IIG University of Freiburg

# Web Security, Summer Term 2012

## Cross Site Request Forgery - CSRF

Dr. E. Benoist

Sommer Semester

## Table of Contents

## Cross Site Request Forgery

- Not a new attack, but simple and devastating
- CSRF attack forces a logged-on victim's browser to send a request to a vulnerable web application
- Target: Perform the chosen action on behalf of the victim

## Sending a request without the consent of the victim?

- **Insert an image in a HTML file**

  <img src="http://www.benoist.ch/image/test.gif">

  Browser: Downloads an image

  GET /image/test.gif HTTP/1.1
  ...

- **An image can be generated by a PHP program (or any program)**

  <img src="http://www.benoist.ch/image/test.php">

  Browser: Downloads an image

## Sending a request? (Cont.)

- **An image can be generated according to some parameters**

  <img src="/barcode.php?number=12345678901">

  Browser: Downloads an image
- **An image tag can contain something else**

  <img src="http://www.benoist.ch/index.php?action=logout">

  Browser ?????

## Very widespread vulnerability

**Touches any web application that**

- **has no authorization checks for vulnerable actions**
- **will process an action if a default login is able to be given in the request**

```
<img src=
"http://www.benoist.ch/doSomething?user=admin&pwd=admin">
```

- **Authorizes requests based only on credentials that are automatically submitted**
  - cookies if currently logged into the application
  - or “Remember me” functionality if not logged into the application
  - or a Kerberos token if part of an Intranet participating in integrated logon with Active Directory.

## State of the art

- **Most of web applications rely solely on automatically submitted credentials**
  - cookies
  - basic authentication credentials
  - source IP addresses
  - SSL certificates
  - or windows domain credentials
- **Vulnerability also known as**
  - Session Riding, One-Click Attacks, Cross Site Reference Forgery, Hostile Linking, and Automation Attack
  - Acronym XSRF is also used together with CSRF

## Vulnerability

- **A typical CSRF attack directs the user to invoke some function**
  - for instance application's logout page
- **The following tag can be inserted in any page viewed by the victim**

  <img src="http://www.benoist.ch/logout.php">

  it generates the same request as clicking on a link containing this address!
- **Example: Online banking transfer**

  <img src="http://www.mybank.de/transfer.do?
  fromAccount=document.form.frmAcct&
  toAccount=4567890&amount=3434.43">

  Could transfer the money from the account of the user, to a given account.

## Vulnerability (Cont.)

- **Jeremiah Grossman[1] Succeeded in making changes in victims DSL routers**
- **Even if the user doesn't know that he can configure his router ;-)**
- **He used the router's default account name to perform his attack** Example

  <img src="http://admin:password@192.168.1.1/">

  Then you just have to reconfigure the system

  <img src="http://192.168.1.1/changeDNS?newDNS=143.23.45.1"

- **Once DNS changed, user will never be able to access a site securely!**

[1]Talk in Blackhat 2006 : "Hacking Intranet Sites from the outside"

## How such a link could reach a victim

- Web-site Owner embedded JavaScript malware
- Web page defaced with embedded JavaScript malware
- JavaScript Malware injected into a public area of a website. (persistent XSS)
- Clicked on, a specially-crafted link causing the website to echo JavaScript malware. (non-persistent XSS)

## CSRF allows to access the intranet

- **The attacker sends requests from inside the Intranet**
  - Doesn't have to go throw the firewall, the victim is already
- **CSRF combined with javascript allows to send many requests sequentially**
  - javascript adds an image in the DOM (possibly invisible).
  - when the request is sent, another image is added
  - and so on

## Internet is protected, but Intranet?

**You would never dare doing this on Internet! But what about Intranet?**

- **Leaving hosts unpatched**
  - Servers are always patched regularly, but local PC's?
- **Using default passwords**
  - What the use of changing the password, the IP address can only be reached from inside my network
  - 192.168.x.y
- **Do not putting a firewall in front of a host**
- **Everything seams OK because the perimeter firewalls black external access**
  - So CSRF attacks can be very fruitful

## Why do CSRF attacks work?

- **User authorization credential is automatically included in any request by the browse**
  - Typical: Session Cookie
- **The Attacker doesn't need to supply that credential**
  - It belongs to the victim's browser
- **Success of CSRF belongs on the probability that the victim is logged in the attacked system**
  - Idea: attack the site the victim visits
  - Mean : XSS

## CSRF and XSS

- **Combine CSRF and XSS**
  - The tag is already posted inside the vulnerable application
- **Risk is increased by that combination**
  - Probability to find a logged in user is higher
- **CSRF does not require XSS**
  - One can attack a site from another one
- **Any application with XSS flaws is susceptible to CSRF**
  - CSRF attacks can exploit the XSS flaw to steal any non-automatically submitted credential
- **When building defenses against CSRF attacks, you must eliminate XSS vulnerabilities**

## Example of attack[2]

- **Javascript Port Scaning**
  - `<script src="http://192.168.1.100/"></script>`

  If a web server is listening: HTML will be returned, causing a JS interpreter error:
  Solution: Capture the Error!
  (`<script ... onerror="myfunction()">`)
- **Javascript can loop on all the possible IP addresses for nearby hosts:**
  - Scanning of the hosts

[2] *Hacking Intranet Sites from the outside*, Jeremiah Grossman

## Example of attack (Cont.)

- **Recognize the server?**
  - Use a URL that is unique for each server
  - Apache Web Server: `/icons/apache_pb.gif`
  - HP Printer: `/hp/device/hp_invent_logo.gif`
  - PHP Image Easter eggs: `/?=PHPE9568F36-D428-11d2-A769-00AA001ACF42`
  - Cycle through unique URL's using Image DOM objects

    ```
    <img src="http://192.168.1.100/unique_image_url"
              onerror="fingerprint()" />
    ```

  - If onerror event DOES NOT execute, then it's the associated platform!

## Example of attack (Cont.)

- **Login**
  - If already authenticated: nothing to do, access is automatic
  - if not: `http://admin:password@192.168.1.1/`
- **Execute command**
  - Suppose we have the following POST form

```
<form action="changePwd.php" method="POST">
   new password <input type="password" name="newpwd">
   retype new password <input type="password" name="newpwd2">
   <input type="submit" value="send">
</form>
```

- **We can forge a URL (GET) faking this POST form**

```
<img src="changePwd.php?newpwd=Toto&newpwd2=Toto">
```

- **It is also possible to generate a POST using JavaScript**

## Possible targets in a LAN (Intranet)

- **Web enabled devices:**
  - Printer, webcam, Phone over IP, WLan access points, switch, ADSL router, etc.
- **Attack on router**
  - Change config password
  - Update DNS
  - => Man in the middle attack

## Protection

- **Application must ensure that they are not only relying on credentials or tokens that are automatically submitted by browsers**
  - Session Cookies
  - Certificates
  - Remember me
  - ...
- **Application should use a custom token that the browser will not “Remember”**
  - So it can not be included in the Requests sent automatically

## Strategies

- **Ensure that there are no XSS vulnerabilities in your application**
  - Otherwise, any protection is useless, since javascript could access the hidden data.
- **Insert custom random tokens into every form and URL**
  - It will not be automatically submitted by the browser
  - Example:

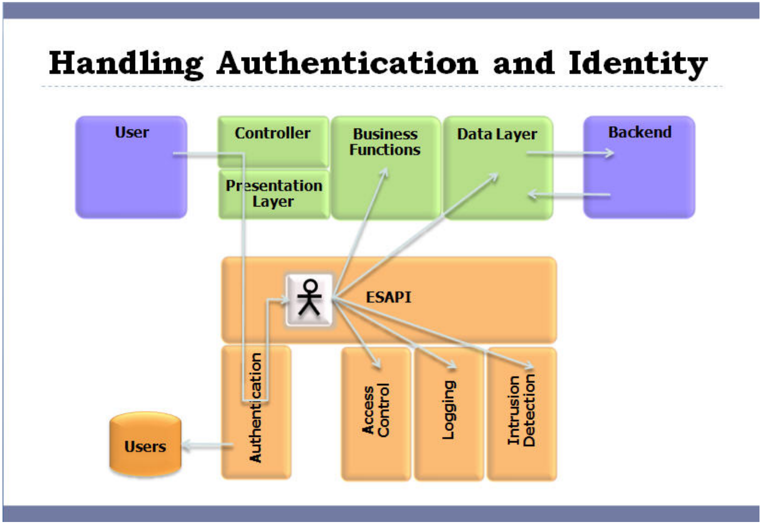
```
<form action="/transfer.do" method="POST">
  <input type="hidden" name="383838" value="1234323433">
  ...
</form>
```

  - Then you have to verify that token
  - Token can be unique for a session or even for each page
  - The more focused the token is, the higher the security is, but the application is then much more complicated to write

## Strategies (Cont.)

- **For sensitive data or value transactions, re-authenticate or use transaction signing**
  - to ensure that the request is genuine.
  - Set up external mechanism to verify requests (phone, e-mail)
  - Notify the user of the request using an e-mail
- **Do not use GET requests for sensitive data or to perform value transactions**
  - Use only POST methods when processing sensitive data from the user.
  - However the URL may contain the random token as this creates a unique URL, which makes CSRF almost impossible to perform
- **POST alone is an insufficient protection**
  - You must also combine it with random tokens

## OWASP ESAPI support Authentication and Identity



## Verifying Security

- **Goal: Verify that the application generates and requires some authorization token that is not resent automatically by the browser**
- **Automated approaches:**
  - Automated approach: few automated scanners can detect CSRF vulnerabilities.
  - Manual Approach: Penetration testing and verification of the code

## Conclusion

- **Cross Site Scripting - XSS**
  - Exploits the trust a user has in a website
  - The website sends content, that the user executes because it comes from this site.
- **Cross Site Request Forgery - CSRF**
  - Exploits the trust the site has in a user
  - by forging the enactor and making a request appear to come from a trusted user[3]
  - The server receives some requests from the user and think it was sent on prupose.

[3] wikipedia

## References

- **OWASP Top 10 - 2010**
  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
- **Hacking Intranet Sites from the outside**, *Jeremiah Grossman*, BlackHat 2006
  http://www.whitehatsec.com/presentations/whitehat_bh_pres_08032006.tar.gz
- **RSnake, “What is CSRF?”**
  http://ha.ckers.org/blog/20061030/what-is-csrf/