



build | integrate | secure

Grow with Denim Group



OWASP Top 10 2007  
A5 – Cross Site Request Forgery (CSRF)

OWASP Austin  
July 31<sup>st</sup>, 2007

## What Is It?

- Occurs when an application is susceptible to forced background requests from users
- Attackers uses a combination of a Cross Site Scripting (XSS) vulnerability along with poorly implemented server side functionality
- XSS payload causes authenticated browser to make unintended requests
- Server side code processes requests as if they came from the authenticated user

# Impact

- Attacker can manipulate the web application as if they were logged in
- Whatever the application lets the authenticated user do – they can do

# Example

- Java-based web application
- Allows user to make money transfers

# Variations

- Can be stored or reflected like XSS
- Can use GET or POST requests

# How Do You Test Applications?

- Start out by looking for XSS flaws
- Testing for many XSS flaws is often easy to automate
  - *Distinctive request/response pattern*
  - *Scanning tools are good at finding these flaws*
- Look at the impact of any XSS flaws
  - *How much HTML/JavaScript can be injected and where?*
- Look for how server-side functionality has been implemented
  - *Can your XSS payloads make requests that would exercise this functionality?*

# Crafting CSRF Payloads

- Start with XSS payload crafting
  - *You may have to deal with pre- and post- HTML*
- The most basic attacks use GET requests
  - `<img src=http://server.com/Resource?p=DoBadStuff />`
  - `<script src="" ... />`
  - `<link REL="STYLESHEET" TYPE="text/css" ... />`
- You can also use POST requests
  - *XMLHttpRequest object*
  - *Requires bigger payload*

# How Do You Guard Against?

- First, protect yourself against XSS vulnerabilities
  - *Positively validate inputs*
    - Length, type, syntax, business rules
  - *Encode application outputs*
    - HTML or XML
    - `<` becomes `&lt;` and so on
- Design server-side functionality with protection in mind
  - *Prefer POST requests to GET requests*
  - *Include random tokens with requests*
  - *Require re-authentication for sensitive transactions*

# POST versus GET

- It is comparatively easy to force a browser to make GET requests versus POST requests
  - *Recall the section on crafting payloads*
  - *Lots of injectable tags cause the browser to make GET requests*
- This alone is not enough protection
  - *But it can help reduce exposure*
  - *Defense in depth*

# Random Tokens

- Have session- or page- specific random tokens that must be passed in with requests
  - *Also called nonces*
  - *Generate using good crypto (SHA-2 family of hashes) and good randomly-generated values known only on the server side*
- Verify these on the server side
- Finer-grained tokens will be more secure, but harder to maintain

```
<form action="/servlet/MakeTransfer">  
<input type="hidden" name="1029384756"  
  value="0192837465" />
```

# Require Re-Authentication

- Make sensitive operations a multi-request process
- Require the user to re-present credentials
- Use out-of-band facilities to confirm
  - *Email, text messages*

# Java-specific Safeguards

- Standard XSS safeguards
  - Avoid using `<%= %>` because that does not encode outputs
  - Escape special HTML characters
    - `<` `>` `'` `"` `/` `&`
    - See <http://www.javapractices.com/Topic96.cjp>
  - Use `URLEncoder` class to encode characters being placed in a URL
  - Use Struts output mechanisms such as `<bean:write ...>`
  - Use JSTL `escapeXML="true"` attribute in `<c:out ...>`

# .NET-specific Safeguards

- Standard XSS safeguards
  - *.NET has built-in blacklist validation against many known XSS attacks*
    - This is good, but not ideal
    - This can be turned off with `ValidateRequest="false"` in the Page tag
  - *Validation framework offers many protection options*
    - `RegexValidator` and others
  - *Avoid using `<%= %>` because that does not encode outputs*
  - *Use `HttpUtility.HtmlEncode()` to encode user-supplied data that is reflected back to users*
  - *Microsoft Anti-XSS Library*
    - <http://www.microsoft.com/downloads/details.aspx?familyid=9a2b9c92-7ad9-496c-9a89-af08de2e5982&displaylang=en>
    - <http://tinyurl.com/fwc7u>

# .NET Specific Safeguards

- Set a ViewStateUserKey
  - *Provides similar protections to using a random token*
  - [http://msdn2.microsoft.com/en-us/library/ms972969.aspx#securitybarriers\\_topic2](http://msdn2.microsoft.com/en-us/library/ms972969.aspx#securitybarriers_topic2)
  - <http://tinyurl.com/2opg4v>

# MySpace Worm

- Based on an assembled JavaScript built up from several substrings
  - *Bypassed numerous negative filters against malicious tags, attributes, etc*
- Relied on the same functionality being served from multiple hostnames
  - *profile.myspace.com and [www.myspace.com](http://www.myspace.com)*
- Used a multi-step background combination of GETs and POSTs
  - *Get the list of friends*
  - *Parse out special POST tokens*
  - *Add samy as a new friend*
- For more info: <http://namb.la/popular/>
- Technical explanation: <http://namb.la/popular/tech.html>

# MySpace Worm: Lessons Learned

- Better input validation – positive versus negative where possible
  - *Interpretable HTML is a “big place” so carving out a couple of tags or syntax bits will not remove the ability of attackers to find workarounds*
- Be careful about what functionality is available from where
  - *GETs versus POSTs*
  - *What applications are available from a given hostname?*
  - *Similar to checking to be sure that HTTPS content is not available via HTTP*

# Recap

- Cross Site Request Forgery (CSRF) vulnerabilities allow attackers to force users' browsers to undertake actions they do not want or even know about
- Several varieties:
  - *Stored and reflected*
  - *GET and POST*
- To guard against:
  - *Eliminate XSS vulnerabilities*
  - *Use random tokens*
  - *Prefer POST to GET*

# Other Materials

- OWASP Top 10 CSRF Page
  - [http://www.owasp.org/index.php/Top\\_10\\_2007-A5](http://www.owasp.org/index.php/Top_10_2007-A5)
- WASC Abuse of Functionality Threat Classification
  - [http://www.webappsec.org/projects/threat/classes/abuse\\_of\\_functionality.shtml](http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml)
- RSnake “What is CSRF?”
  - <http://ha.ckers.org/blog/20061030/what-is-csrf/>

# Questions

Dan Cornell

[dan@denimgroup.com](mailto:dan@denimgroup.com)

Denim Group, Ltd.

(210) 582-4400

[www.denimgroup.com](http://www.denimgroup.com)

[denimgroup.typepad.com](http://denimgroup.typepad.com)