SessionSafe: Implementing XSS Immune Session Handling

ESORICS ‘06
20. September 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 Session Management
- Cross Site Scripting
- Protection Approaches
- Conclusion
Agenda

- Web Application Session Management
- Cross Site Scripting
- Protection Approaches
- Conclusion
Session management in http

- As HTTP is stateless, a web application has to implement its own session tracking mechanism.
- An authenticated user gets assigned a session ID.

Three common methods:

- URL rewriting
- Form based session IDs
- Cookies
Cookies are used to maintain state

- Stored on the client’s browser
- Send to the server along with matching requests
- Example: A request to `www.example.org/index.html` will send all cookies that are stored for
  - `www.example.org` and
  - `example.org`
- Cookies stored for other (sub-)domains are not sent
Session management with cookies

- The server sets a cookie at the client’s browser after the authentication form.
- This cookie is automatically included in all further requests.
- The client’s requests are treated as authorized as long as this cookie is valid.

```
Web browser       Web server
1. POST /login.cgi
2. "Welcome in" Web page
   Set-Cookie: authenticator
3. GET /restricted/index.html
   Cookie: authenticator
4. Content of restricted page
```
Agenda

- Web Application Session Management
- Cross Site Scripting
- Protection Approaches
- Conclusion
An attacker includes malicious JavaScript code into a webpage
This code is executed in the victim’s browser session
Today’s defense against XSS

Input Validation (IV):
- All untrusted input data is checked against predefined specification

Output Sanitization (OS):
- “Cleaning” of HTML
  - Removing malicious data
  - Encoding malicious data
This approach fails frequently

Reported XSS vulnerabilities:

- In the first six month of 2006 over 550 XSS vulnerabilities were reported to BugTraq
- All major frameworks were affected (php, j2ee, asp.NET,...)

Affected websites:

- Google (16.09.2006)
- MSN (25.06.2006)
- Yahoo (28.04.2006)
- PayPal.com (23.07.2006)
- Amazon.com (27.06.2006)
- Verisign (14.08.2006)
- ibm.com, sun.com, ebay.com,...
Why does the approach fail?

Input Validation (IV):
- IV is not always possible
- Has to be tailored to a specific scenario
- Scattered code

Output Sanitization (OS):
- Context dependent
  - e.g., from data origin, data destiny, role of the application’s user
  - Heterogeneous technologies lead to losing data’s context

IV/OS is not centralized enforceable
Why does the approach fail? (II)

Furthermore, there are XSS attacks that cannot be prevented this way:

- DOM based XSS
- Server induced XSS
  - Expect handler vulnerability

To make things worse:

- One single XSS vulnerability compromises the whole web application
Can we defend a web application against session hijacking after a successful XSS attack?

Approach:
- Determining the available security mechanisms in web architecture
- Classification of XSS session hijacking attacks
- Investigation of the basic requirements of these attack classes
- Applying the security mechanisms to revoke those requirements
Agenda

- Web Application Session Management
- Cross Site Scripting
- Protection Approaches
- Conclusion
The same origin policy

- Defines basic access rights in http
- Two elements have the “same origin” if the
  - protocol,
  - port,
  - and host
  are the same for both elements

- Access rights
  - JavaScript has unlimited access to all elements, which match the script’s origin
  - Cookies are sent along with all requests with matching target

JavaScript Closures
## XSS Session-Hijacking-Attack Classes

<table>
<thead>
<tr>
<th>Attack class</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionID Theft</td>
<td>Accessibility of the SID by the malicious JavaScript</td>
</tr>
<tr>
<td>◆ Leakage of the SID</td>
<td>Deferred Loading</td>
</tr>
<tr>
<td>Browser Hijacking</td>
<td>Pre-knowledge of the application’s URLs</td>
</tr>
<tr>
<td>◆ Execution of the attack within the browser</td>
<td>One-Time URLs</td>
</tr>
<tr>
<td>Background XSS Propagation</td>
<td>Implicit trust between browser documents with the same origin</td>
</tr>
<tr>
<td>◆ Controlling the attack with an outside entity</td>
<td>Subdomain Switching</td>
</tr>
</tbody>
</table>
This is XSS 101

- The malicious JavaScript obtains the SID
  - All presented SID mechanisms (URL, Form, Cookie) are vulnerable
- and communicates it to the attacker.
- The attacker is now able to impersonate the victim

![Diagram of Session ID Theft]

- Requirement: Malicious JavaScript has to be able to access the SID
Deferred Loading: Concept

www.example.org

HTML

Cookies

JavaScript

Attacker

XSS
Deferred Loading: Concept

synchronisation

www.example.org

secure.example.org

HTML
Cookies

JavaScript

JavaScript

XSS

Attacker

© Martin Johns, UHH, FB Inf, SVS, 20.09.2006
Concept: Synchronisation Problem

www.example.org  secure.example.org

Webserver

HTML

Browser

Cookie
Concept: Pageloader

synchronisation

www.example.org secure.example.org

Webserver

HTML

Cookie

Browser

Pageloader

Cookie

<img>
Protection against: SID Theft

- Because of the Deferred Loading process, the SID cookie and potential XSSs belong to different (sub)domains
- For this reason the malicious script has no access to the SID

www.example.org

- HTML
- JavaScript
- XSS

secure.example.org

- Cookie: SID
Implementation

- Transparent implementation for J2EE
- Serverside proxy
- The proxy executes the Deferred Loading process before passing the requests to the server
Agenda

- Web Application Session Management
- Cross Site Scripting
- Protection Approaches
- Conclusion
Conclusion

Implementation

- Transparent server-side proxy for J2EE

Limitations

- The XSS still has full control over the vulnerable webpage
- Further drawbacks:
  - AJAX has to be treated separately
  - One-Time URLs may break the “Back” and “Reload” button

Protection:

- Good protection against “hidden” attacks
- XSS may still be able to steal passwords by spoofing authentication dialogs
  - Client Side SSL authentication could help
Thanks for listening.

Questions? Comments?