Outline

- Background
- Exploits & Impact
- Practical Experiments Outcome
- Solution: Session Fixation Protection Proxy
- Conclusion & Future Work
### Background

- **Session Fixation known for several years** (at the latest from 2002)
- **Little attention compared to XSS, SQLi, CSRF**
  - Little awareness in developers’ world
- **Session Management not provided for HTTP (stateless)**
  - Fallback procedure: session tracking by identifier (ID)
    - Cookie, URL parameter, hidden form field
    - Carry ID with every request
- **Session Management + Authorization Management**
  - Mismatch of responsibilities: framework vs developer
  - Session management done by programming framework/application server
  - User authentication/authorization is application’s duty
Session Hijacking Reloaded – Session Fixation

**Attack sketch**
- Attacker sets victim’s session ID instead of session ID theft
- Victim authenticates using attacker provided session ID
- Attacker resumes authenticated session making use of known session ID

**Session Fixation starts before user authentication**

**Attack vectors: two attack stages**
- 2\(^{nd}\) stage: Session’s authentication level is raised for the provided “fixed” session ID
- 1\(^{st}\) stage: needs other vulnerability to set SID
  - XSS, meta tags, cross-protocol attack, sub domain cookie bakery, http response splitting, http header injection
Session Fixation in a nutshell

1: GET login.jsp

2: SID=1337

3: www.example.com/login.jsp?SID=1337

4: GET login.jsp?SID=1337

5: POST username, password

6: GET intern.jsp?SID=1337

www.example.com
Impact & Discussion

■ First stage attack preconditions
  ▸ Mislead victim into clicking on a link
  ▸ Set cookie via other vulnerability
  ▸ Make the victim log into his account and meet that time frame

■ Session Fixation preconditions
  ▸ Application is vulnerable
  ▸ If session is bound to IP or browser: additional obstacle
  ▸ Individual session ID needed for every victim

■ But: if all conditions are met Session Fixation is severe attack
  ▸ Full impersonation of victim mostly without any notice
1st case study: open source CMS

Default configuration vulnerable to Session Fixation?

- If yes, we “only” need first stage attack

SID before Auth?

- No: Not vulnerable
- Yes: Reissues SID after Auth?

- Yes: Not vulnerable
- No: Accepts SIDs in URLs?

- No: Vulnerable to attacker controlled Cookies
- Yes: Vulnerable
## 1st case study: open source CMS

<table>
<thead>
<tr>
<th>Application</th>
<th>Version</th>
<th>Cookie</th>
<th>URL</th>
<th>SID</th>
<th>Lang</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joomla</td>
<td>1.5</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>PHP</td>
</tr>
<tr>
<td>CMSmadesimple</td>
<td>1.6.6</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>PHP</td>
</tr>
<tr>
<td>PHPFusion</td>
<td>7.00.06</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>PHP</td>
</tr>
<tr>
<td>Redmine</td>
<td>0.9.2</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>PHP</td>
</tr>
<tr>
<td>XWiki</td>
<td>2.0.2.24648</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>Java</td>
</tr>
<tr>
<td>JAMWiki</td>
<td>0.9</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>Java</td>
</tr>
<tr>
<td>Wordpress</td>
<td>2.9.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>PHP</td>
</tr>
<tr>
<td>Novaboard</td>
<td>1.1.2</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>PHP</td>
</tr>
<tr>
<td>PHPBB</td>
<td>3.0.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>PHP</td>
</tr>
<tr>
<td>SimpleMachinesForum</td>
<td>1.1.11</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>PHP</td>
</tr>
<tr>
<td>Magento Shop</td>
<td>1.3.4.2</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>PHP</td>
</tr>
<tr>
<td>OSCommerce</td>
<td>2.2 RC 2a</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>PHP</td>
</tr>
</tbody>
</table>

- **Cookie:** CMS accepts foisted cookies
- **URL:** CMS accepts session ID via URL parameter
- **SID:** CMS accepts arbitrary SID values
2nd case study: HTTP header injection

- First stage attack: attacker sets cookie on client side
- Our case: user defined data taken for redirection

```
```

![Diagram of HTTP header injection]
2nd case study: HTTP header injection

Results:

- PHP: vulnerable in version < 4.4.2, < 5.1.2
- J2EE: not vulnerable
- CherryPy: vulnerable
- Perl: partially vulnerable (name ended with colon)
- Ruby on Rails: recently patched
Case studies:

- 9 out of 12 open-source Content Management Systems (CMS) vulnerable to session fixation
- 2 out of 5 web application frameworks (at least partially) vulnerable to http header injection
- 5 out of 8 web application frameworks vulnerable to session fixation (different work)
Session Fixation - Solution

- Fixing vulnerability straightforward: renew session ID if authorization level raises
- However: vulnerability on server side, risk on client side (like cross site scripting case)
  → little interest by application providers to find & fix
- Our proposed solution:
  - a proxy to strip off fixated session identifiers
  - Implements transparent session handling between client and proxy
  - Either on client side or on server side
Proxy links PSID \(a\) and SID \(x\)

If proxy receives request with unknown SID \(y\), all session ids are stripped off and a new session is established.

AppServer never sees PSID
Conclusion

- Public level of attention still rather low
- Despite given results: real world applications tested
  - Popular web services vulnerable (2 out of 4)
  - Online Banking web sites vulnerable (2 out of 5)
  - Internet access provider (1 out of 1)
- Risk exists though fixing is fairly easy
- Business partner uses proxy to buy time
- Proxy on server side – no big deal
- Proxy on client side – session ID detection not trivial
  - Future Work!
### Case Study

<table>
<thead>
<tr>
<th>Framework</th>
<th>API</th>
<th>AutoRotate</th>
<th>Conf. Fallback</th>
<th>AutoDisable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Server Faces</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Struts 2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Spring (Security)</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Zend</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Cake PHP</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>ASP.NET</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Web2py</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Django</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

- **API**: provides API to rotate SID
- **AutoRotate**: SID is rotated on every request (default configuration)
- **Conf. Fallback**: URL parameter fallback behavior configurable
- **AutoDisable**: URL parameter fallback is disabled per default