

Security Audit

Building safe applications

by Aaron Bedra

Security Audit

©2008 Aaron Bedra

Every effort was made to provide accurate information in this document. However, neither Aaron Bedra nor Topfunky Corporation shall have any liability for any errors in the code or descriptions presented in this book.

"Rails" and "Ruby on Rails" are trademarks of David Heinemeier Hansson.

This document is available for US\$9 at PeepCode.com (<http://peepcode.com>). Group discounts and site licenses can also be purchased by sending email to peepcode@topfunky.com.

OTHER PEEPCODE PRODUCTS

- RSpec (<http://peepcode.com/products/rspec-basics>) – A three part series on the popular behavior-driven development framework.
- Rails from Scratch (<http://peepcode.com>) – Learn Rails!
- RESTful Rails (<http://peepcode.com/products/restful-rails>) – Teaches the concepts of application design with REST.
- Subscription pack of 10 (<http://peepcode.com/products/subscription-pack-of-10>) – Save money! Buy 10 PeepCode credits.
- Javascript with Prototype (<http://peepcode.com/products/javascript-with-prototypejs>) – Code confidently with Javascript!
- Rails Code Review PDF (<http://peepcode.com/products/draft-rails-code-review-pdf>) – Common mistakes in Rails applications, and how to fix them.

CONTENTS

5 **So You Want to Audit your Rails App?**

- 6 You write tests don't you?
- 6 I don't know the first thing about security!
- 6 Keep Notes!!!
- 7 Be Brutal

8 **Securing Models**

- 8 SQL Injection
- 13 Unapproved Data Manipulation
- 15 Unauthorized Access Escalation

18 **Securing Views**

- 18 Cross Site Scripting (XSS)
- 22 Cross Site Request Forgery (CSRF)

24 **Crawling and Fuzz Testing**

- 26 Running the test
- 28 What does it all mean?
- 29 How do I fix it?

30 Disclaimer!

31 **Keeping your Host on Lockdown**

- 31 Platforms
- 31 Firewall
- 35 Sssshhhh, don't tell them we moved SSH
- 37 General Rules of Thumb

38 **What's the Risk?**

- 38 Why do I care about Risk Analysis?
- 45 Conclusions

47 **Final Thoughts**

- 47 Create Guidelines, not Rules!

So You Want to Audit your Rails App?

CHAPTER 1

Everyday, thousands of applications are hacked.

Bank account numbers are exposed, social security information is compromised, credit cards are revealed, and other types of personal information is accessed. Don't let your Rails application be one of them! Let's pick apart your code and see what places are vulnerable to attacks. We will cover SQL Injection in your models, sanitization logic in your controllers, Cross Site Scripting and Cross Site Request Forgery in your views, and improper setup of your host operating system. The information we cover could very well be the most important thing you learn as a developer aside from writing the code itself.

There are a ton of reasons to audit your application. You could work for a company that is regulated under Sarbanes-Oxley (SOX) or HIPAA. You might be launching a new startup that is going to be taking credit card transactions. Or, you might just want to keep your users' data safe. No matter why you decide to audit your application, the process is the same. Even if you don't have nuclear launch codes being passed over the Internet, you should be auditing your work to make sure you haven't introduced any security holes.

Audits take place every day for many different reasons. It's a great idea for you to learn how to perform an audit. Once you are comfortable with the process you can start scheduling regular audits. Along with increasing the security of your application, you will more than likely increase the overall quality of your code as well.

You write tests don't you?

Just think of this as another test. Some of it can be automated, but eventually there will be manual verification. This can be off-putting to some people, but I assure you it is worth it. During the process you will laugh a little, cry a little, and probably make a few Yaks really really cold (to be explained later). It is a process that will change the way you develop your next application.

I don't know the first thing about security!

That's OK, you don't have to be a security expert to audit your application. It does help to know a thing or two about haxoring, or to be able to think like a hacker, but it's not necessary. This book will get you well on your way to being an effective auditor and will probably inspire you to read more about information security.

Keep Notes!!!

One of the most important things you can do during an audit is keep a very good set of notes. Document everything you do. If you have even the slightest notion that something is awry, make a note of it, because you will need these notes later. I recommend using an electronic solution so you can easily search your notes for a keyword or phrase. Your notes will probably end up being a giant ball of nonsense that only you can make sense of, so having the ability to search through it will help when you go back to write a sane version in report form.

There are many applications that can help you with taking notes. Desktop programs like Voodoo Pad (<http://flyingmeat.com/voodooopad>), Yojimbo (<http://www.barebones.com/products/yojimbo>), or TaskPaper (<http://hogbaysoftware.com/products/taskpaper>) are great for this task. You can

even keep it simple and just use a text file or even the trusty emacs scratch buffer.

Be Brutal

It can be a hard thing to audit your own code. I actually recommend against it! If you audit your own application, you'll be thinking about how much work it will be to fix all the holes you find, which may cause you to treat it more gently than you should. So if you wrote it, let someone else audit it.

If you have no other choice, you need to be honest with yourself about things that are wrong. If it's not your code, it's much easier to be a jerk about what's wrong. You do however, need to be as brutal as you can be when it comes to auditing software. If something seems even the slightest bit wrong, document it for further scrutiny.

Securing Models

CHAPTER 2

The most important part of any application isn't your code or the graphics, it's the application's data. You can rewrite code, but it's very hard to recreate data that has been deleted or manipulated. And it's impossible to retrieve data once it has been leaked!

SQL Injection

Your application is running smoothly on the web. Everything is going nicely until one morning you get an email as you are reading your RSS reader. The site doesn't seem to be working.

As you sit down to troubleshoot, you quickly notice that the data you had yesterday is no longer there. In fact, all your data is missing. You restore from the backups you pulled last night and all is well. Then, a look at your logs reveals some nasty SQL as the culprit of your disaster.

SQL injection accounts for a significant number of web related application break-ins. In the past few years there have been huge improvements in the effort to stop SQL injection related exploits.

SQL injection is a technique that exploits a security vulnerability occurring in the database layer of an application. The vulnerability is present when

- User input is incorrectly filtered.
- Escape characters embedded in SQL statements are not correctly sanitized.
- User input is not strongly typed and thereby unexpectedly exe-

SQL INJECTION RESOURCES

There are a lot of great SQL injection related resources on the web. OWASP has a great section (http://www.owasp.org/index.php/Testing_for_SQL_Injection) on testing for SQL injection if you want to learn more on the subject.

cuted.

SQL injection is in fact an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

How does it happen?

There are a lot of different ways to attack an application that is vulnerable to SQL injection attacks. Let's take a look at how to exploit insecure code using the aforementioned dangerous group of methods. We will use examples from a few and demonstrate what can happen if injection code is passed in unsanitized.

```
Asset.find(:all, :limit => "#{params}")
```

If we passed something similar to this:

```
10 procedure help()
```

We can expect an output and result similar to this:

```
Unknown procedure 'help' : SELECT * FROM 'assets'
```

This is not good! It looks like we are able to directly call stored procedures from the `limit` call. If we could enumerate what stored procedures are on this database, we could start executing those procedures.

Another thing that could happen would look like this:

```
Asset.find_by_sql("SELECT * FROM assets WHERE id=#{params}")
```