# Deep Dive In The Cross Site Request Forgery (CSRF) Vulnerability

**Cincinnati Chapter Meeting**
**May 27th , 2008**
**Marco.Morana@gmail.org**

**OWASP**

# Agenda

1. CSRF: Uncovering the Silent Beast
2. Issue, Threats, Root Causes
3. Learning CSRF
   - Webgoat Lesson Demo
4. Testing for CSRF
   - OWASP CSRF Tester
5. Countermeasures
   - Best Practices: Client and Application
   - Tool: OWASP CSRF Guard
6. Final Questions

# Notable CSRF Flaws Examples

- **Netflix CSRF flaw** enabled an attacker to add a movie to the victim's movie rental queue, change the name and address, the password and cancel the account
- **Gmail CSRF** flaw enabled an attacker to show all your GMail contacts to someone else
- **Routers CSRF flaws** have being shown to allow someone else changing the configuration, hijack the entire Internet traffic (**Does the device at 192.168.1.1 on *your* network have a username/password of admin/admin?)**

# Place of CSRF in the OWASP Top 10

1. Cross Site Scripting (XSS)
2. Injection Flaws
3. Insecure Remote File Include
4. Insecure Direct Object Reference
5. Cross Site Request Forgery (CSRF) (CWE-352)
6. Information Leakage and Improper Error Handling
7. Broken Authentication and Session Management
8. Insecure Cryptographic Storage
9. Insecure Communications
10. Failure to Restrict URL Access

http://www.owasp.org/index.php/Top_10

# Cross Site Request Forgery Defined

What is a Cross Site Request Forgery anyway?

(Also known as CSRF, XSRF, Sea-Surf, One-click attack, cross site reference forgery, session riding, etc...)

- Not to be confused with Cross Site Scripting!
- Involves two key components to be successful:
  - A willing victim
  - A vulnerable website
- Exploits a users privileges and trusts to a particular website.

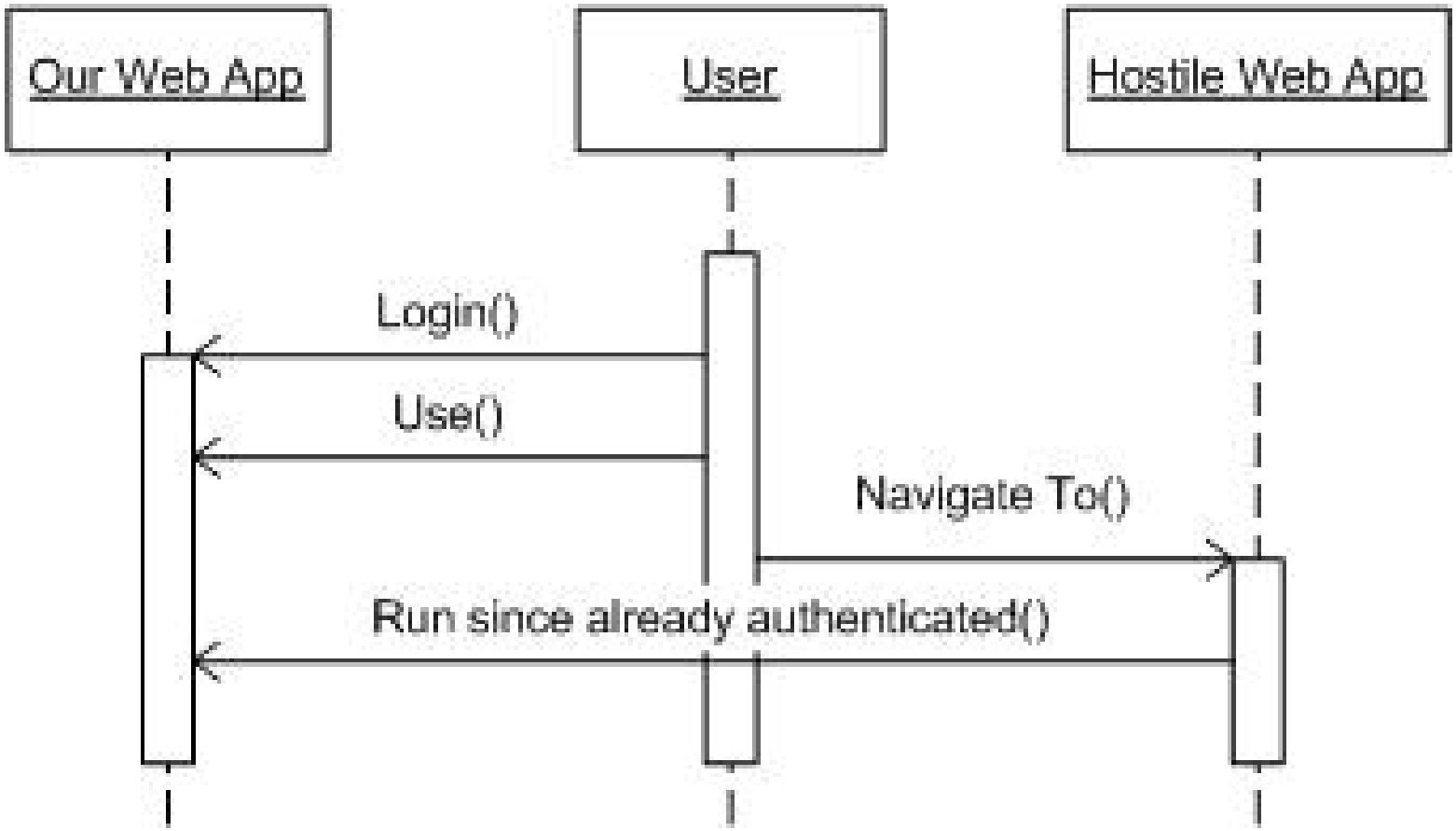
# CSRF Defined

- **A CSRF attack exploits the trust** that applications have on authenticated sessions.
- **A browser logged into an application vulnerable to CSRF can be forced to send a request** to perform and action on behalf of the victim (i.e. bank transfer, logout, shopping, disclose information)
- **Any web application without a build in CSRF control is vulnerable**

# CSRF Root Causes

- The way CSRF is accomplished relies on the following facts:
  1) **Web browser behavior** regarding the handling of session-related information such as cookies and http authentication information;
  2) **Knowledge of valid web application URLs** on the side of the attacker;
  3) **Application lack of re-authentication for business actions, password management**,
  4) **Existence of HTML tags to access an http[s] resource**; for example the image tag *img*.

# CSRF Exploit: Sequence Diagram



# Analyzing the CSRF Attack

How can this attack be successful?

1. The target site uses persistent cookies
2. SessionIDs are not changed after authentication
3. The user has an **auto login** or "**remember me**" invoked
4. The target site **allows static POST or GET requests**
5. **Target site does not re-authenticate** for specific requests (e.g. POST of confidential information)
6. There is **no session time out**
7. Cookies are exposed because of **XSS vulnerabilities**

# CSRF: Client-Application Root Cause

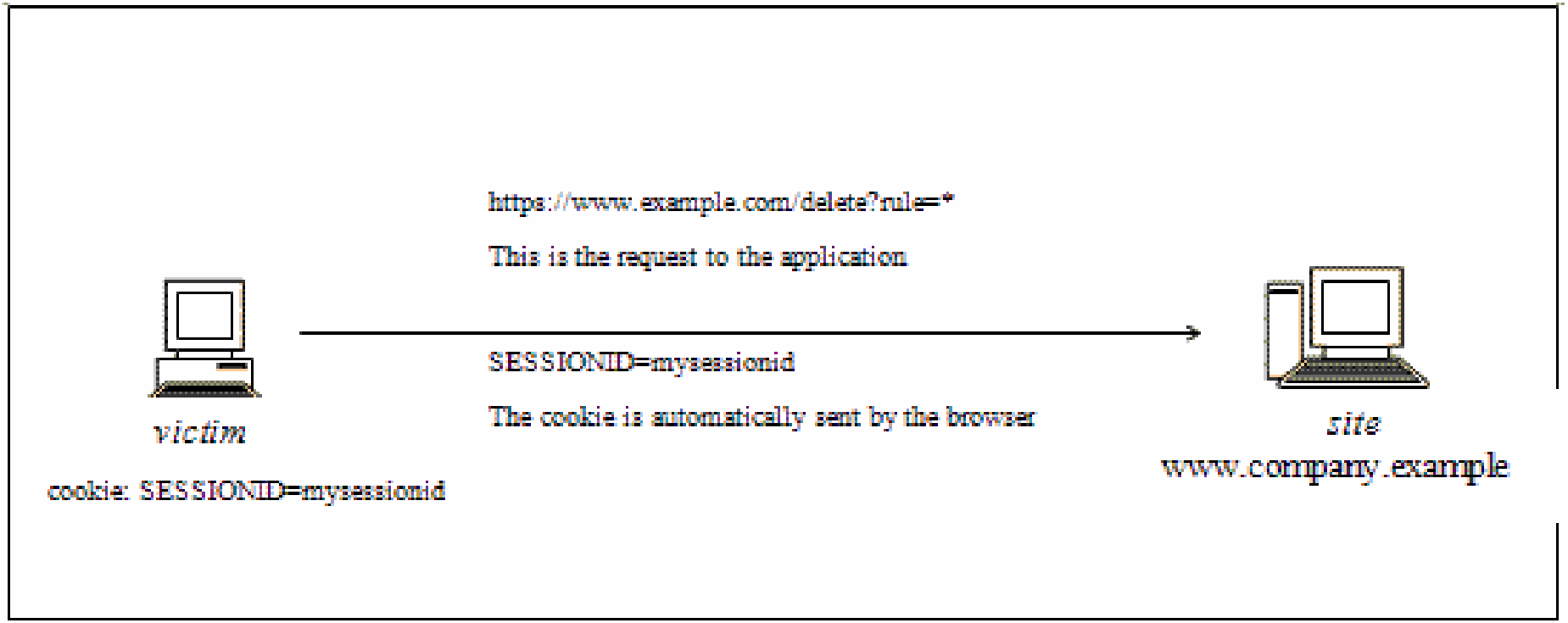
**These GET invocations are all the same for the application**:

1. Using the actual web application;
2. Typing the URL it directly in the browser;
3. Selecting a link (external to the application such as email, posting on a blog) pointing to the URL.

**…but there is a problem:**

**The browser automatically sends the cookie (e.g the session id cookie) that was set for site along with any further request:**

# CSRF Root Cause: Automatic cookie re-send

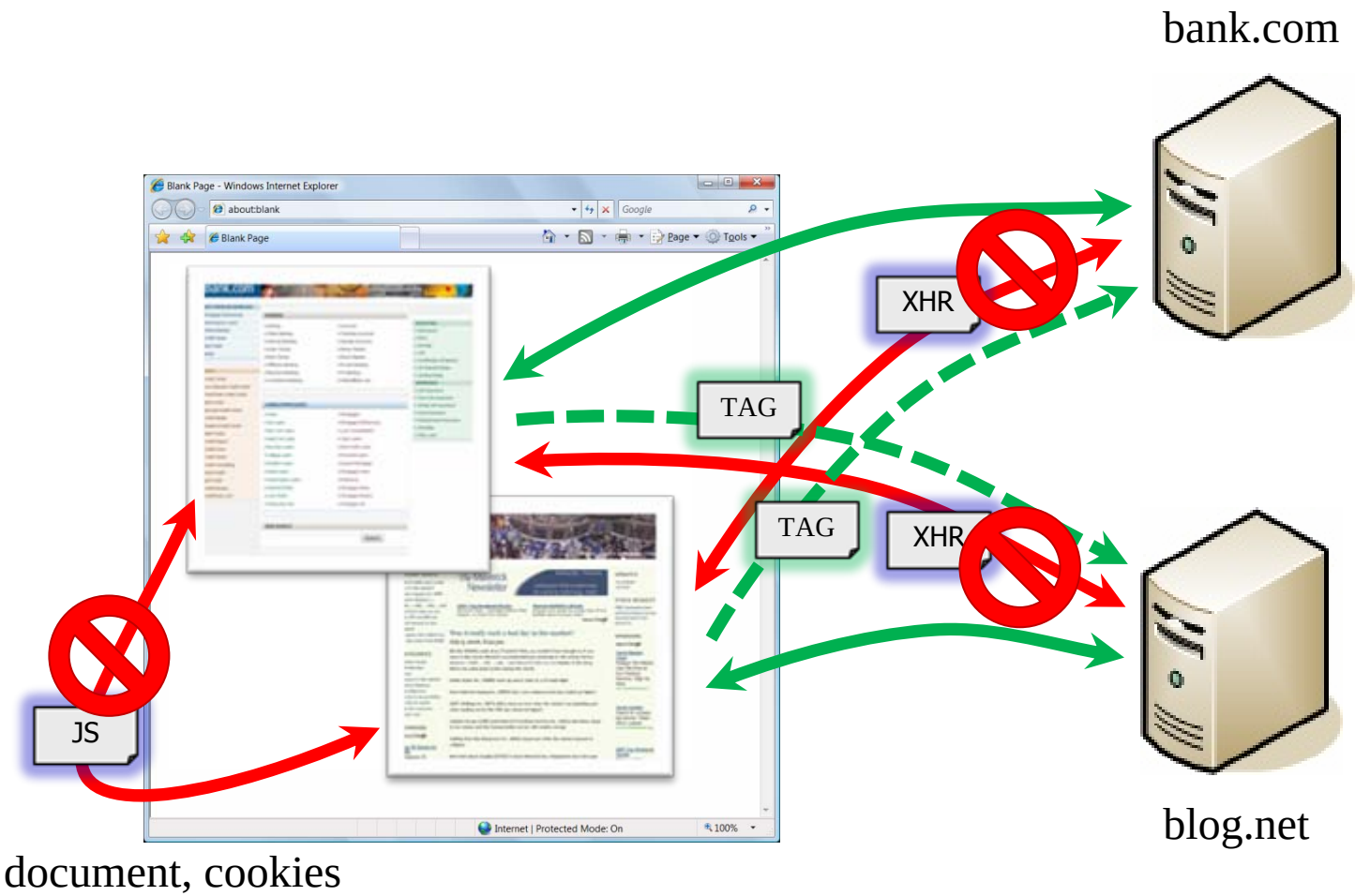


# CSRF:Bypassing Same-Origin Policy

- **Browser control prevents a document or script loaded from one site of origin from manipulating properties with a document loaded from another site of origin**
- But, applies only to:
  - Manipulating browser windows
  - URLs requested via the XmlHttpRequest
  - Manipulating frames (including inline frames)
  - manipulating documents (included using the OBJ tag)
  - Manipulating cookies

  **But, it does not applies to HTML TAGS!**

# The Browser “Same Origin” Policy



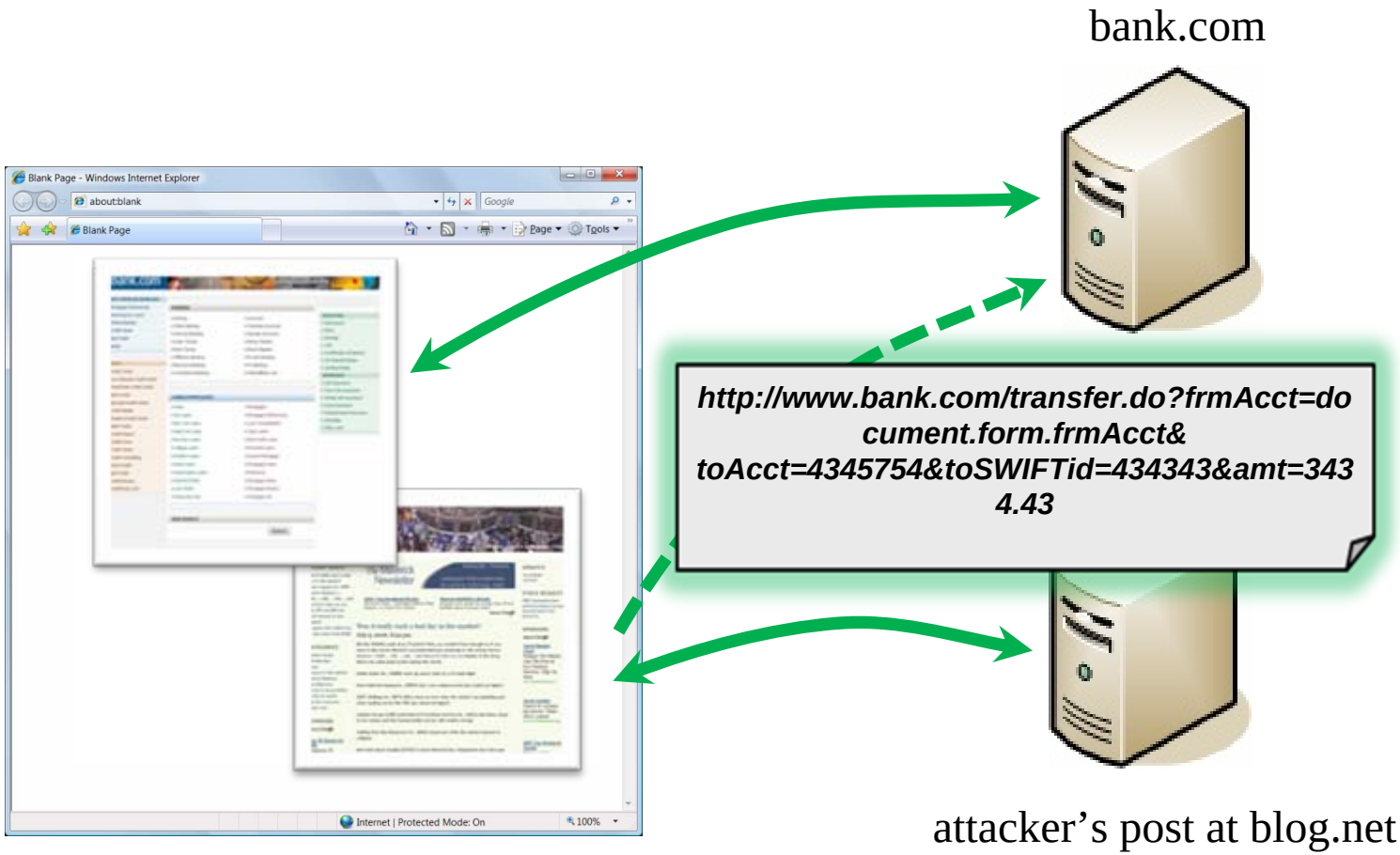
# CSRF In-secure software root cause

- Embedded links in forms with HTML tags
- Business transactions are not re-authenticated

***Check your bank account transfer form and try to forge it putting this link:***

```
<img
src="http://www.bank.com/transfer.do?frmAcct=document.form.frmAcct&
toAcct=4345754&toSWIFTid=434343&amt=3434.43">
```

# CSRF Attack Via Malicious Blog Post



# CSRF: Crafting The Attack

1) Auto-POST discovery

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

2) Attacker tries same as a GET request

```
http://bank.com/transfer.do?acct=BOB&a
mount=10000 HTTP/1.1
```

3) Attacker tries the attack vector in a HTML doc

```
<ahref="http://bank.com/transfer.do?a
cct=MARIA&amount=100000">View my
Pictures!</a>
```

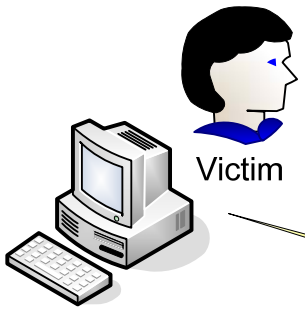
# CSRF: Crafting The Attack (cont)

4) Finally the attacker sends a phishing email to the victim with the attack vector:

```
<img
src="http://bank.com/transfer.do?acct
=MARIA&amount=100000" width="1"
height="1" border="0">
```

NOTE The attack vector will include the link in a zero-bit image that displays no information to the user (just an error of non rendered image)

# CSRF Attack Via Malicious Email





**Malicious Email**

To: Joe.Victim@user.com
From: Spoofed@victimsbank.com
Subject: ABC bank updates

New updates to ABC bank! Click below to see detailed changes to our system.
<img src="http://victimsbank.com/transfer?account=victim&amount=10000destination=attackers_account"></img>

Sincerely,
ABC Bank Staff

**Malicious Request**

http://victimsbank.com/transfer?account=victim&amount=10000&destination=attackers_bank_account

# How to learn about CSRF

**Download Webgoat** from **www.OWASP.org**

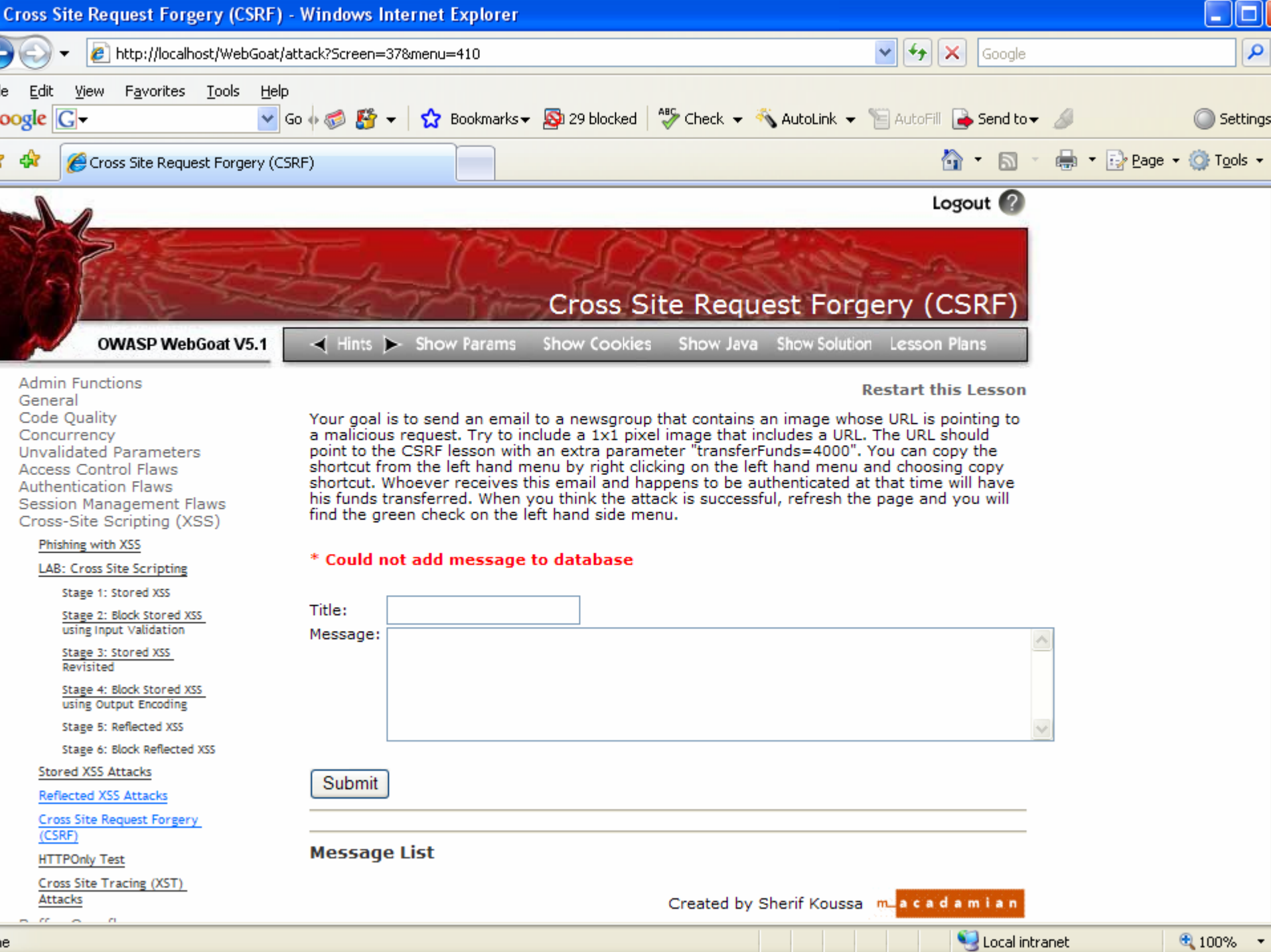
**Unzip the WebGoat-OWASP_Standard-x.x.zip** to your working environment

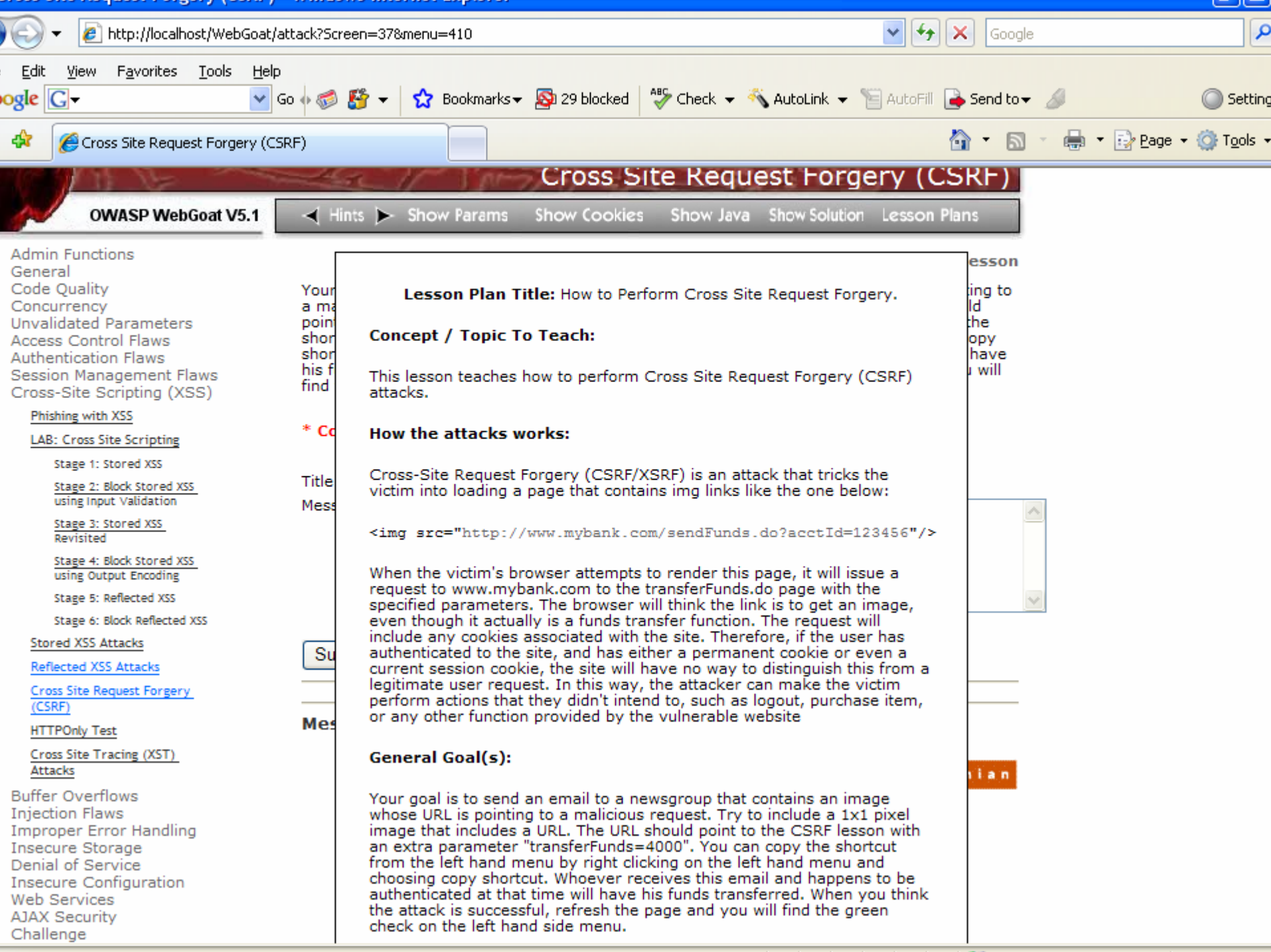
**Start Tomcat**, browse to the WebGoat directory unzipped above and double click "webgoat.bat"

**Browse to http://localhost/WebGoat/attack**

**Login** in as: user = guest, password = guess
CSRF Lesson

From XSS lessons **select: CSRF**


Cross Site Request Forgery (CSRF) - Windows Internet Explorer
http://localhost/WebGoat/attack?Screen=37&menu=410
Google
File Edit View Favorites Tools Help
Go
Bookmarks
29 blocked
Check
AutoLink
AutoFill
Send to
Settings
Cross Site Request Forgery (CSRF)
Page
Tools
Logout
Cross Site Request Forgery (CSRF)
OWASP WebGoat V5.1
Hints
Show Params
Show Cookies
Show Java
Show Solution
Lesson Plans
Admin Functions
General
Code Quality
Concurrency
Unvalidated Parameters
Access Control Flaws
Authentication Flaws
Session Management Flaws
Cross-Site Scripting (XSS)
Phishing with XSS
LAB: Cross Site Scripting
Stage 1: Stored XSS
Stage 2: Block Stored XSS using Input Validation
Stage 3: Stored XSS Revisited
Stage 4: Block Stored XSS using Output Encoding
Stage 5: Reflected XSS
Stage 6: Block Reflected XSS
Stored XSS Attacks
Reflected XSS Attacks
Cross Site Request Forgery (CSRF)
HTTPOnly Test
Cross Site Tracing (XST) Attacks
Restart this Lesson
Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.
* Could not add message to database
Title:
Message:
Submit
Message List
Created by Sherif Koussa
macadamian
Local intranet
100%


http://localhost/WebGoat/attack?Screen=37&menu=410
Google
Edit View Favorites Tools Help
Go Bookmarks 29 blocked Check AutoLink AutoFill Send to Setting
Cross Site Request Forgery (CSRF)
Page Tools
Cross Site Request Forgery (CSRF)
OWASP WebGoat V5.1
Hints Show Params Show Cookies Show Java Show Solution Lesson Plans
Admin Functions
General
Code Quality
Concurrency
Unvalidated Parameters
Access Control Flaws
Authentication Flaws
Session Management Flaws
Cross-Site Scripting (XSS)
Phishing with XSS
LAB: Cross Site Scripting
Stage 1: Stored XSS
Stage 2: Block Stored XSS using Input Validation
Stage 3: Stored XSS Revisited
Stage 4: Block Stored XSS using Output Encoding
Stage 5: Reflected XSS
Stage 6: Block Reflected XSS
Stored XSS Attacks
Reflected XSS Attacks
Cross Site Request Forgery (CSRF)
HTTPOnly Test
Cross Site Tracing (XST) Attacks
Buffer Overflows
Injection Flaws
Improper Error Handling
Insecure Storage
Denial of Service
Insecure Configuration
Web Services
AJAX Security
Challenge
Lesson Plan Title: How to Perform Cross Site Request Forgery.
Concept / Topic To Teach:
This lesson teaches how to perform Cross Site Request Forgery (CSRF) attacks.
How the attacks works:
Cross-Site Request Forgery (CSRF/XSRF) is an attack that tricks the victim into loading a page that contains img links like the one below:
<img src="http://www.mybank.com/sendFunds.do?acctId=123456"/>
When the victim's browser attempts to render this page, it will issue a request to www.mybank.com to the transferFunds.do page with the specified parameters. The browser will think the link is to get an image, even though it actually is a funds transfer function. The request will include any cookies associated with the site. Therefore, if the user has authenticated to the site, and has either a permanent cookie or even a current session cookie, the site will have no way to distinguish this from a legitimate user request. In this way, the attacker can make the victim perform actions that they didn't intend to, such as logout, purchase item, or any other function provided by the vulnerable website
General Goal(s):
Your goal is to send an email to a newsgroup that contains an image whose URL is pointing to a malicious request. Try to include a 1x1 pixel image that includes a URL. The URL should point to the CSRF lesson with an extra parameter "transferFunds=4000". You can copy the shortcut from the left hand menu by right clicking on the left hand menu and choosing copy shortcut. Whoever receives this email and happens to be authenticated at that time will have his funds transferred. When you think the attack is successful, refresh the page and you will find the green check on the left hand side menu.

# CSRF Web Goat Lesson Demo

- file:///C:/WEBGOAT/Lessons/WebGoat_XSS_XSRF/WebGoat_XSS_XSRF.html

# How You Can Find If You Are Vulnerable

- **Check source code for forms that authorize requests** on automatic credentials (session cookies, remember me functionality, SSO tokens)
  - Auto-Posting forms
  - ***<img>, <iFrame> and <script> tags that submit confidential data, perform non re-authenticated transactions***
  - XMLHTTPRequests
- Some automated scanners (Appscan) can detect CSRF today
- Record and replay transactions, manually check for attack vectors
  - ***<img src="http://www.example.com/logout.php">***

# CSRF Black Box testing and example

1. **Assume URL being tested is** http://www.example.com/action
2. **Build a html page** containing the http request referencing the URL embedded in an image tag
3. **Log into the application**;
4. **Assume social engineering attack**, craft an email with a reference to the URL (it can be HTML email with the URL embedded in an tag)
5. **Select the link and observe the result**
6. **Check if the web server executed the request**.

# CSRF Gray Box testing

- **Audit the application** to ascertain if its session management is vulnerable
  - Session management rely on client-side values (e.g. cookies)
  - Basic Authentication (credentials sent in each request)
- Does the application rely on HTTP POST requests?
  - Do POST requests use new sessionIDs?
- Does the application uses ViewState (.NET)?

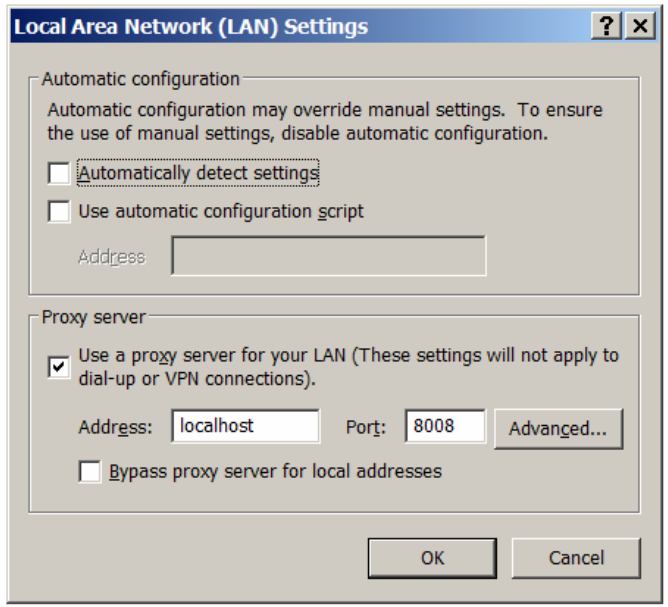
# New Tool: OWASP CSRFTester

- Test your applications for CSRF
  - Record and replay transactions
  - Tune the recorded test case
  - Run test case with exported HTML document
- Test case alternatives
  - Auto-Posting Forms
  - Evil iFrame
  - IMG Tag
  - XMLHTTPRequest
  - Link

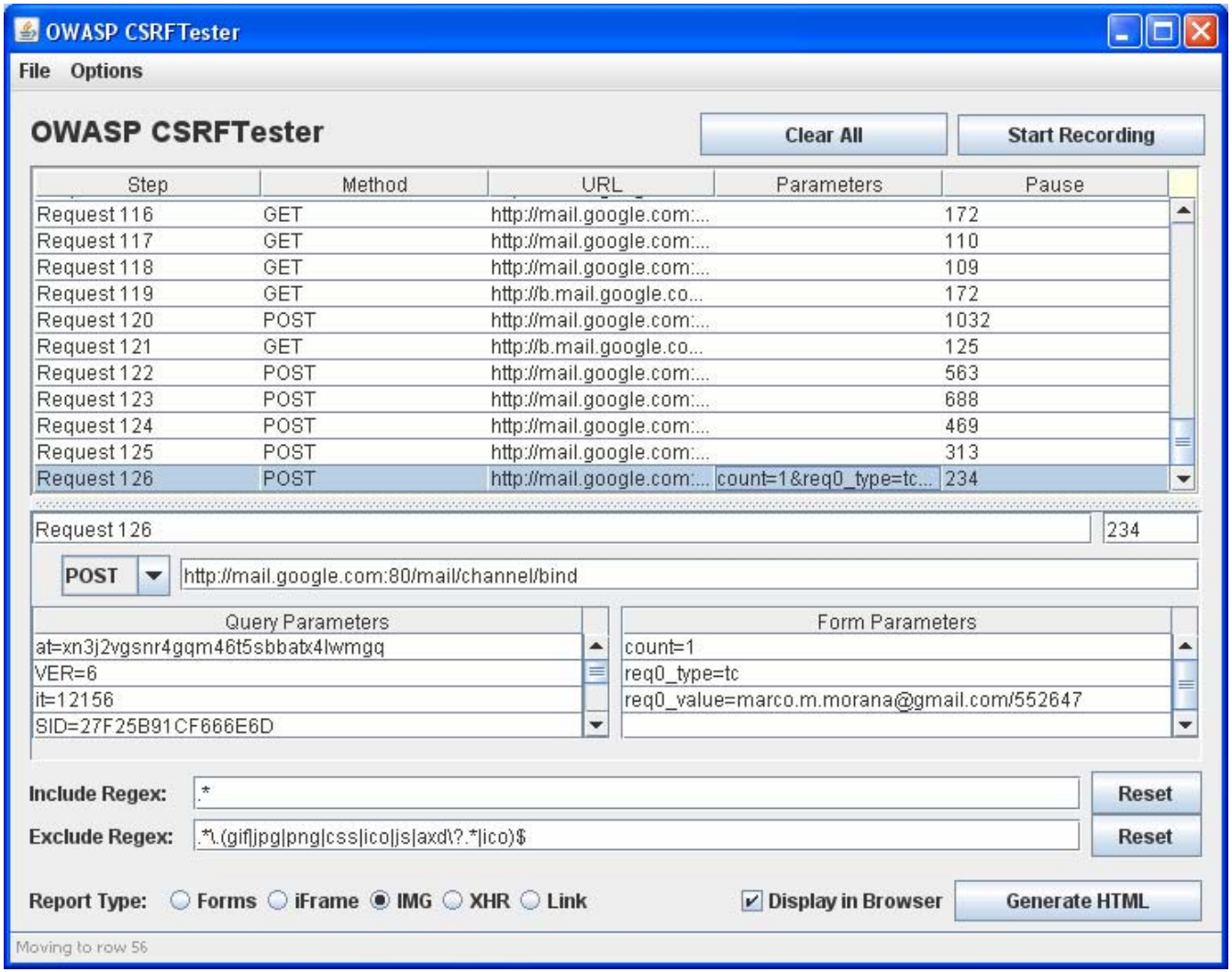
# OWASP CSRF Tester: Installation\Configuration

1) Run the batch file:
C:\CSRFTester\CSRFTester-1.0\run.bat

2) Configure browser to proxy through CSRFTester



# OWASP CSRF Tester: Recording



# OWASP CSRF Tester: Creating the Test Cases

- The report type determines how we want the victims browser to submit the previously recorded requests
  - **Forms**: using auto-posting forms
  - **iFrame**: using auto-submitting iframe tag.
  - **IMG**: using the <img src="..."/> tag
  - **XHR**: using XMLHttpRequest. (Note that this is subject to the same origin policy_
  - **Link**: when the user clicks a link

# OWASP CSRF Tester: Running the Test

1. Open a new browser instance
2. Authenticate with access to the same business function (URL)
3. Have that user/browser launch the newly created HTML report file

**If the action was carried out after viewing the file in the same browser window that was used to authenticate then the tested URL is vulnerable to CSRF.**

# CSRF Mitigation Best Practices For the User

Some mitigating actions are:

1. **Logoff** immediately after using a web application
2. **Do not save username/passwords**, no "remember me" your login
3. **Do not use the same browser to access sensitive applications and to surf freely the Internet**; if you have to do both things at the same machine, do them with separate browsers.
4. **Using web mail pose additional risks** since simply viewing a mail message might lead to the execution of an attack.

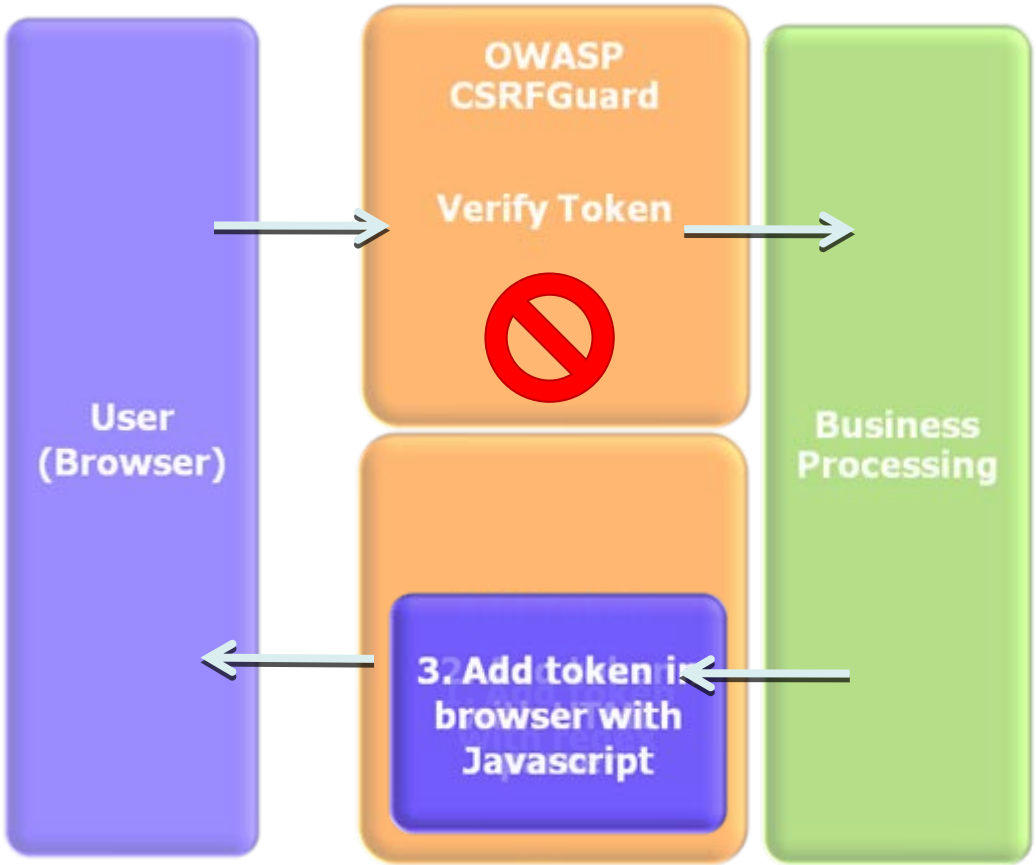
# CSRF Mitigation Best Practices For the App

- **Insert custom random tokens** into every form and URL
  - `<form action="/transfer.do" method="post"> <input type="hidden" name="8438927730" value="43847384383"> … </form>`
- Make sure there a **no XSS vulnerabilities**
- **Re-authenticate** and **perform out of band** verification when performing high risk transactions
- **Do not use GET requests for sensitive data** or to perform high risk transactions
- For ASP.NET **set ViewStateUserKey** (similar check as random token)

# Misconceptions – Defenses That Don't Work

- Only accept POST
  - Stops simple link-based attacks (IMG, frames, etc.)
  - But hidden POST requests can be created with frames, scripts, etc…
- Referer checking
  - Some users prohibit referers, so you can't just require referer headers
  - Techniques to selectively create HTTP request without referers exist

# Countermeasures: OWASP CSRFGuard 2.0




- Adds token to:
  - href attribute
  - src attribute
  - hidden field in all forms
- Actions:
  - Log
  - Invalidate
  - Redirect

http://www.owasp.org/index.php/CSRFGuard

# How CSRF Guard works

1. **Server creates a random session key token** and passed to the browser
2. **The token is appended in each URL request** using javascript on the browser
3. **Server check if the URL request has to be validated** by CSRF (this is configurable a priori)
4. **If the request needs checking, then Session compares the passed CSRFGuard Session Token to the one stored** in Server session.
5. **If they do not match**, or if the token is not present, then we've got a CSRF attempt and **denies the request**.

# Similar Implementations

- PHP CSRFGuard
  - PHP Implementation of CSRFGuard
  - http://www.owasp.org/index.php/PHP_CSRF_Guard
- .NET CSRFGuard

  CSRF can actually be prevented in .Net already -- but you have to be using ViewState, CSRF .Net otherwise:
  - http://www.owasp.org/index.php/.Net_CSRF_Guard
- JSCK (Javascript Cross Site Protection Kit)
  - PHP & JavaScript implementation
  - http://www.thespanner.co.uk/2007/10/19/jsck/

# CSRF Myths and Reality

- Myth: CSRF is a special case of XSS
  - **Fact**: Different vulnerability, root causes and countermeasures. XSS can faciliate CSRF
- Myth: POSTs are not vulnerable to CSRF
  - **Fact**: It is more difficult to exploit but they can lead to automatic submission
- Myth:CSRF is low risk vulnerability
  - **Fact**: Can perform any un-autorized transaction such as change passwords, force logouts, transfer money, disclose information

# Enterprise CSRF Mitigation Strategy

- Balance Between Security, Usability, and Cost

# Q&A