteral: 'userdata'
```

```
var a='userdata';
```

Userinput: `userdata'; alert(1); //`

```
Declaration
  Identifier: a
  StringLiteral: 'userdata'
ExpressionStmt
  Type: CallExpression
  Callee:
    Identifier: alert
    Arguments:
      Literal: 1.0
```

```
var a='userdata';
alert(1); //'
```

Policies

- No **tainted value** may generate anything other than a **Literal** in the JavaScript tokenizer
- No element that can reference an **external resource** may have **tainted origin**(e.g. script.src or embed.src)
  - enforced in the HTML parser and DOM bindings
  - single exception to rule: SAME origin as current page

Evaluation

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

False negatives

- Took known vulnerabilities
  - ... with matching exploit URLs
- Disabled the XSSAuditor
  - ... to avoid interference

- Caught every exploit

False positives

- Compatibility crawl of Alexa Top10k with policies in place
  - **981,453** URLs, **9,304,036** frames

Blocking component	documents
JavaScript	5,979
HTML	8,805
DOM API	182
Sum	14,966 (0.016%)

False positives

- Compatibility crawl of Alexa Top10k with policies in place
  - **981,453** URLs, **9,304,036** frames

Blocking component	documents	domains
JavaScript	5,979	50
HTML	8,805	73
DOM API	182	60
Sum	14,966 (0.016%)	183 (1.83%)

False positives

- Compatibility crawl of Alexa Top10k with policies in place
  - **981,453** URLs, **9,304,036** frames

Blocking component	documents	domains	exploitable domains
JavaScript	5,979	50	22
HTML	8,805	73	60
DOM API	182	60	8
Sum	14,966 (0.016%)	183 (1.83%)	90

Performance

- Evaluation using standard benchmarks
  - Dromaeo, Octane, Kraken, Sunspider
- Two modes (benchmarks usually don't use tainted values)
  - normal operation
  - all strings tainted
- Overhead between **7 and 17%**
  - optimization possible

Conclusion

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

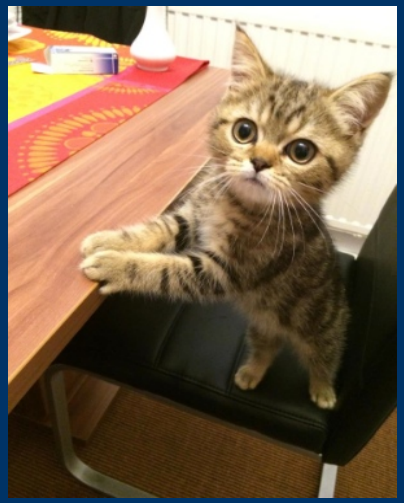
TECHNISCHE FAKULTÄT

Conclusion

- SotA filters can be bypassed for DOM-based XSS
- We propose filter inside JavaScript parser
  - using precise taint information, allowing only tainted Literals
  - **No false negatives**
  - **Low false positives**
    - "XSS by design"
    - untaint API built in
  - **performance impact exists**
    - optimizations possible
    - deployable next to the Auditor if optimized

Thank you

Questions?



@kcotsneb

ben@kittenpics.org

https://kittenpics.org

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT