SAP: Session (Fixation) Attacks and Protections
(in Web Applications)

Raul Siles
raul@taddong.com
March 17, 2011
Black Hat Europe 2011
Outline

• Session management and web security
• Session fixation
  – Discovery and exploitation (for pen-testers)
• Case studies
  1. Joomla! open-source CMS
  2. Commercial web application server
  3. World’s leader in business software (SAP)
• Conclusions and future research
Sessions in Web Applications

- A web session is a sequence of HTTP request and response transactions associated to the same user.
- Modern and complex web applications require to retain information or keep the state of each user for the duration of multiple requests.
- Sessions provide the ability to establish variables, such as access rights and localization settings, which will apply to every and each interaction a user has with the web application until she terminates her session.
Session Management in Web-Apps

- HTTP is a stateless protocol (RFC2616)
- Session tracking capabilities built on top of HTTP (session IDs or tokens)
- Key & core component of web-apps:

Are there any security risks? 😊
The Top 10 Most Critical Web Application Security Risks:

<table>
<thead>
<tr>
<th>OWASP Top 10 – 2007 (Previous)</th>
<th>OWASP Top 10 – 2010 (New)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2 – Injection Flaws</td>
<td>A1 – Injection</td>
</tr>
<tr>
<td>A1 – Cross Site Scripting (XSS)</td>
<td>A2 – Cross-Site Scripting (XSS)</td>
</tr>
<tr>
<td>A7 – Broken Authentication and Session Management</td>
<td>A3 – Broken Authentication and Session Management</td>
</tr>
<tr>
<td>A4 – Insecure Direct Object Reference</td>
<td>A4 – Insecure Direct Object References</td>
</tr>
<tr>
<td>A5 – Cross Site Request Forgery (CSRF)</td>
<td>A5 – Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>&lt;was T10 2004 A10 – Insecure Configuration Management&gt;</td>
<td>A6 – Security Misconfiguration (NEW)</td>
</tr>
<tr>
<td>A8 – Insecure Cryptographic Storage</td>
<td>A7 – Insecure Cryptographic Storage</td>
</tr>
<tr>
<td>A10 – Failure to Restrict URL Access</td>
<td>A8 – Failure to Restrict URL Access</td>
</tr>
<tr>
<td>A9 – Insecure Communications</td>
<td>A9 – Insufficient Transport Layer Protection</td>
</tr>
<tr>
<td>&lt;not in T10 2007&gt;</td>
<td>A10 – Unvalidated Redirects and Forwards (NEW)</td>
</tr>
<tr>
<td>A3 – Malicious File Execution</td>
<td>&lt;dropped from T10 2010&gt;</td>
</tr>
<tr>
<td>A6 – Information Leakage and Improper Error Handling</td>
<td>&lt;dropped from T10 2010&gt;</td>
</tr>
</tbody>
</table>

### A3 Broken Authentication and Session Management

<table>
<thead>
<tr>
<th>Attack Vectors</th>
<th>Security Weakness</th>
<th>Technical Impacts</th>
<th>Business Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitability</td>
<td>Prevalence</td>
<td>Detectability</td>
<td>Impact</td>
</tr>
<tr>
<td>Average</td>
<td>Common</td>
<td>Average</td>
<td>Severe</td>
</tr>
</tbody>
</table>

**Exploitability Average**
- Consider anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others. Also consider insiders wanting to disguise their actions.

**Prevalence Common**
- Attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users.

**Detectability Average**
- Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique.

**Impact Severe**
- Such flaws may allow some or even all accounts to be attacked. Once successful, the attacker can do anything the victim could do. Privileged accounts are frequently targeted.

**Business Impacts**
- Consider the business value of the affected data or application functions.
- Also consider the business impact of public exposure of the vulnerability.
WASC Threat Classification v2.0

• WASC-18: Credential & Session Prediction
  – Session ID disclosure and/or interception
  – Session ID prediction or brute-forcing
  – Session hijacking (sidejacking)
• WASC-37: Session Fixation
• WASC-47: Insufficient Session Expiration

http://www.webappsec.org/projects/threat/
Session Fixation

• Discovered and/or publicized at the end of 2002 by Mitja Kolšek
  – Obtaining vs. “Fixing” a valid session ID
• The attacker fixes the session ID before the victim logs in to the target web-app
• Types: permissive and strict session mgmt.
• State-of-the-art (after 9 years)?

What Session Fixation Should Be?

Fixation

n. a strong, often unhealthy attachment or preoccupation

Obsession &
An habit of activity or practice

Session Fixation Discovery

• Evaluate session tracking pre and post-authentication (and compare)
  – Identify the session ID transport or exchange mechanism (web interception proxy)
  – Get a valid session ID (pre/post-authentication)
  – Fix the session ID playing the victim user role
  – Authenticate into the target web-app
  – Analyze the response post-authentication

Same session ID, or no session ID, in the response?
Multiple mechanisms are available in HTTP to maintain session state.

Session ID sent as a...
- Cookie (standard HTTP header)
- URL parameter (URL rewriting) – RFC 2396
- URL argument: GET request (URL rewriting)
- Body argument: POST request
- Hidden form field (HTML forms)
- Proprietary HTTP header
Session ID Exchange (2)

- Cookie (standard HTTP header):
  - Cookie: id=012345; ...

- URL parameter: (URL rewriting)
  - https://portal.example.com/private;id=012345?...

- URL argument (GET request):
  - https://portal.example.com/private?id=012345&…

- Body argument (POST request):
  - id=012345&…

- Hidden form field (HTML):
  - <INPUT TYPE=“HIDDEN” NAME=“id” VALUE=“012345”>

- Proprietary HTTP header:
  - Portal-session-ID: id=012345
Session ID Exchange
Used vs. Accepted

• Method used by the application vs. method(s) accepted by the application

• Example:
  – Application uses cookies to exchange IDs, but also accepts session IDs in URLs
    • Can use both: automatic URL rewriting
    • Clients w/o cookie capabilities or not accepting them
  – Session ID disclosure
  – Facilitates session fixation attacks
Session Fixation Discovery Summary

HTTP request (w/o session ID) → ID (pre-authentication)
HTTP response (session ID) → ID

Authentication
Response (post-authentication)
HTTP request (token)
Session specific data
HTTP request (token)
...

Authentication or any application privilege level change

Pen-tester
Web-App
Session tracking
The Attacker is After the…
Session Fixation Exploitation

• Active attack for session hijacking and user impersonation
  – Targeted attacks against sensitive users
  – Indiscriminate attacks as any legitimate user

• Unauthorized access (or privilege escalation attacks) as victim user

• Fixation and exploitation phases
  – Wait till the victim user authenticates
Session Fixation Attacks

Attack Vectors (1)

• Web references or links (URLs):
  – Social engineering tricks: entice user to follow the link with the attacker’s session ID
    https://portal.example.com/private;sessionid=012345?...

• HTTP meta tags (e.g. cookies):
  – Cannot be disabled in web browsers
    https://portal.example.com/<meta%20http-equiv=Set-Cookie%20content="SESSIONID=012345;%20path=/;...">

• Untrusted client shared environments
Attack Vectors (2)

• Web traffic interception & manipulation:
  – MitM attacks over unencrypted HTTP traffic to add or replace legitimate session IDs
  – Any exchange mechanisms (single request)

  ```
  Set-Cookie: SESSIONID=012345; expires=Friday, 17-May-13 18:45:00 GMT; ...
  ```

• Cross-subdomain cooking: (design)
  – “domain” cookie attribute from vuln servers

  ```
  Set-Cookie: SESSIONID=012345; domain=.example.com; ...
  ```
Attack Vectors (3)

- HTTP response splitting:
  - Inject session IDs (as HTTP headers)
  - E.g. HTTP redirection

REQ: https://portal.example.com/login
Set-Cookie: SESSIONID=012345
Dummy-Header:

RESP:
HTTP/1.1 302 Found
Server: Vulnerable Server 1.0
Location: https://portal.example.com/login
Set-Cookie: SESSIONID=012345
Dummy-Header: /login
...
Attack Vectors (4)

• Cross-Site Scripting (XSS):
  – Set the session IDs through JavaScript
  – Target web applications (or subdomain apps)
  – Persistent and reflective XSS

https://portal.example.com/search?q=<script>
document.cookie="SESSIONID=012345;%20path=/; %20domain=.example.com";</script>

• SQL injection:
  – Session management database (subtle attacks)
Session Fixation Benefits

• Bigger attack window
  – Initial fixation occurs pre-authentication
  – Victim user authenticates (long time afterwards)
  – Attack is exploited post-authentication (active)

• Extended attack lifetime
  – Persistent cookies (e.g. 10 years)
  – Web application terminates the session
  – Session ID remains on the user browser waiting for the session to be resumed (or re-launched)
Session Fixation Exploitation Summary

HTTP request (w/o session ID) → ID (pre-authentication)
HTTP response (session ID) ← ID

Pen-tester → ID → Victim user

Web-App
Vulnerable Web-App

Attck vector(s): combined & target dependant
Case Studies
Three Case Studies

• From real-world penetration tests
  – Past two years: 2009-2010
  – Three different session fixation vulnerabilities on three separate target web environments

• How they were discovered & exploited

• Real impact

• Vulnerability disclosure timeline

• Protections
Discovering Security Vulnerabilities

I DISCOVERED A HOLE IN OUR INTERNET SECURITY.

GOOD GRIEF, MAN! HOW COULD YOU PUT A HOLE IN OUR INTERNET?

WHAT?!!

I DIDN'T PUT IT THERE. I FOUND IT... AND IT'S NOT...

IT'S YOUR JOB TO FIX THAT HOLE. I WANT YOU TO WORK 24-7!

ACTUALLY, THAT'S NOT MY JOB. BUT I'LL INFORM OUR NETWORK MANAGEMENT GROUP.

PASSING THE BUCK!!! YOU'RE A BUCK PASSER!!!

FORGET IT! THERE'S NO HOLE! IT GOT BETTER!

THAT'S MORE LIKE IT.

I FIXED THE INTERNET.

Case Study #1
Joomla! Open-Source CMS
#1 Summary

- Session fixation in Joomla!, a widely used open-source CMS
- Affected versions: 1.5.x – 1.5.15
- Vulnerability ID: 20100423 (TAD-2010-001)
- Notified: November 2009
- Release date: April 2010

#1 Discovery and Exploitation

- Target **HTTPS-only** web application
  - Public & private sections (registered users)
  - Built-in Joomla! core session management
  - Authentication: e-National ID card or user/pass

- **MD5 hashes** for session ID and value
  - Ignore it: meaning & purpose are not required
  - Discovered through a blackbox pen-test but…
  - Source-code available: whitebox pen-test
#1 Impact

• **Open-source CMS**
  – Non-profit organizations, academic institutions, and non-business related and …
  – … business critical web-applications
    • Commercial companies and governments
    • Standalone, source-code customizations, and other frameworks (internally and publicly)

• **All 1.5.x Joomla! versions up to 1.5.15**
  – Depending on criticality of web application
#1 Vulnerability Disclosure Timeline

- Lessons learned from vulnerability notifications, handling, and disclosure
  - Definitely, open for improvement!!
- Advisory says reported on March 25, 2010, when it should say Nov 2009
- “The Seven Deadly Sins of Security Vulnerability Reporting” blog post
  
#1 Protections

- Web applications based on Joomla! must upgrade to the latest Joomla! version (1.5.16 or later)
Case Study #2
Commercial Web Application Server
#2 Summary

• Session fixation vulnerability on a widespread commercial web application server (HTTP vs. HTTPS misbehavior)

• Affected versions: ?
• Vulnerability ID: ?
• Notified: December 2010
• Release date: ?

No references yet...
#2 Discovery and Exploitation (1)

- Complex & recently redesigned web-app
- Public section + private section (auth)
- Java-based cookie (JSESSIONID)
  - Pre-authentication
  - “domain” & “path” attributes
  ```
  Set-Cookie: JSESSIONID=Fz5f…qMql;
domain=.example.com; path=/
  ```
- Authentication (POST & HTTPS & cookie):
  ```
  https://portal.example.com/private/miPortal
  ```
#2 Discovery and Exploitation (2)

- Successful authentication (post-auth):
  
  ```
  Set-Cookie: AUTH_JSESSIONID=G4lD…vQ14; path=/; secure
  ```

- Any previous AUTH_JSESSIONID value is renewed

- Very common scenario: two cookies
  - Pre-auth (unsecure): always (www, portal, etc)
  - Post-auth (secure): portal only, renewed

  Session ID (auth users) = JSESSIONID + AUTH_JSESSIONID
• All links from “www” to “portal” are HTTPS
  – But HTTP is also allowed in “portal”
• What is used for session ID verification when accessing “authenticated resources”?
  – Common sense: both cookies (! in reality)
• HTTPS behavior:
  1. Both cookies: OK
  2. JSESSIONID bad: redirect to login & renewed
  3. AUTH_JSESSIONID bad: 401 Basic?

  Missing or expired
#2 Discovery and Exploitation (4)

- HTTP behavior:
  - Once authenticated, HTTPS requires both
  - HTTP only makes use of JSESSIONID
- All resources available through HTTP 😞
- JSESSIONID is enough to associate the web request to an authenticated session (via HTTP)!!
- Remember, JSESSIONID is not renewed

Even simpler attacks as JSESSIONID is disclosed via HTTP
#2 Impact

• Three possible scenarios:
  – *High*: commercial web-app server found vulnerable (all web-apps)
  – *Mid*: vulnerability due to misconfiguration of the commercial web-app server. *How easy is to introduce the wrong setting?*
  – *Low*: only the specific web-app it was discovered in is vulnerable

• Even if not 0-day, subtle sample of HTTP(S) misconfiguration and session management
#2 Vulnerability Disclosure Timeline

• Notified to vendor in early December 2010
  – Quick analysis & limited target information
  – Conclusion: Specific to target environment

• Mid-February 2011: with full config details
  – Being re-analyzed for confirmation
  – Web-app source code required for depth analysis and ratification?
    – Under study at the time of preparing presentation…
#2 Protections

• Waiting for full ratification to provide accurate and specific countermeasures

• Separate public & private web environments
  – Server, IP, hostname, and domain
  – Session management infrastructure

• Pen-testers must try this!! (lessons learned)
  – HTTP vs. HTTPS inconsistencies
  – Session management verifications (# cookies)
    • Even if available only through HTTPS?
Case Study #3
World’s Leader in Business Software
#3 Summary

- Session fixation in the SAP J2EE Engine affecting the core SAP NetWeaver platform
- Affected versions: 6.40 - 7.20
- Vuln ID: SAP Security Note 1310561 (TAD-2011-002)
- Notified: July 2009
- Release date: December 2010 (SAP SMP)

https://websmp130.sap-ag.de/sap/support/notes/1310561
#3 Discovery and Exploitation (1)

• Large penetration test (net, web-app, wi-fi)
• Some of the target servers were the Intranet website and the SAP systems
  – Critical business processes and activities
• This website contained a link (used by employees) to the SAP Portal (HTTP)
  – http(s)://intranet.example.com (NTLM auth)
  – http://portal.example.com (SAP NW Portal)
• SAP Portal redirects to HTTPS version
#3 Discovery and Exploitation (2)

- **HTTP 307: “Temporary Redirect”**
  - `https://portal.example.com/irj/portal`

- The common & “innocent” HTTP redirection discloses all the session cookies: (network traffic)
  - `saplb_*`, `PortalAlias` & `JSESSIONID`

- Even if the reference is HTTPS, the lack of the “secure” attribute makes possible to MitM it and relay fictitious HTTP to HTTPS (e.g. SSLstrip)

- Target SAP Portal supported client-based digital certificates (smart card ID) or user/password auth
#3 Discovery and Exploitation (3)

- Pen-tester obtains a valid session ID (pre)
- The session ID is “fixed” in the victim browser (ARP poisoning & traffic control)
  - MitM by injecting the session ID in the cookie headers of the HTTP response (307 redirect)
- The user authenticates in the SAP Portal
  - Session ID does not change (session fixation)
- Pen-Tester gets full access to victim’s session (business critical data and actions)
#3 Discovery and Exploitation (4)
• Attacker only had to reuse the following specific set of target cookies:

Cookie:
- saplb_*= (J2EE01234567)01234567;
- PortalAlias=portal;
- JSESSIONID=(J2EE01234567)
  ID0123456789DB01234567890123456789End;
- MYSAPSSO2= AjEx…(very long string)…ewCw%3D;
- SAPWP_active=1
#3 Discovery and Exploitation (6)

- SAP NW Portal version 6.4.200607310245:
  - Server: SAP Web Application Server (ICM)
  - Server: SAP J2EE Engine/6.40
  - PortalVersion:"6.4.200607310245"

- SAP Portal session IDs available pre-authentication

- Post-authentication, session IDs do not change (session fixation)

- Choose targets selectively (business role)
#3 Impact (1)

• Hijack any SAP user (or admin) session
  – Unauthorized access to SAP Portal and other SAP applications and modules
  – SAP NetWeaver is SAP’s integrated technology platform & technical foundation for all SAP apps
  – Key business users (target core business)

• Real-world impact: who could be affected?
  – SAP AG: world’s leader in enterprise biz SW
  – +109,000 customers in 120 countries
  – +140,000 installations & +2,400 cert partners
SAP Architecture

User adaptation
- Duet / Alloy / Portal / Mobile

Flexibility, extensibility
- Business process management – Composite applications

Business insights
- SAP BusinessObjects portfolio
  - Business intelligence – Information management –
  - Enterprise performance management – Governance, risk, and compliance

Industry core processes
- Industry-specific extensions
  - SAP Business Suite
  - On-demand extensions
  - SAP Business All-In-One
  - SAP Business ByDesign

Horizontal core processes
- SAP NetWeaver – Process Integration –
  - Master Data Management – Information Lifecycle Management

Integration
- Large Enterprises
- Midsize Companies
- Small Businesses

Copyright © 2011 Taddong S.L. www.taddong.com
#3 Impact (2)

- Direct impact of software-based and web services-based business activities of thousands of organizations and companies worldwide

- Session fixation might impact web-app design
  - In-depth architecture analysis & 3rd-parties & redesign
  - Minor change can break other components
  - E.g. User impersonation between applications
    - SSO (Single Sign On) or session management tricks
  - E.g. Software components that receive and use IDs
    - Without capabilities to discern if it is valid or not

Bypass the most advanced authentication mechanisms
#3 Impact (3)

- SW maintenance & support strategy: 7-2
  - 7 years mainstream + 2 years extended
  - Fixes for new & legacy versions (production)

Security Response Process has been enhanced and aligned across all product areas: Monthly Patch Day introduced in September 2010. Service Marketplace supports crediting external researchers
#3 Vulnerability Disclosure Timeline (1)

- Complexity of modern web architectures and broad vulnerability scope = 1,5 years
- Reported on early July 2009 & ratified
  - First deadline: 2 months (best case scenario)
- Mid Sep’09 difficulties identified (stability)
- Nov’09: estimated release on Jan/Feb’10
  - Responsible disclosure (plans) & real impact
  - Initial technical solution being tested

Meanwhile environments remain vulnerable...
#3 Vulnerability Disclosure Timeline (2)

• End Jan’10: solution still not available
  – Issue escalated internally
  – Several months required (all affected releases)

• Mar’10: fixes for all cases expected +Sep’10
  – Issues found on legacy releases
  – Partial fixes for specific CUs under evaluation

• Aug’10: meeting date for Nov’10 (disclosure)

• Dec’10: vuln & fix releases (CUs & partners)

• Mar’11: implementation time of 3 months
SAP Disclosure Guidelines (1)

• SAP disclosure guidelines details:
  – Published after this specific finding
  – “Since the integrity and security of business operations is crucial for businesses in all industries, SAP as a provider of business software is absolutely committed to maintaining the highest possible level of security within its products.”
  – What is the right balance between full security and fast disclosure?

Other researchers can find it: != motivations (see case #1)
SAP Disclosure Guidelines (2)

• Fix and vuln disclosure details and timing:

PLEASE GIVE SAP SUFFICIENT TIME TO DEVELOP SUITABLE FIXES

- Fixing security vulnerabilities can be a long and arduous process as we work to develop a patch, ensure its compatibility with all relevant software versions, run comprehensive tests to ensure that the fixes run well and do not have any side-effects, and provide it to our customers.
- As a vendor of business software we provide security fixes not only to the latest version but also for many older versions of our software products. This means that we need to develop and thoroughly test feasible patches for a broad range of product versions, which can take time.

PLEASE DO NOT PUBLICIZE VULNERABILITIES UNTIL SAP CUSTOMERS HAVE HAD TIME TO DEPLOY FIXES

- The deployment of patches for SAP enterprise systems is usually more complicated than a software upgrade on a consumer PC. Depending on the nature of the vulnerability, the deployment of patches often is not only done by an automated update; in some cases it requires manual configuration work in the system.
- Some of our customers also have regular patching cycles, for instance on a monthly or a quarterly basis.
- In light of these circumstances, we ask all security researchers to give SAP customers sufficient time to implement patches in their SAP systems. As a rule of thumb, we suggest respecting an implementation time of three months. We ask all security researchers to not disseminate any kind of information or tools that would help to exploit the vulnerability during that time.

New SAP security program: highlight security notes, periodic releases & credit

Is the all or nothing approach the right approximation?
#3 Protections (1)

- Monthly Patch Day (since Sep’2010)
- SAP ACK to security researchers:
  - [Taddong, Raul Siles, SAP Security Note 1310561](https://websmp130.sap-ag.de/sap/support/notes/1310561)
  - SAP Security Note 1310561
    - December 2010
    - [https://websmp130.sap-ag.de/sap/support/notes/1310561](https://websmp130.sap-ag.de/sap/support/notes/1310561) (SAP Service Marketplace)

Third oldest #id, after 1175239 (related) & 1151410

Enable “SessionIdRegenerationEnabled”
- SAP Security Note 1310561
- Web Container Service property
- Two cookies required to identify sessions: JSESSIONID & JSESSIONMARKID ("secure")
- The new “secure” session ID is renewed on every successful login
- Disabled by default but…
- Enabled in +7.11 SP06 & all SPs 7.20 & 7.30
- Specific scenarios may require extra steps
#3 Protections (3)

- Use HTTPS-only links & remove HTTP support in SAP Portal
- Enable “SystemCookiesHTTPSProtection”
  - SAP Security Notes 1019335 & 1020365
  - HTTP Provider Service property
  - Sets the “secure” attribute for session and load balancing cookies (JSESSIONID & saplb)
  - Available in 6.40 SP21 & 7.0 SP14
  - Disabled by default

Vendor conservative settings & backward compatibility. Security teams!!
Enable “SessionIPProtectionEnabled”

Web Container Service property

- Manages J2EE web components
- HTTP session cannot be accessed from different IP addresses. Only requests from the IP addr that started the session are processed
- Disabled by default
- If front proxy or load balancer is used

Configure the “ClientIpHeaderName” property of the HTTP Provider Service (e.g. relay “X-Forwarded-For” header)
Conclusions
Session Fixation Protections

- Renew session ID after privilege level changes
- Lack of link between authentication and session management capabilities (best practices only)
  - Web developer’s hands (e.g. PHP or Java or .NET…)
- Limit accepted session tracking mechanisms
- HTTPS everywhere
- Session ID available only post-authentication
- Bind session ID to other user properties
- Isolate critical web-apps on its own domain
- Very restrictive cookie attributes
Conclusions (1)

• Session fixation still prevalent in 2010
  – Open-source projects, commercial web application frameworks, and mission critical business platforms

• Thousands of critical and business-related web environments affected worldwide

• Entry point to get unauthorized access to business critical data and infrastructures
  – Targeted, criminal, and corporate espionage

• Multiple exploitation methods available
Conclusions (2)

• Session attacks can bypass even the most advanced authentication mechanisms

• Session ID is equivalent to…
  – Password
  – Passphrase
  – Digital certificates
  – Smart cards
  – Fingerprint
  – Eye retina
Conclusions (3)

• Impact on the web-app design and on multiple modules (and 3rd-party components)
  – Complexity of web-apps and core nature of session management infrastructures
  – Minor misconfiguration introduces vulnerability?
  – How easy is to fix session fixation?
  – Plan and test early in design and development

• Promote (continuous) testing for session fixation flaws, development awareness, and improve vulnerability handling and disclosure
Future Research

• Session fixation state-of-the-art on the wild
  – Widely used Internet services and selected sample of critical web applications
  – Valid user account on the target web-app

• Manual techniques vs. semi-automated tool for discovery and basic exploitation
  – Automate verification and extend testing

• Authentication and privilege level changes
I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions. I will use Google before asking dumb questions.