On XSRF
And Why You Should Care

PacSec 2006

Martin Johns
Me, Myself and I

- Martin Johns
- johns at informatik.uni-hamburg.de
- Security researcher at the University of Hamburg
- Member of the secologic project
  - Research project carried out by SAP, Commerzbank, Eurosec and the University of Hamburg
  - Sponsored by the German Ministry of Technology (BMWi)
  - Goal: Improving software security
  - Visit us at http://www.secologic.org
Agenda

- Web Application Authentication
- XSRF / Session Riding
- Server Side Countermeasures
- Client Side Protection
- Conclusion
Agenda

- Web Application Authentication
- XSRF / Session Riding
- Server Side Countermeasures
- Client Side Protection
- Conclusion
Explicit Authentication

The authentication credentials are communicated by the web application

- URL rewriting: Session token is included in every URL
- Form based session tokens

Immune against XSRF

(actually only almost immune)
Implicit Authentication

Automatically executed by the browser

- Cookies
- http authentication (Basic, Digest, NTLM)
- IP based schemes
- Client side SSL

Potentially vulnerable to XSRF
Session management with cookies

- After the authentication form the server sets a cookie at the client’s browser
- As long as this cookie is valid, the client’s requests are treated as authorized
http authentication (Basic, Digest)

- The client requests a restricted resource
- The server answers with a “401 Unauthorized” response
- This causes the client’s browser to demand the credentials
- The client resends the request
- The user’s credentials are included via the “Authorization” header
- Every further request to that authentication realm contains the credentials automatically
Client side SSL authentication

- The client web browser possesses a X.509 certificate that was signed by an authority that is trusted by the web application.

- Initial authentication:
  - The client has to prove his identity.
  - For this reason, the web server demands a valid signature from the client.
  - “SSL handshake”
  - Depending on the browser, the user may or may not confirm the initial handshake by entering a password (only once).

- If the handshake was successful, a SSL session is established between the client’s browser and the web server.

- As long as the SSL session is valid, all request to the web server are transmitted using the negotiated credentials.
IP based authentication

Intranet webserver

Firewall
Agenda

- Web Application Authentication
- XSRF / Session Riding
- Server Side Countermeasures
- Client Side Protection
- Conclusion
Exploits implicit authentication mechanisms

- Known since 2001
- XSRF a.k.a. CSRF a.k.a. “Session Riding” (a.k.a. “Sea Surf”)
- Unknown/underestimated attack vector (compared to XSS or SQL injection)

The Attack:

- The attacker creates a hidden http request inside the victim’s web browser
- This request is executed in the victim’s authentication context
www.bank.com

Cookie: auth_ok
GET transfer.cgi?am=10000&an=3422421

Cookie: auth_ok
Cause: The web application does not verify that state changing request were created “within” the web application

Attack methods:
- Forging GET requests:
  - Image tag with SRC attribute that points to a state changing URL
  - The URL might be obfuscated by a http redirect
- Forging POST request:
  - Attacker creates an IFRAME (or a pop-up window)
  - The frame is filled with a HTML form
  - This form is submitted via JavaScript
Reflected:
- Attacker has to manufacture a website that hosts the origin of the hidden request

Local / stored:
- Origin of the malicious http request is hosted on the attacked website
- Example: Users are allowed to post images with foreign URLs
Example 1: Breaking Applications

Vulnerable: digg.com

- digg.com’s frontpage is determined by the number of “diggs” a certain story gets
- Using XSRF a webpage was able to cause the victim’s browser to “digg” an arbitrary URL
- The demo page “diggged” itself
Example 2: Causing Financial Loss

Vulnerable: Netflix.com

- Add movies to your rental queue
- Add a movie to the top of your rental queue
- Change the name and address on your account
- Change the email address and password on your account (i.e., takeover your account)
- Cancel your account (Unconfirmed/Conjectured)
Example 3: Owning the Server

Vulnerable: Wordpress 2.02

- Wordpress’ theme editor was susceptible to XSRF
- Wordpress theme-files can be php-files
- Via XSRF an attacker could modify those files to contain arbitrary php-code
**Example 4: Exploring the Intranet**

**Vulnerable: (most) intranet webservers**

- By dynamically including external images and using timed JavaScript events, a malicious website can, e.g.:
  - Portscan the intranet
  - Fingerprint existing web servers and installed applications

→ “JavaScript Malware”

![Diagram](image-url)
General problem:

- Session Riding vulnerabilities are NOT caused by programming mistakes
- Completely correct code can be vulnerable
- The reason for Session Riding lies within http:
  - No dedicated authentication credential
  - State-changing GET requests
  - JavaScript

“Preventing Session Riding” is actually “fixing the protocol”
Agenda

- Web Application Authentication
- XSRF / Session Riding
- Server Side Countermeasures
- Client Side Protection
- Conclusion
Misconceptions

Only accepting POST requests

- Defends against local attacks
- On foreign web pages hidden POST requests can be created with frames

Referrer checking

- Some users prohibit referrers → referrerless requests have to be accepted
- Techniques to selectively create http request without referrers exist:

<table>
<thead>
<tr>
<th>Method/Browser</th>
<th>IE 5</th>
<th>IE 6*</th>
<th>IE 7**</th>
<th>FF 1.07</th>
<th>FF 1.5</th>
<th>O 8</th>
<th>S 1.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>META Refresh</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic filled frame</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pop up window (regular)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pop up window (dynamically filled)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Generating referrerless requests ("X" denotes a working method)

- Furthermore, referrers can be spoofed with Flash
Method 1: Switch to explicit authentication

URL rewriting: Session token is included in every URL
- Attention: Token leakage via proxy-logs/referrers
- Does not protect against local attacks
  - All URLs produced by the application contain the token

Form based session tokens
- Session token is communicated via “hidden” form fields
- Attention: May break the “Back” button

Combination of explicit and implicit mechanisms
- e.g. Cookies AND URL-rewriting
- Has to be supported by the used framework
- SID-leakage may still be a problem
Method 2: Manual Protection

Reflected attacks:

- Allow only POST requests to commit state changing requests
  - (as it was intended by the inventors of http)
- Use one-time form tokens (nonces)
  - This assures that the POST request’s origin was an HTML form that was provided by the web application in the first place

Example:

```html
<form action="submit.cgi" method="POST">
  <input type="text" name="foo">
  <input type="hidden" name="nonce"
        value="xulkjsf22enbsc">
</form>
```
Method 2: Manual Protection (II)

Local attacks:

- Mirror all foreign content
- Don’t allow arbitrary URLs
- Only serve images from your own servers
- Careful: Do not allow attackers to store arbitrary data on your computers
Method 3: Automatic protection

NoForge [1]
- Reverse Proxy
- Positioned between web application and internet
- Parses http responses and adds tokens to all internal URLs
- Drops requests, that do not contain a token
- Only protects applications that use cookies for session tracking

Agenda

- Web Application Authentication
- XSRF / Session Riding
- Server Side Countermeasures
- Client Side Protection
- Conclusion
RequestRodeo: Concept

- Client Side Proxy or Browser Extension (“RequestRodeo”)
- Identification of potential fraudulent requests
- Removal of implicit authentication

“fixing the browser”
Identification of suspicious requests

- The origin determines the state
- **Definition:** entitled

An http request is classified as **entitled** only if:
- It was initiated because of the interaction with a web page (i.e. clicking on a link, submitting a form or through JavaScript) and
- the URLs of the originating page and the requested page satisfy the “same origin policy”

- Only entitled requests are permitted to carry implicit authentication information

A request’s state can be established by the browser
Proxy Solution: Processing http responses

Adding tokens to URLs

- HTML code in http responses is processed
- Identification of elements that potentially causes http requests
- The target URLs of these elements are enhanced with a URL token
- The tokens are kept together with the repose’s URL in a table

→ This way the proxy is able determine a request’s origin

“a reliable referrer”

- Every http request is checked for a URL token
- If such a token exists, the originating URL is retrieved
- If the domains of the request and its origin do not match, implicit authentication information gets removed
Removal of implicit authentication

- **Cookies:**
  - Remove all “Cookie”-fields from unentitled requests
  - A cookie’s domain value has to be respected

- **http authentication:**
  - Browser Extension: Triggers a new authentication dialogue for unentitled requests
  - Proxy: Adds token to the URL before removing the “authentication header”

- **Client Side SSL:**
  - Display warning web site before sending the request
  - Non-obstrusive: hidden and image-tag-based attacks are not noticeable by the user
IP-based authentication

- Unentitled requests are only allowed if their target is worldreachable
- For this reason a “reflection sever” is employed
- The reflection server is hosted in a DMZ (i.e. on the other side of the cooperate inner firewall)
- Before allowing an unentitled request, the proxy verifies that the reflection server is able to access the request’s target
- Caching of checked IPs to minimize the performance penalty

→ Such a reflection server also protects against “JavaScript-Malware”
Removal of implicit authentication: IP-based auth.
Removal of implicit authentication: IP-based auth.
Implementation

Proxy [1]:
- Implemented in Python
- Using the “Twisted” framework
- Will be released on www.secologic.org

Browser Extension:
- Extension for Firefox
- Under development

Discussion

Conservative Approach:
- Connections are allowed
- Only removal of implicit authentication
- Request to public resources are uninhibited
- With small modifications also usable at the server side

Limitations:
- No protection against “local” attacks
- Proxy solution:
  - NTLM / Client Side SSL not implemented

Future work
- Addition of anti XSS techniques
Agenda

- Web Application Authentication
- XSRF / Session Riding
- Server Side Countermeasures
- Client Side Protection
- Conclusion
Conclusion

XSRF is problematic!

Correct looking code can be vulnerable!

Programmers:
- Use nonces whenever possible

Users:
- Logout!
The end

Thank you for your attention

Questions?
Comments?