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

Sebastian Lekies (@sebastianlekies), Ben Stock (@kcotsneb) and Martin Johns (@datenkeller)

Attention hackers!

These slides are preliminary!

For updated material please check http://kittenpics.org

Meow!

Agenda

Technical Background

- XSS 101
- Chrome's XSS Auditor

Bypassing the XSS Auditor

- Scope-related Issues
- String-matching-based Issues
- Empirical Study

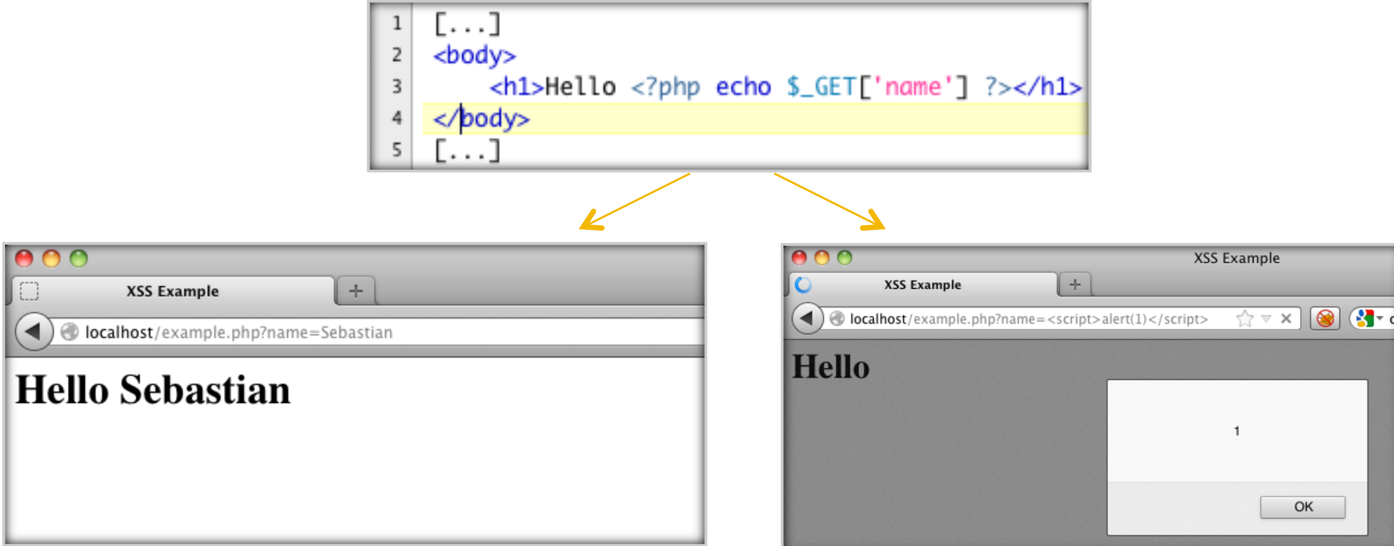
Conclusion

Technical Background

Cross-Site Scripting 101
What is XSS?

Underlying Problem

- Web applications process data that was passed to them via GET or POST requests
  – User input such as: Form fields, parts of the URL, HTTP headers, etc.
- Often this data is included / echoed somewhere in the application's UI
  – E.g. within HTML:



Cross-Site Scripting 101

Types of Cross-Site Scripting I

Caused by server-side code (Java, PHP, etc.)

1. Reflected
2. Persistent

(1–2: Traditional XSS)

Caused by client-side code (JavaScript, VB, Flash)

3. Reflected
4. Persistent

(3–4: DOM-based XSS)

(1–4: Application-specific)

Caused by the infrastructure

5. Client-side infrastructure (e.g. Universal XSS)
6. Server-side infrastructure (e.g. Response Splitting)
7. Network (e.g. Off-path Attacks, Active Network Attacker)

Caused by the user

8. Self-XSS

(5–8: Application-independent)

Cross-Site Scripting 101

Types of Cross-Site Scripting II

	Reflected	Persistent
Server	see code below	see code below
Client	see code below	see code below

Server – Reflected

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

Server – Persistent

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

Client – Reflected

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

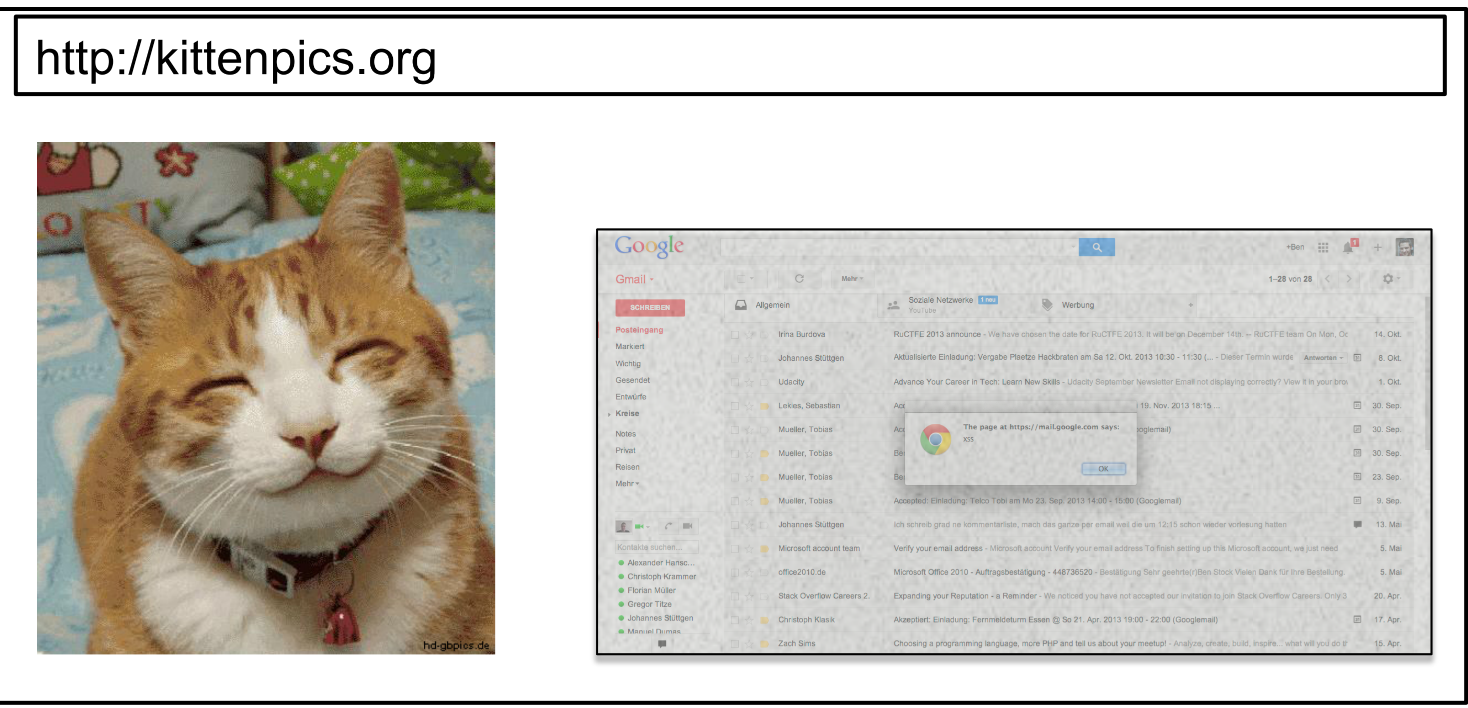
Client – Persistent

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

Cross-Site Scripting 101
Exploitation (Reflected XSS)

Reflected Cross-Site Scripting

1. Craft malicious link
2. Embed link with payload within a innocent looking page



Source: http://www.hd-gbpics.de/gbbilder/katzen/katzen2.jpg

Cross-Site Scripting 101
Exploitation (Persistent XSS)

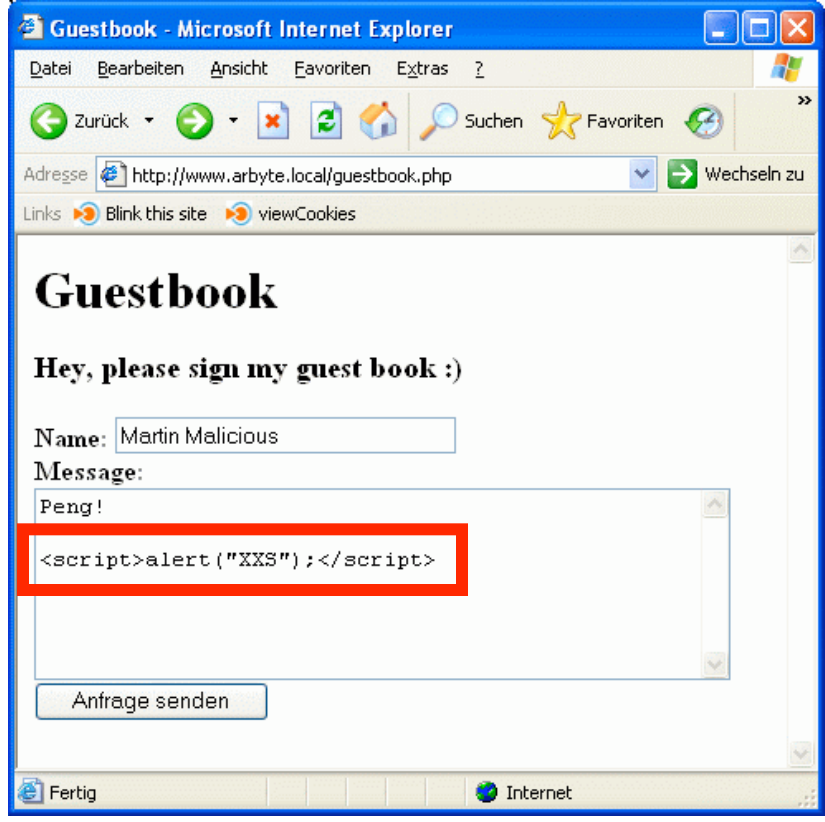
Persistent Cross-Site Scripting

- The web application permanently stores user provided data
- This data is included in the website
- Every time the vulnerable web page is visited, the malicious code gets executed

Cross-Site Scripting 101
Exploitation (Persistent XSS)

Persistent Cross-Site Scripting

- The web application permanently stores user provided data
- This data is included in the website
- Every time the vulnerable web page is visited, the malicious code gets executed
  - Example: Guestbook

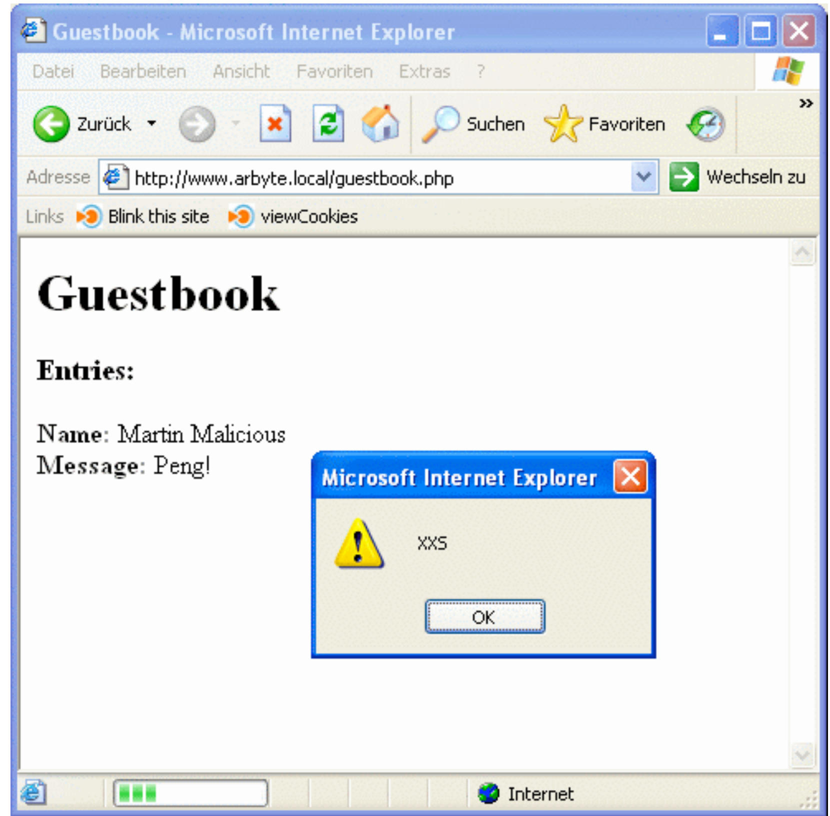


Cross-Site Scripting 101
Exploitation (Persistent XSS)

Persistent Cross-Site Scripting

- The web application permanently stores user provided data
- This data is included in the website
- Every time the vulnerable web page is visited, the malicious code gets executed
  - Example: Guestbook

After injecting the attack code the adversary only has to sit back and wait…

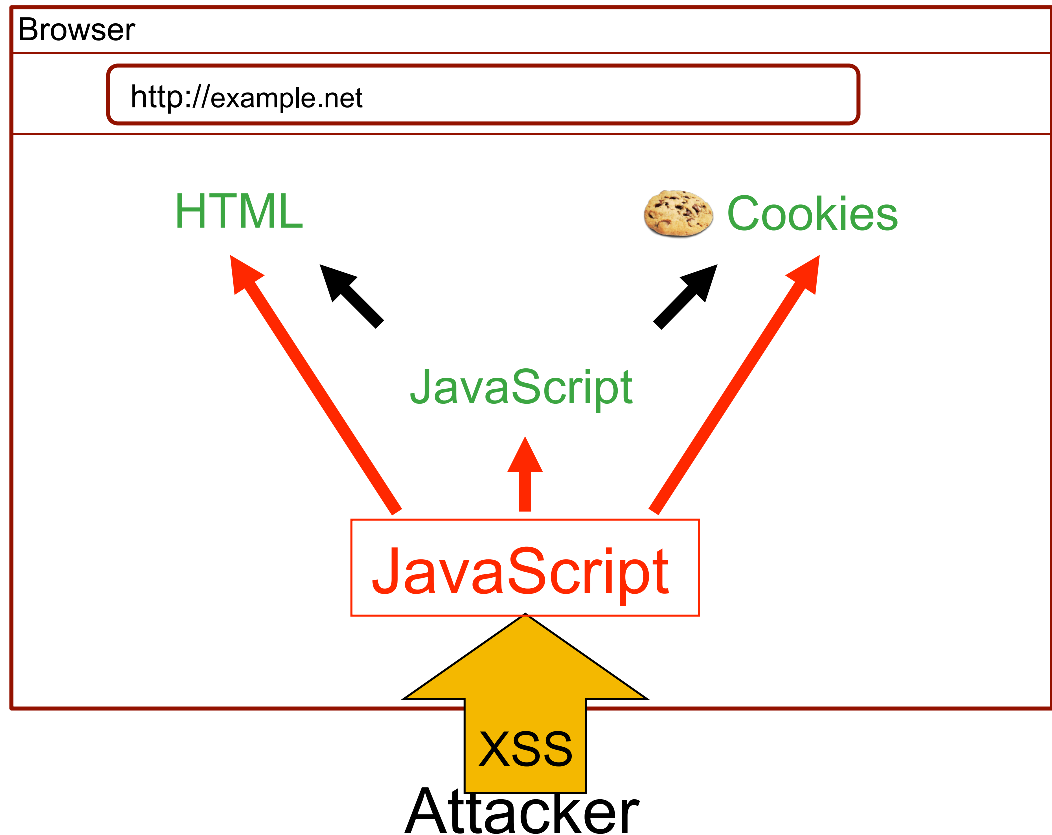


Technical Background

Cross-Site Scripting - Exploitation

The effects of a successful attack:

- An attacker includes malicious JavaScript code into a webpage
- This code is executed in the victim's browser session. In the context of the application



Cross-Site Scripting 101

Example

Ubuntu Forums Hacked – 1.82 Million Usernames Stolen

In a press release on their website, Canonical Ltd announced that on 14 July there was a breach of Ubuntu's forums leading to the theft of 1.82 million of it's users' details.

The attacker used a method known as "cross site scripting" or "XSS" which is a string of code that executes a command, in this case, to steal cookies from a logged in user. By sending this code, disguised as a hyperlink in a message to an administrator, the attacker was able to login.

Often websites use cookies to 'remember' whether a user has logged in, by stealing the cookie of a logged in administrator, the attacker was able to take on their identity and never become asked for a password.

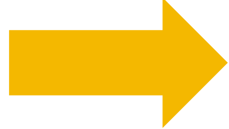
Canonical has announced that "They used this access to download the 'user' table which contained usernames, email addresses and salted and hashed (using md5) passwords for 1.82 million users."

What the hacker exhibited is a sophisticated mixture of techniques and a deep knowledge of the underlying forum software, vBulletin.

Cross-Site Scripting 101
Attacker Capabilities

Malicious Capabilities

- Web content alteration
  - Displaying faked content
  - Spoofing of login dialogues
    - Phishing of Username / Password
- Session Hijacking
  - Cookie Theft → Session Hijacking
  - Browser Hijacking → Creating HTTP requests



Impersonating the user (towards the server)

Impersonating the server (towards the user)

Chrome’s XSS Auditor

Chrome's XSS Auditor

Best protection against XSS is to **avoid vulnerabilities**…

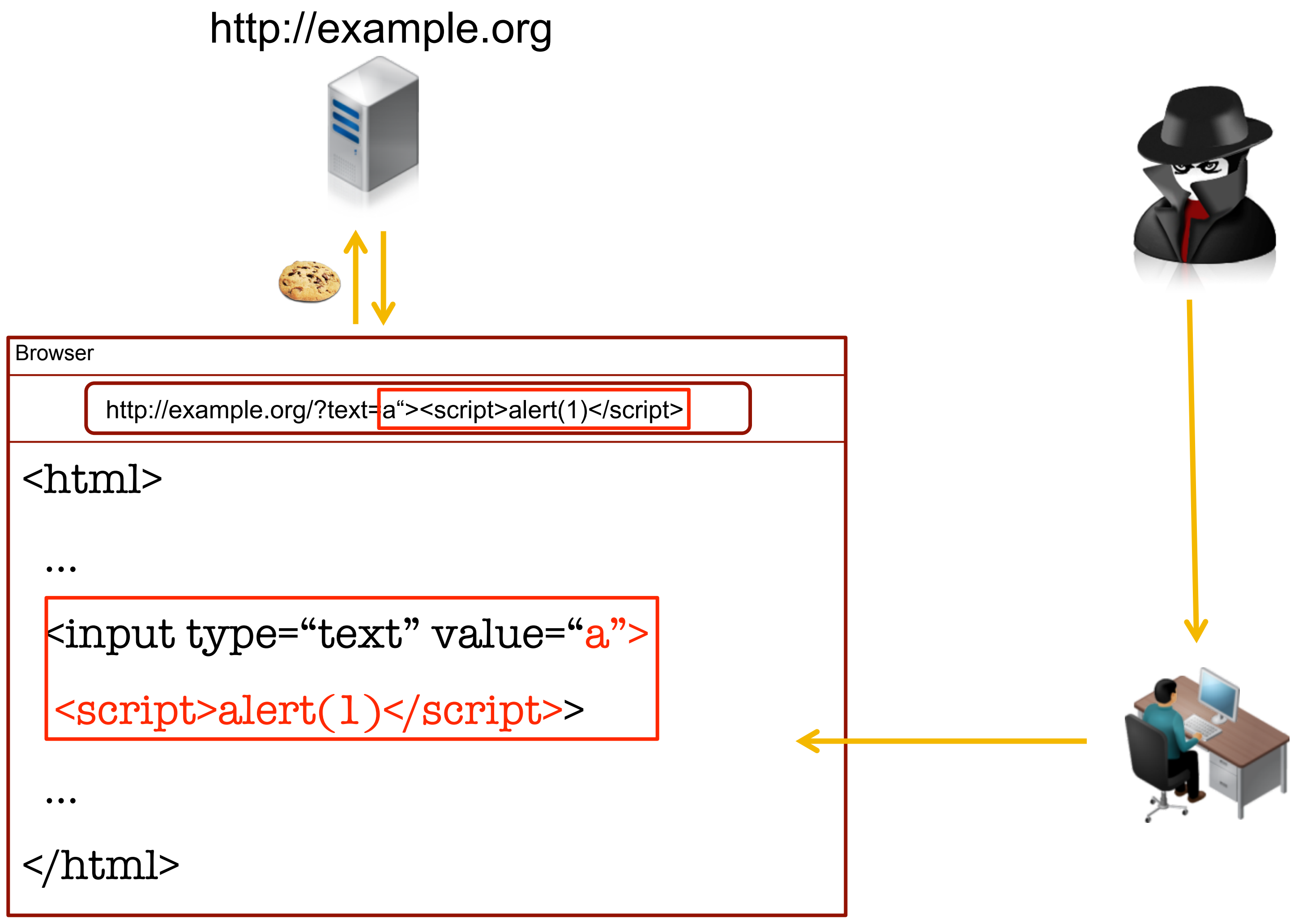
…But**: XSS vulnerabilities are omnipresent** in the Web

NoScript and Microsoft introduced first client-side countermeasures

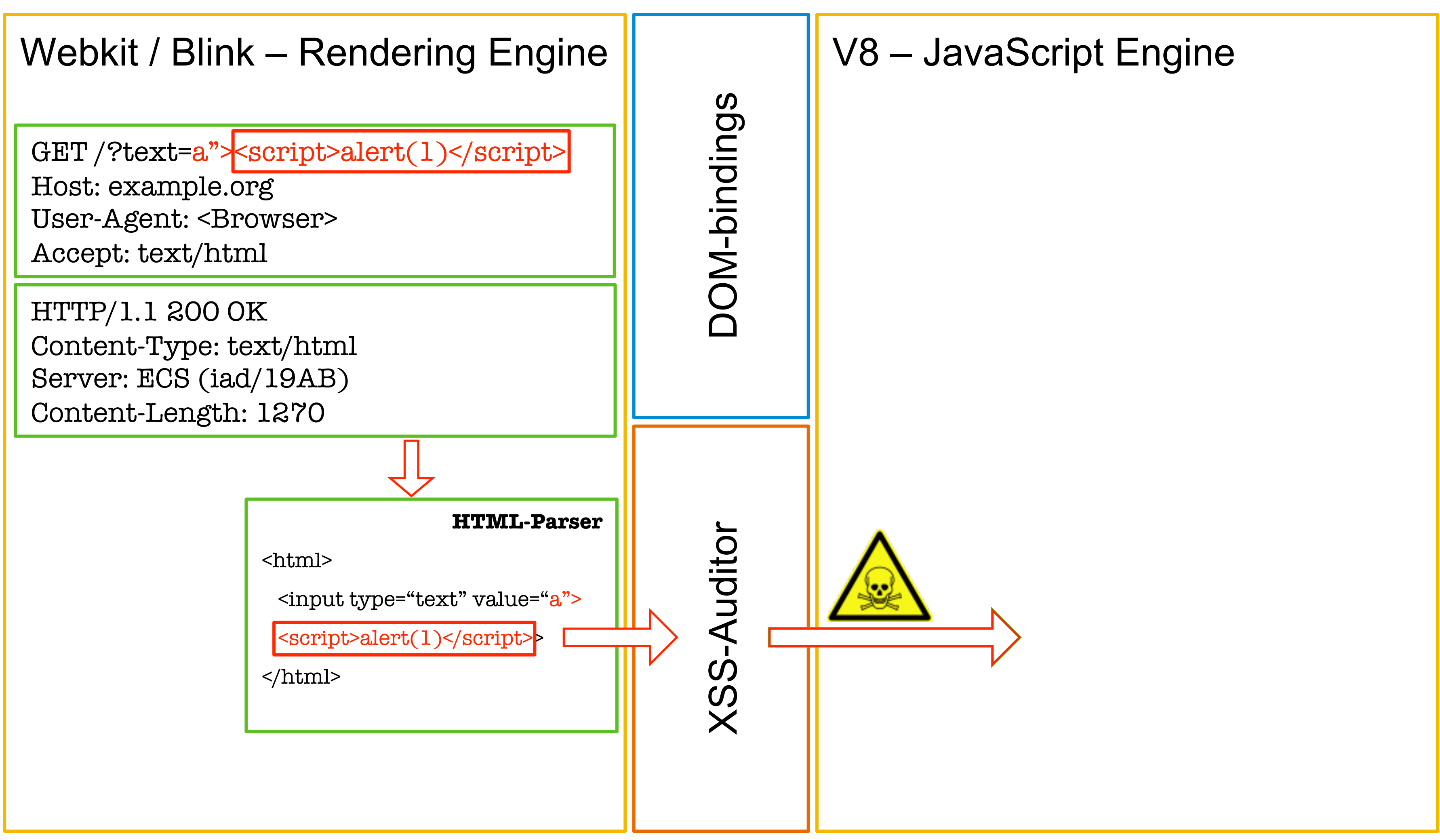
Google introduced the XSS Auditor in 2010.

- Client-side system to prevent exploitation of existing XSS vulnerabilities
- Primarily aims at reflected XSS
- Goals: Low false positive rate, low performance impact

Chrome’s XSS Auditor – Attacker Model



Chrome’s XSS Auditor – Placement



Chrome's XSS Auditor – Decision Logic

Ways to Invoke JavaScript Engine:

- Inline Scripts
  - <script>alert(1);</script>

FilterCharacterToken

- Event handler
  - onload, onerror, onclick, oncut, onunload, onfocus, onblur
  - e.g.: <img src="foo" onload="alert(1)"></img>
- Attributes with JavaScript URLs
  - frame.src, a.href
  - e.g.: <iframe src="javascript:alert(1)"></iframe>

EraseDangerousAttributes

- External Content
  - e.g.: <script src="http://evil.com/script.js"></script>
  - e.g.: <embed src="http://evil.com/flash.swf"></embed>
  - e.g.: <applet code="http://evil.com/java.class"></applet>
  - e.g.: <object><param name="source" value="http://evil.com/silverlight.xap"></object>

FilterTagSpecificAttributes

Chrome's XSS Auditor – Matching Rules (Simplified)

If one of these situations is present, the Auditor performs its checks…

- **For Inline Scripts** (e.g. <script>alert(1)//test</script>)…
  - …the Auditor checks whether the **content of the script is contained within the request**

- **For each attribute** (e.g. <div onclick="alert(1)">)…
  - … the Auditor checks whether the attribute **contains a JavaScript URL**
  - … or whether the attribute **is an event handler**
  - …and if the **complete attribute is contained in the request**

- **For special attributes** (e.g. <script foo="bar" src="http://evil.com/evil.js"></script>)
  - … the Auditor checks whether the **tag name is contained within the request**
  - … and if the **complete attribute is contained in the request**

Bypassing Chrome's XSS Auditor

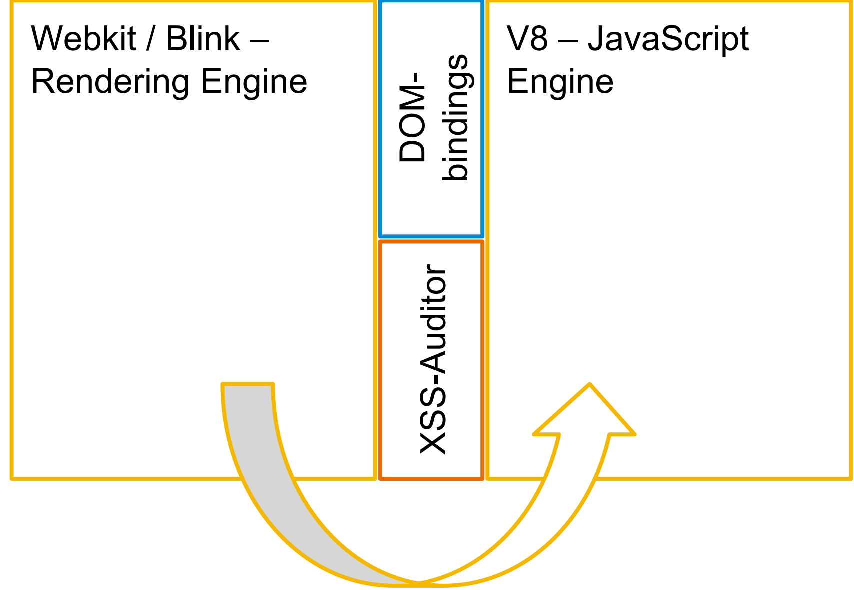
Chrome’s XSS Auditor – Decision Logic

Filter Character Token – Matching Rule

- <script>/* some comment */ eval("\x61\x6c\x65\x72\x74\x28\x31\x29”) /* […] */ var foo=“bar”; </script>
- Skip initial comments and whitespaces
- Use any character until the next comment, opening script tag or comma
  - eval("\x61\x6c\x65\x72\x74\x28\x31\x29”)
- Fully decode the string
  - eval(“alert(1)”)
- Fully decode the URL

Bypassing the XSS Auditor

Scope Related Issues



Bypassing the XSS Auditor

String-matching-related Issues

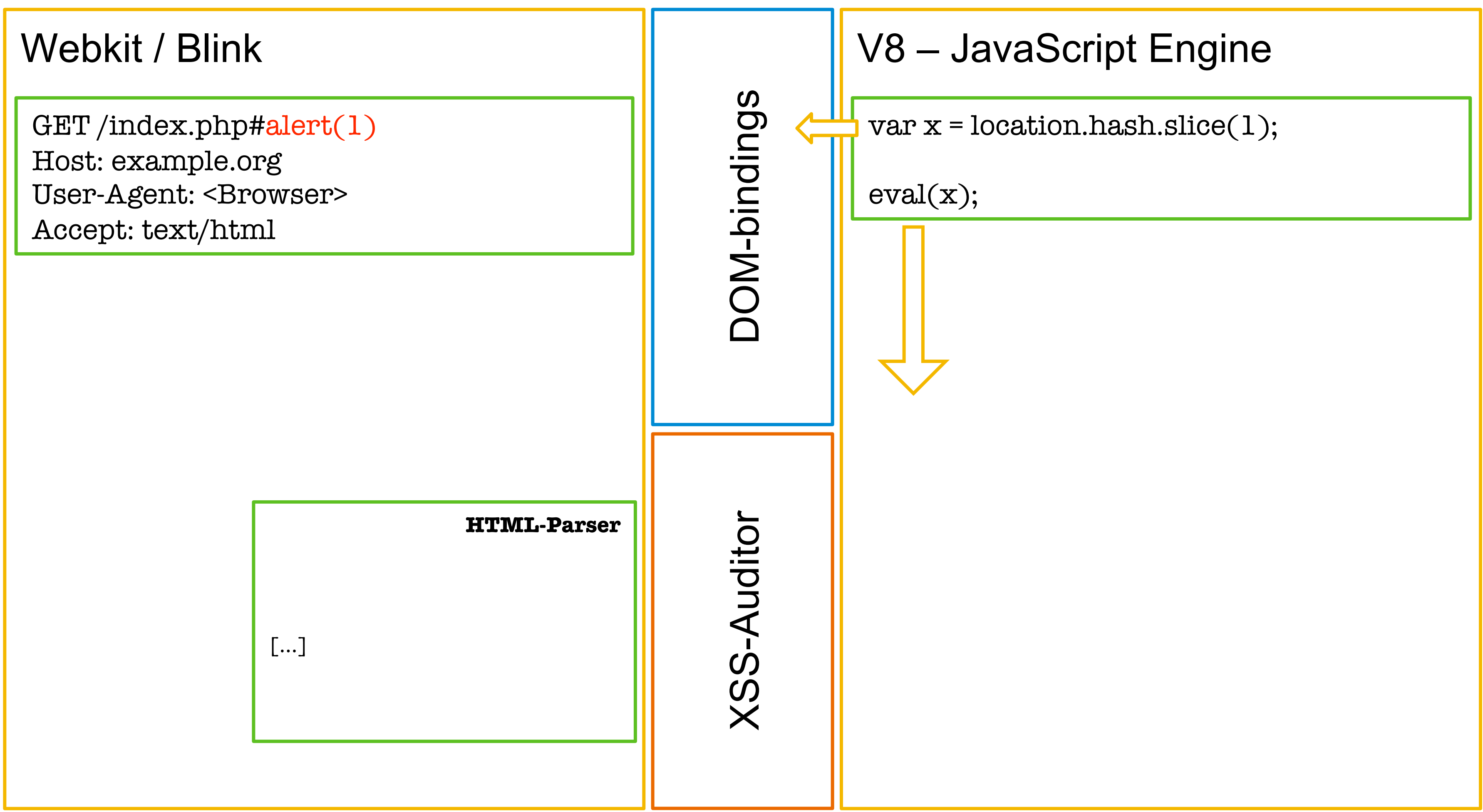
```
GET /?text=a”; alert(1);//”;
Host: example.org
User-Agent: <Browser>
Accept: text/html
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: ECS (iad/19AB)
Content-Length: 1270


<html>
  <script> var x = “a”; alert(1);//”;</script>
</html>
```

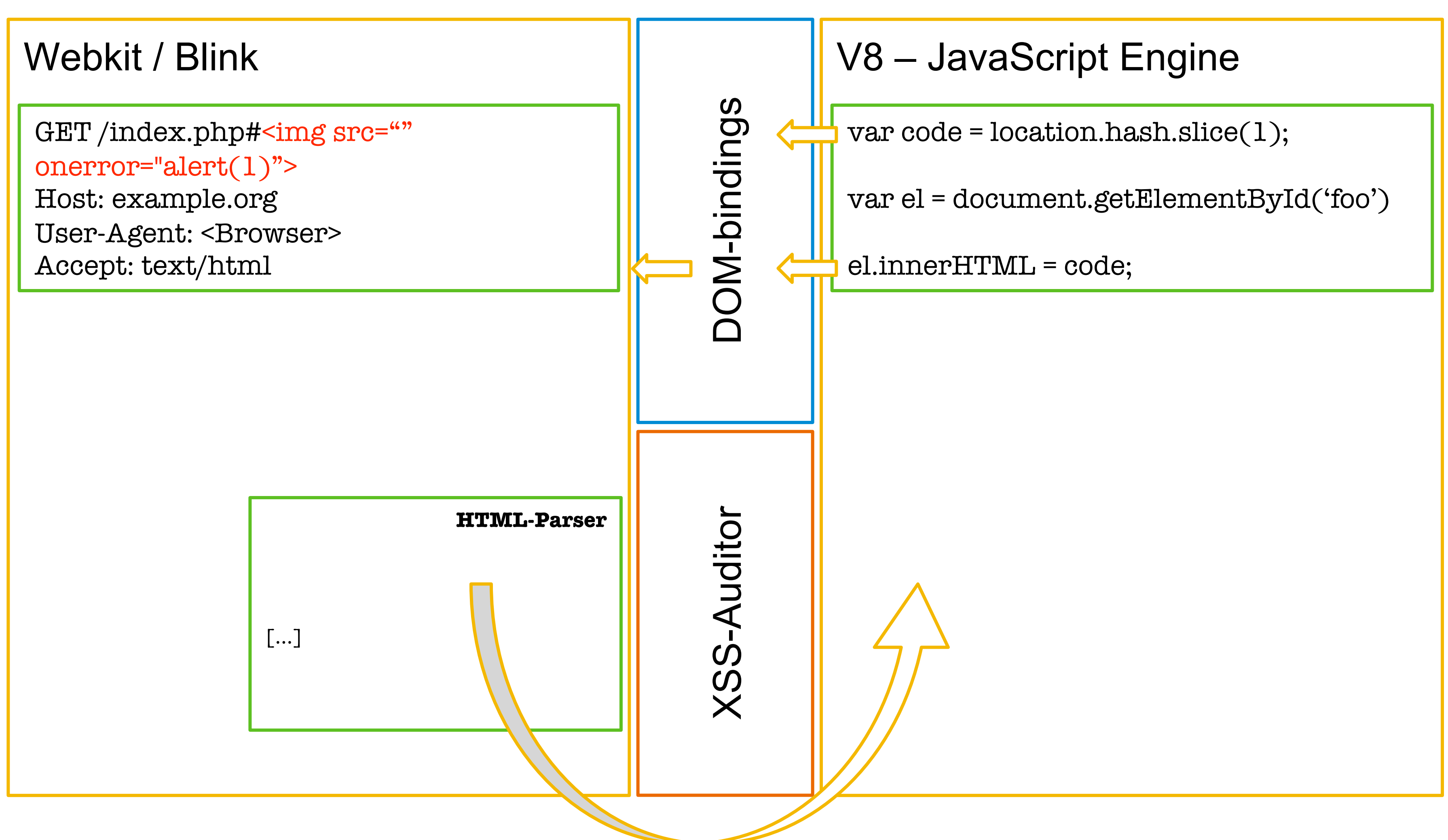
Chrome's XSS Auditor – Scope Related Issues

Eval



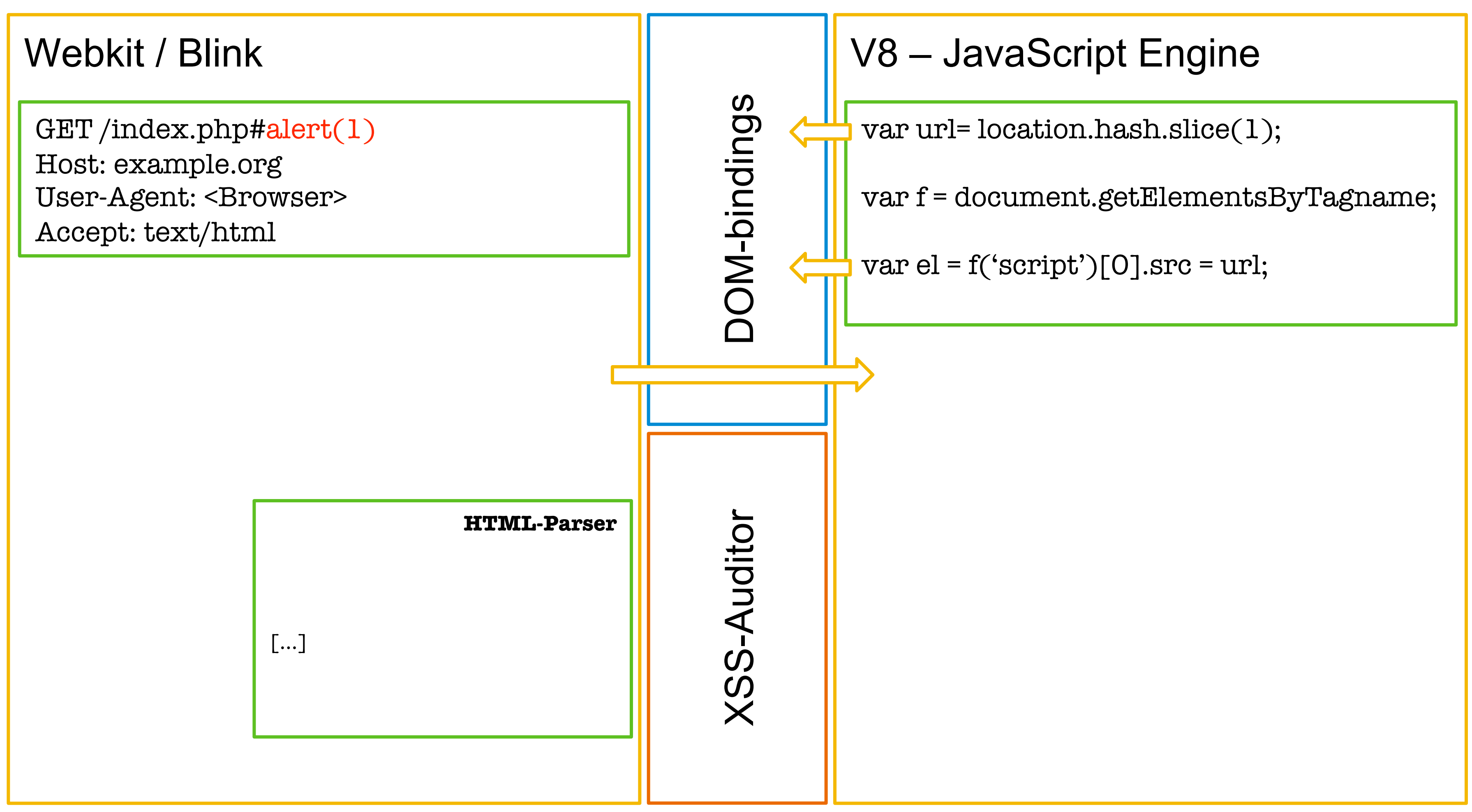
Chrome’s XSS Auditor – Scope Related Issues

InnerHTML, outterHTML, insertAdjacentHTML



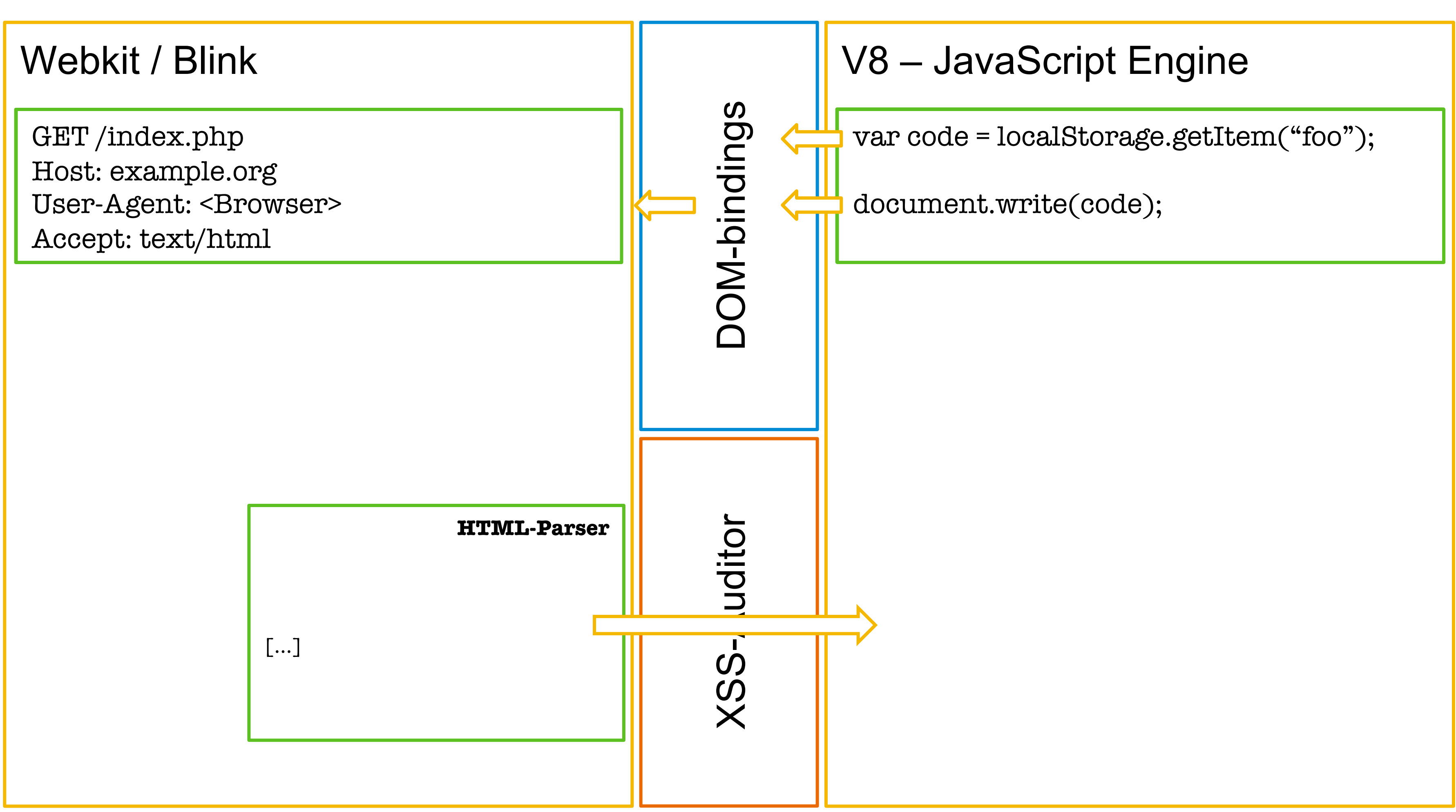
Chrome’s XSS Auditor – Scope Related Issues

Access via DOM-bindings



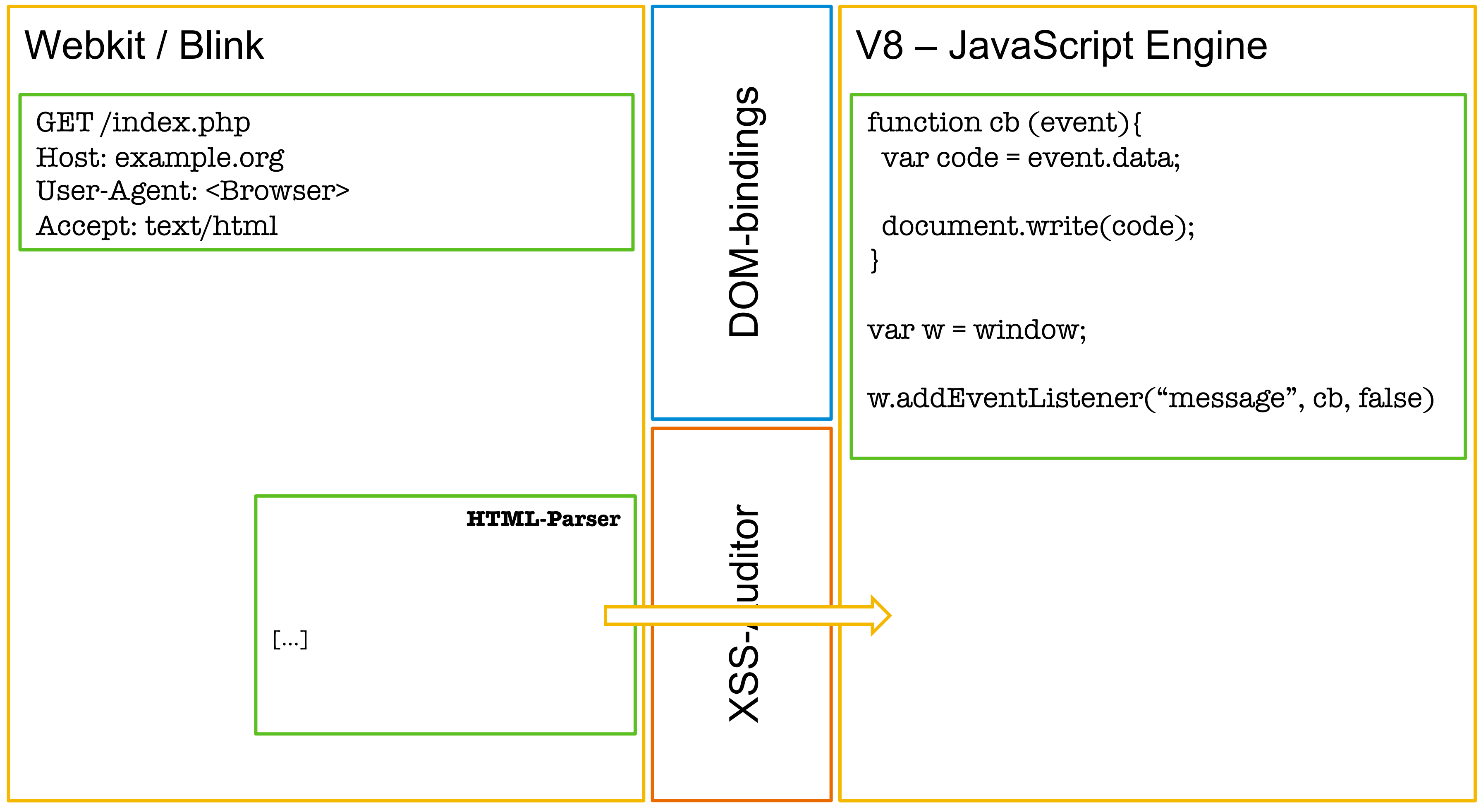
Chrome's XSS Auditor – Scope Related Issues

Second Order Flows



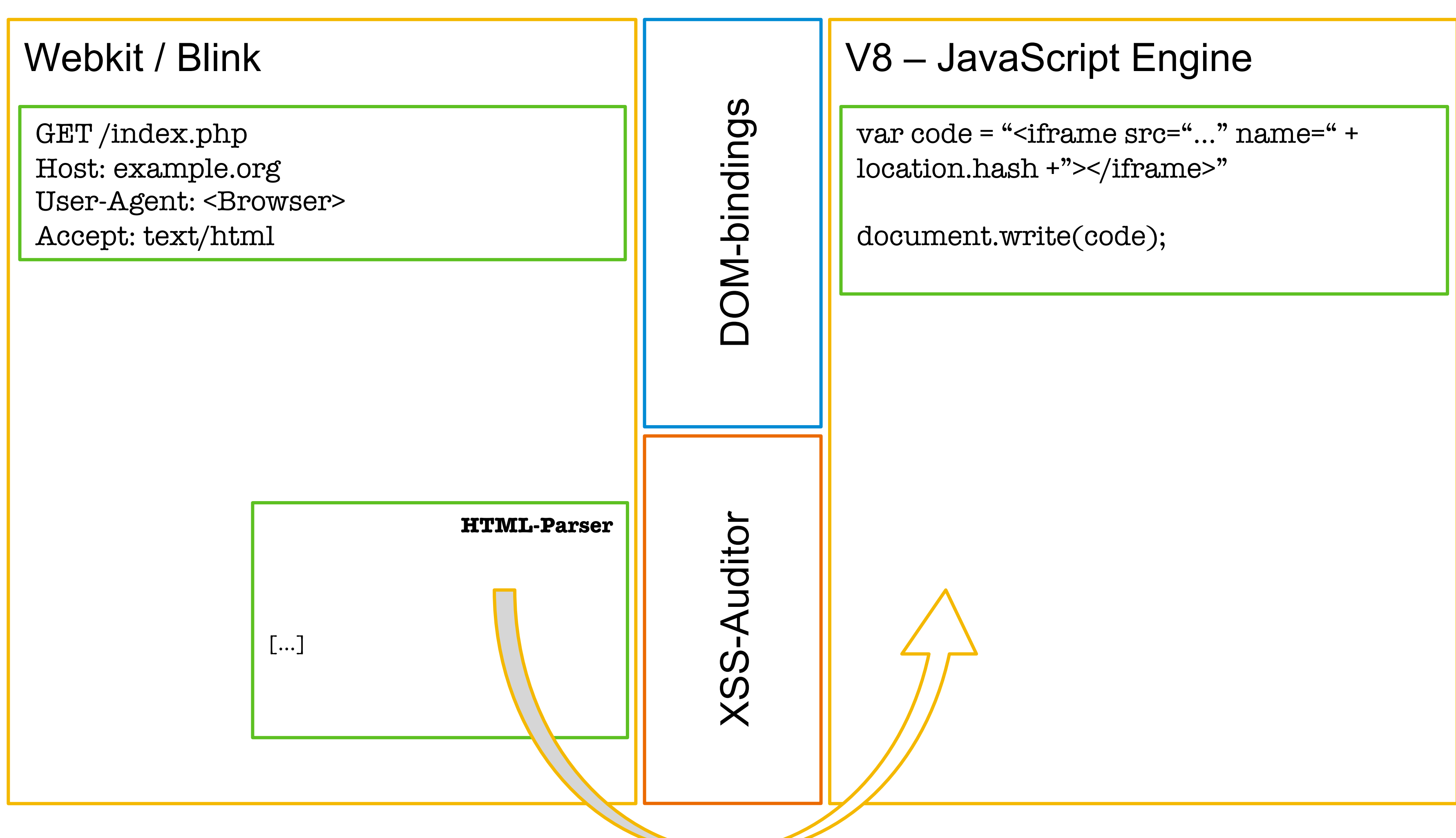
Chrome’s XSS Auditor – Scope Related Issues

Alternative Attack Vectors



Chrome's XSS Auditor – Scope Related Issues

Unquoted Attribute



String-Matching-based Issues

1. **Partial Injections**
   - Tag Hijacking
   - Attribute Hijacking
   - In-script Injections

String-Matching-based Issues

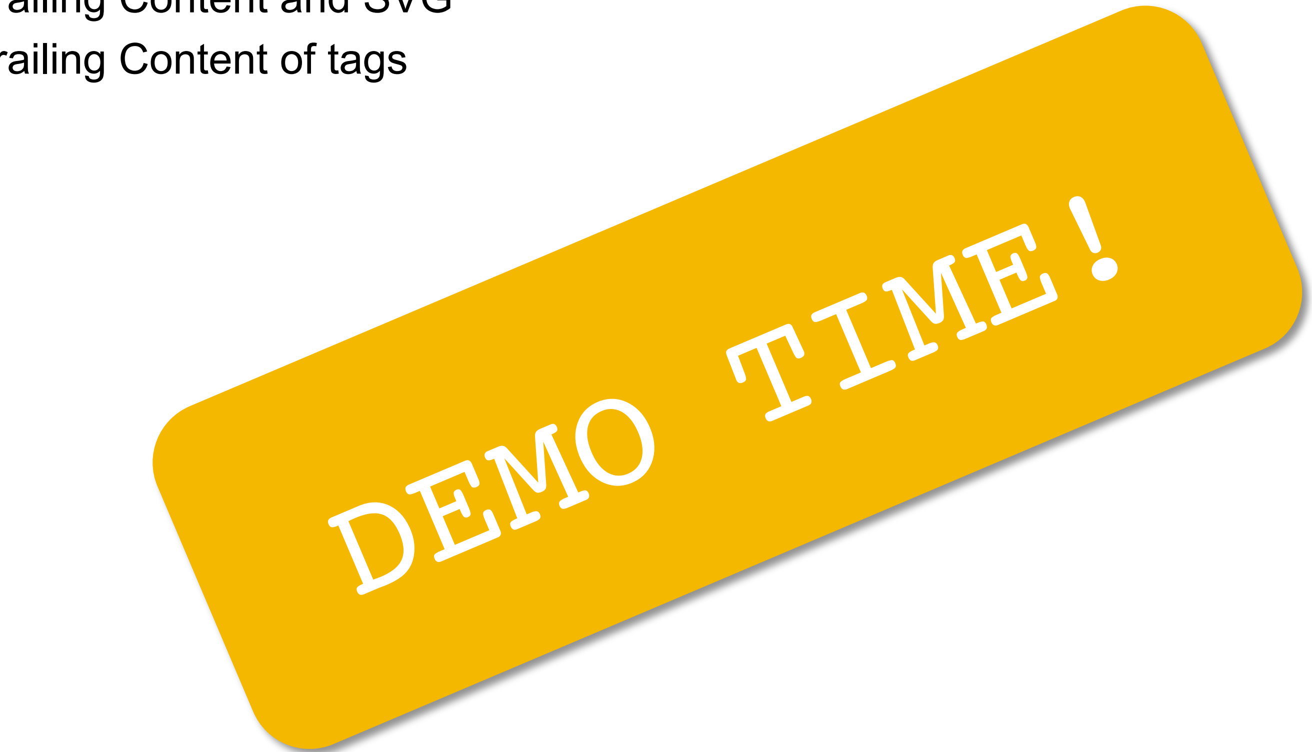
1. **Partial Injections**
   - Tag Hijacking
   - Attribute Hijacking
   - In-script Injections

String-Matching-based Issues

1. **Trailing Content**
   - Trailing Content within Attributes
   - Trailing Content and SVG
   - Trailing Content of tags

String-Matching-based Issues

1. **Trailing Content**
   - Trailing Content within Attributes
   - Trailing Content and SVG
   - Trailing Content of tags

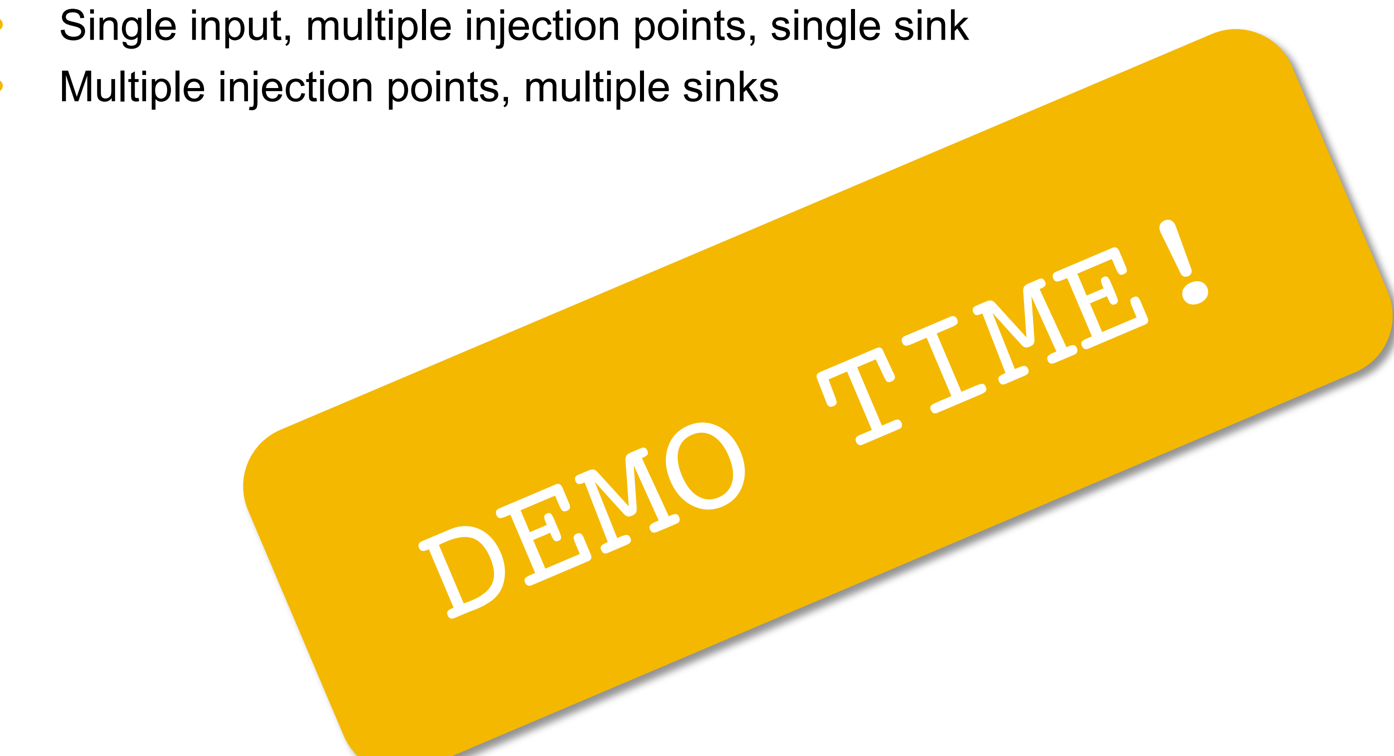


String-Matching-based Issues

1. **Double Injections**
   - Multiple inputs, multiple injection points, single sink
   - Single input, multiple injection points, single sink
   - Multiple injection points, multiple sinks

String-Matching-based Issues

1. **Double Injections**
   - Multiple inputs, multiple injection points, single sink
   - Single input, multiple injection points, single sink
   - Multiple injection points, multiple sinks



String-Matching-based Issues

Application-specific input mutation

String-Matching-based Issues

Application-specific input mutation

Empirical Study

In a previous study we collected…

- …1,602 DOM-based XSS vulnerabilities
- … on 958 domains

We built a tool to generate bypasses for these vulnerabilities

Results

- We successfully exploited 73% of the 1602 vulnerabilities despite of the Auditor
- We exploited vulnerabilities on 81% of all vulnerable applications

Conclusion

Conclusion

XSS is a wide-spread problem

- Many different types of XSS exist
- DOM-based XSS is one serious subclass of XSS

Browser-vendors introduced client-side XSS filters

- …to protect users from being exploited successfully
- All major browsers offer XSS filter

We conducted a security analysis of Chrome’s XSS Auditor

- …and found 18 bypasses
- …7 scope-related Issues
- …9 string-matching-related issues
- …allowing us to bypass XSS vulnerabilities on about 80% of all vulnerable applications

Thank you

Contact information:

Sebastian Lekies	Ben Stock	Martin Johns
SAP AG	FAU Erlangen	SAP AG
@sebastianlekies	@kcotsneb	@datenkeller