Berner Fachhochschule, Technik und Informatik

Advanced Web Technology
11) Web Security : ESAPI

Dr. E. Benoist

Fall Semester 2010/2011

Table of Contents

OWASP

- **The Open Web Application Security Project (OWASP)**
  - is a worldwide free and open community focused on improving the security of application software.
- **Products developed within OWASP**
  - OWASP Top 10 : 10 most present vulnerabilities for web sites
  - Web Goat : a deliberately insecure J2EE web application
  - Web Scarab : framework for analysing applications that communicate using the HTTP and HTTPS protocols.
  - Application Security Verification Standards (ASVS) : defines four levels of application-level security verification for Web applications.
- **ESAPI**
  - Enterprise Security API: helps software developers guard against security-related design and implementation flaws.

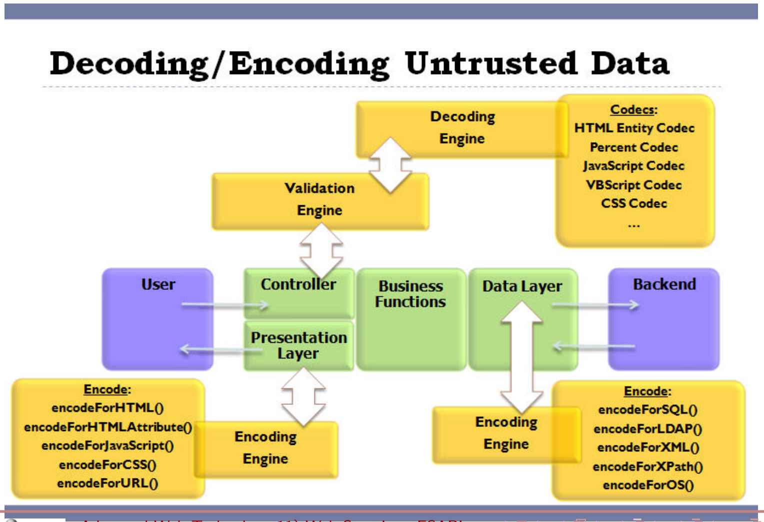
Architecture of a Web Application

- **Presentation Layer**
  - Responsible for decoding requests and encoding html responses
  - For us uses JSF
  - Can be a templating system in PHP (Smarty for instance)
- **Business Layer**
  - Responsible for business logic
  - Written in POJO for us
  - Contains PHP commands
- **Persistence Layer**
  - Responsible for transferring objects in DataBase and vice-versa
  - For use a JPA implementation (Hibernate or TopLink for instance)
  - In PHP the PEAR MDB2 library
- **DataBase Layer**

Security Needs in a Web Application

- **Presentation Layer**
  - Encoding in HTML (against XSS attacks)
  - Verification of authentication for accessing resources
  - Validate the strings as numbers or valid passwords (string or weak)
  - Encode reference to resource (transform direct to indirect reference)
- **Business Layer**
  - Rights managements of users on functions
  - Encryption of configuration parameters
  -
- **Data / Database layer**
  - Encode SQL (against SQL injection)
  - Access to resource using an alias (transform indirect to direct reference)
  - Verify the rights a user has on a specific resource
- . . .

Decoding / Encoding Untrusted Data[1]

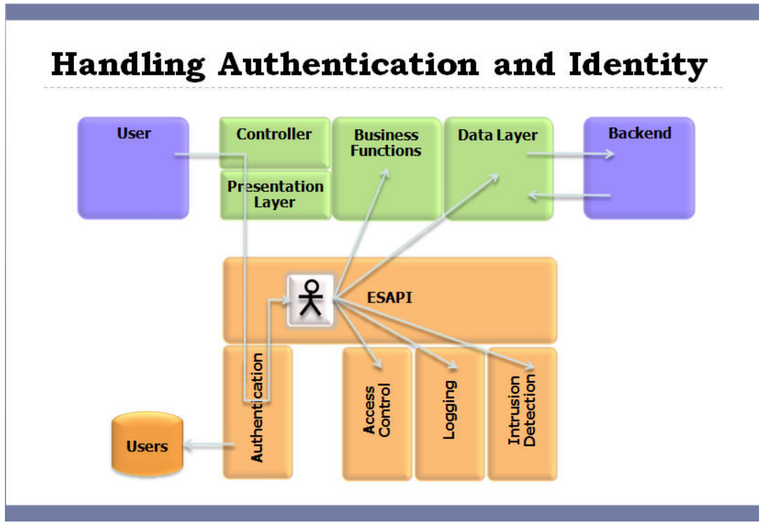


Source: Javadoc documentation of the ESAPI package

Decoding / Encoding Untrusted Data (Cont.)

- **One should never trust input given by the user**
  - It must be first canonicalized using function `canonicalize()`
  - It reduces a possibly encoded string down to its simplest form.
- **Work with the simplest string**
- **May be stored encoded**
  - In the DataBase using `encodeForSQL()`
  - In a LDAP server `encodeForLDAP` or `encodeForDN` (for distinguished name).
- **Encode for the output**
  - For use in a HTML document `encodeForHTML` or `encodeForHTMLAttribute`
  - For use in a Javascript program `encodeForJavascript`

Handling Authentication and Identity



The Authenticator Interface

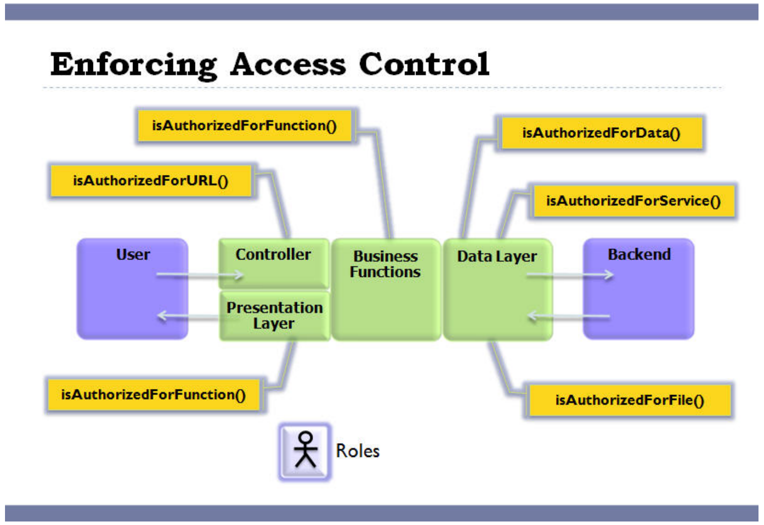
- **defines a set of methods for generating and handling account credentials and session identifiers.**
- **Application must set current user as soon as possible**
  - The value of `getCurrentUser()` is used in several other places in this API.
- **Method for login uses request and or session parameter to retrieve the user**

```
try {
User user = ESAPI.authenticator().login(request, response);
// continue with authenticated user
} catch (AuthenticationException e) {
// handle failed authentication (it's already been logged)
}
```

The Authenticator Interface (Cont.)

- **Handle password**
  - Can change the password of the current user with `changePassword()`
  - Can generate a new strong password `generateStrongPassword()`
  - Method `verifyPasswordStrength()` ensures that the pwd site-specific complexity requirements, like length or number of character sets.
  - Generate a hash of the password using account name as a salt `hashPassword()`
- **Handles login / logout**
  - Get username, password or User in session from request information in method `login()`
  - `logout()`

Enforcing Access Control



Enforcing Access Control

- **Use the login done by `Authenticator`**
- **Interface `AccessController` must be extended according to firm policy**
  - An existing `DefaultAccessController` class exists
  - Reads its rules out of a configuration file
  - Uses `AccessControlRules` described in the file

AccessController

- **Each time a resource is accessed, one must "Assert" its availability**

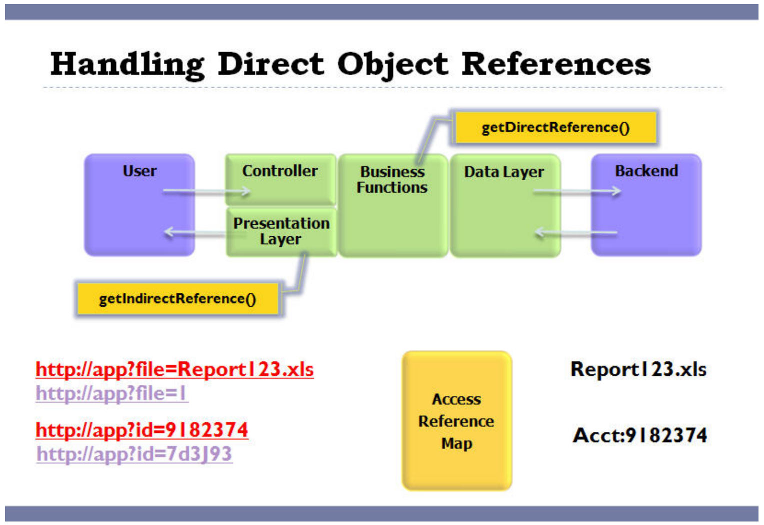
```
try {
      ESAPI.accessController().assertAuthorized("↘
      →businessFunction",
      runtimeData);
      // execute BUSINESS_FUNCTION
} catch (AccessControlException ace) {
... attack in progress
}
```

- **an attacker can attempt to invoke any business function or access any data in your application.**
  - Access control checks in the user interface should be repeated in both the business logic and data layers.

Direct Access Reference

- **Vulnerability: gives access to internal structure**
  - Can be files `www.toto.com/download.do?res=mysecretfile.pdf`
  - Can be a database index `edit.do?page=123`
  - and other types of direct object references
- **As a rule, developers should not expose their direct object references as it enables attackers to attempt to manipulate them.**

AccessReferenceMap



AccessReferenceMap

- **Indirect references are handled as strings, to facilitate their use in HTML**
- **When a reference is sent to the browser**
  - Internal reference is stored in the map,
  - the indirect reference is sent to the browser
  - it is random generated
- **When a reference is received from the browser**
  - The indirect reference is received
  - It is converted back to a direct reference using the map.
- **If per-user AccessReferenceMaps are used, then request forgery (CSRF) attacks will also be prevented.**

AccessReferenceMap (Cont.)

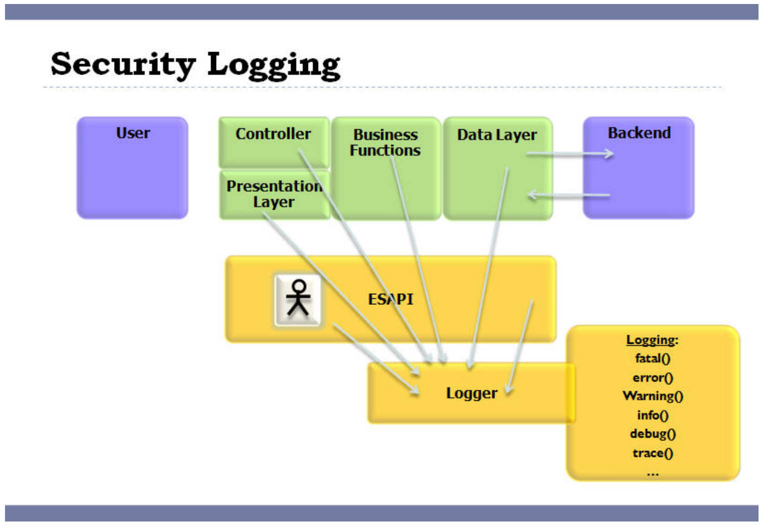
- **Example of use**

```
Set fileSet = new HashSet();
 fileSet.addAll(...); // add direct references (e.g. File objects)
 AccessReferenceMap map = new AccessReferenceMap( fileSet )↘
 →;
 // store the map somewhere safe − like the session!
 String indRef = map.getIndirectReference( file1 );
 String href = "http://www.aspectsecurity.com/esapi?file=" + ↘
 →indRef );
 ...
 // if the indirect reference doesn't exist, it's likely an attack
 // getDirectReference throws an AccessControlException
 // you should handle as appropriate
 String indref = request.getParameter( "file" );
 File file = (File)map.getDirectReference( indref );
```

Intrusion Detection

- **Based on two classes**
  - `Logger` charged to write all the information gathered in the application
  - `IntrusionDetector` that "analyses" the values gathered and reacts according to given "rules".

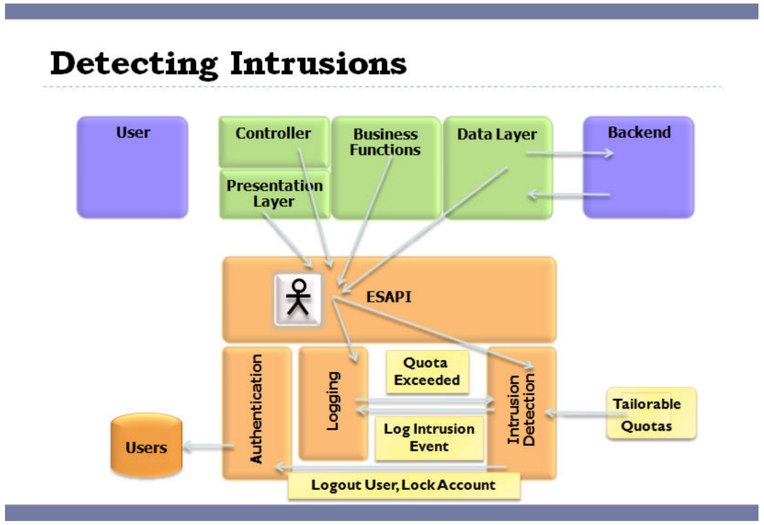
Logger



Logger (Cont.)

- **The Logger interface defines a set of methods that can be used to log security events.**
- **hierarchy of logging levels**
  - Can be configured at runtime level
  - All events bellow a the current threshold are discarded.
  - Levels are :fatal (highest value), error, warning, info, debug, trace (lowest value)
  -
- **Loggers must fulfil the following requirements**
  - Ensure that HTML characters are encoded (for persons looking at stats in a browser)
  - Encode and CLRF in order to prevent log injection attacks.
  - Record for each event : the identity of the user, the description of the event, success/failure of the event, security level, IP address, a time stamp

IntrusionDetector



IntrusionDetector (Cont.)

- **Track security relevant events and identify attack behavior.**
- **The interface is currently designed to accept exceptions as well as custom events.**
- **Implemented in the `DefaultIntrusionDetector` class**
  - This implementation monitors EnterpriseSecurityExceptions to see if any user exceeds a configurable threshold in a configurable time period.
  - For instance, if a user exceeds 10 input validation issues in a 1 minute period.
  - Or if there are more than 3 authentication problems in a 10 second period.
- **More complex implementation are possible**

Configure ESAPI for your business

- **Default implementation can be configured**
  - Use config files for defining users and roles, such as business rules
  - Very generic and simplified
  - Needs to be extended
- **Develop your own implementation of Interfaces**
- **Need to integrate the new classes in the framework**
  - Done using "(pseudo)-singleton pattern" or "(pseudo)-Factory Pattern"

Pseudo-Singleton Pattern

(uses only static methods)

```
myauthenticator = new MyAuthenticator();
ESAPI.setAuthenticator(myauthenticator); //register with ↘
→locator class
authenticator = ESAPI.getAuthenticator();
authenticator.login(...); //use your implementation
```

Change the instances

- **At runtime change the instance used in the**
- **The instance of class ESAPI contains the default values / changed values**

Conclusion

- **ESAPI groups all the security items in one place**
  - Easyer to maintain than code in all the application
- **ESAPI has been tested and developed by security specialists**
  - Never reinvent the wheel
  - Amateurism in security is no security
- **ESAPI can easily be tailored for your business needs**
  - Implement the interfaces
  - Replace the default implementation in the ESAPI class (using set methods).

References

- The ESAPI Toolkit web pages `http://www.owasp.org`
- OWASP Javadoc of ESAPI `http://owasp-esapi-java.googlecode.com/svn/trunk_doc/2.0-rc4/index.html`