

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

Raúl Siles - raul@taddong.com

Black Hat Europe 2011 (March, 17-18 – Barcelona, Spain)

ABSTRACT

Session fixation is an old and well-known web application vulnerability since 2002, but still today, open-source projects, widely deployed web application frameworks, and mission critical commercial business platforms are vulnerable to it, exposing thousands of production web environments worldwide. In particular, the exposure of business platform web interfaces on the Internet, as well as internally, makes this type of vulnerability the entry point to get access to unauthorized business critical data and infrastructures through targeted, criminal (blackmail, fraud, extortion, sabotage, theft and abuse), and industrial and corporate espionage attacks.

The discovery of session fixation and management flaws in web applications can have a devastating impact, allowing attackers to bypass even the most advanced authentication mechanisms. Due to its nature as a core component on web application architectures, plus the complexity of modern web solutions and too broad session management requirements on industry specifications, fixing session fixation vulnerabilities may require a full reassessment and in-depth analysis of the web application design, impacting third party modules and products also, and requiring (in some cases) several months to get them fixed; meanwhile environments remain vulnerable.

This white paper/presentation provides an updated in-depth look at session fixation attacks through case studies from real-world penetration tests, including the details of how these vulnerabilities were discovered and exploited, the vendor timelines from initial reporting until fix and disclosure, and its impact. Following responsible disclosure and best practices during the last two years, the examples detail vulnerabilities in the open-source Joomla! CMS, plus the public disclosure of a session fixation in a widely used web application server, and a 0-day vulnerability in the core platform of the world's leader in business software.

INTRODUCTION

The white paper/presentation analyzes the state-of-the-art of session fixation vulnerabilities and attacks in web applications using three case studies from real-world penetration tests. The details include an in-depth analysis of different session fixation flaws discovered by the author. These case studies offer an updated look at session fixation vulnerabilities and attacks nowadays and their real impact on production environments.

After briefly introducing session fixation vulnerabilities, the white paper describes how these flaws can be discovered and provides pen-testers and security researchers with multiple attack and exploitation methods to demonstrate the impact of session fixation. The cutting-edge case studies from real-world penetration tests dissect three different vulnerabilities on an open-source CMS project, a commercial web application server, and a commercial business software platform, including the complexity, implications, and real impact of a core session fixation vulnerability that has taken more than one and a half years to get fixed. Finally, the presentation inspects best practices and defenses for web developers and web architects to eliminate or reduce the impact of session fixation vulnerabilities.

A continuous testing on session fixation vulnerabilities by pen-testers, auditors, and web developers is mandatory to emphasize and reflect the real threat and significant impact of this type of vulnerability in web-based commercial products, web applications, and critical production environments. Unfortunately, the software industry cannot relax thinking this is a vulnerability from the past. Collaterally, the discovery of these vulnerabilities corroborates a serious need for improvement among the information technology and software industry regarding standard vulnerability handling practices notified by third parties.

SESSION FIXATION

Session management is a key component of modern web applications, mainly because the HyperText Transfer Protocol (HTTP) is a stateless protocol (RFC2616) and, therefore, a complete session tracking mechanism was built on top of this protocol afterwards. The session management capabilities of web applications link the authentication layer with the access control mechanisms implemented throughout the application, and provide the session tracking features required by complex web applications in order to implement the user session concept, granular authorization mechanisms, and manage different user privilege levels without requiring a continuous authentication exchange:



A web session, or user session, is a sequence of HTTP request and response transactions associated to the same user. Modern and complex web applications require to keep the state or other information about 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 Fixation Definition

Session fixation is a web-based security vulnerability that was discovered and its term publicized at the end of 2002 by Mitja Kolšek [1]. Although it is an old and well-known vulnerability, it is still prevalent in modern and crucial web applications nowadays. Most of the session management vulnerabilities that affect web applications, used in session ID disclosure, interception, prediction, or brute-force attacks, are based on the attacker obtaining a valid session ID of a victim user in order to impersonate her. Session fixation seats apart, as its goal is just the opposite, the attacker sets (or “fixes”) the session ID of the victim user in advance to impersonate her afterwards. In both cases, the attacker’s end goal is to possess a valid user session ID to hijack her session, an attack known as session hijacking (or sidejacking).

Session fixation is accurately defined on the original session fixation paper abstract [1]:

*“Many web-based applications employ some kind of session management to create a user-friendly environment. Sessions are stored on server and associated with respective users by session IDentifiers (IDs). Naturally, session IDs present an attractive target for attackers, who, by obtaining them, effectively hijack users’ identities. [...] In a **session fixation attack**, the attacker fixes the user’s session ID before the user even logs into the target server, thereby eliminating the need to obtain the user’s session ID afterwards. There are many ways for the attacker to perform a session fixation attack, depending on the session ID transport mechanism (URL arguments, hidden form fields, cookies) and the vulnerabilities available in the target system or its immediate environment.”*

Session Fixation Types

There are two types of session fixation attacks based on the behavior of the session management implementation of the target web application:

- *Permissive session management*: The web application allows web clients to specify any session ID, accepting arbitrary (session ID) values as valid and incorporating them within the session tracking mechanisms. The application might require session IDs with the same format as the legitimate ones, or even accept any (or a random) session ID format.
- *Strict session management*: The web application only accepts server-side generated session ID values. Some basic checks are executed to verify that the session ID provided by the client has been previously created by the web application and it is currently available on the session tracking infrastructure. These legitimate session IDs could have associated a session ID timeout, although it is rather unlikely before the authentication takes place.

How does the attacker get a valid session ID (for the fixation attack)? In the permissive case, the attacker has more flexibility to select the session ID and can generate her own values. In the strict case, most

common nowadays, the attacker has to connect to the target web application first to gather a legitimate session ID.

Session Fixation Discovery (for Pen-Testers)

The discovery of session fixation vulnerabilities requires to evaluate the target web application response (in regards to session tracking) before the authentication takes place (pre-authentication) and upon a successful authentication (post-authentication) and compare the two.

The first step is to identify what specific mechanism is used by the web application for session tracking purposes, as it might use cookies, URL parameters (RFC 2396, section 3.3), URL arguments (HTTP GET request), body arguments (HTTP POST request), proprietary HTTP headers, or hidden form fields (HTML). The pen-tester simply needs to establish a connection to the target web application to identify the session tracking mechanism and record the value of the valid session ID (attacker's ID) set by the target web application on the response.

Playing the victim user role, the pen-tester manually sets (or fixes) the recorded session ID into the victim user browser and starts a new session using that ID. Once the (allegedly) victim user successfully authenticates, the analysis of the HTTP response received post-authentication confirms if the victim session is bound to the same session ID or not, that is, if the session ID value used during the authentication has not changed afterwards. If the HTTP response upon a successful authentication contains the same session ID value, or no value at all (the previous value is still valid), it means the same session ID is being (re)used and, thus, the web application is vulnerable to session fixation. At this point, the pen-tester can resume her original session, and get full access to the web application as the victim user with her privilege level.

If the target web application only provides the session ID after a successful authentication (post-authentication), the pen-tester will need in advance a legitimate user account in the application in order to collect a valid session ID. The same requirement applies to mixed environments, where the session ID is made up of multiple tokens (such as various cookies), some obtained pre-authentication and some post-authentication. Even in these scenarios, if the victim user logs in to the web application with the attacker's (session) ID and the context of this session ID is reassigned from the attacker's authenticated session to the (new) victim user's authenticated session, the application is vulnerable.

Most of these discovery steps can be easily executed by using a web interception proxy that allows the pen-tester to manually inspect and manipulate HTTP requests and responses.

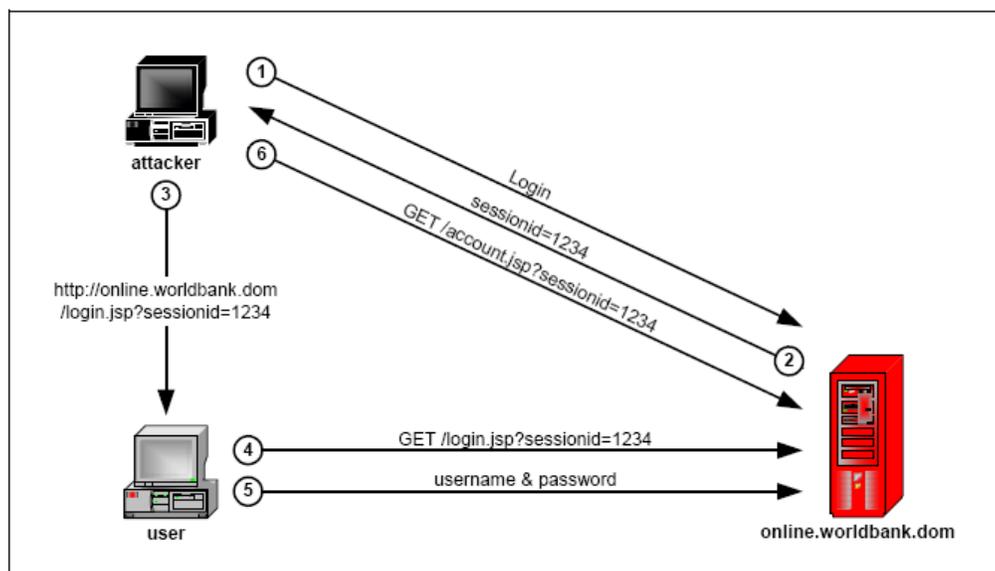
Although the discovery example above focuses on the authentication process, session fixation applies to any web application privilege level change, such as when an already authenticated regular user obtains administrative privileges within the application. Thus, this vulnerability can be used for unauthorized access or privilege escalation attacks.

Session Fixation Exploitation

Session fixation is an active attack that can be exploited to gain access to the web application as a currently logged in victim user by hijacking her session, thus, impersonating a valid authenticated user. This vulnerability can be exploited on targeted attacks against specific and sensitive users of a web application, or to launch indiscriminate attacks to access the target web application as any valid user. One of the requirements is the existence of an active session, therefore, insufficient session expiration practices increase the feasibility and impact of different session management attacks.

Once the attacker obtains a valid session ID she must fix that value (or ID) into the victim user, and wait till the user authenticates into the web application using that ID. Although the typical session fixation attack example involves social engineering tricks to force the victim to click on a malicious link that contains the attacker’s session ID (assuming the web application accepts session IDs within URLs), this is not the only way this vulnerability can be exploited.

The “fixation” of the session ID on the victim user, step 3 on the image below (extracted from the original session-fixation paper [1]), is the key step for the successful exploitation of session fixation attacks, as all the other steps involve standard HTTP requests and responses:



Although most web applications nowadays make use of cookies as their session tracking mechanism, lots of them also accept session IDs within other alternative exchange mechanisms, been URLs the second most accepted method (known as URL rewriting). Therefore, apart from the implications of disclosing the session ID in the URL, an attacker can get the session ID from the web application in the form of a cookie, and use the session ID value within a URL to easily launch session fixation attacks:

The session ID is set by the web application in the form of a cookie:

```
Set-Cookie: SESSIONID=abcdefghijklmnopqrstuvwxy0123456789
```

However, the web application also accepts session IDs in URLs:

```
https://portal.example.com/private;sessionid=abcdefghijklmnopqrstuvwxyz0123456789?lang=es&vuln=yes
```

There are multiple attack and exploitation vectors available for pen-testers, security researchers, and attackers to exploit session fixation vulnerabilities in web applications:

- **Web references or links (URLs):** The attacker fixes the session ID by enticing an unsuspecting victim into following a malicious link (or URL) that contains the attacker's session ID as a URL parameter or URL argument (HTTP GET request). The malicious link can be actively sent to the victim user inside an e-mail, SMS, or instant message, or can be passively published in any web resource, such as a web page, forum, blog, social networking site, document, etc:

```
https://portal.example.com/private?sessionid=0123456789&... (URL argument)
```

```
https://portal.example.com/private;sessionid=0123456789?... (URL parameter)
```

A similar scenario applies to session IDs exchanged through POST parameters if the attacker uses HTML forms (to fix the session ID) within a web page or e-mail.

- **HTTP meta tags:** An attacker can take advantage of HTTP meta tags within URLs to set new cookies, and as a result, fix a new session ID on the victim user following the malicious URL:

```
https://portal.example.com/<meta%20http-equiv=Set-Cookie%20content="SESSIONID=0123456789;%20path=/;...">
```

HTTP meta tags are processed by browsers anywhere within an HTML document and even when other web browser capabilities are disabled, such as scripting. OWASP states the processing of HTTP meta tags cannot be disabled in web browsers¹.

- **Untrusted client shared environments:** The attacker gets access to a shared computer, fixes the session ID on the web browser (or any other web client), and leaves the web browser open and ready (for example, pointing to the vulnerable login webpage), waiting for the victim user to log in. This scenario is very common on cybercafés, libraries, or Internet kiosks. The next user that arrives to the shared computer and authenticates to the target web application will become victim of the attack. Closing and reopening the web browser before starting to use it does not protect against the attack when the session ID is persistent.
- **Web traffic interception and manipulation:** The attacker can launch a (layer-2 to layer-5) man-in-the-middle attack against the victim user to intercept and modify all her web traffic. The goal of the attack is to add a new or replace the existing session ID received from the target web

¹ Session fixation (OWASP): http://www.owasp.org/index.php/Session_fixation

application within HTTP responses. The attacker only needs a single unencrypted HTTP request against the target web application to modify the associated response and include the fixed session ID, for example, as a cookie (standard HTTP header):

```
Set-Cookie: SESSIONID=0123456789; expires=Friday, 17-May-13 18:45:00 GMT; ...
```

This attack method can be used to manipulate any session tracking exchange mechanism, such as hidden form fields (HTML) or URL-based session IDs, as the attacker has full control over the web traffic.

- **Cross-subdomain cooking:** This attack vector takes advantage of an implicit weakness associated to cookies, where any web application can set a cookie for all other web applications in the same domain (or subdomains). By specifying the “domain=.example.com” cookie attribute, any web server on the same domain (or any subdomain) can fix a session ID in the form of a cookie for the target web application (or subdomain) vulnerable to session fixation. For example, when the victim user browses the server at “tests.example.com” the website assigns the following cookie for “example.com”, that will be sent by the victim user when she accesses the target web application at “portal.example.com”:

```
Set-Cookie: SESSIONID=0123456789; domain=.example.com; ...
```

This attack vector is easily exploitable if subdomains are managed by an untrusted third-party or if the security stance and protection level of each subdomain differs. Vulnerabilities in a non-critical subdomain (e.g. “tests.example.com”), like code injection or cross-site scripting (XSS), can be leveraged by an attacker to inject malicious domain level cookies and exploit the main target subdomain (“portal.example.com”). This attack vector can be combined with other vulnerabilities, such as DNS spoofing or poisoning attacks, where the attacker's web server impersonates a legitimate system on the target domain and issues malicious domain cookies.

By default, when a cookie is set without the “domain” attribute, it is only associated to the specific host from where it has been obtained, such as “portal.example.com”.

- **Cross-site cooking:** Due to (old) vulnerabilities in certain web clients², a malicious web server could set cookies for a different domain. The attacker, controlling the malicious web server, can fix a session ID in the form of a cookie on the victim web browser for the target web application (or domain) vulnerable to session fixation. When the victim user browses the malicious server at “www.blackhat.com” the website sets the following cookie for the “example.com” domain:

```
Set-Cookie: SESSIONID=0123456789; domain=.example.com; ...
```

² Browsers face triple threat: <http://news.techworld.com/security/5276/browsers-face-triple-threat/>

This attack vector is similar to the previous one (cross-subdomain cooking) but it is only exploitable if the victim user web browser is vulnerable to cross-site cooking or implements very relaxed third-party cookie controls.

- **HTTP response splitting:** If the target web application is vulnerable to HTTP response splitting, for example, within a user redirection inside the application when a requested resource is not found, an attacker can take advantage of this vulnerability to inject a new session ID in the form of a cookie (a new HTTP header):

```
The attacker exploits an HTTP response splitting flaw to inject a new session ID (cookie):
https://portal.example.com/unexistent\r\nSet-Cookie:SESSIONID=012345678
9\r\nDummy-Header:

The corresponding poisoned HTTP response from the web application to the victim user looks like:
HTTP/1.1 302 Found
Server: Vulnerable Server 1.0
Location: https://portal.example.com
Set-Cookie:SESSIONID=0123456789
Dummy-Header: /login
...
```

- **Cross-Site Scripting (XSS):** If the target web application is vulnerable to Cross-Site Scripting (XSS), like in a single GET parameter of any web resource, an attacker can take advantage of this vulnerability to fix the session ID in the form of a cookie by using JavaScript code:

```
https://portal.example.com/search?q=<script>document.cookie="SESSIONID=
0123456789;%20path=/%20domain=.example.com";</script>
```

Both reflective and persistent XSS vulnerabilities can be used to inject the malicious JavaScript containing the cookie into the victim user web browser and fix the session ID.

- **SQL injection:** If the session management implementation makes use of a back-end relational database, and that database is somehow exposed from within the target web application, an SQL injection vulnerability would (potentially) allow an attacker to modify the session's database and fix new session IDs for victim users. This attack vector is very dependent on the specific session management implementation used. Although through the SQL injection vulnerability the attacker might have a more privileged access to the web application, it could be used to launch very subtle session fixation attacks to impersonate sensitive users within the application.

The attack vector to use will depend on the session ID exchange mechanisms supported by the target web application, the level of privilege the attacker has on the victim network, and the existence of other vulnerabilities on the target environment. Multiple attack methods can be combined into the same attack, and in particular, multiple cookie attributes. Once the ("malicious") session ID has been fixed, the attacker simply needs to wait for the victim user to log in to the target web application.

Session fixation attacks provide some benefits for the attacker compared to other session hijacking attacks based on obtaining a valid session ID, such as session ID disclosure, interception, prediction, or brute-force attacks. By fixating a session ID into a victim user before the authentication takes place (pre-authentication), the attack window is bigger, as the session ID known by the attacker can remain fixed (or be “refixed” subsequently) in the victim web browser for long periods of time until the authentication takes place. The other session management attacks require the acquisition of a session ID that is still valid, that is, belongs to a currently active session. Thus, session fixation attacks can be initially launched plenty of time in advance (before the victim user logs in), and be exploited post-authentication while the user session is still active; the other session management attacks must be launched and exploited post-authentication while the user session is still active.

Furthermore, most session-based attacks have a limited lifetime, especially when the web application sets an absolute timeout for any valid session. Once the session is terminated by the web application, the attack must be launched again from scratch targeting a different session (for the same or for a different user). Session fixation attacks might extend the attack lifetime by setting session IDs that remain longer in the victim user environment, such as setting (or fixing) persistent cookies (instead of session cookies) that expire in, for example, 10 years. The malicious cookie set by the attacker simply needs to include a standard “expires” cookie attribute to become persistent (see the interception attack vector above). Even when the web application terminates the session being attacked, the session ID known by the attacker will remain on the victim web browser, and the same user can be victim of the same attack through the same session ID the next time she authenticates again on the web application.

CASE STUDIES

This section provides an in-depth analysis of three case studies from real-world penetration tests performed during 2009 and 2010 on separate target environments. Three different session fixation vulnerabilities on an open-source CMS project, a commercial web application server, and a commercial enterprise software business platform were discovered and are dissected, analyzing how they were found and exploited, their real impact, and how to avoid similar vulnerabilities.

Case Study 1: Open-Source Joomla! CMS

Summary: Session fixation vulnerability in Joomla!³, an open-source Content Management System (CMS), affecting all 1.5.x versions prior to and including 1.5.15, and publicly released as Joomla! vulnerability number 20100423 [3] in April 2010.

Note: Due to a typo in the advisory, “Google” says I’m the first one that found a “sessation fixation” vulnerability in the security industry ;) Based on the official security advisory [3], the vulnerability was also reported to the Joomla! Security Strike Team by another researcher.

Discovery and Exploitation

This vulnerability was discovered at the end of 2009 during a web application pen-test on a target HTTPS-only web application based on Joomla!, a widely used open-source web-based Content Management System (CMS). The Joomla! core session management system was prone to a session-fixation vulnerability, that was released as Taddong security advisory TAD-2010-001 [2] in April 2010.

The target web application had a public section available to unauthenticated users and visitors, and a separate private section only available to registered users. Users need to authenticate against the web application in order to get access to the private section. The target web portal used the Joomla! built-in session management capabilities, before and after authentication. The authentication methods supported by the target web application were client-based digital certificates (through the usage of an electronic National ID card) or username and password.

The Joomla! session identifiers (session IDs) are implemented as HTTP cookies pre-authentication, and use a pair "name=value" format, where both cookie name and value are MD5 hashes. E.g.:

```
Set-Cookie:  
f2345657e5f302e02d18922ba903a4ef=74d0a95cfd16feb8a9678510686ba63; path=/  

```

“Security through obscurity” is involved in the Joomla! session management capabilities through the usage of MD5 values for the session ID name and value. However, an attacker does not need to fully

³ <http://www.joomla.org>

analyze and understand the meaning and purpose of the session ID format, , but the underlying technology, to launch certain session management attacks.

The availability of the Joomla! project source code can also be used to understand how the specific session IDs are created and assigned. Although this session fixation vulnerability was discovered using manual methods during a blackbox pen-test, the availability of the source code in open-source web applications makes possible to find the same vulnerability through the analysis of the associated (or an approximate version of the) code (e.g. PHP), turning the security assessment into a whitebox pen-test.

Although Joomla! had session management access controls already in place, the implementation didn't work as expected, and the framework was vulnerable to session fixation.

Impact

An attacker can exploit this vulnerability to hijack a user's session and gain unauthorized access to the protected portions (those requiring authentication and authorization) of the affected web application.

All web applications based on Joomla!, from version 1.5 to 1.5.15, and using the Joomla! built-in session management capabilities are affected. Open-source web-based Content Management Systems (CMS), such as Joomla!, Drupal, OpenCMS, or Plone, are commonly associated to non-profit organizations, academic institutions, and non-business related web sites and applications. However, nowadays this kind of open-source web application frameworks are used by commercial companies and governments (both internally and externally) to support their business critical web applications, in a stand-alone setup, integrated with proprietary source-code customizations, or as part of wider web application frameworks. Therefore, the impact of security vulnerabilities in open-source CMS systems can be very relevant depending on the criticality of web applications and environments running on top of them.

Vulnerability Disclosure Timeline

This vulnerability, and the lessons learned while dealing with the vulnerability notification, management tasks, and final disclosure, influenced (together with other findings) the publication of a related vulnerability reporting blog post, called "The Seven Deadly Sins of Security Vulnerability Reporting" [5]. To sum up, there are multiple aspects open for improvement in organizations and companies when they are notified and have to act upon security vulnerabilities in their products or production environments.

Although the official Joomla! security advisory states the vulnerability was reported on March 25, 2010, it was tested on Joomla! 1.5.14 (the latest version available at that time) and reported in November 2009.

Protections

Web applications based on Joomla! must upgrade to the latest Joomla! version (1.5.16 or later).

Case Study 2: Web Application Server

Note: Due to the timing between the moment this white paper was written, and the time the “Case Study 2” vulnerability is being reevaluated and ratified, the specific vendor name, product name and version affected by this session fixation vulnerability is not reflected in this document on purpose. The plan is to disclose the vendor, product, and version details for this case study during the associated Black Hat presentation, mid March 2011, if the vulnerability ratification process is completed before that time.

Summary: Session fixation vulnerability on a widespread commercial web application server due to misbehavior in the way a specific web application (running on top of this web application server) manages access to HTTPS private resources through HTTP.

Due to the fact the vulnerability has not been fully ratified at the time of this writing, three separate scenarios with a very different impact are possible (see the associated “Impact” section too):

1. [*High impact*] This vulnerability might affect the commercial web application server itself, and as a result, most (if not all) web applications based on this widely used web application framework would be affected.
2. [*Mid impact*] This vulnerability can be introduced due to a misconfiguration of the web application settings within the commercial web application server. The number of affected environments depends on how easy is for applications to introduce or reproduce the vulnerable settings on this web application server. As a result, there could be a significant number of vulnerable web applications based on this widely used commercial framework.
3. [*Low impact*] This vulnerability only affects the specific target web application it was discovered in. It has been so badly misconfigured that the chance of other environments implementing a similar configuration is negligible. The commercial web application server is not impacted by the vulnerability, so as a result, only this target web applications is vulnerable.

Discovery and Exploitation

This vulnerability was discovered during a web application pen-test of a complex and large target web environment. The organization in charge of the web application recently redesigned the web infrastructure, application, and associated web technologies, in order to accommodate new business-related services and increase the web application capacity to manage thousands/millions of potential users. To evaluate the security of the new platform a security assessment was requested.

The target web application has a public section available to unauthenticated users and visitors, and a separate private section only available to registered users. Users need to authenticate against the web application in order to get access to the private section, including their personal profile. The target web application (www.example.com) provides a standard Java-based session cookie once a client establishes an HTTP connection to it, named “JSESSIONID”. The cookie has very permissive “domain” and “path” attributes assigned to it, as the cookie is sent to any path of any web application in the target domain. The cookie is transmitted for both HTTP and HTTPS connections against the target web application:

```
Set-Cookie: JSESSIONID=Fz5fMSfQJFTNxpHdp6Kdy63S4f6snLM5tLjJyQ592mJTvnWmqMql;  
domain=.example.com; path=/
```

Most web applications nowadays set pre-authentication cookies to keep track of anonymous user sessions for different purposes, such as to collect usage statistics or to set certain properties for users, like the user's preferred language in multi-language web applications.

Although the cookie length is bigger than 50 characters and the value is random enough (both properties help to mitigate session guessing and brute-force attacks), these properties do not protect against session fixation attacks.

The supported authentication methods are username and password, and client-based digital certificates, standalone or stored in an electronic National ID card. The authentication (or login) page is available under the following URL: <https://portal.example.com/private/miPortal>.

If the client directly accesses the login page without a cookie, the web application sets the same cookie in the response as in the main webpage, "JSESSIONID", using the same attributes. The authentication process makes use of a POST request, and sends the user credentials over the HTTPS encrypted tunnel.

If the authentication succeeds, the web application response contains a new protected cookie, only available to authenticated users, named "AUTH_JSESSIONID". This new cookie is transmitted for any path on "portal.example.com" but only via HTTPS connections to the target web application due to the existence of the "secure" cookie attribute:

```
Set-Cookie: AUTH_JSESSIONID=G41Dl07gF5s1KJ8AvQ14; path=/; secure
```

If the user had a previously set AUTH_JSESSIONID pre-authentication, the web application renews the cookie value, protecting the environment against session fixation attacks.

The JSESSIONID cookie is contained on every web client request against <http://www.example.com> as well as against any other web application in the target domain, such as <http://portal.example.com> or <https://portal.example.com>. However, the AUTH_JSESSIONID cookie will only be transmitted on requests against <https://portal.example.com> (HTTPS encrypted connections).

This behavior reflects a very common practice in lots of modern web applications, where the addition of a renewed and protected (or "secure") cookie is used to mitigate session ID disclosure and session fixation attacks. In theory, both cookies are supposed to be used for session tracking purposes on authenticated users, and because the value of the protected cookie always changes post-authentication, the whole cookie set changes and, from a session fixation perspective, it is similar to having a single cookie whose value gets renewed after an application privilege level change (e.g. authentication).

All links from www.example.com pointing to portal.example.com make use of the HTTPS protocol, but the environment also allows to access portal.example.com through HTTP.

Although common sense makes anybody think that both cookies, JSESSIONID and AUTH_JSESSIONID, are going to be fully verified when a protected resource within portal.example.com that requires authentication (named “authenticated resource” for simplicity) is requested, this is not exactly the case, and in particular, when the web application is accessed through HTTP (versus HTTPS).

An HTTPS request against an authenticated resource requires both cookies. If the request contains both cookies and they are both valid, the access is allowed. If the JSESSIONID cookie in the request is missing or is not valid (for example, the associated value has expired), then the user is redirected to the authentication page and a new value for JSESSIONID is provided. This is the expected behavior for a secure web application. However, if the AUTH_JSESSIONID cookie in the request is missing or is not valid (for example, the associated value has expired), then the application returns a 401 HTTP response (“Authorization Required”) using HTTP Basic authentication. This is a non-expected behavior, especially because the web application does not make use of HTTP Basic or Digest authentication, but custom web form authentication.

In this scenario, the session fixation vulnerability is associated to the HTTP access. Once a user is authenticated into the web application, any valid request going through HTTPS requires the exchange of both cookies. However, any valid request going through HTTP will only include, and only requires, the JSESSIONID cookie. The AUTH_JSESSIONID cookie is “secure”, and therefore, will never be sent over non-HTTPS connections. The key point is that HTTP request only require the JSESSIONID to associate the request to the authenticated session. Thus, an attacker can access all the available authenticated resources through HTTP by simply using the JSESSIONID cookie value of the authenticated victim user, currently bound to the user’s session and the non-required AUTH_JSESSION value.

Because the JSESSIONID never changes before and after authentication, this misbehavior in the HTTP and HTTPS access opens the door to session fixation attacks. The attacker simply needs to get a valid JSESSIONID value from the web application, by requesting any resource as an anonymous user, and fix the value into the victim user. Once the victim user has been authenticated into the web application, he will get a valid AUTH_JSESSIONID cookie, but her JSESSIONID value will not change, keeping the attacker’s value. At this point, the attacker can perform any request for authenticated resources over HTTP using the previously fixed JSESSIONID value and get direct access as the authenticated victim user, without never been asked for the AUTH_JSESSIONID cookie (because the HTTPS protocol is not involved in the attack).

The target web application this vulnerability was discovered in offered all the authenticated resources through both HTTPS and HTTP, thus, an attacker could get full access to the web application as the victim user.

The vulnerability is two-fold. On the one hand, the web application allows access through HTTP to private resources that should only be available through HTTPS. On the other hand, solely the JSESSIONID

cookie, whose value never changes, identifies an authenticated session. Instead, the AUTH_JSESSIONID (alone or in combination with JSESSIONID) must be mandatory to identify a valid authenticated session.

Note: This HTTP vs. HTTPS misbehavior also makes the target web application vulnerable to other but simpler session management attacks, Because the JSESSIONID is disclosed on every HTTP (unencrypted) request sent by the victim user, the attacker simply needs to get access to the JSESSIONID value to be able to access the web application via HTTP as the victim user.

Impact

Even in the case this misbehavior is not finally confirmed as a new vulnerability (0-day) affecting the commercial web application server and, it is somehow, merely associated to the specific target web application where it was discovered in, it demonstrates how a subtle misconfiguration can make the whole web application vulnerable to serious session management attacks, including session fixation.

This is an interesting real-world case study of inconsistency flaws in the usage of HTTP and HTTPS regarding session management in web applications, and provides interesting lessons to learn. Pen-testers should verify the behavior of web applications when HTTPS protected resources, considered as private from a business logic perspective, are accessed through HTTP, and in particular, in regards to session management weaknesses and cookie verification and enforcement on the server side. Potentially, similar vulnerabilities are still possible if the application does not verify the existence of both cookies for authenticated sessions when HTTPS is the only available transport mechanism on the target web application.

Vulnerability Disclosure Timeline

In early December 2010, this candidate vulnerability was reported to the commercial web application server vendor. After a quick analysis and discussion, around mid December, and with very limited information available from the affected web application, it was assumed that the vulnerability was specific to the target environment it was discovered in (until additional details were available).

In mid February 2011 the vulnerability analysis was taken up again once it was possible to get full access to the configuration details from the target web application. At the end of February 2011 (time of this writing), all the required information from the target web application has been collected and it is being analyzed to confirm the vulnerability and identify the reason for it and vulnerable software component.

Protections

Until this vulnerability is not fully ratified and the specific vulnerable configuration or software component identified, it is not possible to provide accurate and in-depth countermeasures.

As a general advice, web applications must clearly differentiate and establish a real separation between their public environment, accessible through HTTP and plainly serving public web contents to anonymous users and visitors, and their private environment, (in theory) only accessible through HTTPS and available just to authorized users that have been previously registered and are forced to authenticate into the web application to get access to sensitive data. Still today hundreds of web

environments mix both worlds under the same host, such as <http://www.example.com> (public) and <https://www.example.com> (private). Instead, it is highly recommended to use www.example.com for the public environment and a different host (desirably running on a separate IP addresses and even server), in the same or (preferably) a different domain, for the private environment, like secure.example.com. The “domain” and “path” cookie attributes open the door to misconfigurations and session management vulnerabilities in the same host or domain. The goal of this separation is to implement completely independent session management infrastructures for both environments and, failing to do so, might end up in vulnerable web architectures like the one described.

Case Study 3: World's Leader in Business Software

Summary: Session fixation vulnerability in the SAP J2EE Engine affecting the core SAP NetWeaver platform, thus, multiple SAP software components (J2EE Engine API, SAP J2EE Engine Core, and Core Server Components) and versions (v6.40 - 7.20; see details on the official SAP Security Note). This vulnerability was publicly announced as SAP Security Note 1310561 [6] in December 2010.

Note: After the Black Hat presentation the author plans to release the associated Taddong security advisory (TAD-2011-xxx) on Taddong’s Security Blog, that will be available at blog.taddong.com.

Discovery and Exploitation

The vulnerability was found during an extensive pen-test (including network, web application, and wireless activities) of a target organization in the Spring/Summer of 2009. Some of the target servers included the Intranet website and the systems supporting the SAP environment for the target company, deployed to support most of the critical business processes and activities, such as financials, human resources, business warehouse, or budget management.

The company staff had multiple internal and external resources available through the company Intranet website, set as the default home page on the employees’ web browsers: <http://intranet.example.com>. The Intranet website is accessed through HTTP (unencrypted web sessions), except for the authentication process, that makes use of HTTPS and NTLM (or HTTP Integrated) web-based authentication using the user credentials in the corporate Windows domain.

One of the links available on this Intranet website, named the “Employee Portal”, is a reference to the SAP NetWeaver Portal web interface (referred as SAP Portal henceforth) available at <http://portal.example.com>. Any web request to the SAP Portal is automatically redirected through an HTTP 307 response (“Temporary Redirect”) to the HTTPS encrypted version of the SAP Portal, and in particular, to <https://portal.example.com/irj/portal> (the default URL for the SAP Portal – see note below). Employees normally authenticate against the SAP Portal through their employee id smart card (whose smart chip contains the employee identity in the form of client-based digital certificates) or, during exceptional emergency situations, through username and password. This scenario is very common in Intranet websites for lots of modern corporate environments.

Note: IRJ (iView Runtime Java) is the SAP Enterprise Portal running under SAP NetWeaver Java Application Server. URL: <http://wiki.sdn.sap.com/wiki/pages/viewpage.action?pageId=69861795>.

This quick and simple, but very common, web redirection from HTTP to HTTPS is all an attacker or pen-tester (referred as attacker henceforth) needs to intercept the network traffic and get access to the initial HTTP user request in clear text, which might contain the user cookies to access the SAP Portal:

```
Cookie: saplb_*= (J2EE01234567) 01234567; PortalAlias=portal;  
JSESSIONID= (J2EE01234567) ID0123456789DB01234567890123456789End
```

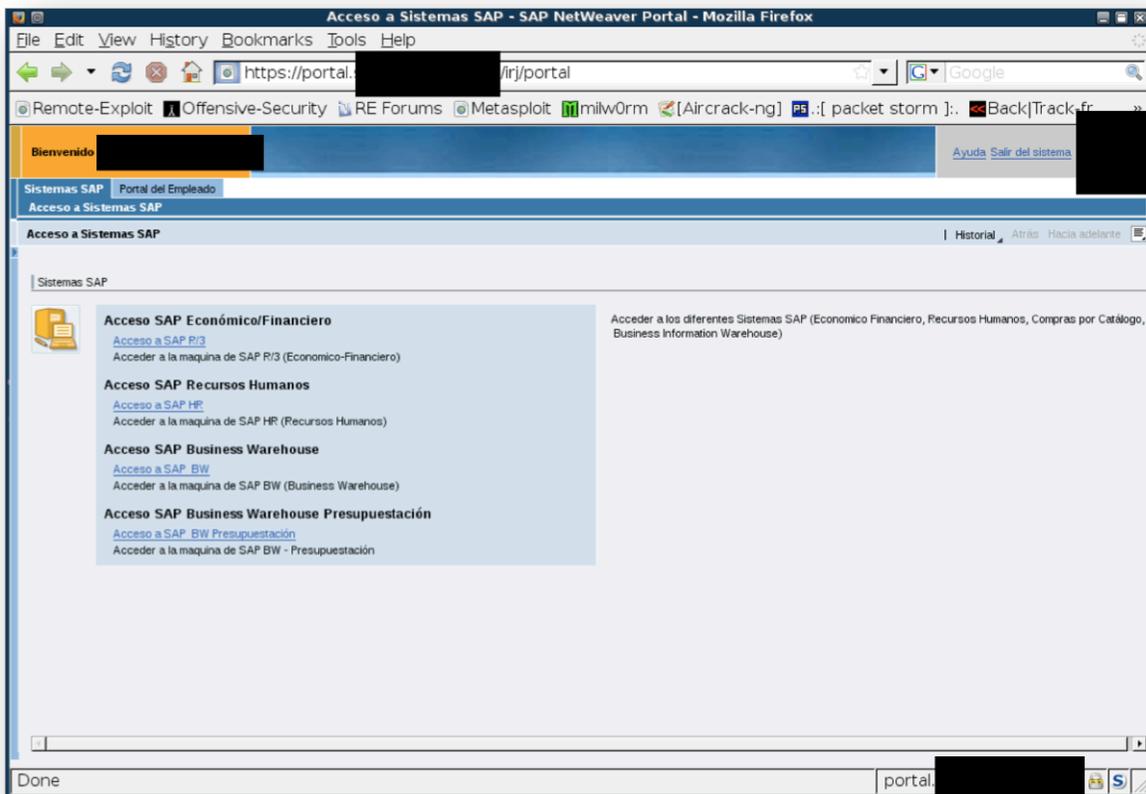
Due to the fact the SAP Portal cookies (obtained during any previous web request to the SAP Portal) were not defined as “secure”, they were transmitted over HTTP, thus exposed to anyone eavesdropping the network traffic. This specific pen-test and attack was performed on a switched environment, therefore the victim user traffic was redirected to the attacker’s system using ARP poisoning techniques. Even if the link from the Intranet website to the SAP Portal were HTTPS-based, simply because the cookies are not marked as “secure”, an attacker can use traffic interception and manipulation techniques to convert all HTTPS links to HTTP on the victim side (e.g. using tools like SSLstrip), and force the victim web browser to disclose the unprotected cookies.

At this point, the attacker has collected all the pre-authentication cookies of the victim user on the SAP Portal. However, because the victim user has not authenticated yet, she has just been redirected to the SAP Portal authentication web page, the attacker does not have any access to the SAP Portal (yet). Once the victim user authenticates into the SAP Portal, using the employee id smart card (or username and password), and because the SAP Portal is vulnerable to session fixation, the session ID or cookies remain unchanged post-authentication. Now, the same set of cookies available pre-authentication is associated to an authenticated session and the attacker can get access to the SAP Portal as the victim user.

In fact, in the pure session fixation attack launched during the pen-test described, the attacker did not need to gather the victim user cookies in advance. Instead, the attacker initially got a valid set of SAP Portal cookies (pre-authentication), later on these cookies were “fixed” into the victim user browser, and when the victim logged in to the SAP Portal, the attacker got access to the victim session. The method used to exploit the session fixation vulnerability in this scenario, that is, to “fix” the cookies into the victim user browser, was through a man-in-the-middle (MitM) attack over the victim user traffic. Once the traffic was intercepted, the attacker’s cookies were injected (as cookie headers) inside the HTTP headers of the initial HTTP 307 response associated to the SAP Portal redirection to HTTPS. The victim user requested access to the SAP Portal from the main Intranet website, but instead of getting a legitimate set of cookies from the SAP Portal, she got the set of cookies injected by the attacker (and replacing the legitimate cookies) in the first HTTP response received.

The attacker simply made use of the specific set of SAP Portal cookies previously “fixed” on the victim user to complete the session fixation attack and log in to the SAP Portal (see screenshot below). In this specific environment, the following set of cookies was required to get access to the SAP Portal:

```
Cookie: saplb_*=(J2EE01234567)01234567; PortalAlias=portal;  
JSESSIONID=(J2EE01234567)ID0123456789DB01234567890123456789End;  
MYSAPSSO2=AjEx...(very long string)...ewCw%3D; SAPWP_active=1
```



The specific SAP NetWeaver Portal version where the vulnerability was discovered was 6.40 (6.4.200607310245), identified by the following strings and HTTP headers extracted from the target environment web traffic:

```
Server: SAP Web Application Server (ICM)  
Server: SAP J2EE Engine/6.40  
PortalVersion:"6.4.200607310245"
```

Once the attack succeeds, the attacker is authenticated in the SAP Portal as the victim user she fixed the session ID to. Through the aforementioned traffic redirection techniques, the attacker can selectively target any SAP Portal user and hijack her session.

By fully impersonating carefully chosen victim identities within a core business application, the attacker can take advantage of the victim user roles and privilege levels to get access to any business critical data and perform any action on behalf of that user. This scenario opens the door to targeted, criminal (blackmail, fraud, extortion, sabotage, theft and abuse), and industrial and corporate espionage attacks.

Impact

This session fixation vulnerability can be exploited to hijack any SAP user session and get unauthorized access to the SAP Portal as well as to other SAP applications and software components, such as the SAP NetWeaver administrative web interface and other non-Portal SAP related modules. As a result, by selectively targeting key business users, an attacker can get full control of the SAP platform and thus, the core business of the target company.

Note: The exploitation of the session fixation flaw in other SAP modules might require to replicate extra cookies not showed previously, such as “SAPPORTALSDB0” or “DSMKeepAliveStamp” (as for the aforementioned SAP NetWeaver administrative web interface).

The real-world impact of this vulnerability can only be thoroughly evaluated by analyzing who could be affected by it worldwide. SAP AG (referred as SAP henceforth) is declared to be the world’s leader in enterprise business software (as of 2009), with significant numbers⁴: +12 billion Euros in revenues (2010), more than 53,000 employees (end of 2010), and more than 12 million daily users from more than 109,000 customers in 120 countries, from Fortune 500 companies to small and medium businesses (SMBs). SAP offers over 25 industry-specific business solutions for large enterprises (and hundreds of micro-vertical solutions for SMBs) deployed in more than 140,000 installations worldwide, directly or via their more than 2,400 certified partners.

Taking into consideration this session fixation vulnerability resides in SAP NetWeaver, SAP's integrated technology platform and the technical foundation for all SAP applications, software suites, and solutions, and that multiple platform versions were found vulnerable, it is realistic to state that this vulnerability directly impacts the software-based and web services-based business activities of thousands of organizations and businesses around the world.

Session fixation vulnerabilities may have a significant impact on the design of web applications, and therefore, their fix might require a full assessment and in-depth analysis of the overall web application core architecture, and, in the worse case scenario and due to the complexity of modern web environments, a redesign of the web solution. Additionally, this detailed analysis can collaterally find that multiple product versions are impacted, as well as other applications, modules, third-party

⁴ SAP business in brief: <http://www.sap.com/about/investor/inbrief/index.epx>

software, and products running on top of the vulnerable web infrastructure, as it is the case with the SAP NetWeaver core platform.

From a functional perspective, a minor change or fix in the session tracking mechanisms of the underlying core platform, such as the SAP infrastructure or a web application server (see “Case Study 2”), can break other software components or web application modules that make use of the session IDs. On the one hand, legitimate software components sometimes make inappropriate use of the session IDs, taking advantage of session management vulnerabilities (or “extra features”) to get access to the current sessions for troubleshooting or administrative purposes. On the other hand, there are software components that, when they receive a session ID, make use of (or create a new session with) it, and do not have the capabilities to identify if it is valid or not, and thus, cannot discern session fixation attacks.

The software maintenance or support strategy also has a relevant impact on vulnerabilities. Since November 2008, SAP implements a 7-2 maintenance strategy (7 years of mainstream maintenance, plus 2 years of extended maintenance) for new releases of the core applications of the SAP Business Suite. This nine years’ maintenance horizon means that legacy software versions, potentially vulnerable to the described session fixation flaw, will be the main core of production environments for several years. Therefore, the associated fix must consider both new and legacy scenarios and provide a solution for all currently supported versions.

This case study (as well as the two previous cases) exemplifies how session fixation attacks can be used to bypass even the most advanced web authentication mechanisms, like smart cards or standalone client-based digital certificates. Once the user is authenticated into the vulnerable web application, the session ID is equivalent to the user credentials. The only difference is the session ID is not valid once the session has expired, while the credentials can be reused for future unauthorized access.

The complexity of modern web architectures and the broad impact scope of vulnerabilities in core web components might require (in some cases) several months to get them fixed and ready for production; meanwhile environments remain vulnerable.

Java Servlet Specification

Session management vulnerabilities and the associated fixes might be influenced by their relationship with too broad session management definitions and requirements on industry specifications, such as version 2.5 of the Java Servlet Specification.

Some of the solutions proposed and investigated to solve this session fixation vulnerability involved renaming the default session tracking cookie (JSESSIONID) or the inclusion of a second protected cookie (e.g. JSESSIONMARKID). This new cookie should only be sent over HTTPS connections, through the use of the “secure” flag, and will complement the default session cookie: the combination of both cookies, the default one and the protected one, would conform the “unique” session ID. Although this approach has been implemented by other widely adopted web applications in the industry, it can break the Java (J2EE) Servlet Specification [8], and as a result, affect third-party applications and components integrated within SAP Portal that follow this standard.

As with most specifications, the Java Servlet Specification is too generic and it is open to the developer interpretation. There are two references to session tracking mechanisms within two sections of the Java Servlet Specification (version 2.5 MR2 [8]) that have a significant impact on the security of the session management mechanisms of web applications following this standard, and specifically, of session tracking disclosure and session fixation attacks:

- SRV .7.1.1 - Cookies (page 59):

“Session tracking through HTTP cookies is the most used session tracking mechanism and is required to be supported by all servlet containers. The container sends a cookie to the client. The client will then return the cookie on each subsequent request to the server, unambiguously associating the request with a session. The name of the session tracking cookie must be JSESSIONID”

The specification obliges to name the session tracking cookie (or session ID) “JSESSIONID”, disclosing the technology being used (Java). Changing the cookie name would be a violation of the specification. It also denotes that, once the client received a cookie, it must be sent subsequently to stay associated to that same session. Depending on the interpretation, it could be assumed that this cookie value cannot be changed for the rest of the session (even after application privilege level changes) and, therefore, help to promote session fixation attacks. If third-party applications or components were developed under this assumption, any change to the cookie value could potentially break their functionality.

- SRV.7.1.3 - URL Rewriting (page 60):

*“URL rewriting is the lowest common denominator of session tracking. When a client will not accept a cookie, URL rewriting may be used by the server as the basis for session tracking. URL rewriting involves adding data, a session ID, to the URL path that is interpreted by the container to associate the request with a session. The session ID must be encoded as a path parameter in the URL string. The name of the parameter must be jsessionid. Here is an example of a URL containing encoded path information:
<http://www.myserver.com/catalog/index.html;jsessionid=1234>”*

Although the rewriting of session IDs from cookies to URL parameters was a common method used in the past by web applications to support web clients without cookie support, such as old mobile devices, most (if not all) client devices today support cookies. Including or accepting session IDs in the URL eases the disclosure of the id and facilitates session fixation attacks. This section also defines that the name of the session tracking cookie must be “jsessionid”.

Note: Although the title inside the document says “Java Servlet Specification Version 2.5 MR6” all other references reflect Maintenance Release 2 (MR2) as the latest version available.

Section 7 of version 3.0 of the Java J2EE Servlet specification [13] focuses on sessions. Although several improvements have been added to it, allowing the customization of the session tracking cookie name, reflecting the availability of the “httpOnly” cookie attribute, and advising against the usage of URL rewriting due to its security implications, still the usage of the “secure” cookie attribute (and guidance on other cookie attributes) or specific advice to avoid session fixation attacks are missing.

Vulnerability Disclosure Timeline

Note: This section includes a brief summarized timeline of the events occurred from the discovery of the vulnerability until full public disclosure.

The vulnerability was reported to SAP AG in early July 2009. During that month the vulnerability was ratified and the currently available related protection mechanisms in SAP Portal were discussed. The first deadline for the release of a fix was set in two months (a best case scenario).

In mid September 2009, the difficulties to quickly and easily fix the vulnerability and the discovery of a related stability issue started to arise. A new deadline was set for January 2010 (6 months after the initial notification). The various responsible disclosure alternatives and the real impact were analyzed throughout the process. In November 2009, a status update still estimated January/February 2010 as the release timeframe, and an initial technical solution was being widely tested. Simultaneously, the vulnerability disclosure plans started to get debated. SAP disclosure model makes use of the SAP Service Marketplace (SMP), which is accessible to customers and partners only. At that time SAP was in the course of re-evaluating its product security response process, security notes, vulnerability disclosure practices, and the relationship with external security researchers [7].

At the end of January 2010, SAP confirmed that the secure solution for all business applications was still not available, and conversations got postponed till February. Then, the issue was escalated internally to conclude that several additional months would be required to fix the vulnerability for all affected releases. The real impact and the necessity of a working fix were discussed, trying to minimize the amount of time without a valid solution. In March 2010, SAP confirmed it did not expect to have fixes for all releases and scenarios before September 2010 (15 months after the initial notification); committing to a specific date was difficult, especially due to unresolved issues on legacy releases, but the option of providing partial fixes for some customers was evaluated.

In August 2010, the status of the fix was evaluated and finally, a meeting date to collaborate on the disclosure details was set on November 2010. Due to the criticality of this vulnerability on SAP environments worldwide, and after all the reasoning and thoughts, the extensive testing required, the evaluation of affected entities and businesses, and the controversies of a vulnerability fix that has taken a long time to be released, in December 2010, SAP published the fix as SAP Security Note 1310561 [6] to its customers and partners.

The vulnerability was notified to SAP in July 2009, and due to its core nature, impact, and broad scope, it has taken one and a half years to have a fix ready for release and deployment into production environments. On the one hand, one would expect to get fixes for critical vulnerabilities in key industry products in a 3 to 6-month period at the most. On the other hand, when you look at the big picture, you realize that for large companies like SAP, with maintenance strategies supporting a large amount of

legacy releases, providing fixes is something that cannot be easily done; providing fixes for all releases under maintenance⁵ and have these fixes properly tested could take a significant amount of time.

Following SAP public disclosure recommendations, “...we suggest respecting an implementation time of three months” [7], in March 2011 (more than 20 months after the initial notification, and 3 months after the official release), the vulnerability is going to be released during the Black Hat Europe 2011 conference in Barcelona, Spain.

During the period between the discovery of the vulnerability and the moment the fix was available, one of the mitigation techniques applied was to keep the vulnerability details private. However, as in “Case Study 1” (Joomla!), there is a significant chance of other security researchers finding the same flaw. Depending on the motivations of these researchers, the vulnerability could be reported to the vendor or be used in a fraudulent way to exploit critical systems and obtain economic benefit. This is always a considerable risk that must be evaluated when deciding on the timing to get a working fix ready and the different vulnerability disclosure alternatives.

In 2010, SAP started a new security program to highlight security issues from other software fixes, and to periodically release security related updates to its customers the second Tuesday of each month, aligning the release date (or Monthly Patch Day, introduced by SAP in September 2010) with other common security industry practices. As part of this new customer security program, SAP started in January 2010 to publicly credit researchers on vulnerability discovery and SAP Security Notes⁶.

Protections

In order to prevent this session fixation vulnerability, the recommended fix is to apply the solution described in the SAP Security Note associated to this vulnerability [6]. The solution suggests to enable the “SessionIdRegenerationEnabled” property of the Web Container, which involves HTTP sessions to be identified by two cookies, JSESSIONID and JSESSIONMARKID. Although this property is present in earlier SPs of the J2EE Engine, the session fixation protection handling has been significantly improved in the latest version of the J2EE Engine. There are specific SAP scenarios, such as in the case of producer and consumer portal, where extra features, like the “SAP Logon Tickets for Multiple Domains” [10] or the “SecuritySessionIdGracePeriod” property, must be enabled or adjusted.

SAP provides additional guidance to secure web-based sessions and protect the SAP platform against session fixation and management attacks within that security note and other documents, covering from Single Sign-On (SSO) with logon tickets scenarios [10], to additional secure cookie properties [11], or timing issues of parallel HTTP requests [12].

There are additional SAP security features and general best practices that can be enabled to mitigate session fixation (and management) attacks. The following recommendations do not pretend to be an exhaustive list of SAP session security protections.

⁵ Is this a very ambitious goal? What is the right approach to fix vulnerabilities as quickly and reliably as possible? The debate is open!

⁶ <https://service.sap.com/securitynotes> - Registration on the SAP Service Marketplace (SMP) is required.

Any web link or reference to the SAP Portal, or link to any critical web application for that matter, such as the one available on this case Intranet website, should be HTTPS-based (versus HTTP-based). Besides that, HTTP should be disabled on the SAP Portal and HTTPS be the only web transport protocol available. Besides following these two best practices, the SAP Portal cookies must be defined as “secure”. One of the reasons the attack described didn’t require extra actions on the pen-tester side was that the session IDs (or cookies) were not protected against eavesdropping, that is, the setup didn’t force them to be exchanged just over HTTPS channels. In order to increase the security of the environment, the SAP Java Server provides an HTTP service property named “SystemCookiesHTTPSProtection” to set the “secure” attribute in system cookies (JSESSIONID and load balancer cookies, saplb). The property is available in NetWeaver 6.40 as of SP21 but is disabled by default (see details from SAP Note 1019335 below). There is a corresponding SAP Note 1020365 for NetWeaver 7.0 SP14.

Extracted from SAP Note 1019335:

A new property called "SystemCookiesHTTPSProtection" is available in the HTTP Provider Service to offer support for the "secure" attribute for system cookies (JSESSIONID and saplb). By default, this property is disabled. If enabled, during the creation of the JSESSIONID and saplb cookies, a new "secure" attribute will be added.

Although vendors have to be very conservative about the default security settings and backward compatibility scenarios, as (unfortunately) functionality will always have prevalence over security, the organization’s security teams must evaluate and enable specific settings (disabled by default) in order to apply best security practices and protect their production environments.

Another SAP security related feature available is the “SessionIPProtectionEnabled” property of the Web Container Service [9]. When this property is enabled, an HTTP session cannot be accessed from different IP addresses. Only requests from the IP address that started the session are processed. This forces the attacker to get access or share the public (or external) IP address of the victim user in order to be able to hijack her session. However, once again, this property is disabled by default. When this property is set on an environment where there is a proxy server or load balancer in front of the J2EE Engine, it is required to configure the “ClientIpHeaderName” property of the HTTP Provider Service, so that the SAP engine processes the corresponding HTTP header that contains the IP address of the client.

Although security constraints based on client features (such as the client IP address or user-agent string) are not bullet-proof against session management attacks in web applications, they add an extra layer of security that forces the attacker to perform additional steps to launch a successful attack, and have demonstrated to be very effective from a detection perspective. If the attacker has to identify the right IP address or user-agent string from a victim user to hijack her session, potentially, these extra attacker tests and failed connection attempts could be detected.

SESSION FIXATION PROTECTIONS

The recommended protection to avoid session fixation in web applications is to enforce the renewal of the session ID after any privilege level change, being the authentication process the most common case. When a session has already been established and the privilege level of the session changes (e.g. upon successful authentication), the web application cannot trust the previous session ID value and must generate a new one. The web application must also invalidate on the server side the value of the previous session ID that has been replaced by the new value.

Due to the fact most web programming frameworks do not have an intrinsic link between the authentication and session management capabilities, there is a lack of automated features to defend against session fixation vulnerabilities. Thus, the manual protection is exclusively on the web application developer's hands; one of the reasons session fixation is still prevalent in today's web applications.

The built-in session management frameworks available in some of the most common web application development languages already provide some capabilities to mitigate session fixation attacks: Java provides the session "invalidate()" method, PHP supplies the "session_regenerate_id()" function, and ASP.NET has notable deficiencies, as the "regenerateExpiredSessionId" web.config property does not help against session fixation⁷:

Java	PHP
<pre>HttpSession s = request.getSession(); s.invalidate(); s = request.getSession(true);</pre>	<pre>session_start(); session_regenerate_id(true);</pre>

The following set of extra best practices, countermeasures, and practical defenses can also help to mitigate session fixation vulnerabilities. These will help to protect web applications and victim web clients from session fixation and other session management security vulnerabilities, or at least, increase the attacker's prerequisites to launch these attacks. Some of these recommendations only apply to certain web applications while some others simply help to reinforce a better defense-in-depth posture:

- **Limit the session tracking exchange mechanisms accepted by the web application:** The web application should define the (unique) session tracking mechanism accepted, being cookies the preferred method due to their multiple security-related attributes, and enforce that other methods won't be accepted, such as session IDs within URLs, by disabling URL rewriting.
- **Make use of HTTPS for the whole web session:** Nowadays, critical applications must enforce the usage of fully encrypted web sessions through HTTPS (SSL/TLS) for the whole duration of the user session within the web application, not involving HTTP at all. The extremely common initial HTTP to HTTPS redirection used on lots of websites must be removed. Although HTTPS won't protect against most of the session fixation attack vectors discussed, it is critical against traffic interception and manipulation attacks and session ID disclosure.

⁷ Session Attacks and ASP.NET - Part 1: <http://software-security.sans.org/blog/2009/06/14/session-attacks-and-aspnet-part-1/>

- **Make session IDs available only post-authentication:** By generating valid session IDs only after the authentication has been successfully completed, the attacker would need to have access to a legitimate user account in the target web application to collect a valid session ID that can be used in session fixation attacks. For example, multiple web application assign cookies pre-authentication to keep track of anonymous users, and then provide new but separate cookies post-authentication to keep track of authenticated users.
- **Bind the session ID to other user properties:** Web applications can bind the session ID to other user properties at the session creation time, such as the IP address or user-agent of the user, with the purpose of verifying that binding for the whole duration of the session. Although a savvy attacker can manipulate her corresponding user properties to bypass these bindings and access controls, this kind of countermeasure has demonstrated to be very effective from a detection perspective to identify potential session hijacking attacks.
- **Isolate critical web applications on their own domain:** Critical web applications with high security requirements, such as e-banking websites or web-based business platforms, should be deployed on their own independent domain (e.g. www.secure-example.com). The goal is to avoid that other vulnerable systems within the same domain (or subdomains) could be used to launch attacks against the critical web application through domain level cookies. For the very same reason, multiple web applications should not be deployed on the same host.
- **Use very restrictive cookie attributes:** From a security perspective, when session IDs are implemented in the form of cookies, it is highly recommended to make use of all the security features available, as well as define very restrictive attributes for the session ID cookie(s). Cookies must use the “secure” and “httpOnly” attributes, so that they are only sent over HTTPS connections and won’t be accessible from client scripts, respectively. Frequently, the “expires” attribute won’t be set for session IDs, implemented as session cookies. It is highly recommended not to set the “domain” attribute and use instead the default value that associates the cookie just to the specific host it was received from⁸. Finally, the application should specify a very restrictive “path” property associating the session IDs just to the URI where the web application resides, e.g. “path=/private” (particularly if multiple applications reside on the same host).
- **Deploy Web-Application Firewalls (WAF) in front of the vulnerable web application:** There are situations where the vulnerable web application source code cannot be modified, or the modifications will require a redesign of the web application and take a considerable amount of time until completed. In these cases, WAFs can be used as a complementary countermeasure in front of the vulnerable web application to mitigate session fixation attacks, translating from the user perspective the vulnerable session behavior by secure and well-behaved session management practices. For example, the open-source mod_security WAF includes session fixation protections in the default Core Rule Set⁹, and in particular, within the “base_rules/modsecurity_crs_40_generic_attacks.conf” file.

⁸ The common practice of resolving through DNS the domain name, “example.com”, to the same IP address associated to the website, “www.example.com” has serious implications from a cookie perspective; the default “domain” attribute could be assigned to the domain.

⁹ Mod-security Core Rule Set Project: http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project.

CONCLUSIONS AND FUTURE RESEARCH

Session fixation vulnerabilities are extremely prevalent in web applications and web-based products still today, and their real impact on production environments can be devastating from a business perspective. Session fixation allows attackers to bypass even the most advanced authentication mechanisms, such as client-based digital certificates, biometric, or multi-factor authentication.

The complexity of fixing session fixation vulnerabilities affecting core components of critical web applications has been analyzed, specifically, through a real-world case study that has taken about one and a half years to be ready for production. Although in theory session-fixation issues can be easily solved by using a different (or renewing the) session ID after authentication, it turns out that in reality fixing session-fixation (as well as other session management) vulnerabilities might not be a trivial task. This kind of vulnerabilities typically affect multiple modules of the web application, including the authentication and access control sections, as well as potentially other external session related web applications and software components. As a result, there are scenarios where the solution might impact the design of the web application architecture. Therefore, it is highly recommended to plan session management countermeasures and protections during the web application design phase, and test them in the early stages of the web application development, with the goal of trying to avoid the pain and complexity of fixing them afterwards.

While investigating the prevalence of session fixation vulnerabilities in today's web applications, the only recent reference to research on this topic is from a presentation in the OWASP AppSec Research 2010 conference [4], and a few session fixation vulnerabilities released in the last couple of years (this is a non-exhaustive list from the NVD¹⁰; references omitted), in the OAuth Core v1.0 protocol (Apr'09), SquirrelMail (May'09), IBM Tivoli Identity Manager (Jul'09), RT – Request Tracker (Nov'09), WikyBlog (Feb'10), Apache Axis2 (Jun'10), Devise for Rails (Nov'10), TIBCO Collaborative Information Manager (Jan'11), or Adobe ColdFusion (Feb'11 & Aug'09).

Further research regarding session fixation in modern and widely used Internet facing web applications is on the works to evaluate the overall state-of-the-art of this vulnerability on the wild. This research is limited by a significant prerequisite, having a valid user account in the target web applications under study. Nowadays, session fixation discovery and exploitation involve the use of manual techniques on the pen-tester side. The proposed research would be facilitated by the usage of a (semi-automated) session fixation discovery and exploitation tool that could help the developer and information security communities to automate the process of discovering and verifying this kind of vulnerability. The initial research should focus on the web application authentication process, although similar vulnerabilities might be found on any web application privilege level change.

Protecting today's web applications against session fixation vulnerabilities is a key step for the information technology and software industry!

¹⁰ Search (the last 3 years) by "session fixation" on the NIST National Vulnerability Database (NVD): <http://web.nvd.nist.gov/view/vuln/search>.

REFERENCES

- [1] “Session Fixation Vulnerability in Web-based Applications” (version 1.0). Mitja Kolšek. ACROS Security. December 2002. URL: http://www.acrossecurity.com/papers/session_fixation.pdf
- [2] “TAD-2010-001: Session-fixation vulnerability in Joomla! (20100423) “. Raul Siles. Taddong. 10 May 2010. URL: <http://blog.taddong.com/2010/05/session-fixation-vulnerability-in.html>
- [3] “[20100423] - Core - Sessation Fixation”. 23 Apr 2010. URL: <http://developer.joomla.org/security/news/309-20100423-core-sessation-fixation.html>
- [4] “Session Fixation - the Forgotten Vulnerability?”. Michael Schrank and Bastian Braun, University of Passau; Martin Johns, SAP Research. OWASP AppSec Research 2010. June 2010. URL:http://www.owasp.org/images/7/7a/OWASP_AppSec_Research_2010_Session_Fixation_by_Schrank_Braun_Johns_and_Poehls.pdf
- [5] “The Seven Deadly Sins of Security Vulnerability Reporting”. Raul Siles. Taddong. 15 Aug 2010. URL: <http://blog.taddong.com/2010/08/seven-deadly-sins-of-security.html>
- [6] “SAP J2EE Engine Session Fixation Protection”. SAP Security Note 1310561 ¹¹. SAP AG. December 2010. URL: <https://websmp130.sap-ag.de/sap/support/notes/1310561>
- [7] “Acknowledgments to Security Researchers”. SAP AG. (Since) January 2010. URL : <http://www.sdn.sap.com/irj/sdn/index?rid=/webcontent/uuid/c05604f6-4eb3-2d10-eea7-ceb666083a6a>
- [8] “JSR-000154 Java™ Servlet 2.5 Specification (Maintenance Release 2)”. Java Community Process (JCP) Program. Sun Microsystems, Inc. 28 Jul 2007. URL: <http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html>
- [9] “Web Container Service”. SAP Documentation. SAP AG. URL: http://help.sap.com/saphelp_nw70/helpdata/en/ac/2bc55a78e54d60b561140048eaa80c/frameset.htm
- [10] “Configuring Logon Tickets for Multiple Domains” (UME). SAP Documentation. SAP AG. URL: http://help.sap.com/saphelp_nw2004s/helpdata/en/e0/fa984050a13354e10000000a1550b0/frameset.htm
- [11] “Session Security Protection (SAP Library - Administrator's Guide)” (DOC-102853). SAP AG. 22 Oct 2010. URL: <https://cw.sdn.sap.com/cw/docs/DOC-102853>
- [12] “Parallel HTTP Requests and Session Fixation Protection” (DOC-122241). SAP AG. 1 Jun 2010. URL: <https://cw.sdn.sap.com/cw/docs/DOC-122241>
- [13] “JSR-000315 Java™ Servlet 3.0 Specification (Final Release)”. Java Community Process (JCP) Program. Sun Microsystems, Inc. 10 Dec 2009. URL: <http://jcp.org/en/jsr/summary?id=315> ; URL: <http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html>

AUTHOR

Raúl Siles Founder & Senior Security Analyst with Taddong (www.taddong.com)

¹¹ Registration on the SAP Service Marketplace (SMP) is required.