Volume 3, Issue 12, December 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

# Study and Analysis of Session Security on Web Based Applications

**Umesh Kumar Singh, Rakhi Sunhare, Kailash Chandra Phuleriya**
Institute of Computer Science, Vikram University,
Ujjain (M.P.)-456010, India

***Abstract: Uses of web based applications are increasing day by day. Session vulnerabilities are very common in web applications and their real impact on production environments can be devastating from a business perspective. Session allows attackers to bypass even the most advanced authentication mechanism. The most common, and most important piece of information stored in session state is the authenticated identity of the user. For this reason, session state represents a very valuable target for attackers. A large fraction of attacks on web applications involve stealing, replacing, or co-opting a user's session. All the session management solutions have significant drawbacks in one way or another. Even then some common techniques can be applied to provide a secure and reliable session in a general way that can also defend against session-based attacks. We can emphasizes on studying how session can be handled in many different ways for web applications and on analyzing the different security measures taken to session attacks. Finally, we addressing the result of this study and also define the future aspects.***

***Keywords: Security, Web Based Application, Session vulnerabilities, Attacks and Networks.***

## 1. Introduction

All over the world Internet users are using web based systems that follows the client server paradigm. The web browser acts as the client for a web server that provides the service. These web based systems rely on the HTTP protocol for communication, but it is a stateless protocol. Over the last few years, the web has shifted from being a collection of pages containing static information to a dynamic and fully interactive platform. Where the Internet was once used only as an information repository, today it powers complex web applications, developed both to replace programs that used to run locally on a user's computer, and to provide whole new functionality that is possible only on the web. Most web applications handle user authentication via the concept of web sessions. These allow users to use a web application without having to enter their login credentials for every action taken. Unfortunately, web sessions have many security weaknesses. OWASP, a leading organization in the field of web application security, rates 'Broken Authentication and Session Management' as the third most important web application security risk [1].
A problem is that users of a web application have to trust the developer of the application to take the necessary security precautions. The web developer, on the other hand, may consider such precautions to be too difficult or too costly, leaving the users unprotected. Moreover, the web developer might not even know that his web application contains security vulnerabilities. To enable users to protect themselves against session attacks, regardless of the web applications being secure, a client-side tool offering protection against these attacks is needed.
In the rest of this paper is organized in such a way that it helps the reader follow the paper in a graceful manner. In section 1 introduction of session security and web based applications. In section 2 define the comparative study for session security in web application framework. In section 3 describe about web sessions work. Finally, conclude the paper and define the future aspects.

## 2. Comparative Study for Session security in web application framework

In this section, we describe the some widely used frameworks to ensure security against session hijacking and session fixation attacks and compare this framework using some parameters. The list of frameworks is by no means complete, but gives a good comparison of how popular frameworks handle session attacks.

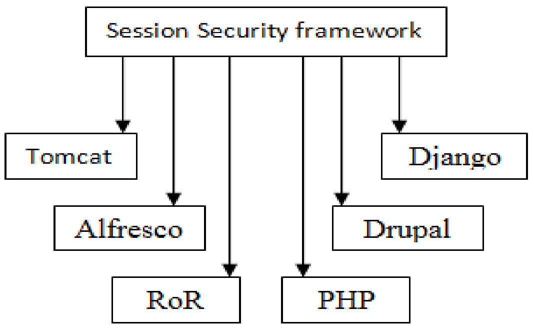


Figure 1: List of frameworks.

**2.1 Tomcat:** Apache Tomcat is a much-used software implementation of the Java Servlet and JavaServer Pages technologies. Renewing of the session identifier on authentication is automatically done since Tomcat [2]. Tomcat uses cookies for session management, but also accepts SIDs that is included in the URL. URL rewriting is also used by default when the client's browser does not support cookies.

**2.2 Alfresco:** Alfresco is a complete content management system that runs on a J2EE application server like Tomcat [3]. It is used by companies like France Air Force, Cisco, Fox and KLM [4]. Requesting more information about session management in Alfresco on their IRC-channel or in the alfresco forums proved fruitless.

**2.3 Ruby on Rails (RoR):** Ruby on Rails is a web framework written in the in 1995 conceived Ruby programming language. It is used in popular web applications like Twitter, Groupon [5]. Renewing the session identifier is not automatically done on each authentication state change in RoR. RoR supports only cookie-based session management by default. If the web developer wants his web application to use URL rewriting instead, he needs to specifically enable this [6].

**2.4 Django:** Django is a web framework built using the Python language. It is used by A. O. ArsTechnica and The Washington Post [7]. Django renews the session identifier automatically when a user logs in. For this, two possible cases are considered; first, if the login request contained a session identifier that corresponds to another logged in user, or if the request did not contain a SID at all, a completely new session is created. Second, if the login request contained a session identifier that is not yet associated with a logged in user, a new SID is created. This SID is then associated with the state corresponding to the old SID [8]. The old SID is removed from the database, so it cannot be used in the future to access the session information.

**2.5 PHP:** PHP is a scripting language used for web development. While PHP is technically not a web framework, we include a discussion of PHP's session module in this section because PHP is estimated to be the server scripting language for over 75% of all websites [9].

**2.6 Drupal**: Drupal is a complete content management system written in PHP. It powers the websites of The Economist, Symantec, and even The White House [10]. Drupal automatically renews the session identifier when the user's authentication state changes [11]. Drupal makes sure that PHP's underlying session mechanism will only accept session IDs via cookies.

A summary of session security in the discussed web application frameworks is given in Table1. As we can see, the popular high-level frameworks in our list provide pretty good protection against session attacks, which reinforces our recommendation to use a high-level framework whenever possible. Using a framework, however, does not relieve the web developer from making sure that all components are configured correctly.

Table 1: Comparison of session security in different web application frameworks

| | **Renews SID on authentication** | **Only accepts cookie-based SIDs** | **secure flag for SSL session cookies** |
|---|---|---|---|
| **Tomcat** | since version | implementable | default |
| **Alfresco** | no | implementable | default |
| **RoR** | configurable | yes | configurable |
| **Django** | Yes | Yes | configurable |
| **PHP** | No | configurable | configurable |
| **Drupal** | Yes | Yes | via module |

## 3 Server-side countermeasures

In this section, we describe some standalone server-side countermeasures to session hijacking and session fixation. These should be deployed separately, and are not part of any web framework.

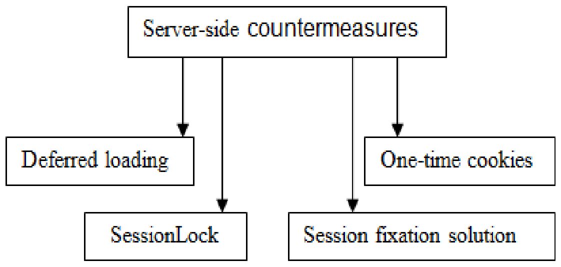


Figure 2: List of Server-side countermeasures

**3.1 Deferred loading:** The reasoning behind deferred loading is that cookies should be separated from the content of a page, to prevent an attacker from using the page content to steal the cookie. This is achieved by setting the cookie on a different sub domain than the web page itself.

**3.2 SessionLock**: SessionLock, proposed by Adida et al. in 2008 [12], tries to solve session hijacking by making session identifiers only available at the client side. For this purpose, fragment identifiers are used. Fragment identifiers are mentioned in the URL specification [13] as a means for making the user's web browser scroll to a certain part of a web page. The fragment identifier is included in the URL as the part that follows the# character. A major advantage of this approach is that, while users use HTTPS to authenticate, a secure connection is not required for subsequent requests.

**3.3 One-time cookies:** According Dacosta et al. [14], a solution to session hijacking is proposed wherein session identifiers are replaced by one-time cookies which are renewed on every request. This is done as follows:

1. The user authentication happens over a secure (HTTPS) connection. In this authentication, a secret seed and a session secret are generated.
2. Subsequent requests are made over an insecure (HTTP) connection. In the i$^{th}$ request, the user's browser attaches the outcome of a hash function applied I times to the secret seed.

**3.4 Session fixation solution:** Johns et al. [15] present a solution to login session fixation attacks in the form of a server-side reverse proxy. This allows web applications that have already been deployed in the past to be patched against session fixation afterwards. The proxy works by introducing a second session identifier, called the 'proxy session identifier' (or PSID), which is tightly secured against login session fixation.

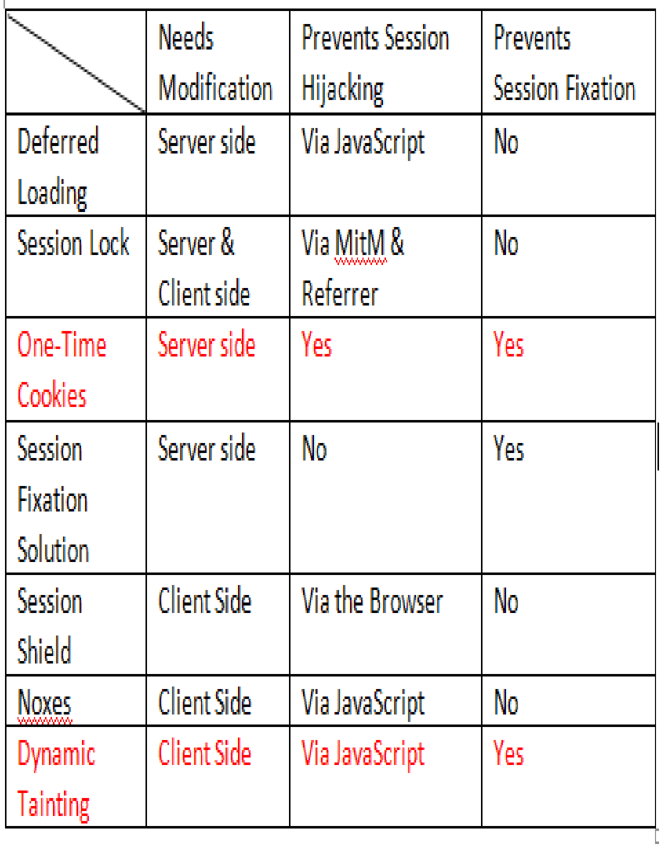
## 4. Client-side countermeasures

In this section, we focus on session hijacking countermeasures at the client side. The advantage that these approaches have over server-side countermeasures is that they protect a user against session hijacking, even when the web application itself is not secure. Because of this, it is the user who can make sure that he is protected against this kind of attack, without having to rely on the web developers of all web applications he uses to implement session hijacking protection. Client side countermeasures to session fixation attacks were, to our knowledge, non-existent at the time of writing [16].

- SessionShield [17]
- Noxes [18]
- Dynamic tainting and static analysis [19]

In Table 2, an overview of the different server-side and client-side session attack countermeasures is given. From this table, we can draw two conclusions: firstly, while lots of work has been done to mitigate session hijacking attacks, the session fixation attack is often overlooked. Secondly, none of the discussed countermeasures handle session fixation attacks entirely at the client side. In fact, we do not know of any countermeasure to session fixation that only requires modifications to be made at the client side.

Table 2: Comparison of different session attack countermeasures



| | Needs Modification | Prevents Session Hijacking | Prevents Session Fixation |
|---|---|---|---|
| Deferred Loading | Server side | Via JavaScript | No |
| Session Lock | Server & Client side | Via MitM & Referrer | No |
| One-Time Cookies | Server side | Yes | Yes |
| Session Fixation Solution | Server side | No | Yes |
| Session Shield | Client Side | Via the Browser | No |
| Noxes | Client Side | Via JavaScript | No |
| Dynamic Tainting | Client Side | Via JavaScript | Yes |

## 5. Conclusion and Future Research Direction:

In this paper, we looked at the security of session management in web applications. We started off by showing the different methods of session handling that exist on the web today. We discussed how session identifiers can be accessed, and which properties a secure session identifier should possess. Since session management is essentially an ad hoc approach, not developed with security in mind, it is prone to exploitation by an attacker. In session attacks several solutions exist both at the end in client and server side: some offer protection against specific attacks, while others try to improve session security in general. Thoroughly examining these solutions showed that, while most of them suffer shortcomings, some good countermeasures to session attacks are available: especially popular web frameworks provide reasonably good protection. A solution offering client-side protection against session fixation attacks is, however, inexistent. As a consequence, users have to rely on the developers of all web applications they are using to implement this protection at the server side. As our study we recommend client-side solution is better to prevent session fixation attacks. The client-side solution provides protection against both session fixation and session hijacking attacks, where an attacker uses un-trusted channels to access or modify the session cookie. It does prevent session fixation and session hijacking via one of these un-trusted channels. This allows a user to protect him against session hijacking and session fixation attacks without requiring the web developer to secure his web application. In future we develop a best security mechanism for prevent a session attack because client-side solution are no provide pure security.

**References:**

[1]. Jeff Williams and Dave Wichers, OWASP, year 2010.
[2] Tomcat Developers. The Valve Component, Available from: http://tomcat.apache.org/tomcat.
[3] Tomcat Developers. Sites, Applications, and Systems that are powered by Tomcat, 2011. Available from: http://wiki.apache.org/tomcat.
[4] Alfresco Software. Alfresco Customer Overview, Year-2011, Available from: http: www.alfresco.com/customers.
[5] Ruby on Rails developers. Real applications live in the wild, year- 2011. Available from: http://rubyonrails.org/applications.
[6] Paul McMahon. Enabling url parameter based sessions in Ruby on Rails, year-2010. Available from: http://www.mobalean.com/blog/2010/04/09/enabling-url-parameterbased- sessions-in-ruby-on-rails.
[7] Django Developers. Django-powered sites, year-2007. Available from: http://code.djangoproject.com/wiki/DjangoPoweredSites?version=540.
[8] Django Developers, Source: http://code.djangoproject.com.
[9] W3Techs, Usage of server-side programming languages for websites, year-2010. Available from: http://w3techs.com/technologies/overview/programming_language/all.
[10] Thomas Schreiber. Session Riding: A Widespread Vulnerability in Today's Web Applications. Technical report, SecureNet GmbH, München, Germany, year-2004.
[11] Drupal Developers. user_authenticate_finalize, 2011. Available from: http://api.drupal.org.
[12] Ben Adida. Sessionlock: securing web sessions against eavesdropping. In Proceeding of the 17th international conference on World Wide Web, pages 517–524, Year-2008.
[13] T. Berners-Lee, R. Fielding, and L. Masinter, Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005.
[14] ItaloDacosta, SaurabhChakradeo, MustaqueAhamad, and Patrick Traynor. One- Time Cookies: Preventing Session Hijacking Attacks with Disposable Credentials. Technical report, Georgia Institute of Technology, year-2011.
[15] Martin Johns, Bastian Braun, Michael Schrank, and Joachim Posegga. Reliable Protection Against Session Fixation Attacks. In SAC '11 Proceedings of the2011 ACM Symposium on Applied Computing, pages 1531–1537, New York, New York, USA, 2011.
[16] Bram Bonné, Philippe De Ryck, Nick Nikiforakis, LievenDesmet, and Frank Piessens. Client-Side Protection against Session Fixation Attacks, Oakland, CA, 2011.
[17] Mike TerLouw, Jin Soon Lim, and V.N. Venkatakrishnan, Extensible Web Browser Security, In 4th International Conference on Detection of Intrusions, Malware, and Vulnerability Assessment (DIMVA), Lucerne, Switzerland, 2007.
[18] EnginKirda, Christopher Kruegel, Giovanni Vigna, and NenadJovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In Proceedings of the 2006 ACM symposium on applied computing, pages 330–337. Year 2006.
[19] Philipp Vogt, Florian Nentwich, NenadJovanovic, EnginKirda, Christopher Kruegel, and Giovanni Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In Proceeding of the Network and DistributedSystem Security Symposium (NDSS), volume 42. Citeseer, year-2007.
[20] www.osun.com
[21] www.google.com