Call To Arms:

A Tale of the Weaknesses of Current Client-Side XSS Filtering

Martin Johns, **Ben Stock**, Sebastian Lekies

About us

- Martin Johns, Ben Stock, Sebastian Lekies
- Security Researchers at SAP, Uni Erlangen and Google
- More and stuff at http://kittenpics.org

About this talk

- Results of a practical evaluation of client-side XSS filtering
- Technical analysis of the Chrome XSS filter
- Presentation of various techniques to bypass the filter

Cross-Site Scripting

a.k.a. XSS (duh)

The Same-Origin Policy

- Question: why can't attacker.org read the visitors emails from GMail?

- Answer: the Same-Origin Policy is "in the way"
  - Only resources with matching protocol, domain and port may gain access

- That makes for a sad attacker (and his kitten)

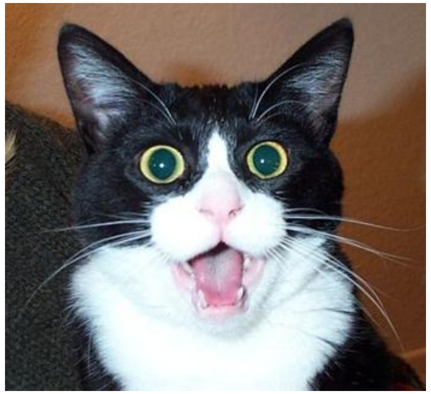
http://andshesaidit.files.wordpress.com/2010/05/sad_kitten1.jpg

XSS – the underlying problem

- Web Apps process **data**
  - Which was provided by the user
  - POST, GET, headers, ….
- **Data** might be stored, or echoed back directly
- **Data** `<script>alert(1)</script>` is actually **Code**
- **…** interpreted by the victim's browser, executed in the origin of vulnerable application
- Attack method
  - Find flaw in Web application that allows injection of CODE, not just DATA
  - (we will elaborate in a minute)
  - Make victim visit that site

➔ We can read your GMails ☺

XSS – what an attacker can do



http://images.sodahead.com/polls/003820731/1313763967_CAT_SHOCKED_xlarge.jpeg

- Open an alert box!
- Hijack a session
  - Oldest trick in the book: steal their cookies
  - Force victim to "click" a link (or post something about BlackHat on Twitter)
- Alter content
  - Display fake content
  - Spoof login forms
- .. Steal your password manager's passwords
  - See our AsiaCCS paper if you are interested ☺
- **Do everything with the Web app, that you could do – under your ID**

Types of XSS

Reflected

Server

```php
<?php
  echo "Hello ".$_GET['name'];
?>
```

Client

```html
<script>
  var name = location.hash.slice(1));
  document.write("Hello " + name);
</script>
```

http://upload.wikimedia.org/wikipedia/commons/f/f1
/Kitten_and_partial_reflection_in_mirror.jpg

Stored

Server

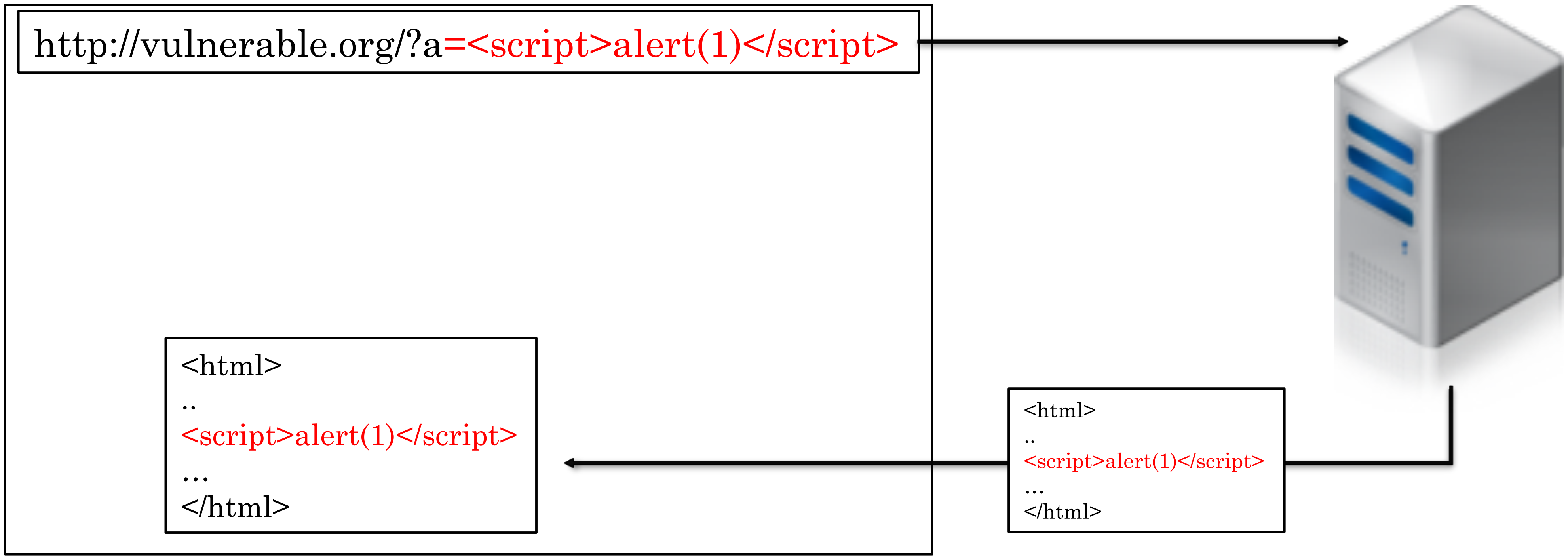
```php
<?php
  $res = mysql_query("INSERT…".$_GET['message']);
  […]
  $res = mysql_query("SELECT…");
  $row = mysql_fetch_assoc($res);
  echo $row['message'];
?>
```

Client

```html
<script>
  var html= location.hash.slice(1);
  localStorage.setItem("message", html);
  […]
  var message = localStorage.getItem("message");
  document.write(message);
</script>
```

http://www.cat-lovers-only.com/images/kittens-in-a-box.jpg

Reflected XSS



Stopping XSS attacks

If you are the application's owner:

- Don't use user-provided data in an unencoded/unfiltered way
- Use secure frameworks or other magic
- Use Content Security Policy, sandboxed iframes, …

Stopping XSS attacks

If you are the application's owner:

- Don't use user-provided data in an unencoded/unfiltered way
- Use secure frameworks or other magic
- Use Content Security Policy, sandboxed iframes, …

If you are the application's user:

- Turn of JavaScript
- Client-side XSS Filters
  - NoScript
  - IE
  - Chrome (the "XSS Auditor")

Quick digression: finding a lot of DOMXSS vulns

Finding and exploiting DOMXSS vulnerabilities automatically at scale

- … using byte-level taint tracking in Chromium
  - each character in a string has its source information attached to it
- … Chrome extension to crawl given set of Web sites
  - also the interface between taint engine and central server
- … and an exploit generator
  - using taint information
  - and HTML and JavaScript syntax rules
  - to generate exploits fully automatic

Results (many many ~~cats~~ XSS)

- For our study, we analyzed **Alexa Top 5k**
  - Found **480** domains with vulnerabilities

- Reran experiment against **Alexa Top 10k**
  - Found a total of **1,602 unique vulnerabilities**
  - .. On **958** domains

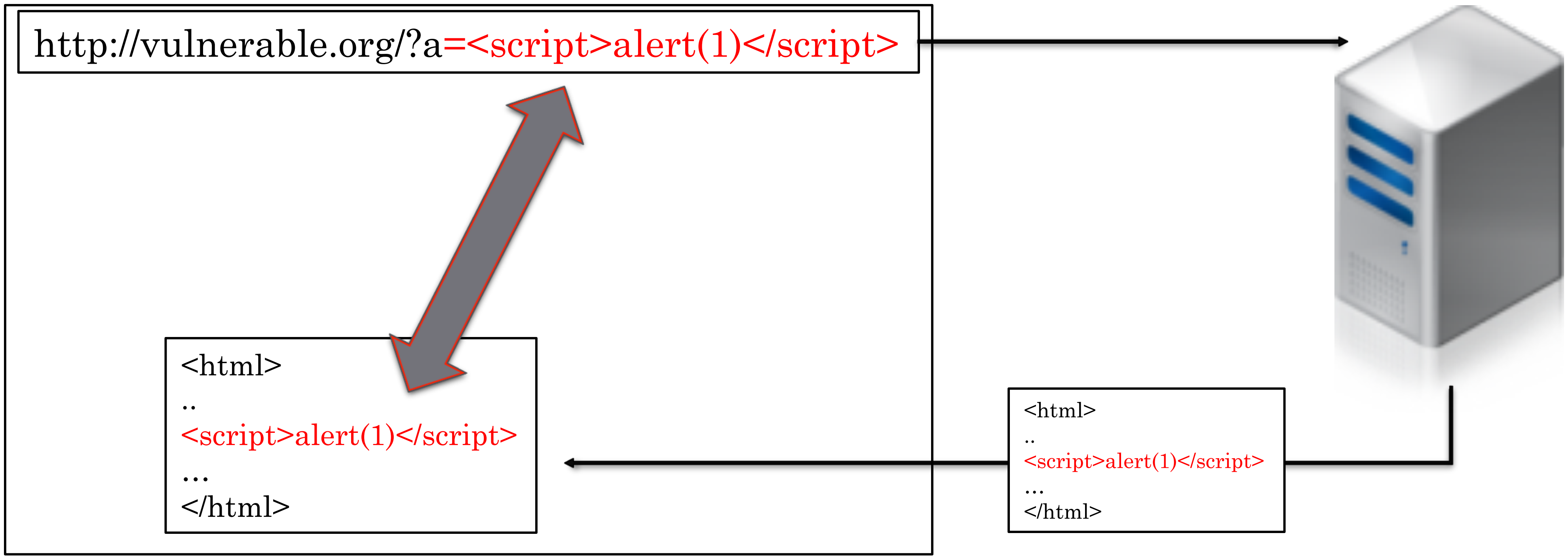
- Auditor turned off at that point

Motivation

- So, we had this considerable amount of real-world XSS vulnerabilities
- And our prime testing platform was built onto the Chrome browser
- Hence, we got curious: How well does the Chrome Auditor protect us?
- We reran our experiment, with the Auditor turned on
- The Auditor did not catch all of our exploits
- This made us even more curious…
  - Why were the exploits not blocked?
  - And can we increase the number of bypasses?

Bypassing the XSSAuditor

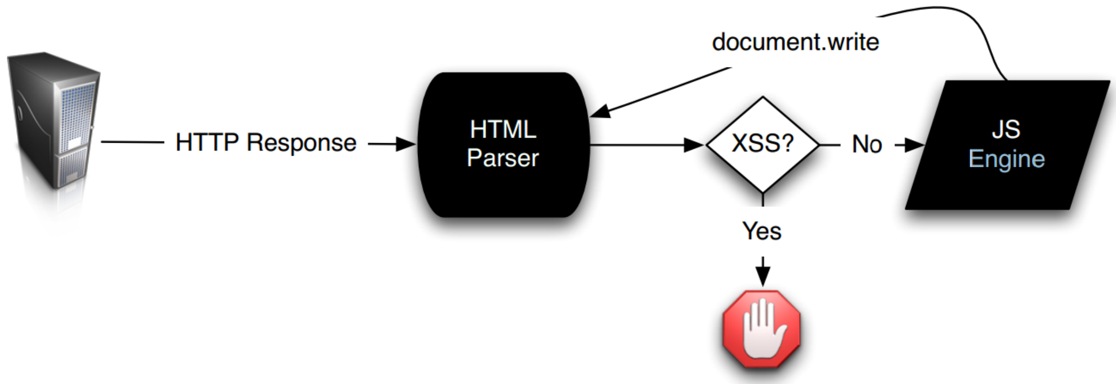
Reflected XSS (revisited)



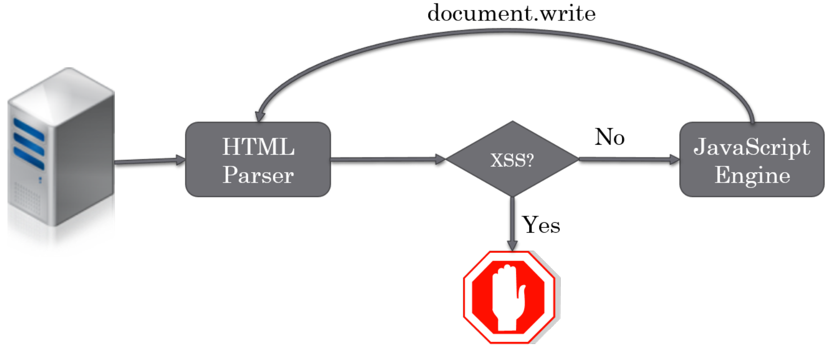
XSS Payload is contained **in** the request (i.e., in the URL)!

XSS Filter Strategies

- NoScript: Check outgoing requests for JavaScript
- IE: Use regular expression to compare HTTP requests and responses
- **XSSAuditor**
  - Don't look at requests
  - When response comes in, invoke HTML parser (actually, tokenizer)
  - When a "dangerous" element or attribute is found during parsing, check the corresponding request's URL



How the XSS Auditor works



- An incoming HTTP response is parsed
- Every time the parser encounters an HTML construct that potentially executes JavaScript, the Auditor is invoked
  - Important fact one: Only during the initial parsing process
  - Important fact two: This check is done **only** if certain characters are contained in the URL: <, >, “ and ‘
- The auditor checks the HTTP request, if the encountered HTML/JavaScript can be found in the request’s URL (or body)
  - Important fact three: Depending on the HTML construct, the matching algorithm differs
- If a match is found, the parser replaces the potential attack with a harmless placeholder

Auditor matching rules (simplified)

- **Inline scripts**

  `<script>alert(1)</script>`

- **Matching rule**
  - … the Auditor checks whether **content of script** is contained in the request
  - … skipping initial comments and whitespaces,
  - …only using up to 100 characters
  - …stop if encountering a “terminating character”:
    - `# ? //` …

Auditor matching rules (simplified)

- **HTML attributes**
  - Event handlers

```
<img onerror="alert(1)" src="//doesnot.exist">
```

  - Attributes with JavaScript URLs

```
<iframe src="javascript:alert(1)"></iframe>
```

- **For each attribute**
  - … the Auditor checks whether the attribute contains a **JavaScript URL**
  - … or if the attribute is an **event handler**
- **Matching rule**
  - Check if the **complete attribute** is contained in the request

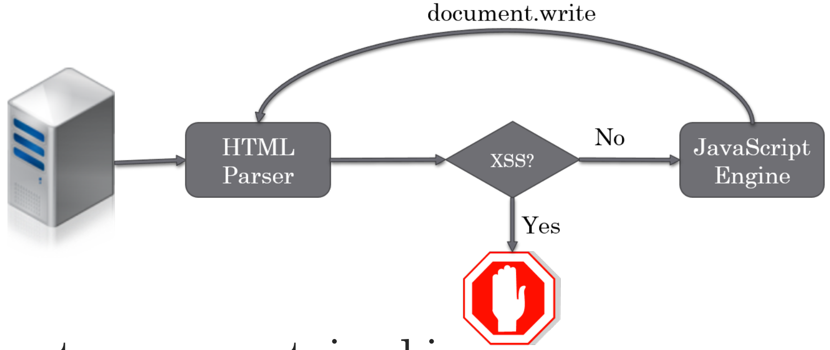
Auditor matching rules (simplified)

- **For HTML elements that can reference external content**

```
<script src="//attacker.org/script.js"></script>
<embed src="//attacker.org/flash.swf"></embed>
```

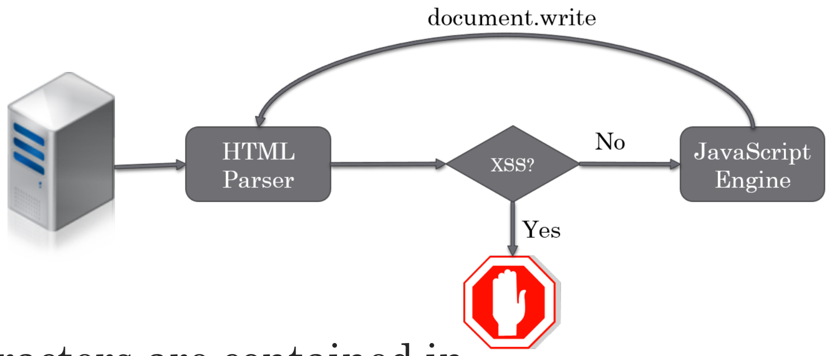
- **Matching rule**
  - … the Auditor checks whether the **tag name** is contained in the request
  - … and whether the **complete attribute** is contained in the request

How the XSS Auditor works



- An incoming HTTP response is parsed
- Every time the parser encounters an HTML construct that potentially executes JavaScript, the Auditor is invoked
  - Important fact one: This check is done **only** if certain characters are contained in the URL: <, >, “ and ‘
- The auditor checks the HTTP request, if the encountered HTML/JavaScript can be found in the request’s URL (or body)
  - Important fact two: Depending on the HTML construct, the matching algorithm differs
- If a match is found, the parser replaces the potential attack with a harmless placeholder

How the XSS Auditor works



- An incoming HTTP response is parsed
- Every time the parser encounte construct that pot Audito
  - Impo ck is done **only** if certain characters are contained in the U , and
- The auditor che tered HTML/JavaScript can be f
  - Impor ing on the HTML construct, the matching algorithm differs
- If a match is f attack with a harmless placehol

Invocation

Matching

Blocking

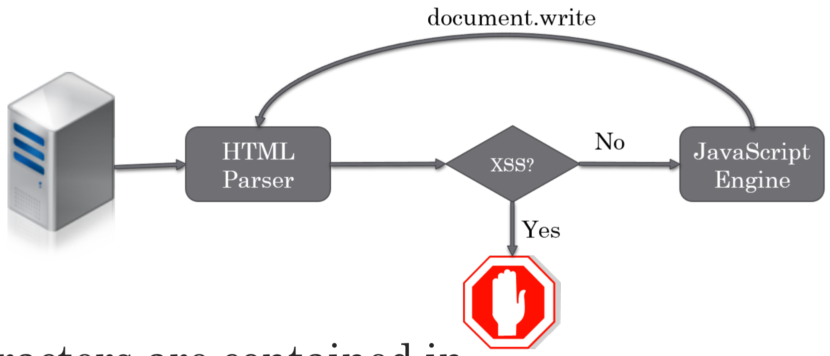
How to bypass the XSS Auditor

- An incoming HTTP response is parsed
- Every time the parser encounter construct that pot Audito
  - Impo ck is done **only** if certain characters are contained in the U , , and
- The auditor che tered HTML/JavaScript can be
  - Impor ing on the HTML construct, the matching algorithm differs
- If a match is f attack with a harmless placehol

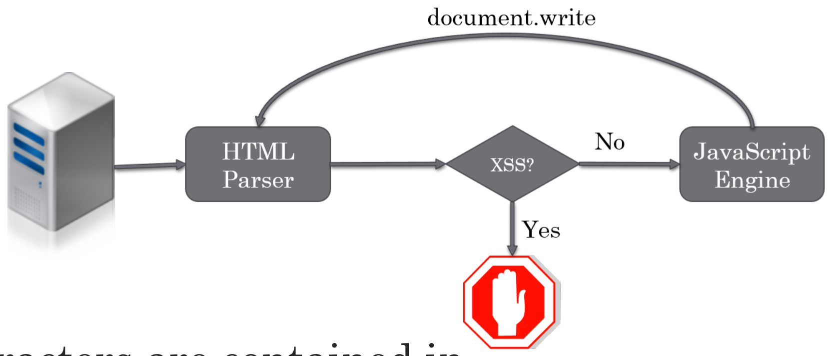
Invocation

Matching

Blocking



How to bypass the XSS Auditor



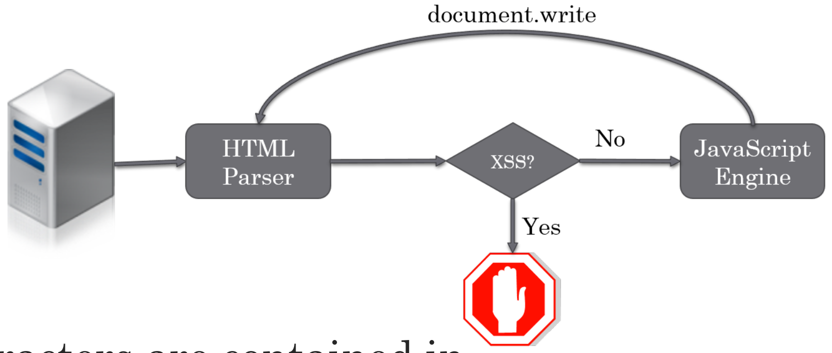
- An incoming HTTP response is parsed
- Every time the parser encounte…
  construct that …
  Audito…
  - Impo… …is done **only** if certain characters are contained in
    the U… and
- The auditor che… …tered HTML/JavaScript
  can be …
  - Impor… …on the HTML construct, the matching algorithm
    differs
- If a match is f… …attack with a harmless
  placehol…

Invocation

Matching

Blocking

How to bypass the XSS Auditor



- An incoming HTTP response is parsed
- Every time the parser encounte… construct that p… Audito…
  - Impo… …ck is done **only** if certain characters are contained in the U… , … and …
- The auditor ch… …tered HTML/JavaScript can be f…
  - Impor… …ing on the HTML construct, the matching algorithm differs
- If a match is f… …d attack with a harmless placehol…

Invocation

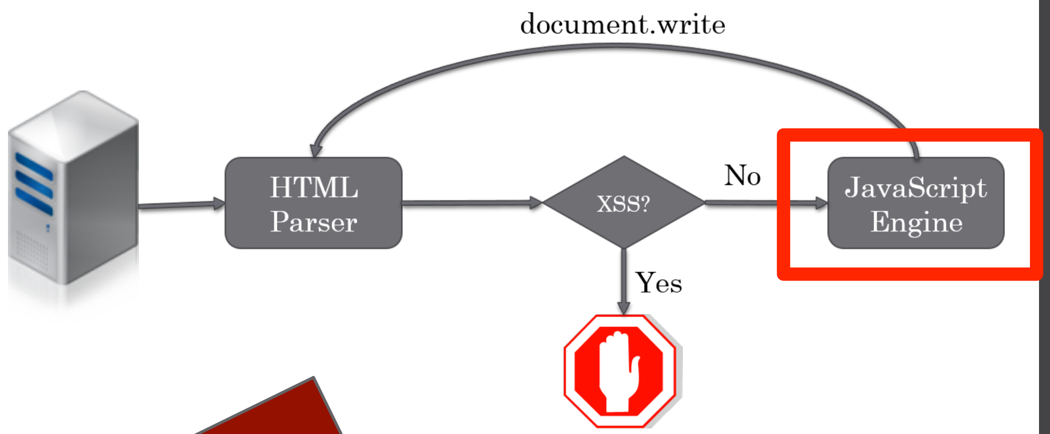
Matching

Blocking

Avoiding Auditor Invocation

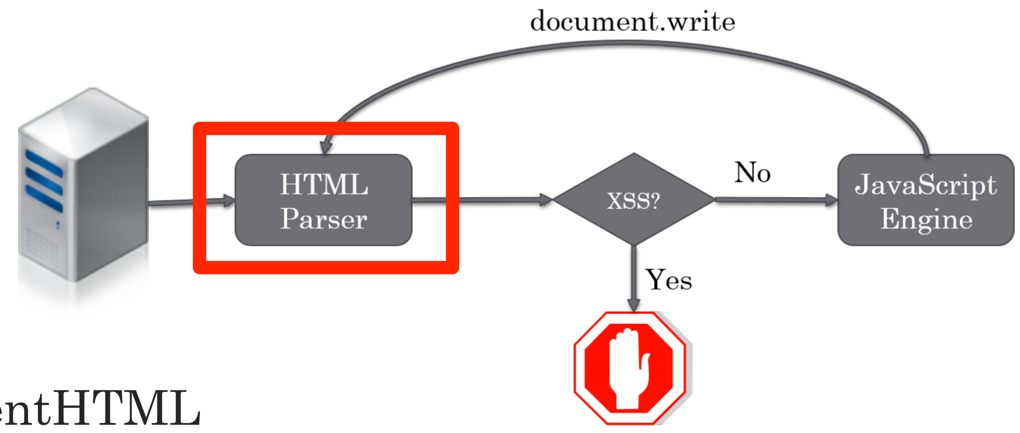
Bypass invocation using eval

- Filter works only for injected HTML
- … not for injected JavaScript



DEMO

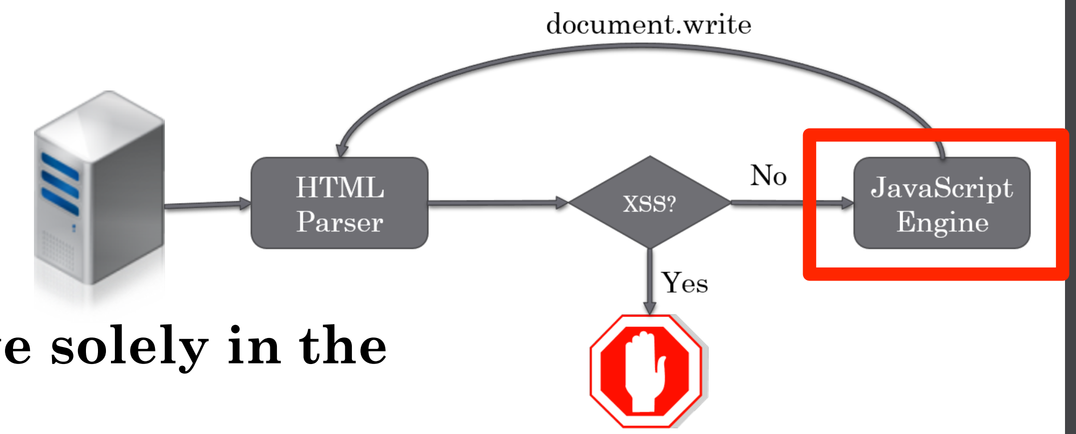
Bypass invocation in the HTML Parser



- **Parsing "document fragments"**
  - i.e. innerHTML, outerHTML, insertAdjacentHTML
  - For performance reasons, Auditor is off for document fragments
  - ➔ all vulnerabilities targeting these sinks go through

- **Unquoted attribute injection**
  - Auditor is disabled if <, >, “ and ‘ are not found in the request
  - All injections that lead to JS execution, that do not require these characters evade the Auditor

HTML-free injections



Various injection techniques that live solely in the JavaScript space

- As the HTML parser is not involved, the Auditor is not activated

1. DOM bindings

- e.g. assigning src attribute of existing script tag
- No HTML parsing, as the injection affects the already parsed DOM

2. Second-order flows

- e.g. cookies or Web Storage
- Injection vector cannot be found in the request

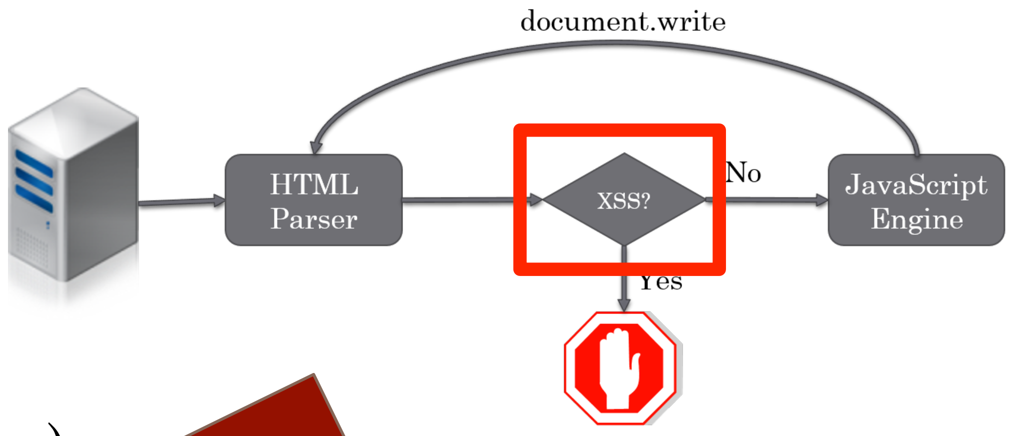
3. Alternative data sources

- e.g. postMessages
- Attack vector enters the page through non-request channels

String-matching issues

Create situations, in which the injected vector does not match the parsed JavaScript

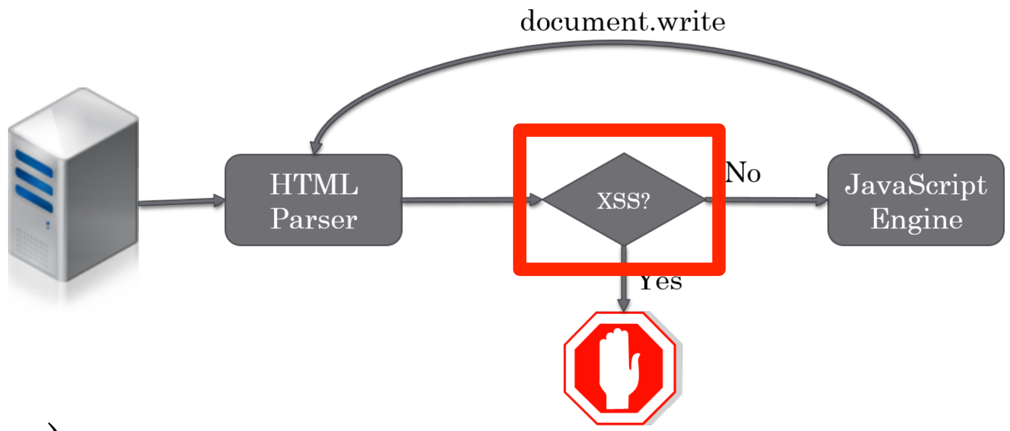
Partial Injections



- Hijack an existing tag
- Hijack an existing attribute (e.g. script.src)
- **Hijack an existing script node**

DEMO

Partial Injections

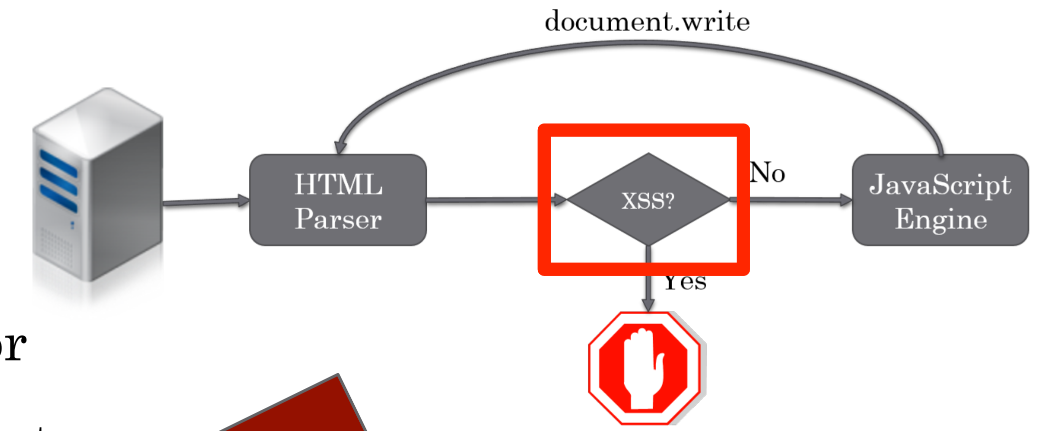


- Hijack an existing tag
- Hijack an existing attribute (e.g. script.src)
- **Hijack an existing script node**

```
http://www.vuln.com/partial.html#someValue'; cat();//
```

```
var x = 'someValue'; cat();//';
```

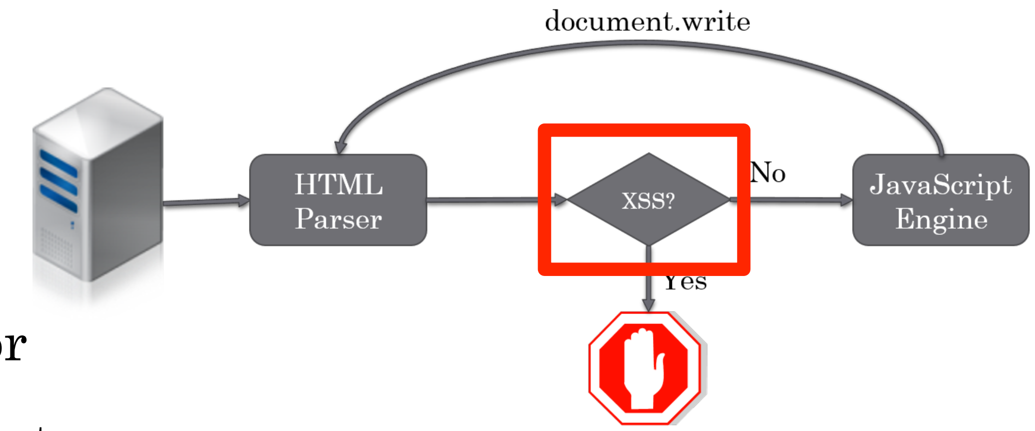
Trailing content



- Idea: use existing content to fool Auditor
- … while still resulting in valid JavaScript

DEMO

Trailing content

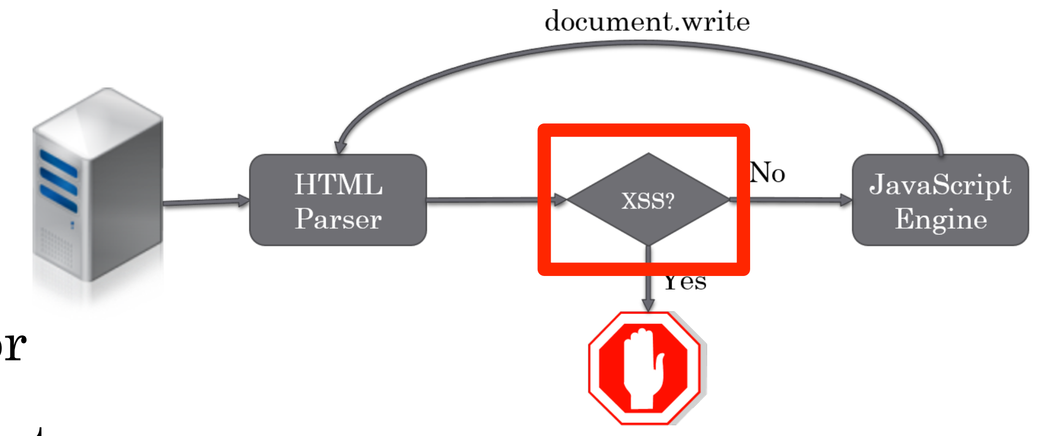


- Idea: use existing content to fool Auditor
- … while still resulting in valid JavaScript

```
http://../trail.html#'><img src=//a onerror='cat();
```

```
<img src=//a onerror='cat();px'>
```

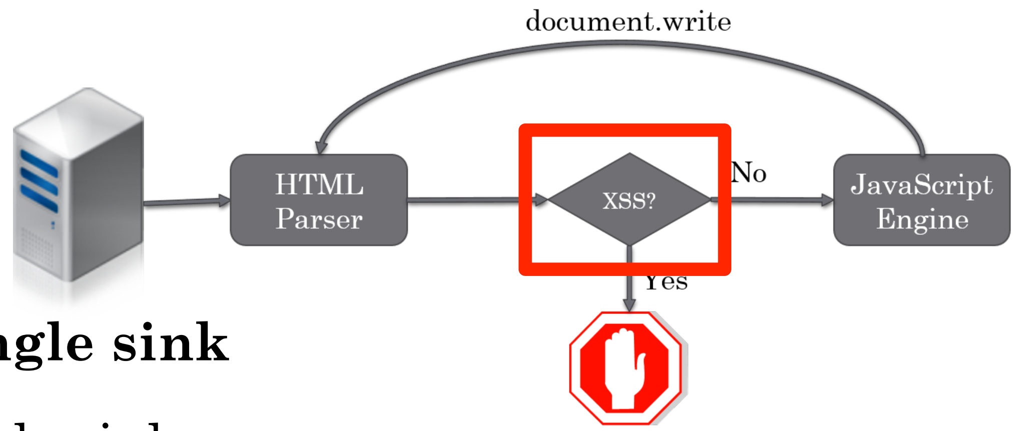
Trailing content



- Idea: use existing content to fool Auditor
- … while still resulting in valid JavaScript

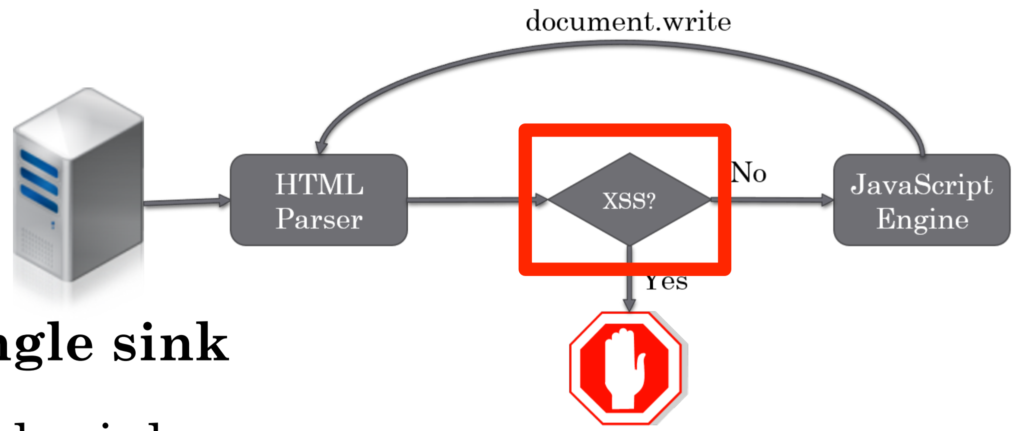
- Further trailing content-based bypasses
  - Trailing slashes (Auditor stops search for payload after **second** slash)
  - Trailing SVG (using Semicolon)

Double injections



- **Single input, multiple injections, single sink**
- Multiple inputs, multiple injections, single sink
- Multiple injection points, multiple sinks

Double injections

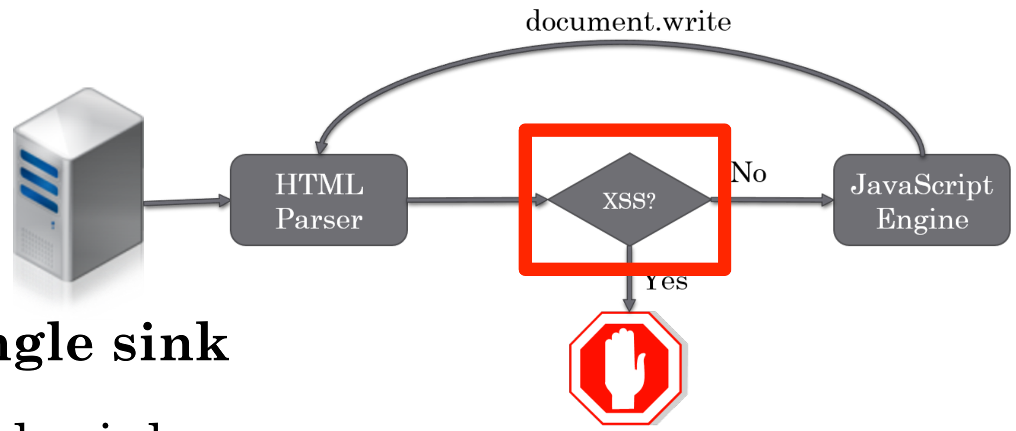


- **Single input, multiple injections, single sink**
- Multiple inputs, multiple injections, single sink
- Multiple injection points, multiple sinks

```
...multi.html#")</script>'><script>cat(); void("
```

```
<img height='250
")</script>'><script>cat(); void("
' src='c.jpg'><img height='250
")</script>'><script>cat(); void("
' src='c.jpg'>
```

Double injections

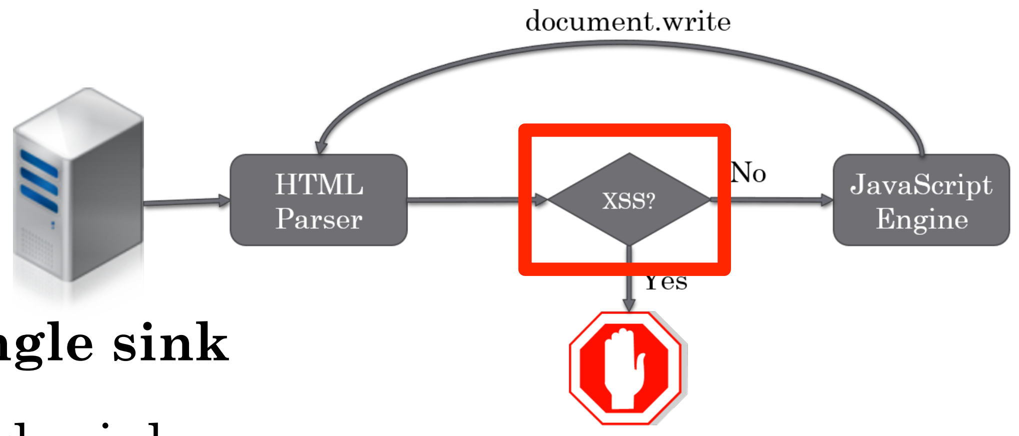


- **Single input, multiple injections, single sink**
- Multiple inputs, multiple injections, single sink
- Multiple injection points, multiple sinks

```
...multi.html#")</script>'><script>cat(); void("
```

```
<img height='250")</script>'>
<script>
cat(); void("' src='c.jpg'><img height='250")
</script>
'><script>cat(); void("' src='c.jpg'>
```

Double injections



- **Single input, multiple injections, single sink**
- Multiple inputs, multiple injections, single sink
- Multiple injection points, multiple sinks

```
...multi.html#")</script>'><script>cat(); void("
```

```
<img height='250")</script>'>
<script>
cat(); void("' src='c.jpg'><img height='250")
</script>
'><script>cat(); void("' src='c.jpg'>
```

Bypasses in the wild

Empirical study

- Using our existing infrastructure, we found
  - … **1,602** DOM-based XSS vulnerabilities
  - … on **958** domains

- We enhanced our exploit generator to target **bypassable** vulnerabilities
  - Not targeting DOM bindings, second-order flows or alternative attacks

Results of our study

- **776 out of 958 domains with bypassable vulnerabilities**

Bypass type	Domain count
innerHTML	469
eval	78
srcdoc (tag hijacking)	146
Trailing content	80
Multi flows	42
Unquoted attribute	7
Inscript injection	7
Assignment to existing script src	7

Conclusion

What to take away?

- **XSS still is a problem**
  - Attack potential maybe bigger than you thought
  - DOM-based XSS on about 10% of the Alexa Top 10k domains
- **Browsers deploy countermeasure to protect users**
  - IE and Chrome built-in, Firefox as a plugin
  - Chrome arguably best filter
- **Security analysis of the Auditor shows that**
  - … there are many bypasses, related to both
  - ... invocation and
  - … string-matching issues

What else to take away?

- **We built a fully-automated system to find DOMXSS**
  - Taint-aware browser
  - Context-aware exploit generator
- **We enhanced the generator to target known issues in the Auditor**
  - Allowing for more exploits to bypass the Auditor
- **We evaluated the impact of the issues**
  - Bypassing the filter on **776 out of 958 domains (81%)**
  - … **1,162 out of 1,602 vulnerabilities (73%)**

Thank you

visit us at kittenpics.org

Martin Johns

@datenkeller

Ben Stock

@kcotsneb

Sebastian Lekies

@sebastianlekies