



OWASP Testing Guide v3: training

Matteo Meucci

OWASP Testing Guide Lead

OWASP London
28th May 2010

Copyright © 2010 - The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License.

The OWASP Foundation
<http://www.owasp.org>

Agenda:

- ➊ The OWASP Testing Guide: the standard for verifying the security of a web application
- ➋ How to use the Guide
- ➌ The Categories of testing and main controls to test: methodology and examples
- ➍ How the Guide can improve your SDLC and help the security industry



Who am I?

Research

- ▶ OWASP-Italy Chair
- ▶ OWASP Testing Guide Lead



Work

- ▶ CEO @ Minded Security Application Security Consulting
- ▶ Working from 2001 on Information Security focusing on Application Security
- ▶ CISSP, CISA



Minded
security



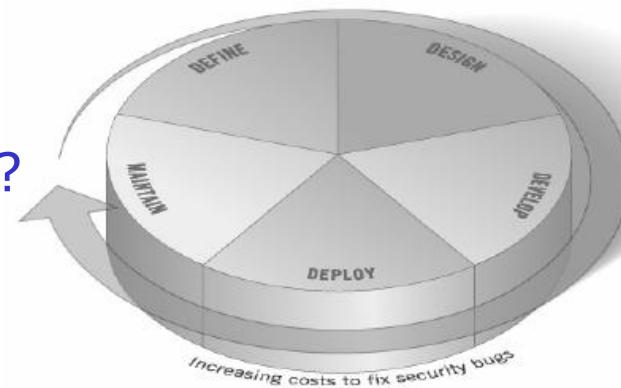
SDLC and Security processes

- ➊ Software Development Life Cycle, SDLC comprises:

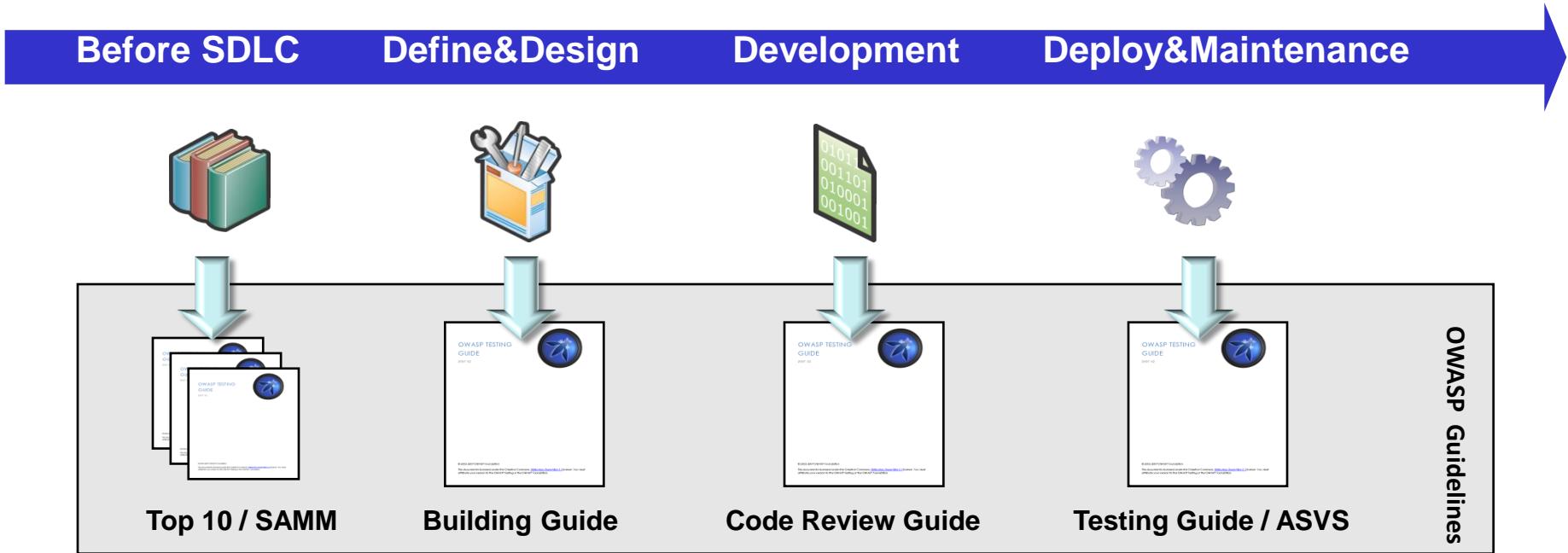
- ▶ Define
- ▶ Design
- ▶ Develop
- ▶ Deploy
- ▶ Maintain

- ➋ Which security processes to implement?

- ▶ Awareness
- ▶ Secure Code Guidelines
- ▶ Code Review
- ▶ Application Testing



SDLC & OWASP Guidelines e tools



Web Goat	.NET CSRGuard ESAPI AntiSamy AppSensor	Orizon O2 LAPSE	WebScarab SWF Intruder SQLNinja SQLMap DirBuster	OWASP Tools
----------	--	-----------------------	--	-------------

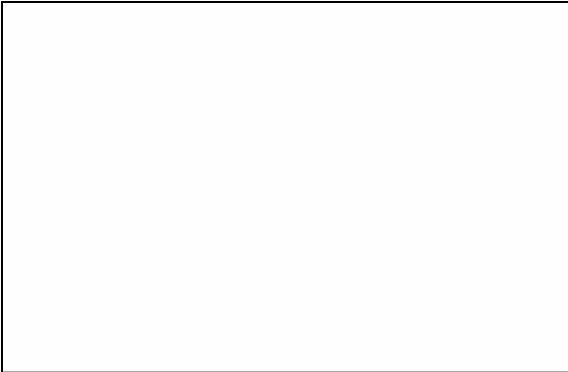


OWASP Guidelines

- Free and open source
- Books at low cost
- Cover all security checks
- Hundreds of experts
- All aspects of application security



Why do we need the OWASP Testing Guide?

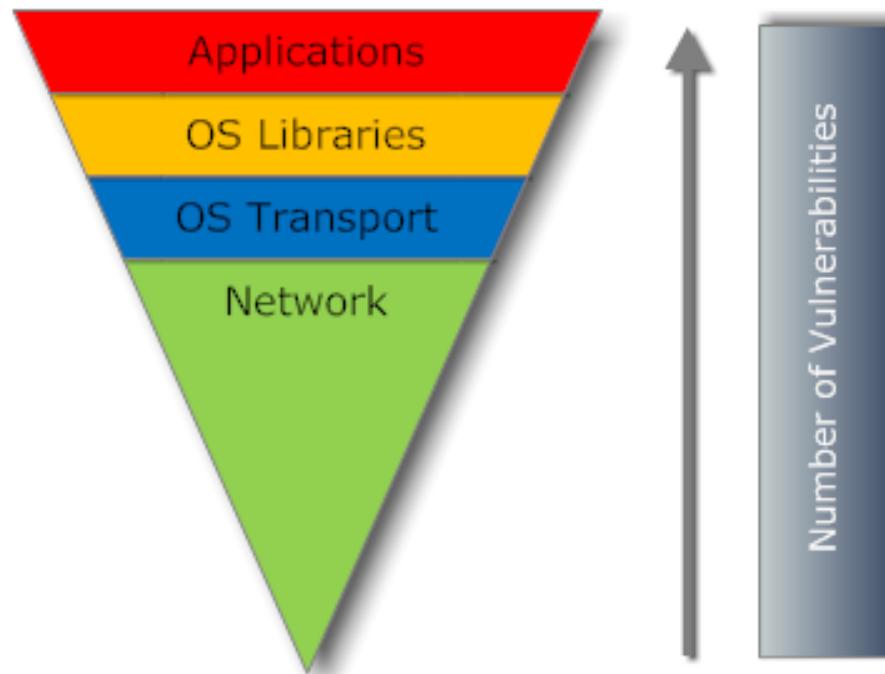


```
public class HelloWorld extends HttpServlet {  
  
    public void doGet(  
        HttpServletRequest request,  
        HttpServletResponse response)  
        throws IOException, ServletException  
{  
    response.setContentType("text/html");  
    PrintWriter out = response.getWriter();  
    out.println("<HTML><HEAD>");  
    out.println("<TITLE>Hello World</TITLE>");  
    out.println("</HEAD><BODY>");  
    out.println("Hello, " + }  
}
```



The control of defects in software security should be considered part of the process of software development.

Where are the problems for the IT?



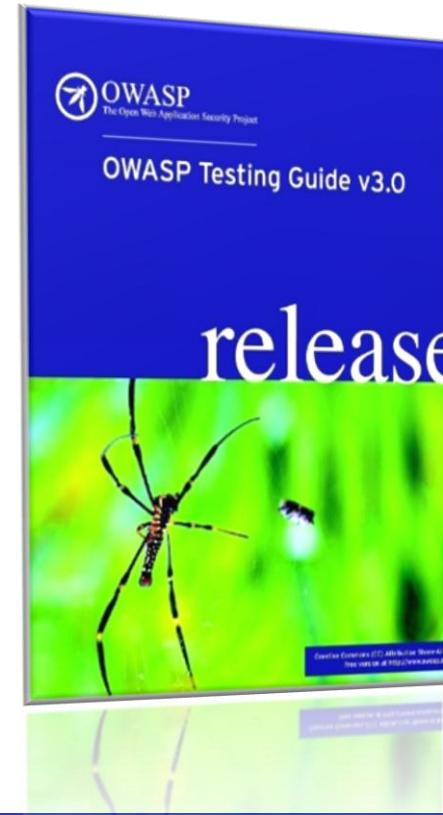
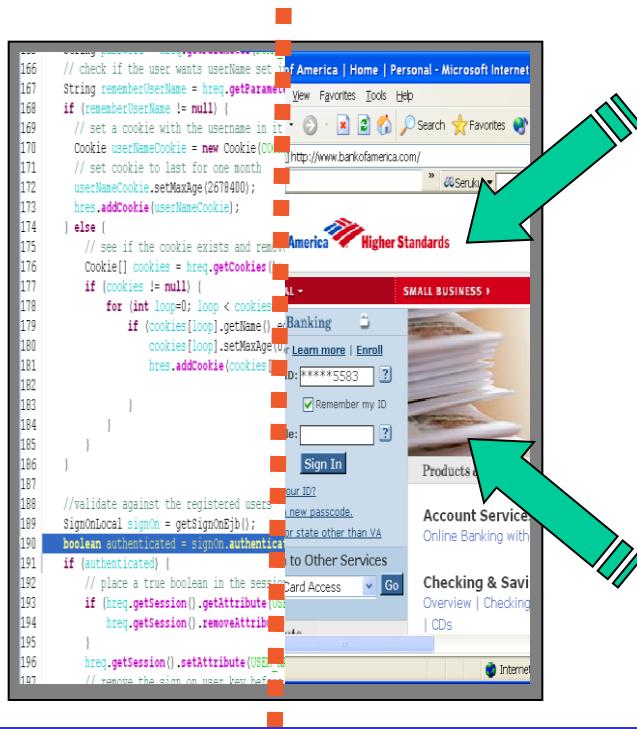
Source SANS : The Top Cyber Security Risks (Set 09)

What are the possible vulnerabilities of a web application?

- ❑ Information Disclosure
- ❑ SSL Weakness
- ❑ Configuration management weakness
- ❑ Old, backup and unreferenced files
- ❑ Access to Admin interfaces
- ❑ HTTP Methods enabled, XST permitted, HTTP Verb
- ❑ Credentials transport over an encrypted channel
- ❑ User enumeration
- ❑ Guessable user account
- ❑ Credentials Brute forcing
- ❑ Bypassing authentication schema
- ❑ Vulnerable remember pwd weak pwd reset
- ❑ Logout function
- ❑ browser cache weakness
- ❑ Bypassing Session Management Schema, Weak Session Token
- ❑ Cookies not secure
- ❑ Session Fixation
- ❑ Exposed sensitive session variables
- ❑ CSRF
- ❑ Path Traversal
- ❑ Bypassing authorization schema
- ❑ Privilege Escalation
- ❑ Bypassable business logic
- ❑ Reflected XSS, Stored XSS, DOM XSS
- ❑ Cross Site Flashing
- ❑ SQL, LDAP, ORM, XML, SSI, Code Injection
- ❑ OS Commanding
- ❑ Buffer overflow
- ❑ Locking Customer Accounts
- ❑ Buffer Overflows
- ❑ WSDL Weakness



OWASP Testing Guide



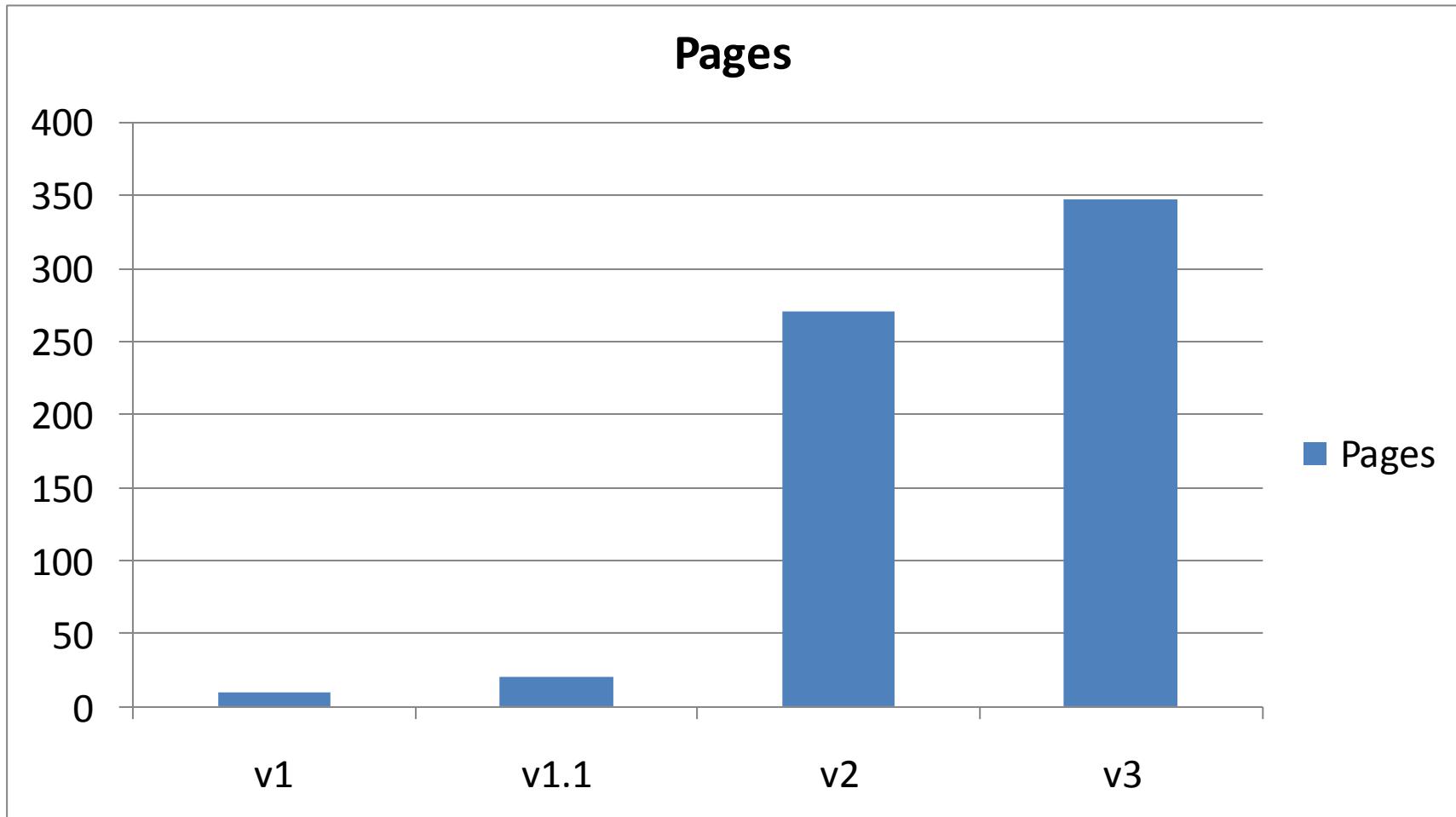
- SANS Top 20 2007
- NIST “Technical Guide to Information Security Testing (Draft)”
- Gary McGraw (CTO Digital) says: “In my opinion it is the strongest piece of Intellectual Property in the OWASP portfolio”



Testing Guide history

- ➊ January 2004
 - ▶ "The OWASP Testing Guide", Version 1.0
- ➋ July 14, 2004
 - ▶ "OWASP Web Application Penetration Checklist", Version 1.1
- ➌ December 25, 2006
 - ▶ "OWASP Testing Guide", Version 2.0
- ➍ December 16, 2008
 - ▶ "OWASP Testing Guide", Version 3.0 – Released at the OWASP Summit 08

Project Complexity



Testing Guide v3: Index

1. Frontispiece
2. Introduction
3. The OWASP Testing Framework
4. Web Application Penetration Testing
5. Writing Reports: value the real risk

Appendix A: Testing Tools

Appendix B: Suggested Reading

Appendix C: Fuzz Vectors

Appendix D: Encoded Injection



Web Application Penetration Testing



What is a Web Application Penetration Testing?

- ▶ The process involves an active analysis of the application for any weaknesses, technical flaws or vulnerabilities
- ▶ It's a Black Box process (we don't know the source code of the application)
- ▶ **Methodology** + tools (OWASP WebScarab, SQLmap, etc..)



Our approach in writing this guide

- ▶ Open
- ▶ Collaborative



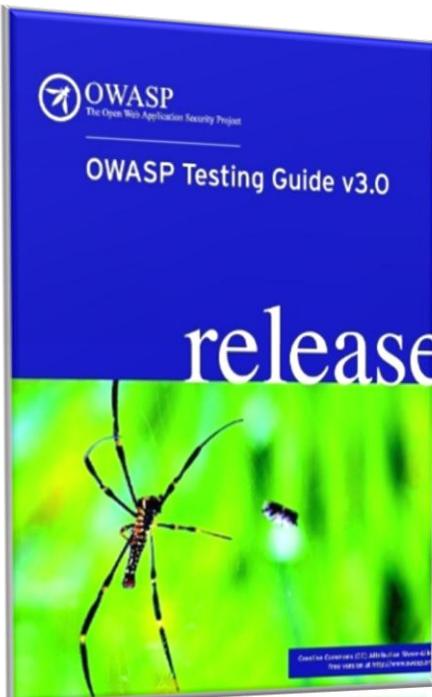
Defined testing methodology

- ▶ Consistent
- ▶ Repeatable
- ▶ Under quality



How to use the Testing Guide

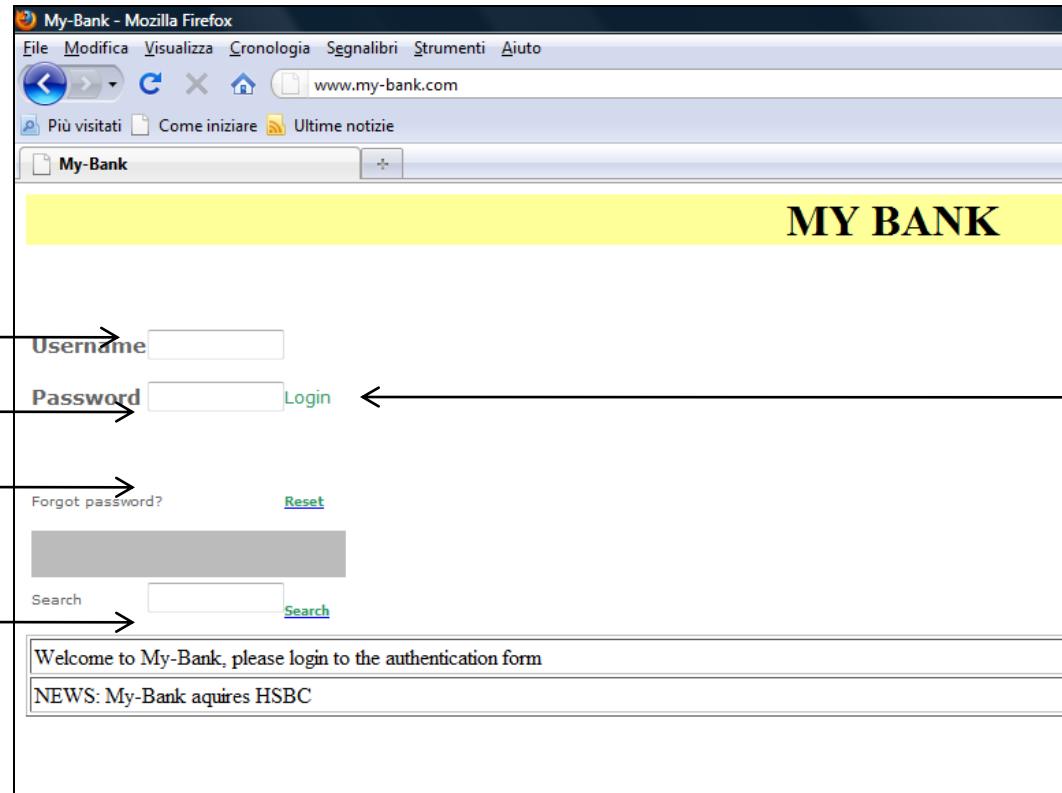
Focus: the methodology

A screenshot of a Firefox browser window showing a login page for "My-Bank". The title bar says "My-Bank - Mozilla Firefox". The main content area has a yellow header with "MY BANK". Below it is a form with "Username" and "Password" fields, a "Login" button, and links for "Forgot password?" and "Reset". There is also a search bar with a "Search" button. At the bottom of the page, there are two news items: "Welcome to My-Bank, please login to the authentication form" and "NEWS: My-Bank aquires HSBC".

● OWASP Methodology

● Site to verify (PoC)

The methodology: a plan for the testing



Web Application Threats

- 1) Incorrect handling of the environment of the application configurations
 - ▶ Exploiting versions of outdated sw and incorrect configurations can access the file system, read / modify data, access to administration interfaces
- 2) Authentication mechanisms are not robust
 - ▶ Finding username / password for a set of customers
 - ▶ Bypass the authentication scheme (identity theft)
- 3) Weak Authorization mechanisms
 - ▶ Disclosure of confidential information, privilege escalation
 - ▶ Access to information / documents to other users



Web Application Threats (2)

4) Use of weak Session Management mechanism

- ▶ Session hijacking (theft of temporary user session)
- ▶ Forcing a user to perform unintended action (eg transfer)

5) Input / output not properly validated

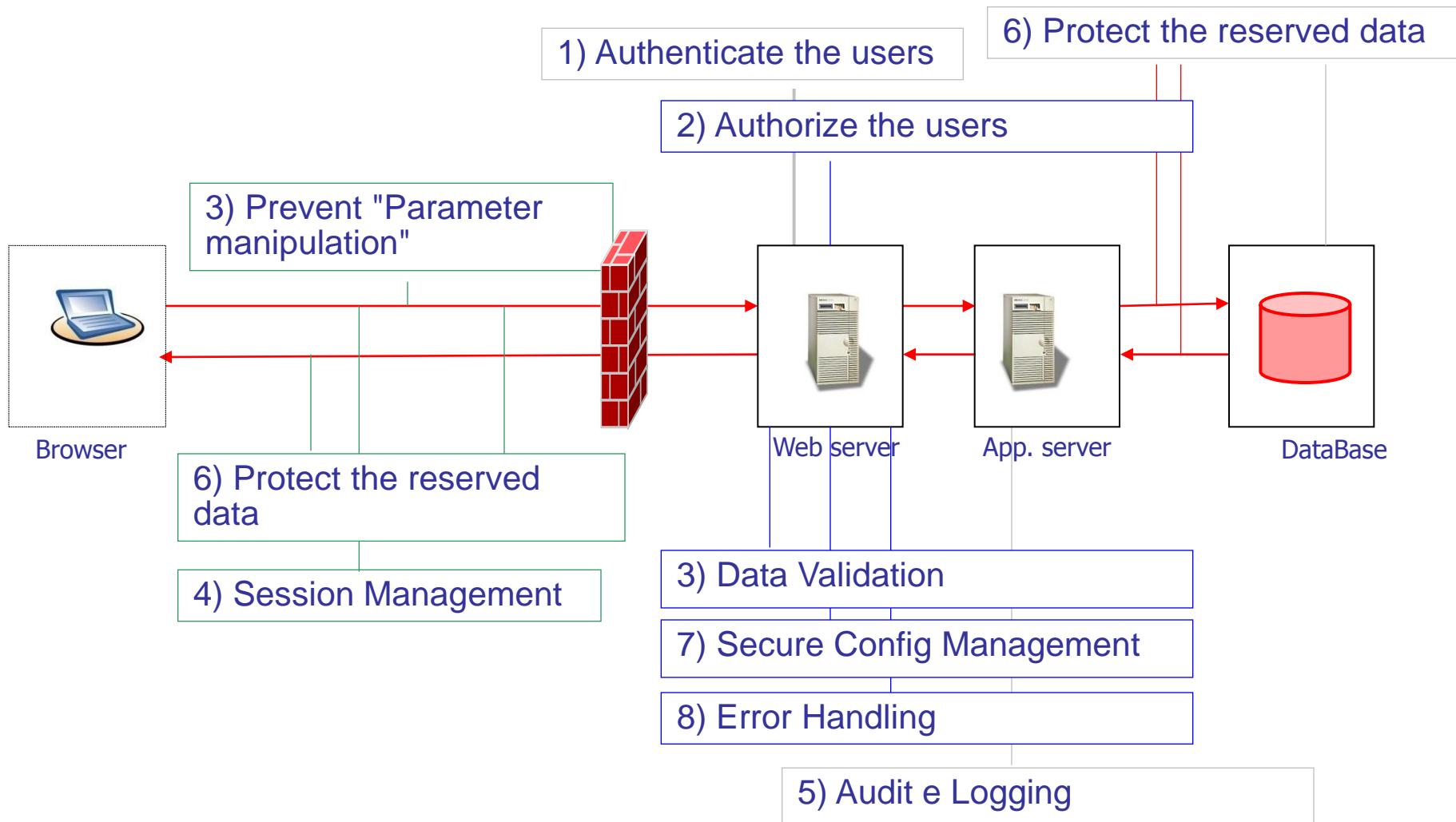
- ▶ Attacks on the browser client (identity theft)
- ▶ Read / edit information on SQL / LDAP server
- ▶ Sending commands to the server
- ▶ Upload code, shell, impaired application

6) Use of weak cryptographic protocols

- ▶ Disclosure of information in transit user to the server
- ▶ Disclosure of information on Data Base



Overview of the security controls



Testing Guide Categories

Category	Ref. Number	Test Name	Vulnerability
Information Gathering	OWASP-IG-001	Spiders, Robots and Crawlers	N.A.
	OWASP-IG-002	Search Engine Discovery/Reconnaissance	N.A.
	OWASP-IG-003	Identify application entry points	N.A.
	OWASP-IG-004	Testing for Web Application Fingerprint	N.A.
	OWASP-IG-005	Application Discovery	N.A.
	OWASP-IG-006	Analysis of Error Codes	Information Disclosure
Configuration Management Testing	OWASP-CM-001	SSL/TLS Testing (SSL Version, Algorithms, Key length, Digital Cert. Validity)	SSL Weakness
	OWASP-CM-002	DB Listener Testing	DB Listener weak
	OWASP-CM-003	Infrastructure Configuration Management Testing	Infrastructure Configuration management weakness
	OWASP-CM-004	Application Configuration Management Testing	Application Configuration management weakness
	OWASP-CM-005	Testing for File Extensions Handling	File extensions handling
	OWASP-CM-006	Old, backup and unreferenced files	Old, backup and unreferenced files
	OWASP-CM-007	Infrastructure and Application Admin Interfaces	Access to Admin interfaces
	OWASP-CM-008	Testing for HTTP Methods and XST	HTTP Methods enabled, XST permitted, HTTP Verb
Authentication Testing	OWASP-AT-001	Credentials transport over an encrypted channel	Credentials transport over an encrypted channel
	OWASP-AT-002	Testing for user enumeration	User enumeration
	OWASP-AT-003	Testing for Guessable (Dictionary) User Account	Guessable user account
	OWASP-AT-004	Brute Force Testing	Credentials Brute forcing

Testing paragraph template



Brief Summary

Describe in "natural language" what we want to test. The target of this section is non-technical people (e.g.: client executive)



Description of the Issue

Short Description of the Issue: Topic and Explanation



Black Box testing and example

- ▶ How to test for vulnerabilities:
- ▶ Result Expected:

...



Gray Box testing and example

- ▶ How to test for vulnerabilities:
- ▶ Result Expected:

...



References

- ▶ Whitepapers
- ▶ Tools

Example



Black Box vs. Gray Box

Black Box

- ✓ The penetration tester does not have any information about the structure of the application, its components and internals

Gray Box

- ✓ The penetration tester has partial information about the application internals. E.g.: platform vendor, sessionID generation algorithm

White box testing, defined as complete knowledge of the application internals, is beyond the scope of the Testing Guide and is covered by the OWASP Code Review Project



Testing Model

- ➊ The testing model consists of:
 - ▶ Tester: Who performs the testing activities
 - ▶ Tools and methodology: The core of this Testing Guide project
 - ▶ Application: The black box target to test
- ➋ The test is divided into 2 phases
- ➌ I) Passive mode: in the passive mode, the tester tries to understand the application's logic, and plays with the application. At the end of this phase, the tester should understand all the access points (*gates*) of the application (e.g., HTTP headers, parameters, and cookies). The Information Gathering section explains how to perform a passive mode test. For example, the tester could find the following:
 - ▶ https://www.example.com/login/Authentic_Form.html
 - ▶ <http://www.example.com/Appx.jsp?a=1&b=1>
- ➍ In this case, the application shows the authc. form and two gates (parameters a and b). All the gates found in this phase represent a point of testing.
- ➎ A spreadsheet with the directory tree of the application and all the access points would be useful for the second phase.



Testing Model (2)

- ➊ II) Active mode: in this phase, the tester begins to test using the methodology described in the follow paragraphs.
- ➋ We have split the set of active tests in 9 sub-categories for a total of 66 controls:
 - ▶ Configuration Management Testing
 - ▶ Authentication Testing
 - ▶ Session Management Testing
 - ▶ Authorization testing
 - ▶ Business Logic Testing
 - ▶ Data Validation Testing
 - ▶ Denial of Service Testing
 - ▶ Web Services Testing
 - ▶ Ajax Testing



The Testing Categories: methodology and PoC

Information Gathering

- ➊ The first phase in security assessment is of course focused on collecting all the information about a target application.
- ➋ Using public tools it is possible to force the application to leak information by sending messages that reveal the versions and technologies used by the application
- ➌ Available techniques include:
 - ▶ Testing: Spiders, robots, and Crawlers (OWASP-IG-001)
 - ▶ Search engine discovery/Reconnaissance (OWASP-IG-002)
 - ▶ Identify application entry points (OWASP-IG-003)
 - ▶ Web Application Fingerprint (OWASP-IG-004)
 - ▶ Application Discovery (OWASP-IG-005)
 - ▶ Analysis of Error Codes (OWASP-IG-006)



Identify application entry points

- ➊ The purpose of this gathering phase is to identify and enumerate all the entry points of the application in order to gain a comprehensive view of the surface to be tested.
- ➋ Focus to the all GET and HTTP POST and all the parameters sent to the application (including Hidden fields in HTML content). Better to use a visual tool that shows the directory tree of published applications and its parameters.
- ➌ Using a proxy WebScarab as you can see all requests and view the headers and parameters sent and received, and put the results into a spreadsheet (the parameters of interest, type of request (POST / GET), authenticated access or not, if SSL is used and any relevant information).



Identify application entry points (2)

□ HTTP GET:

```
GET https://www.my-bank.com/private/Transfer.jsp?ID=123&Q=1000&To=124
Host: www.my-bank.com
Cookie: SESSIONID=cvcxvZCBGlzIGZvbyBhbmQgcGFzc3dvcmQgaXMVd
```

ID, Q, To, SessionID are relevant to this request

□ HTTP POST:

```
POST https://www.my-bank.com
Host: www.my-bank.com.com
Cookie:
SESSIONID=bggbbgbfgCBhcHAgdGhhCBzZXrIHBzYWRpY3RhYmxlIGNvb2tpZXMcYW5kIG1pbmUgaXMgMTIzNA==
CustomCookie=00my00trusted00ip00is00x.x.x.x00

user=Matteo&password=secret!&IP=true
```

user,password,IP, Cookie, CustomCookie are relevant to this request



Configuration Management Testing

Configuration Management Testing

- SSL/TLS Testing (OWASP-CM-001)
- DB Listener Testing (OWASP-CM-002)
- Infrastructure Configuration Management Testing (OWASP-CM-003)
- Application Configuration Management Testing (OWASP-CM-004)
- Testing for File Extensions Handling (OWASP-CM-005)
- Old, Backup and Unreferenced Files (OWASP-CM-006)
- Infrastructure and Application Admin Interfaces (OWASP-CM-007)
- Testing for HTTP Methods and XST (OWASP-CM-008)



Testing for file extensions handling

- ➊ The following file extensions should NEVER be returned by a web server, since they are related to files which may contain sensitive information.
 - ▶ .asa
 - ▶ .inc
- ➋ Therefore, files with these extensions must be checked to verify that they are indeed supposed to be served (and are not leftovers), and that they do not contain sensitive information.
 - ▶ .zip, .tar, .gz, .tgz, .rar, ...: (Compressed) archive files
 - ▶ .java: No reason to provide access to Java source files
 - ▶ .txt: Text files
 - ▶ .pdf: PDF documents
 - ▶ .doc, .rtf, .xls, .ppt, ...: Office documents
 - ▶ .bak, .old and other extensions indicative of backup files (for example: ~ for Emacs backup files)
- ➌ Nessus and Nikto tools



Authentication testing

Testing the authentication scheme means understanding how the application checks for users' identity and using that information to circumvent that mechanism and access the application without having the proper credentials

Tests include the following areas:

- Credentials transport over an encrypted channel (OWASP-AT-001)
- Testing for user enumeration (OWASP-AT-002)
- Default or guessable (dictionary) user account (OWASP-AT-003)
- Testing For Brute Force (OWASP-AT-004)
- Testing for Bypassing authentication schema (OWASP-AT-005)
- Testing for Vulnerable remember password and pwd reset (OWASP-AT-006)
- Testing for Logout and Browser Cache Management (OWASP-AT-007)
- Testing for Captcha (OWASP-AT-008)
- Testing for Multiple factors Authentication (OWASP-AT-009)
- Testing for Race Conditions (OWASP-AT-010)



Session management testing

Session management is a critical part of a security test, as every application has to deal with the fact that HTTP is by its nature a stateless protocol. Session Management broadly covers all controls on a user from authentication to leaving the application

Tests include the following areas:

- Testing for session management scheme (OWASP-SM-001)
- Testing for cookie attributes (OWASP-SM-002)
- Session Fixation (OWASP-SM-003)
- Exposed session variables (OWASP-SM-004)
- Cross Site Request Forgery (OWASP-SM-005)



Cookie collections

1st Authentication:

User = Mario Rossi; password=12aB45cD:

Cookie=TWFyaW8123

2nd Authentication :

User = Mario Rossi; password=12aB45cD:

Cookie=TWFyaW8125

3rd Authentication :

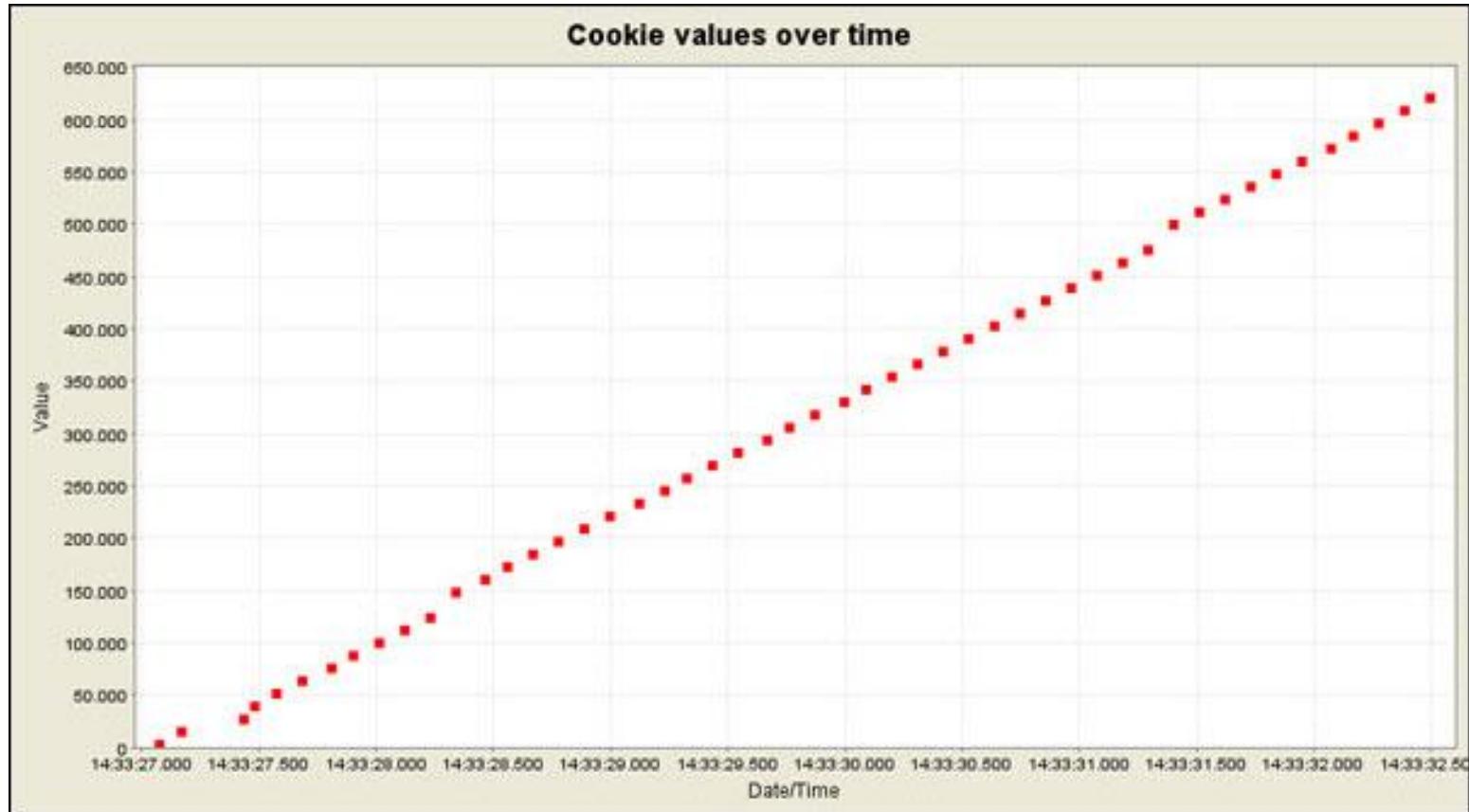
User = Mario Rossi; password=12aB45cD:

Cookie=TWFyaW8127

Cookie Guessable: Cookie=TWFyaW8129



Cookie collection: WS SessionID Analysis



Authorization Testing

Authorization is the concept of allowing access to resources only to those permitted to use them. Testing for Authorization means understanding how the authorization process works, and using that information to circumvent the authorization mechanism.

Tests include the following areas:

- Testing for path traversal (OWASP-AZ-001)
- Testing for bypassing authorization schema (OWASP-AZ-002)
- Testing for Privilege Escalation (OWASP-AZ-003)



Business logic testing

Business logic may include:

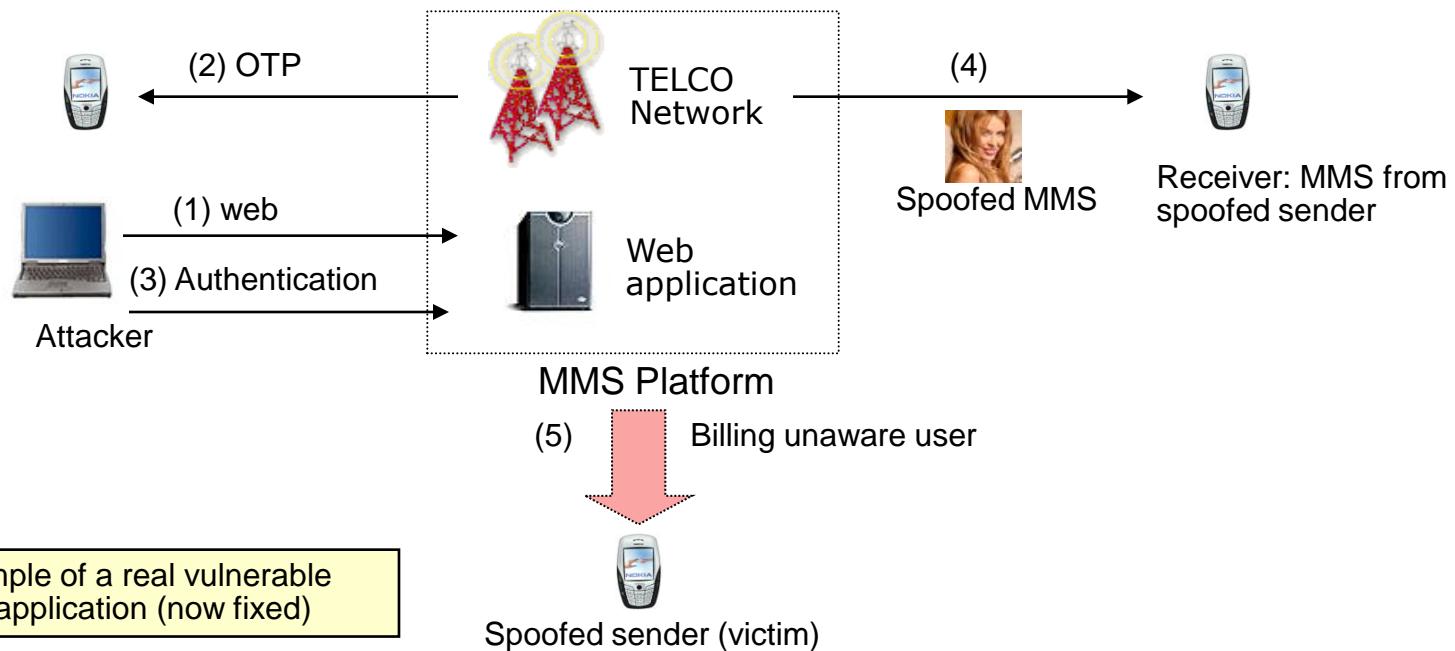
- ⌚ Business rules that express business policy (such as channels, location, logistics, prices, and products); and
- ⌚ Workflows based on the ordered tasks of passing documents or data from one participant (a person or a software system) to another.

This step is the most difficult to perform with automated tools, as it requires the penetration tester to perfectly understand the business logic that is (or should be) implemented by the application

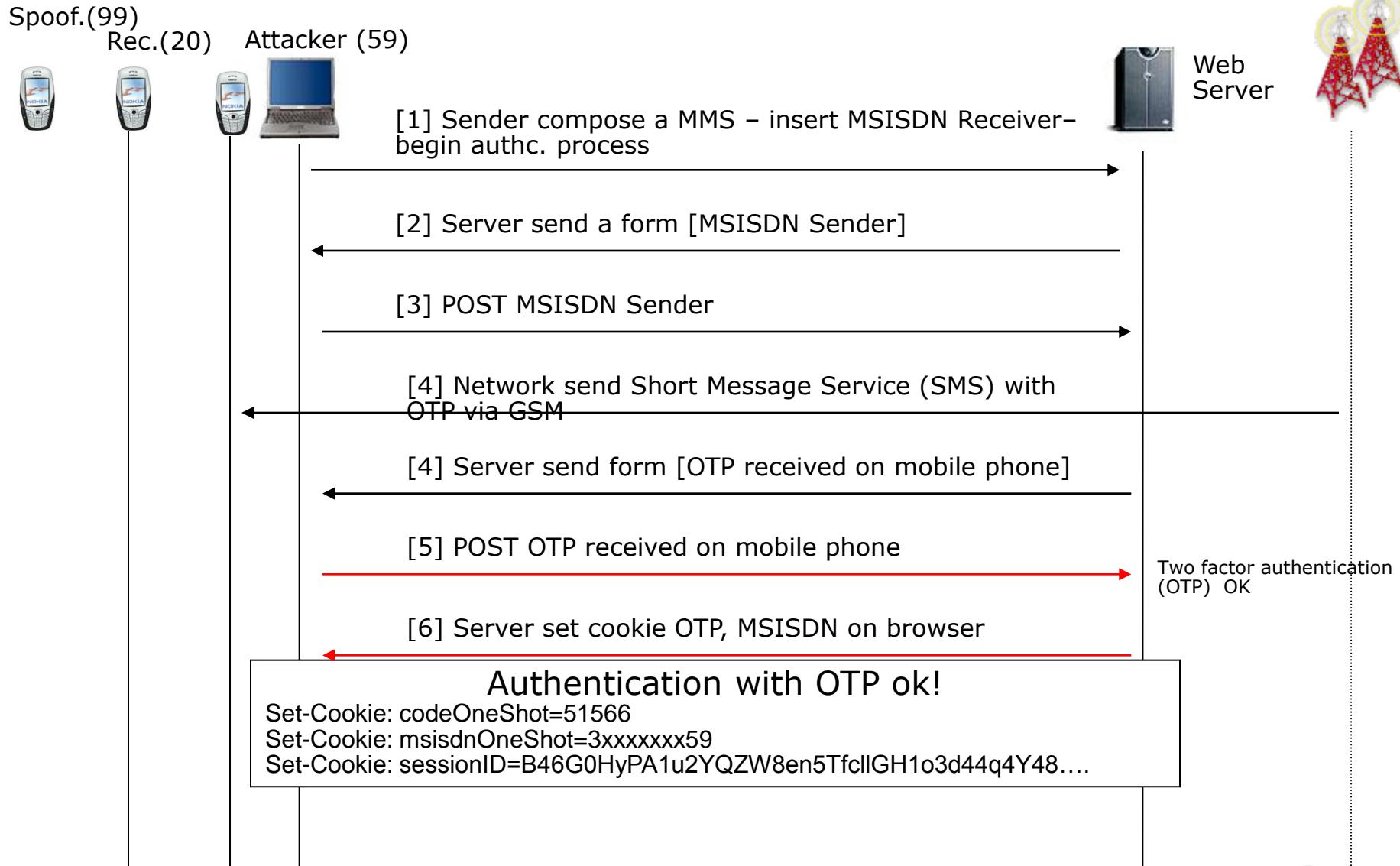


Example: Bypass business logic

Web Application to create MMS.
Two factor Authentication using cellular phone.



[6] Authentication and Set Cookie



[7] Hacking the billing

Modifying the cookie...

Request to http://

forward drop text hex

GET http:// Jp.
url=Q2Fzbz03JkFkdj0wJkRlc3Q9IDM5MzI4MzAxOTU1OSZTaXpIPTE0MzU
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/
application/vnd.ms-excel, application/msword, application/x-shockwave-
Accept-Language: it
Cookie: codeOneShot=51566; ms_sdnOneShot=3; [REDACTED] 159;
sessionID=A2ASVpbNirh79Vt0u2gwwChq1aXuffqJC941TRFsQoqCLm
8001!7002!1082015782428; [REDACTED]; IOLADVACT=ACP0-00-0-00; IOLADVPREF=WCP0000; IOLADVLCT=CLP0
JSESSIONID=A2ASVpbNirh79Vt0u2gwwChq1aXuffqJC941TRFsQoqCL
8001!7002
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 4.0)
Host: [REDACTED]
Proxy-Connection: Keep-Alive
Proxy-Authorization: Basic [REDACTED]

Request to http://

forward drop text hex

GET http:// /p_jnsCodeOneShot.jsp?
url=Q2Fzbz03JkFkdj0wJkRlc3Q9IDM5MzI4MzAxOTU1OSZTaXpIPTE0MzUw HTTP/1.0
Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, application/x-shockwave-flash, */*
Accept-Language: it
Cookie: codeOneShot=51566; ms_sdnOneShot=3; [REDACTED] 199;
sessionID=A2ASVpbNirh79Vt0u2gwwChq1axumquJC941TRFsQoqCLmF1DfVI-855668859I-1062677649I
8001!7002!1082015782428; [REDACTED]; IOLADVID=B155250382;
IOLADVACT=ACP0-00-0-00; IOLADVPREF=WCP0000; IOLADVLCT=CLP0000;
JSESSIONID=A2ASVpbNirh79Vt0u2gwwChq1aXuffqJC941TRFsQoqCLmF1DfVI-855668859I-1062677649I
8001!7002
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 4.0)
Host: [REDACTED]
Proxy-Connection: Keep-Alive
Proxy-Authorization: Basic [REDACTED]

[7] Call the servlet to bill the user

Charge Sender
3xxxxxxxx99 !!



Sn0of (99)

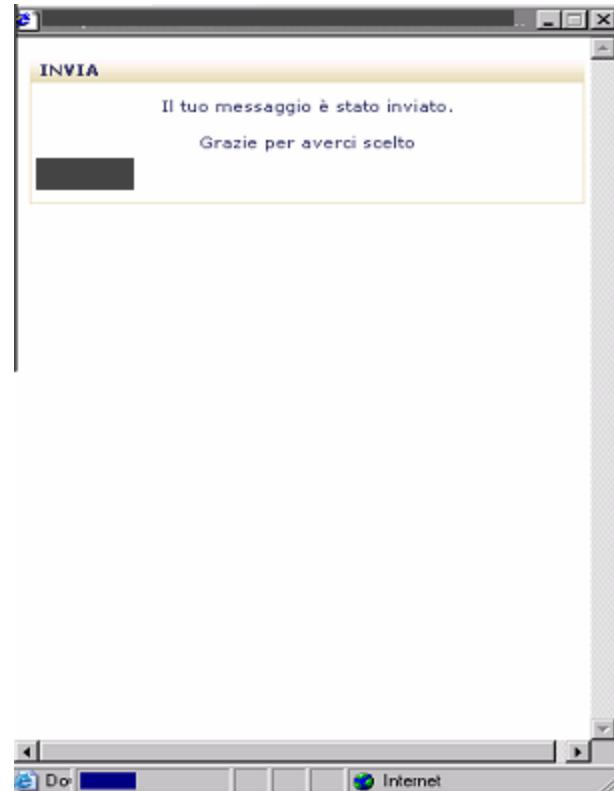
[8] Sent MMS ok



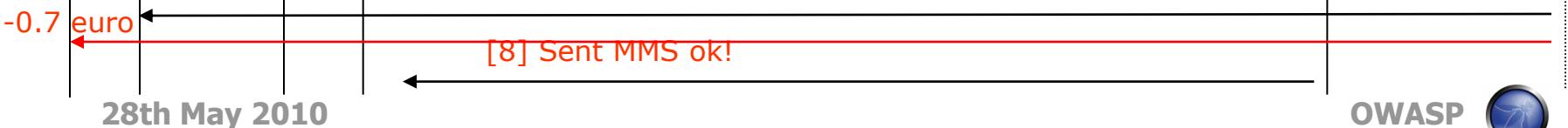
Mobile phone receiver
(3xxxxxxxx20)



Web Browser attacker
(3xxxxxxxx59)



Mobile phone spoofed user
(3xxxxxxxx99)



OWASP



Data validation testing

In this phase we test that all input is properly sanitized before being processed by the application, in order to avoid several classes of attacks

Cross site scripting (Reflected, Stored, DOM, Flashing)

Test that the application filters JavaScript code that might be executed by the victim in order to steal his/her cookie

SQL Injection

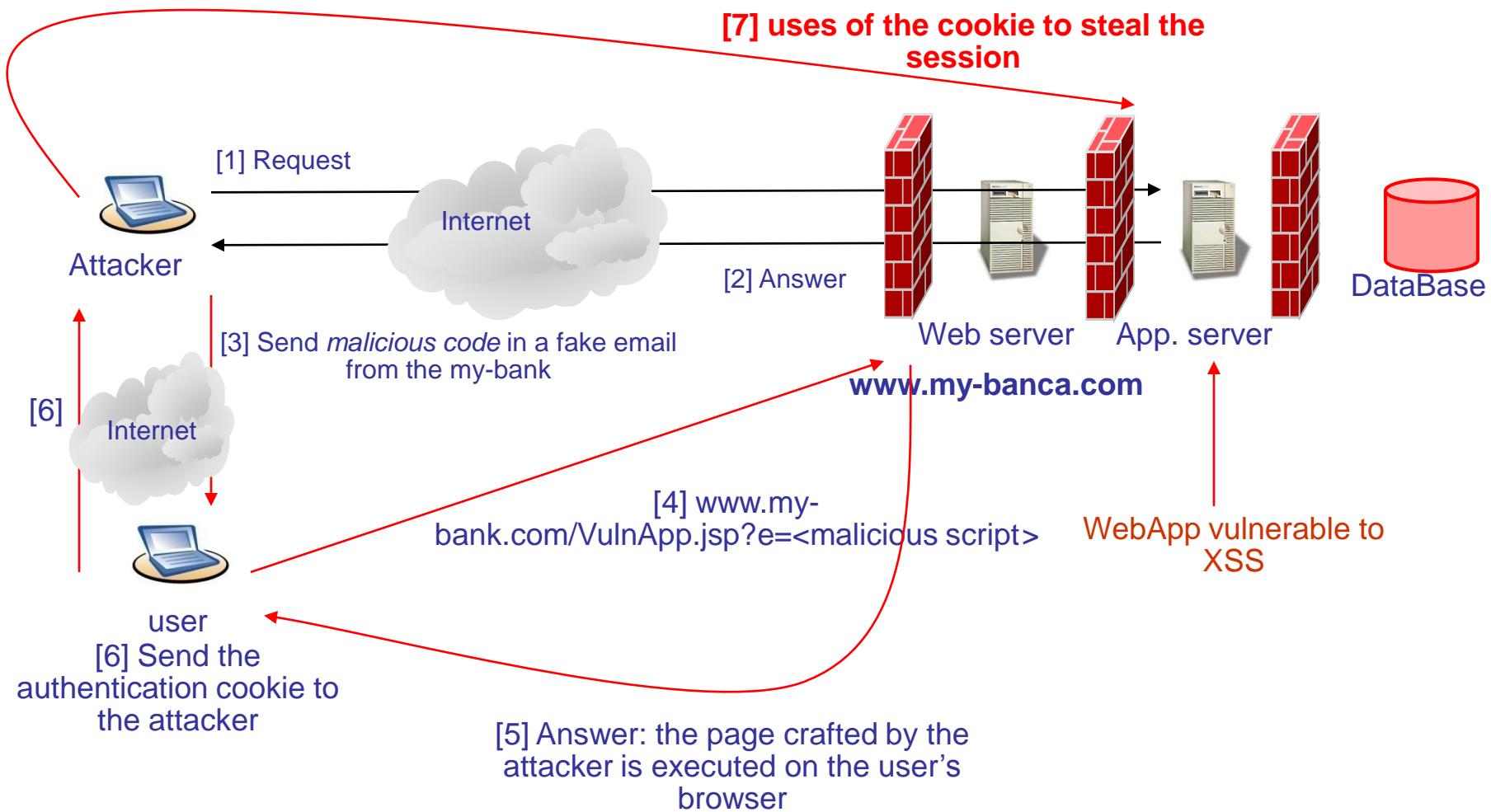
Test that the application properly filters SQL code embedded in the user input

Other attacks based of faulty input validation...

- ▶ LDAP/XML/SMTP/Command injection
- ▶ Buffer overflows



Cross-Site-Scripting (XSS)



<script>alert('XSS')</script>?

• <, ', ", /, (,

The screenshot shows the CAL9000 web application interface. The main window title is "CAL9000" and the sub-page title is "HTTP REQUESTS". The URL in the browser is "file:///CAL9000/CAL9000.html#httpRequests".

The "HTTP REQUESTS" panel contains the following fields:

- Method: GET
- Auth: None
- User: [empty]
- Password: [empty]
- Target URL: http://www.cnn.com/
- Placeholder: [empty]
- Quick Encode: Field: [empty], Encode Type: [empty]
- Add Request Header: Connection: keep-alive
- Add Browser Headers: Firefox
- Add Method Headers: Method: [empty]
- Add Request Parameter: Name: [empty], Value: [empty], Add Parameter: [empty], Request Body Actions: [empty]

The status bar at the bottom shows "Done".

• <http://ha.ckers.org/xss.html>



XSS: what risks?

- ➊ Steal users' cookies and run a session-hijacking (identity theft)
- ➋ Run unintended functionality of the website
- ➌ Attack the end user with malicious code
- ➍ Run a defacement of the site
- ➎ Ability to perform a perfect phishing
- ➏ Forcing the user to perform actions such:
 - ▶ Dispositive action
 - ▶ Scan the internal network

Injection Flaws

- ➊ Injection means ...
 - ▶ Ensure that application includes data additional to those provided in direct flows to an interpreter
- ➋ Interpreters ...
 - ▶ Accept strings as input data and interpret them as commands
 - ▶ SQL, Shell OS, LDAP, XPath, etc ...
- ➌ SQL injection is the most common problem
- ➍ Many applications use SQL and are vulnerable

Web Services Testing

- The vulnerabilities are similar to other “classical” vulnerabilities such as SQL injection, information disclosure and leakage etc but web services also have unique XML/parser related vulnerabilities.
- WebScarab (available for free at www.owasp.org) provides a plug-in specifically targeted to Web Services. It can be used to craft SOAP messages that contains malicious elements in order to test how the remote system validates input

The screenshot shows the WebScarab application window. The menu bar includes File, View, Tools, and Help. The toolbar has tabs for Extensions, SessionID Analysis, Scripted, Fragments, Fuzzer, Compare, Search, Summary, Messages, Proxy, Manual Request, WebServices (which is selected), and Spider. Below the toolbar, the WSDL URL is set to `http://soap.amazon.com:schemas2/AmazonWebServices.wsdl`. The Service is set to `AmazonSearchService` and the Operation is set to `KeywordSearchRequest`. In the main pane, there is a table with columns Node, Type, Nullable, and Value. A row is selected with the value `<userid>myuser</userid> <password>' OR 1=1</password>`.

Node	Type	Nullable	Value
KeywordSearchRequest	KeywordRequest		<userid>myuser</userid> <password>' OR 1=1</password>



Web Services Testing



XML Structural Testing

In this example, we see a snippet of XML code that violates the hierarchical structure of this language. A Web Service must be able to handle this kind of exceptions in a secure way

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note id="666">
<to>OWASP
<from>EOIN</from>
<heading>I am Malformed </to>
</heading>
<body>Example of XML Structural Test</body>
</note>
```



Web Services Testing (cont.)



XML Large payload

Another possible attack consists in sending to a Web Service a very large payload in an XML message. Such a message might deplete the resource of a DOM parser

```
<Envelope>
<Header>
  <wsse:Security>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>...
    <Signature>...</Signature>
  </wsse:Security>
</Header>
<Body>
  <BuyCopy><ISBN>0098666891726</ISBN></BuyCopy>
</Body></Envelope>
```



Web Services Testing (cont.)

Naughty SOAP attachments

Binary files, including executables and document types that can contain malware, can be posted using a web service in several ways

```
POST /Service/Service.asmx HTTP/1.1
Host: somehost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://somehost/service/UploadFile
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<UploadFile xmlns="http://somehost/service">
<filename>eicar.pdf</filename>
<type>pdf</type>
<chunk>x50!P%@AP[4\pzx54(p^)7CC)7}SEICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*</chunk>
<first>true</first>
</UploadFile>
</soap:Body>
</soap:Envelope>
```



OWASP Testing Guide v2: Authors

* Vicente Aguilera
* Mauro Bregolin
* Tom Brennan
* Gary Burns
* Luca Carettoni
* Dan Cornell
* Mark Curney
* Daniel Cuthbert
* Sebastien Deleersnyder
* Stephen DeVries
* Stefano Di Paola

* David Endler
* Giorgio Fedon
* Javier Fernández-Sanguino
* Glyn Geoghegan
* Stan Guzik
* Madhura Halasgikar
* Eoin Keary
* David Litchfield
* Andrea Lombardini
* Ralph M. Los
* Claudio Merloni

* Matteo Meucci
* Marco Morana
* Laura Nunez
* Gunter Ollmann
* Antonio Parata
* Yiannis Pavlosoglou
* Carlo Pelliccioni
* Harinath Pudipeddi
* Alberto Revelli
* Mark Roxberry
* Tom Ryan

* Anush Shetty
* Larry Shields
* Dafydd Studdard
* Andrew van der Stock
* Ariel Waissbein
* Jeff Williams



OWASP Testing Guide v3: Authors

V3 Authors

- | | | |
|----------------------|--------------------------|-------------------------|
| • Anurag Agarwal | • Kevin Horvath | • Matteo Meucci |
| • Daniele Bellucci | • Gianrico Ingrosso | • Marco Morana |
| • Arian Coronel | • Roberto Suggi Liverani | • Antonio Parata |
| • Stefano Di Paola | • Alex Kuza | • Cecil Su |
| • Giorgio Fedon | • Pavol Luptak | • Harish Skanda Sureddy |
| • Adan Goodman | • Ferruh Mavituna | • Mark Roxberry |
| • Christian Heinrich | • Marco Mella | • Andrew Van der Stock |

V3 Reviewers

- | | | |
|--------------|----------------|--------------|
| • Marco Cova | • Kevin Fuller | • Nam Nguyen |
|--------------|----------------|--------------|



Questions?

matteo.meucci@owasp.org

Thanks!!