**What are the security parameters for your Rails app?**

**Introduction:**

This modern world which mostly rely on Web Applications, has become the world of Information Technology. Starting from social networking to e-commerce and online business, we are all surrounded by web applications. As sensitive information and classified data are shared over internet with the help of web applications, security has become a major issue for all. The threats against an app consists of bypass of access control, accessing or modifying classified data, account hijacking etc. Rails provides some useful helper methods to deal with security issues.

In this blog we will understand the attack methods and how it can be handled with Rails.

**Sessions:**

In networking “Sessions” indicates the time of browsing a website by a user. 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.

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

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. Many storage mechanisms are provided in Rails for session hashes, the most important of them is *ActionDispacth::Session::CookieStore.* Another way is to set the expiry time-stamp of the cookie with the session id. Here is an example:

*class Session <ActiveRecord::Base*

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

*iftime.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*

**Cross-Site Request Forgery (CSRF):**

Including malicious code or a link in a page which access a web application that the user is believed to have authenticated, is known as CSRF. If the session for that application is still active, the application becomes vulnerable, as any attacker may execute unauthorized commands.

To prevent CSRF attacks, we should use GET and POST method appropriately. Rails uses a hidden *\_method*to handle additional HTTP verbs such as PUT, PATCH and DELETE. A required security token has introduced to protect against all other forged requests.

*protect\_from\_forgery with: :exception*

This includes a security token in all forms and Ajax requests generated by Rails automatically. An exception will be thrown if the security token doesn’t match.

**Redirection and Files:**

This process involves redirection of user to a fake web application which looks and feels exactly like the original application. Then the attackers can get the information about the user or can originate a self-contained attack. This can be prevented using a legacy action:

*def legacy*

*redirect\_to(params.update(action:'main'))*

*end*

When uploading a file, make sure not to overwrite important files and process the media file asynchronously. The file should be saved and processing request in the database should be scheduled. Users should be restricted not to download arbitrary files. File name like *"../../../etc/passwd"*can download server’s login information. Rails make it possible to check that the requested file is in the expected directory.

*basename = File.expand\_path(File.join(File.dirname(\_\_FILE\_\_), '../../files'))*

*filename = File.expand\_path(File.join(basename, @file.public\_filename))*

*raise if basename !=*

*File.expand\_path(File.join(File.dirname(filename), '../../../'))*

*send\_file filename, disposition: 'inline'*

**Injection:**

Injection is a class of attacks that uses injection of data in to a web application in order to facilitate the execution of harmful data in an unexpected manner. This includes Cross-Site Scripting (XSS), SQL injection, CSS Injection, Command Line injection, Header injection.

Rails provides some helper methods to prevent SQL injcection and Cross-Site Scripting. There is a method called *sanitize()*as a model for whitelist CSS filter, this can be used as a countermeasure of CSS injection. You should update the Rails version to 2.1.2 to prevent header injection.

**Conclusion:**

The number of attacks are increasing to web applications. So we should pay more attention for security. While Rails provide so many built-in mechanism to secure the application of its own, the developers should be conscious about the security. The security features that are not provided by Rails by default, there are many Gems for the same. It is easier to build a secure application in Rails, but a developer should always take basic measures to protect the application.