**How to secure sessions in Rails Applications?**

**Introduction:**

In networking “Sessions” indicates the time of browsing a website by a user. It is a semi-permanent, information exchange between a user and a computer. As HTTP is stateless, Sessions are used to make it stateful. Without sessions the user would have no identity, so user has to authenticate on every request. As webpages are having no memories, sessions are helpful for users to be recognized with in a website/application.

**Sessions in Rails:**

Rails creates a new session when a new user accesses the application and loads the existing session if the user has already used the same. In Rails app a session usually consists of a hash of values and a session id. The session id is a 32-character string, it helps to identify the hash. It is possible to save and retrieve the values using *session* method.

To set data:

*def create*

*# ...*

*session[:current\_user\_id] = @user.id*

*# ...*

*end*

To retrieve the data:

*def index*

*current\_user = User.find\_by\_id(session[:current\_user\_id])*

*# ...*

*end*

There are several ways in which hackers use sessions to exploit sensitive information, such as Session Hijacking, Replay Attack, Session Fixation and Session Expiry.

**Session Hijacking:**

It is the process of stealing a user’s session id so that the attacker can use the application in the user name. Many web applications have an authentication system for user login, in which a user name and password is required for a valid session. The session id in the cookie identifies the session. If someone sniffs the cookie in an insecure network may use the web application in the name of user. It is prevented by providing a secure connection over SSL. In Rails 3.1 and later versions it is done by forcing SSL connection in the config file.

*config.force\_ssl = true*

**Replay Attacks for CookieStore Sessions:**

A replay attack is a form of network attack in which valid data transmission is repeated or delayed maliciously. This is done using the cookie store. To solve replay attacks include a nonce or random value in the sessions. Storing the nonce in a database table is the best solution. This will make the purpose of CookieStore entirely false.

**Session Fixation:**

When the attackers fix a user’s session id known to them, this is known as session fixation. They force the user’s browser to use the fixed id. So after this the attackers don’t need to steal the sessions. It can be dangerous because the victim and the attacker will co-use the application, as the session is valid and the victim can’t even notice it.

To prevent Session Fixation, issue a new session identifier and make the previous one invalid after a successful login. This is how a new session is created in Rails:

*reset\_session*

**Session Fixation:**

There are cases in which sessions never expires. This extend the time-frame for attacks. It can be prevented by setting expiry time-stamp of the cookie with the session id. Here is an example how to expire a session in a database:

*class Session < ActiveRecord::Base*

*def self.sweep(time = 1.hour)*

*if time.is\_a?(String)*

*time = time.split.inject { |count, unit| count.to\_i.send(unit) }*

*end*

*delete\_all "updated\_at < '#{time.ago.to\_s(:db)}'"*

*end*

*end*

**Conclusion:**

The SSL is the only way to prevent sniffing attacks that are done with sessions. Still there are some additional guidelines in Rails to secure the sessions. Large objects shouldn’t be stored in sessions. They should be stored in the database and save their id in session. Also critical data must not be saved in session. Many storage mechanisms are provided in Rails for session hashes, the most important of them is *ActionDispacth::Session::CookieStore.* Rails 2 also has introduced a default session storage called, CookieStore. This is helpful in preventing tempering.