Web application vulnerabilities (part 2)

Questões de Segurança em Engenharia de Software (QSES)
Mestrado em Segurança Informática
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto

Eduardo R. B. Marques, edrdo@dcc.fc.up.pt

DOM — Document Object Model

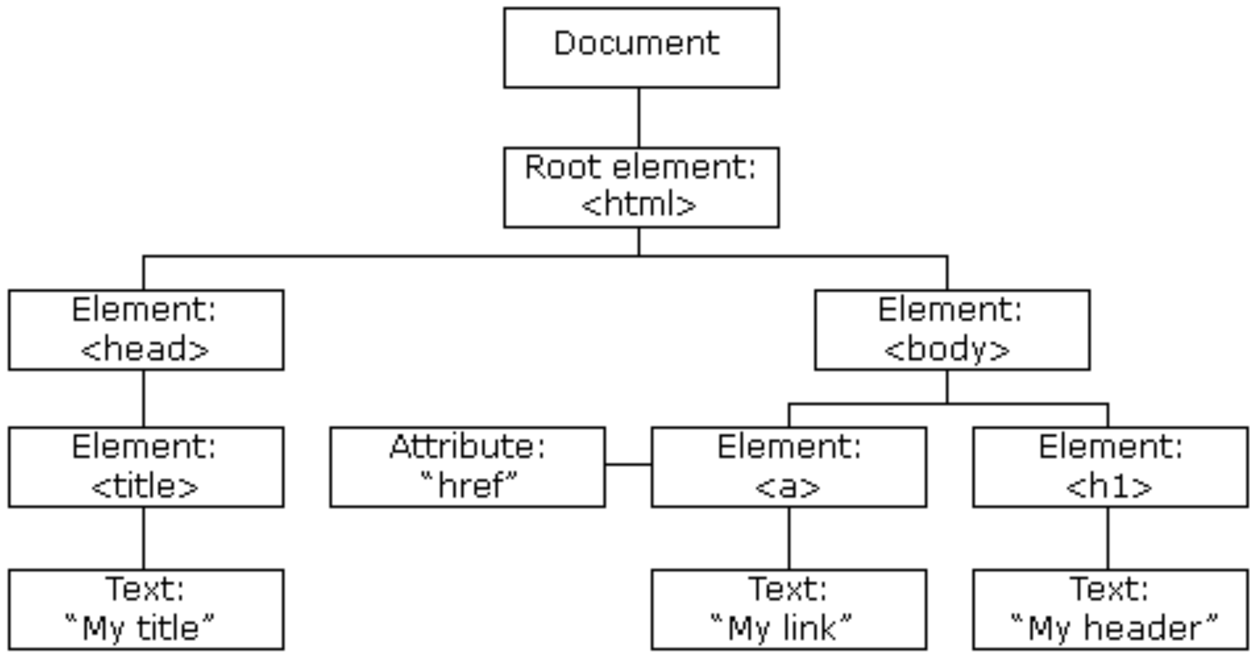
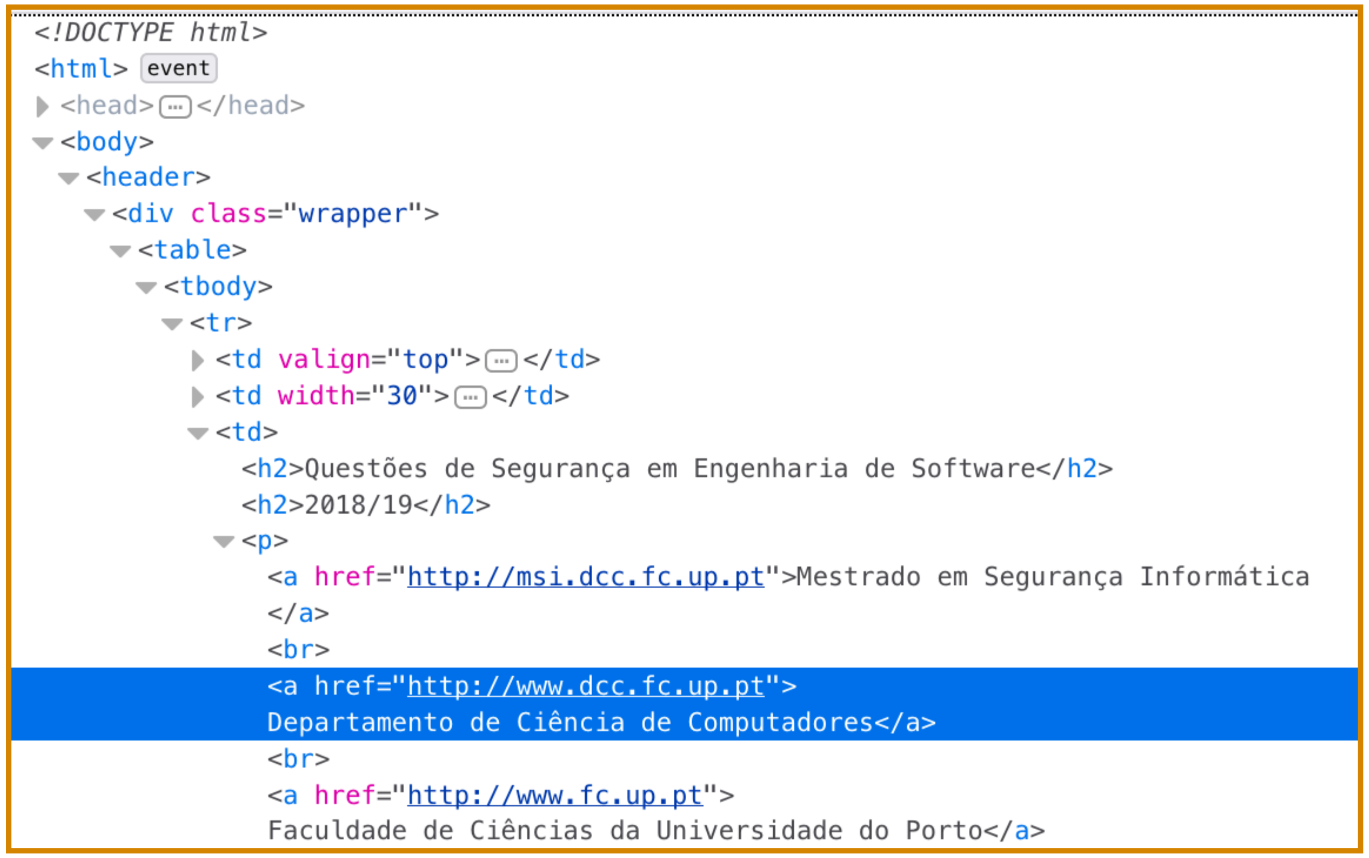


Image source: W3 Schools



```
<!DOCTYPE html>
<html> event
▶ <head>…</head>
▼ <body>
  ▼ <header>
    ▼ <div class="wrapper">
      ▼ <table>
        ▼ <tbody>
          ▼ <tr>
            ▶ <td valign="top">…</td>
            ▶ <td width="30">…</td>
            ▼ <td>
                <h2>Questões de Segurança em Engenharia de Software</h2>
                <h2>2018/19</h2>
              ▼ <p>
                  <a href="http://msi.dcc.fc.up.pt">Mestrado em Segurança Informática
                  </a>
                  <br>
                  <a href="http://www.dcc.fc.up.pt">
                  Departamento de Ciência de Computadores</a>
                  <br>
                  <a href="http://www.fc.up.pt">
                  Faculdade de Ciências da Universidade do Porto</a>
```

Firefox DOM Inspector

- The Document-Object model (DOM) is a tree-abstraction for documents:
  - an HTML (but also XML, XHTML, SVG, …) document is treated as a tree structure where in each node is an object representing a part of the document.
  - tree nodes can be visited, created/added/deleted, and have associated attributes like event handlers and styles.
  - A W3C standard until 2004, now maintained by the WHATWG group — check the live document for the current DOM specification.
  - Browsers represents HTML document in an internal structure similar to the DOM - major browsers use the WebKit Webcore component for that purose

The W3 School form example again

Source: ["PHP - AJAX and MySQL"](), W3 schools

```
<script>
function showUser(str) {
  . . .
  xmlhttp.onreadystatechange=function()  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200) {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
  }
  xmlhttp.open("GET","getuser.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>
<form>
<select name="users" onchange="showUser(this.value)">
<option value="">Select a person:</option>
<option value="1">Peter Griffin</option>
. . .
</select>
</form>
<div id="txtHint"><b>Person info will be listed here...</b></div>
```

- Javascript interaction with the DOM:
  - 1. `showUser()` function invoked as an event handler for the HTML form
  - 2. This function in turn performs an asynchronous server request
  - 3. The response callback changes the document by filling the HTML of the `txtHint` element.

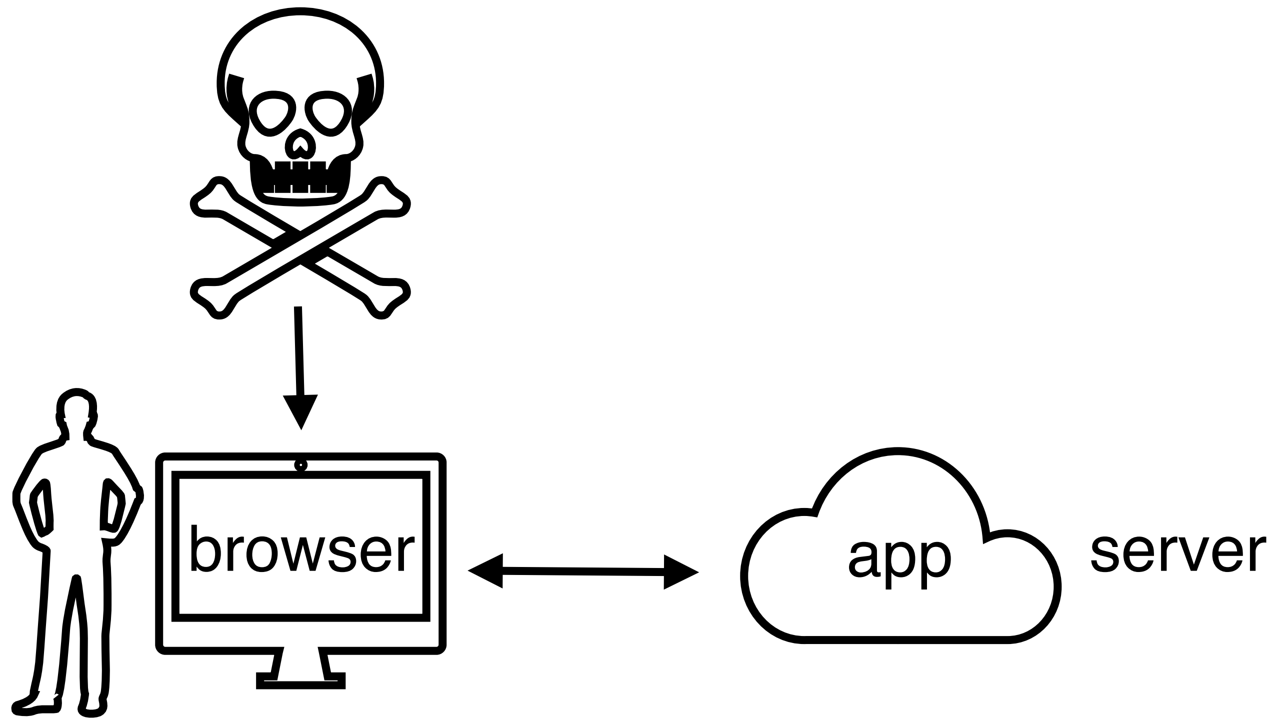
Javascript DOM API — outlook

- A brief overview of some of the functionality in the Javascript DOM API … accessible through **`document,`** the top-level object that represents the DOM:
  - Basic attributes:
    - `title referrer URL hash cookie readyState`
  - Page elements:
    - `head body forms scripts links`
    - `getElementById(elementId)`
    - `querySelector(cssSelector)`
  - Modification methods:
    - `write(anything) writeln(anything)`: append output to the document
    - `createElement() createEvent() execCommand() addEventListener()`
- A wide attack surface for malicious scripts!

Javascript — language & API

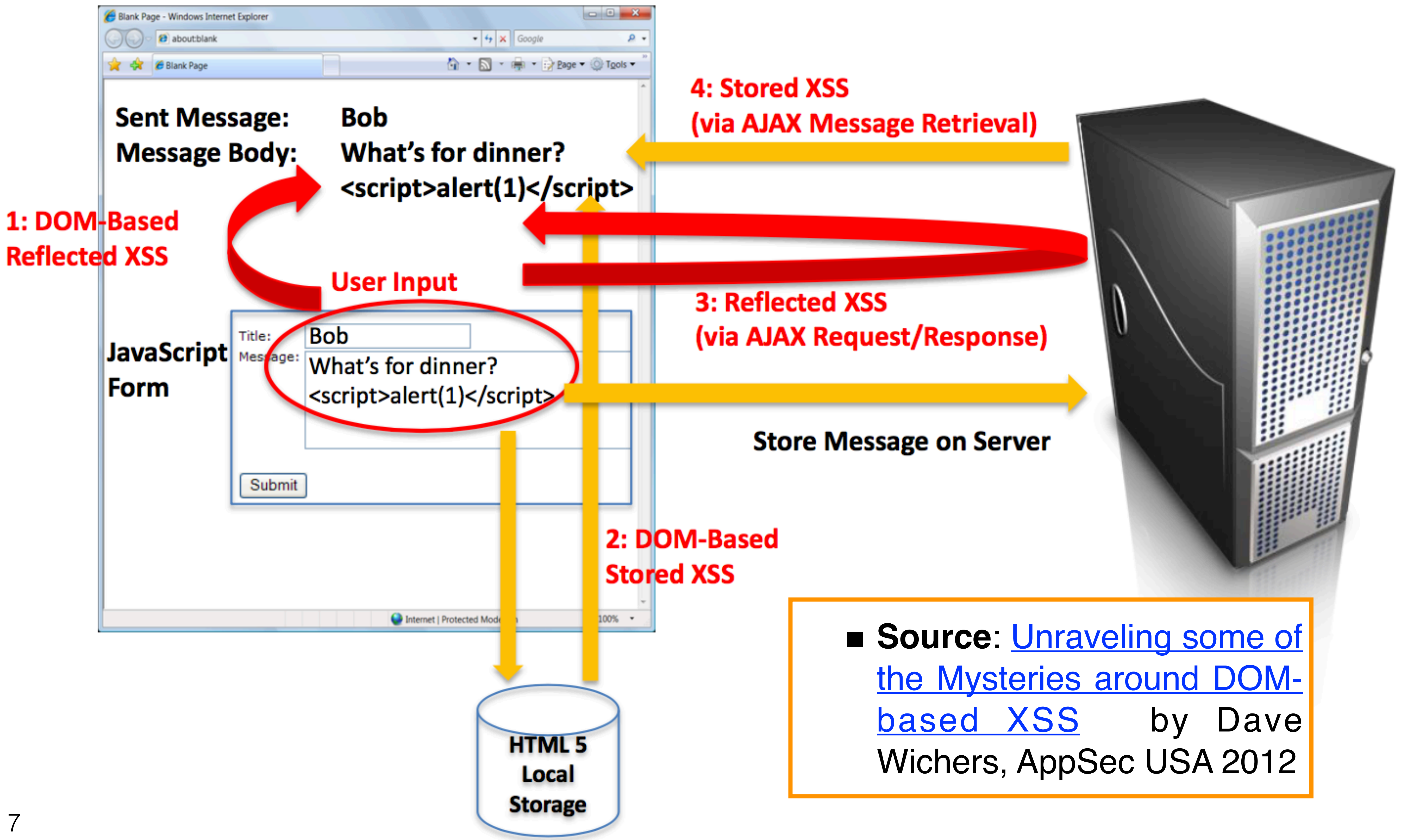
- Java is self-modifiable. Even core functionality may be overridden (e.g. array functionality).
- Like most scripting languages, an `eval()` function is part of the standard library, allowing arbitrary input to be interpreted as code (basic principle: don't use it!)
- The language has quite cumbersome aspects , lack of abstractions, and "strange" corner-cases:
  - James Mickens on Javascript: "Life is Terrible, Let's Talk about the web"
  - John K. Paul — "Javascript: The real bad parts"
  - wtfjs.com
- Javascript is weakly typed coupled with "strange" corner-cases. This also aids in unreliable or insecure behavior.
  - but you can use (or gradually transition Javascript code to) TypeScript!
  - … and consider emerging languages like Elm or PureScript

DOM-based XSS



- Malicious input is delivered in several forms: untrusted Javascript library, email links, etc.
- Attack takes advantage of a **client-side browser vulnerability for executing malicious scripts.**
- Remote (i.e. server) interaction may play some role in serving the adversary's purpose (e.g. storing the malicious link, scripts, etc), but this need not happen.
  - Stored or Persistent XSS: exploits executed when page is sent and rendered - relies on a server vulnerability.
  - DOM-based XSS: exploits executed at some point after page is loaded, relying on a DOM-based vulnerability.

XSS types compared


Blank Page - Windows Internet Explorer
about:blank
Sent Message: Bob
Message Body: What's for dinner? <script>alert(1)</script>
1: DOM-Based Reflected XSS
User Input
JavaScript Form
Title: Bob
Message: What's for dinner? <script>alert(1)</script>
Submit
4: Stored XSS (via AJAX Message Retrieval)
3: Reflected XSS (via AJAX Request/Response)
Store Message on Server
2: DOM-Based Stored XSS
HTML 5 Local Storage

- **Source**: Unraveling some of the Mysteries around DOM-based XSS by Dave Wichers, AppSec USA 2012

Simple code injection example

Malicious script

```
<script>
  // Get malicious input from query string and unescape it
  var pos = document.URL.indexOf("evil=")+9;
  var evilScript=document.URL.substring(pos,document.URL.length);
  // Make it take effect
  document.write(unescape(evilScript));
</script>
```

Code injection possible using:

```
queryStringAttack.html?evil=<script>. . . </script>
```

- Query string associated to HTML page. No server interaction for triggering the exploit.
- Anchor '#' (i.e., document.hash instead of document.URL) also exploitable in similar

Eval-based injection example

```
goodJSON = '{  "x": "1", "y": "2" }';
jsObj =  eval('(' + goodJSON + ‘)');

evilJSON = '{  "x": "1", "y": window.alert(“oho”) }';
jsObj =  eval('(' + evilJSON + ‘)');

// The right way to parseJSON
jsObj = JSON.parse(evilJSON);  // throws SyntaxError
```

- **One of the most common uses of eval was (and ma still is) to parse JSON data onto Javascript objects.**
  - “The Eval that Men Do — A Large-scale Study of the Use of Eval in JavaScript Applications”, Richards et al., ECOOP’11
- JSON encoding/decoding functions were not available initially for Javascript.

Functionality overriding

1
```
<script type="text/javascript">
var leakVar;
var oldArray = Array;
Array = function () {
  a = oldArray();
  leakVar = a;
  return a;
}
</script>
```

2
```
<script>
var myArray = Array();
myArray[0] = 'data 1 ';
myArray[1] = 'data 2';
</script>
```

3
```
<script>
  window.alert(‘The data: ' + leakVar);
</script>
```

- Javascript fragments above:
  - (1) overrides the Array() constructor such that a leakVar will point to the created array
  - (2) normal code that uses Array() to define an array with two entries
  - (3) makes use of leakVar to leak the array data
- Example inspired by Anatomy of a Subtle JSON Vulnerability”, combining functionality overriding with CSRF, JSON parsing and same-origin policy vulnerabilities

The recent ESLint backdoor case

- ESLint is a popular static analysis tool for Javascript. It used by developers to ensure their code adheres to certain standards and coding rules, in some cases including security aspects.
- Attackers gained access to the NPM repository account for ESLint and inserted a “backdoor script” into ESLint package version 3.7.2.
- The “backdoor script” essentially fetched a malicious script from PasteBin, the actual exploit payload. The payload read the NPM authentication token from the local machine and delivered it
- The ESLint 3.7.2 module was subsequently removed from NPM.
- Further info:
  - “ESLint backdoor: revoke all the tokens”, sqreen.io blog
  - “Postmortem for Malicious Packages Published on July 12th, 2018”, eslint.org
- ESLint runs on Node.js rather than a web browser. Still packages used in a web browser context can be compromised similarly.
- In general, malicious Javascript package inclusion is a serious problem. See for instance: “You Are What You Include: Large-scale Evaluation of Remote JavaScript Inclusions”, Nikiforakis et al., CCS’12

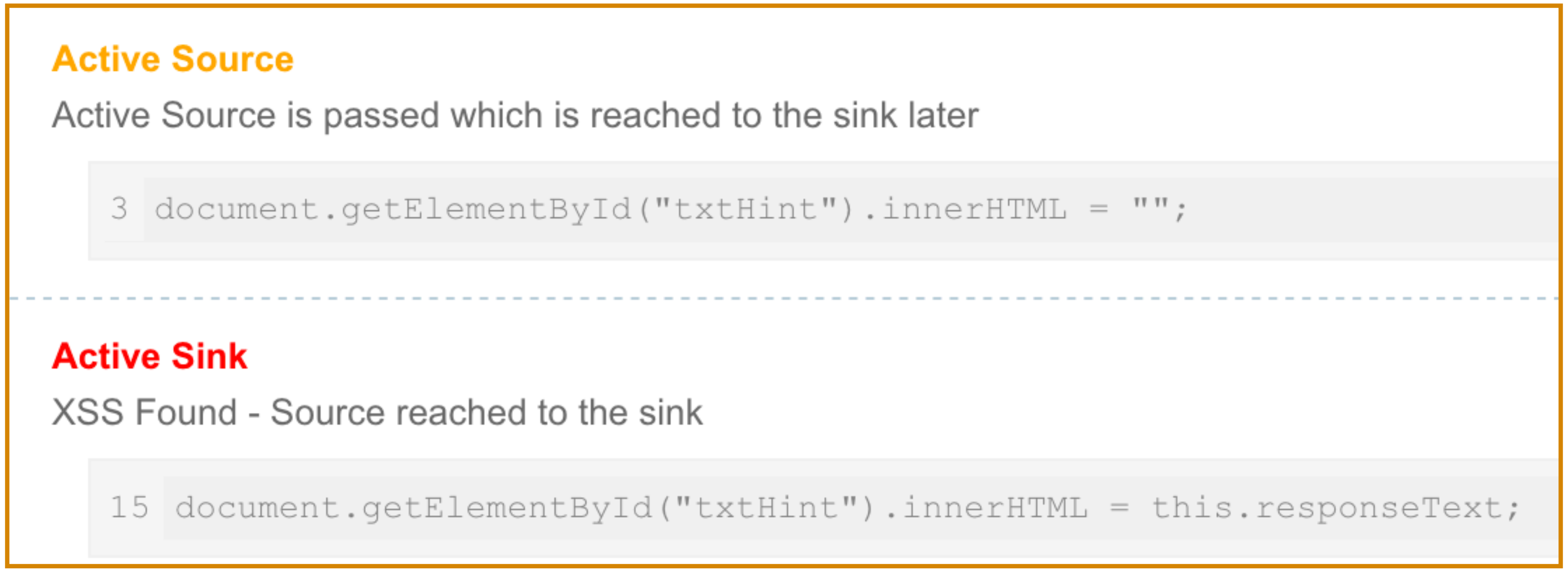
Handling DOM-based XSS

- **Detection** — Dave Wichers remarks that:
    - "*It's like trying to find code flaws in the middle of a dynamically compiled, running, self modifying, continuously updating engine while all the gears are spinning and changing.*"
    - "*Manual code review is hell!*"
    - Source: "Unraveling some of the Mysteries around DOM-based XSS", APPSEC 2012
    - For code reviewing there are few freely-available security-oriented static analysis tools, and comparatively less powerful than for other languages — we will have a quick demo using jsprime next.
- **Prevention** — usual techniques adapted to the context of the DOM-based environment as in OWASP's DOM based XSS Prevention Cheat Sheet
    - **General methodology**: Be careful with untrustworthy input reaching security-sensitive sinks.
    - **Techniques**: input validation/sanitization, output encoding, several coding recipes.

Using jsprime — examples

Javascript examples and jsprime reports are provided online in a ZIP file.
Screenshots are shown below for the various examples we considered:
(1) W3 schools, (**2)** query string attack, **(3)** JSON parsing using eval.

(1)



Active Source

Active Source is passed which is reached to the sink later

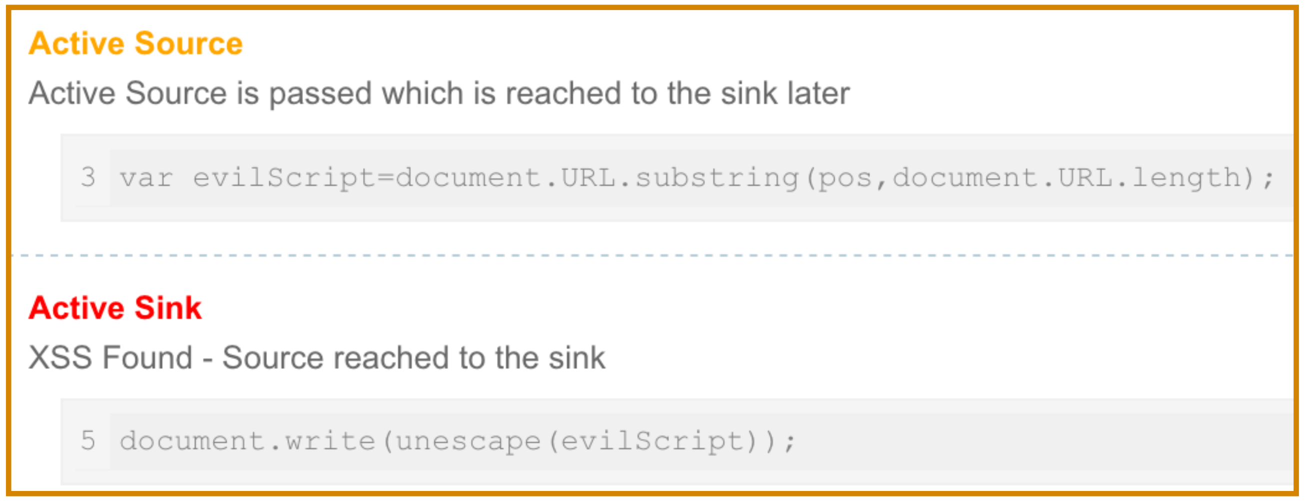
```
document.getElementById("txtHint").innerHTML = "";
```

Active Sink

XSS Found - Source reached to the sink

```
document.getElementById("txtHint").innerHTML = this.responseText;
```

(2)



Active Source

Active Source is passed which is reached to the sink later

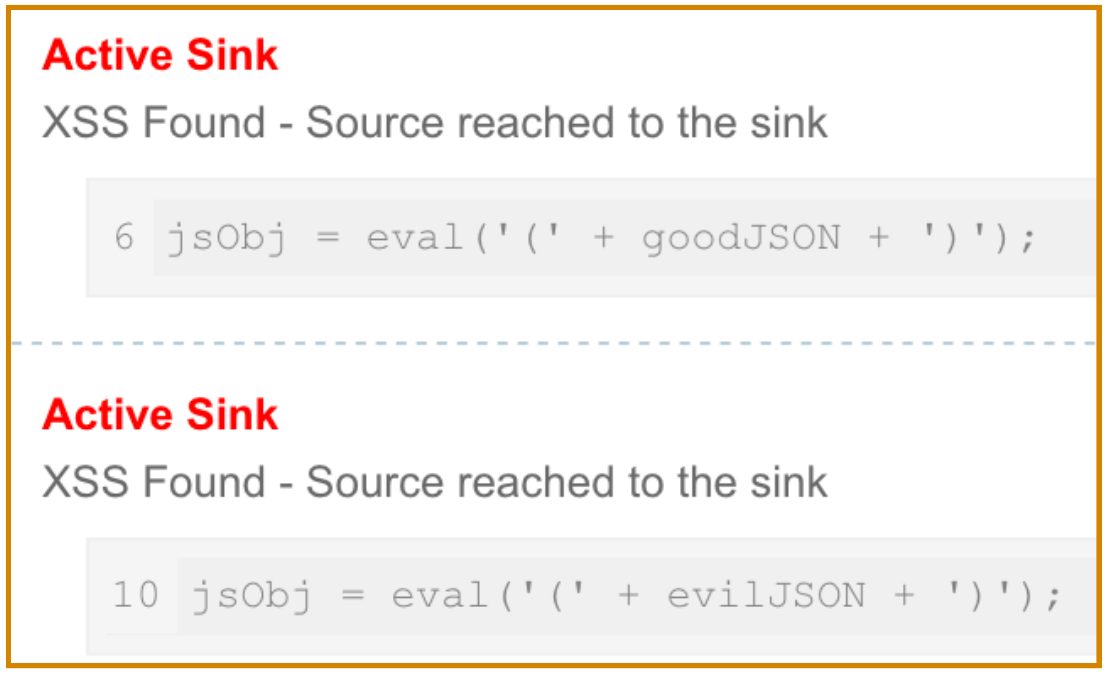
```
var evilScript=document.URL.substring(pos,document.URL.length);
```

Active Sink

XSS Found - Source reached to the sink

```
document.write(unescape(evilScript));
```

(3)



Active Sink

XSS Found - Source reached to the sink

```
jsObj = eval('(' + goodJSON + ')');
```

Active Sink

XSS Found - Source reached to the sink

```
jsObj = eval('(' + evilJSON + ')');
```

Also look out for …

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML = localStorage.getItem("lastname");
```

- HTML 5 introduced local browser storage of key-value pairs (like cookies), an extra attack surface.
  - One more facility to store sensitive or malicious data that can be controlled programatically.
  - Above: a "predictable" usage example from w3schools … (!) How can it go wrong?
  - WebSQL and IndexedDB allow structured databases, though adoption of one or the other has not been peaceful (both out of HTML 5). Firefox only supports IndexedDB, Safari and Google also support WebSQL.
- CSS-based vulnerabilities are not covered here
  - CSS Exfil
  - Microsoft Internet Explorer Cascading Style Sheets Remote Code Execution Vulnerability