

**Secure Programming CA3  
Static Code Analysis**

**James Finglas – B00094138**

Department of Informatics

School of Informatics and Engineering

Institute of Technology Blanchardstown

Dublin 15.

**BN311 Bachelor of Science in Computing**

**Digital Forensics & Cyber Security**

**Penetration Testing**

**28/10/2018**

Contents

[Abstract 3](#_Toc532584870)

[Scope 3](#_Toc532584871)

[Introduction 3](#_Toc532584872)

[Investigation of Bugs 4](#_Toc532584873)

[SQL Injection 4](#_Toc532584874)

[Cross Site Scripting 7](#_Toc532584875)

[Access Control 9](#_Toc532584876)

[Insecure Direct Object Reference (IDOR) 10](#_Toc532584877)

[Scanning for Bugs 12](#_Toc532584878)

[FindBugs 12](#_Toc532584879)

[Visual code Grepper (VCP) V2.1.0 15](#_Toc532584880)

[Fixes Implemented & Scans Revisited 22](#_Toc532584881)

[SQL Injection Fixed 22](#_Toc532584882)

[Cross Site Scripting Fixed 27](#_Toc532584883)

[Access Control Fixed 29](#_Toc532584884)

[Insecure Direct Object Reference (IDOR) Fixed 30](#_Toc532584885)

[Conclusions 32](#_Toc532584886)

[References 32](#_Toc532584887)

Abstract

This report documents my investigation in a Java based program, in an attempt to determine if the program contains any potential security vulnerabilities which may lead to the user’s system potentially being compromised.

Scope

My chosen program is ‘’CyberSecurityProject” by ‘dinesh2043’ **[1]**, a java-based Web application. The application is designed to have several vulnerabilities which can be exploited, examined and fixed, in an effort to help teach developers in training. My chosen IDE is NetBeans. The Brief requires that 2 different static code analysers be used and that 2 fixes be implemented, however; I elected to examine and fix 4 as I felt that the difficult of some the fixes was simply too easy.  
  
 I chose to use the ‘Find Bugs’ security plugin for NetBeans as my first static code analyser. This seemed a natural choice due to its integration into the IDE which I had familiarized myself with this year.  
  
My second choice was ‘Visual Code Grepper V2.1.0’ **[2]**. This is an interesting project, as it is stand alone and presents issues in what I would consider to be a much more old-school manner. However, installation was simple, and utilization also was simple, merely requiring the user to point at the directory and scan.  
  
In an attempt to restrict myself to code relevant to our course material, I have tried to restrict myself to what I consider to be moderate/critical security flaws which could lead to system exposure.

Introduction

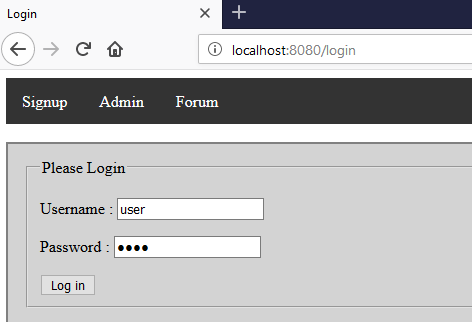
The program has several high-ranking security flaws integrated into its coding I therefore elected to explore these vulnerabilities and demonstrate them. I then scanned the program with each vulnerability scanner to see what was detected and do a direct comparison in order to see which the better scanner was. This seemed the most efficient method.  
  
I then moved on to fixing the code to remove the vulnerabilities or increase security, before rescanning to see if any vulnerabilities which had been detected previously, where now no longer detected.

.

# Investigation of Bugs

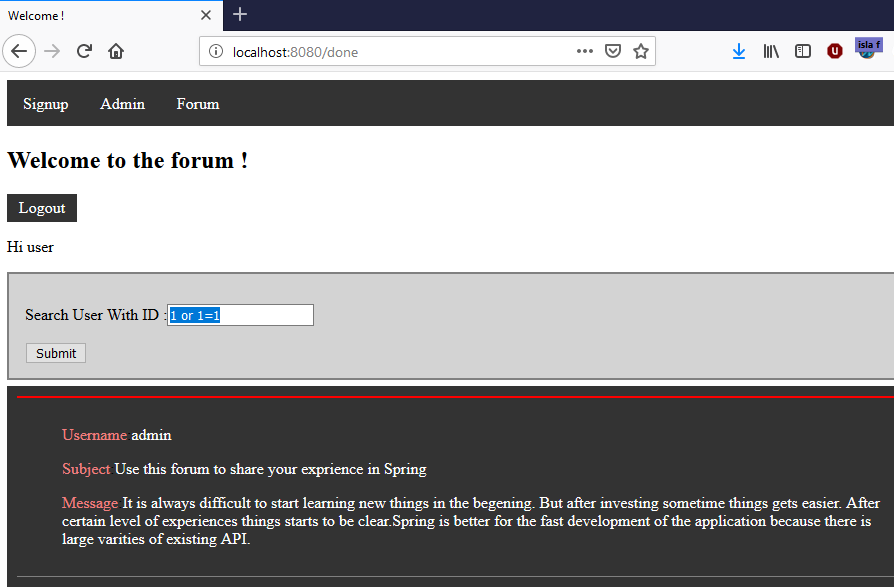
## SQL Injection

My investigation of bugs begins with SQL injection. We first log into our forum page and log in as a user.



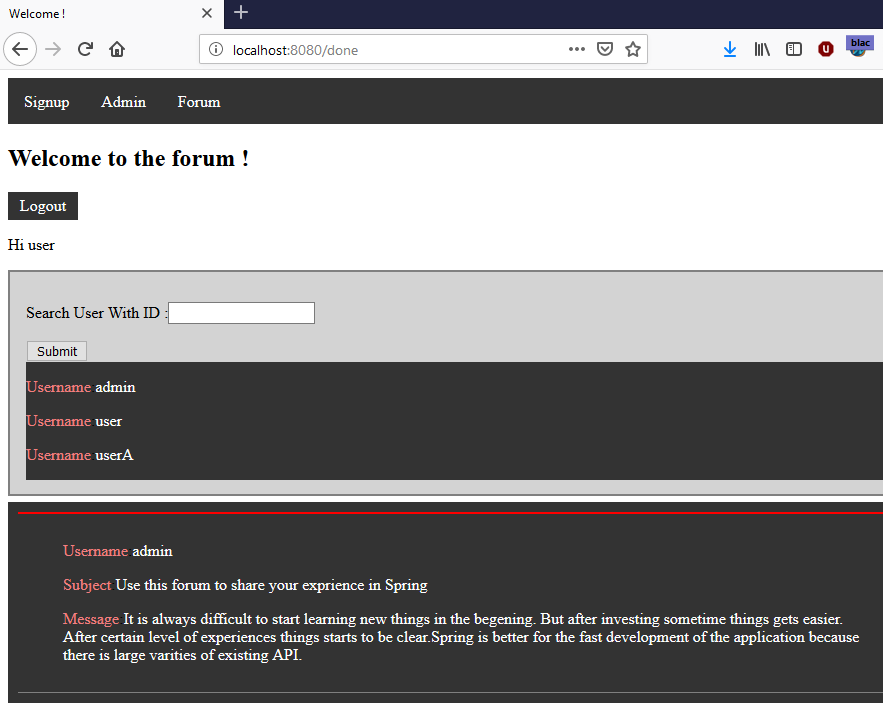
*Here we see our program awaiting login. As you can see we are not logging in as admin.*

Now that we have logged in, we are presented with a “search User With ID” field. Into this, we enter a simple always true SQL statement.



*Here we see our attempt to bypass admin restrictions as user by entering what we hoped would be an always true SQL statement.*

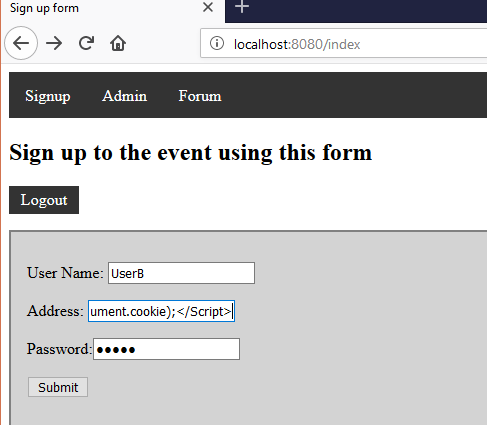
And below we see that our SQL injection has been successful, and we can view the entire user table from the database output directly to our page.



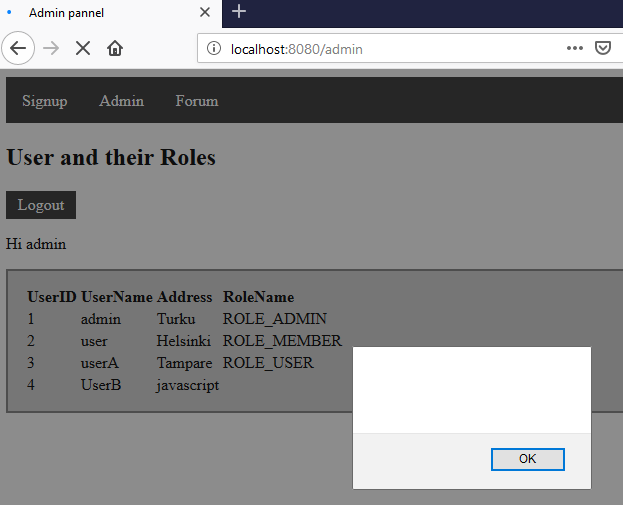
*Here we see the output of our successful SQL injection attack, outputting the entire user table to the page.*

## Cross Site Scripting

Next, we discovered a cross site scripting (XSS) vulnerability. This vulnerability was discovered by attempting the enter a script into the address field of the ‘Signup’ page of the web app.

  
*Here we attempt to enter a XSS script into the address field of the Signup page. The full command entered was: javascript <Script>alert(document.cookie);</*Script>.

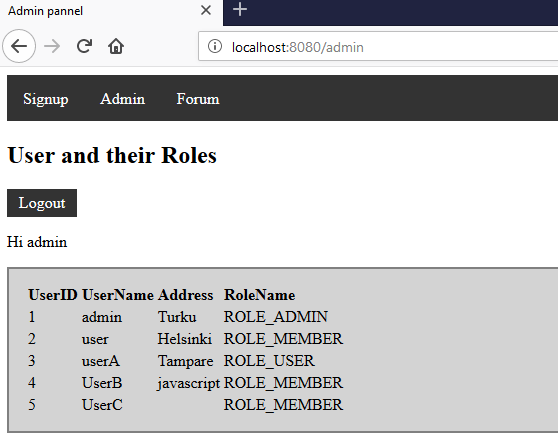
This is successful, and now we relog into the site to view our new user’s information as admin.



*Here we see that not only has our script address been accepted, but our alert box is functional.*

## Access Control

Our next bug we discovered simple by logging in as a user and attempting to access the admin page.

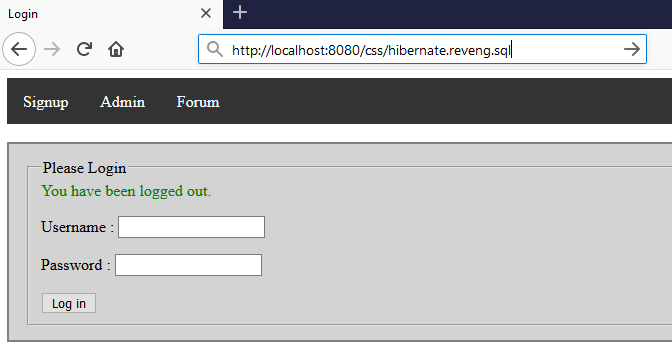


*Here we are, logged in as user, on the admin page.*

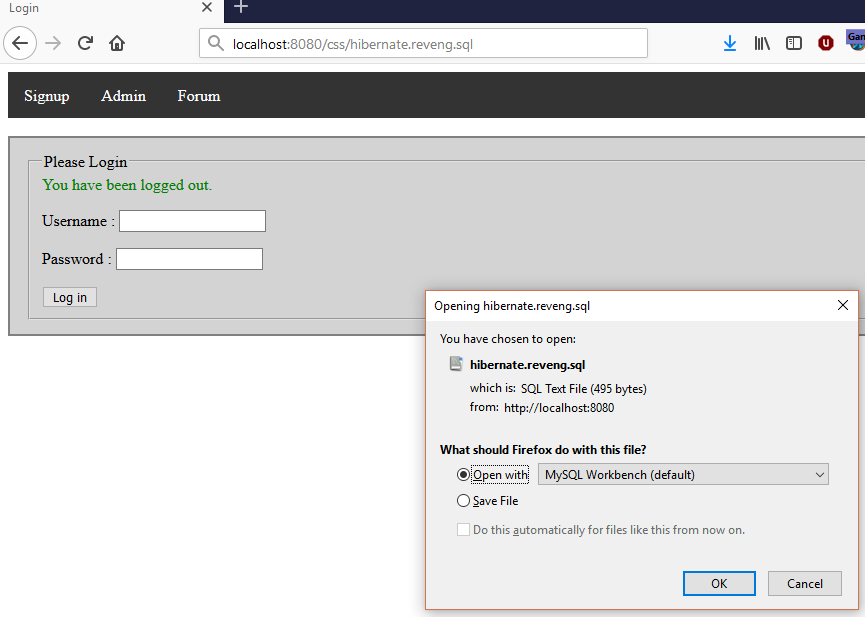
As we see above, this should not be possible if the permissions are set up correctly as our ‘user’ has only the privileges of ‘ROLE\_USER’ and therefore should not be able to view this page.

## Insecure Direct Object Reference (IDOR)

This last vulnerability which I will cover was discovered by attempting to access a page that no user should be able to access. No link exists for this page and it may only be accessed by inputting the URL directly into browsers URL window. This page contains a file which contains sensitive information pertaining to the web apps database.



*Here we are, logged out; as we attempt to both bypass login and directly download a file from a URL no user should have access to.*

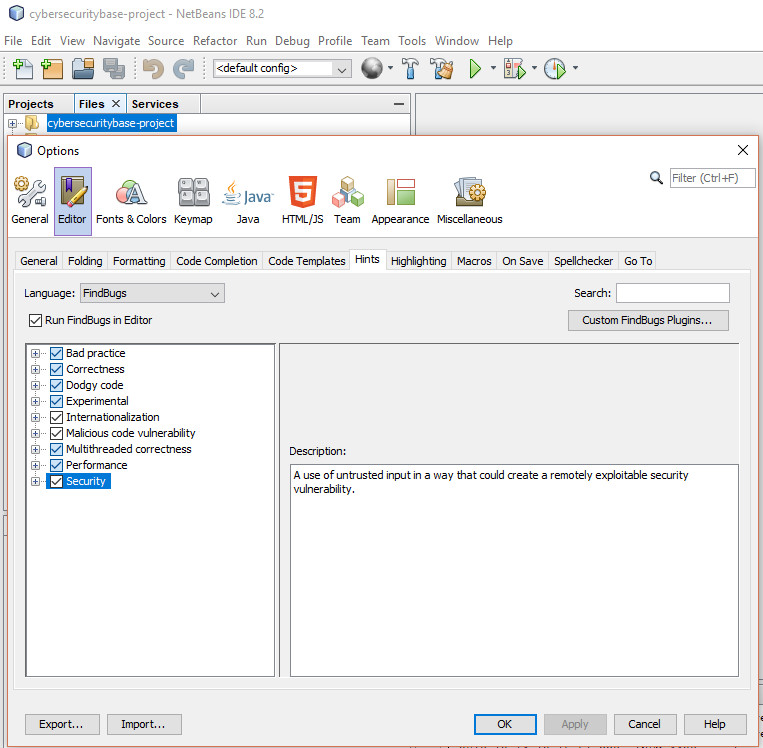
**

*Here we see that our attempt to access this restricted page, and download this file containing sensitive data has been successful.*

# Scanning for Bugs

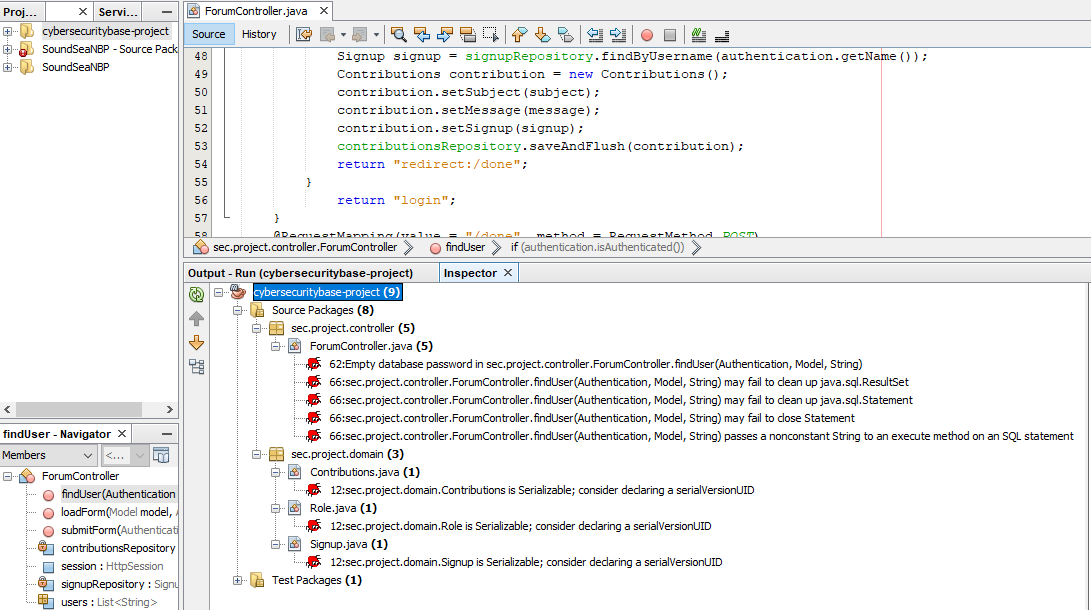
## FindBugs

First, I displayed that the security plugin for FindBugs is correctly installed and functioning as expected.



*Above we can see confirmation that the plugin is installed correctly.*

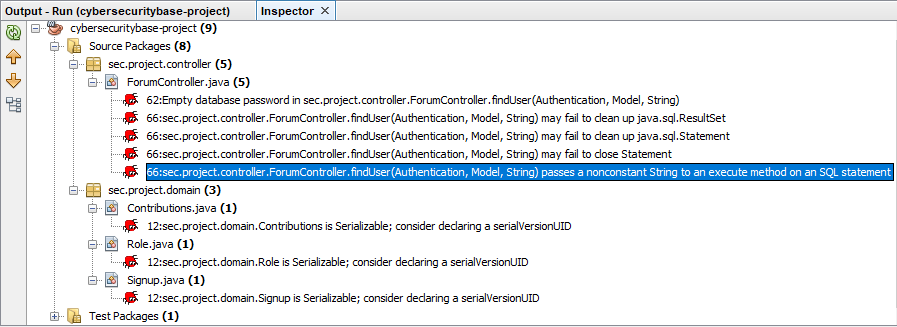
Next, we carried our first san using FindBugs in an attempted to discover if our bugs will be detected, or indeed on any other serious bugs are detected.

**

*Here we see our initial scan using FindBugs.*

Our scan with FindBugs detects a total of 9 bugs. However, one bug is in the test package which is not relevant. The first five bugs detected are located in the ForumController class. The first of these is an unexpected security flaw found on line 62 of the ForumController class, where the database is created without a password. This is indeed a high vulnerability. However, in this case, as the application is only created for testing purposes, I elected not to fix this. Where I to use an external database such as the fix for this would be two-fold, firstly the database would be created using a strong password and the user would be denied permissions to alter or otherwise damage tables, or indeed the schema itself.   
  
The next three bugs found in this class are minor bugs. It could be argued that these bugs are security bugs that could lead to information leakage and as such should be fixed. But seeing as they are minor again I elected to skip over these bugs deeming them to be too insignificant.

At line 66 we find our first indication of one of our previously discovered and demonstrated bugs. This is our SQL injection bug.

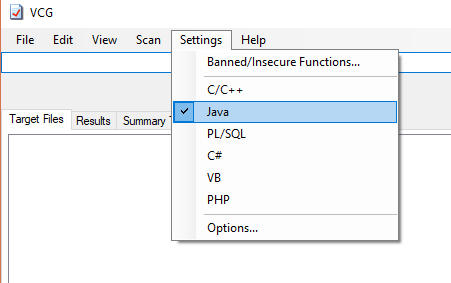


*Here we see pour previously demonstrated SQL Injection vulnerability.*

The final three bugs are located across three classes, Contributions.java, Role.java and Signup.java; which each detect that the class is Serializable. But a serialVersionUID has not been assigned. These are not particularly high-ranking security vulnerabilities.  
  
So, out of our 4 major security vulnerabilities, only one is detected by FindBugs. A rather poor showing and I had expected much better from this plugin.

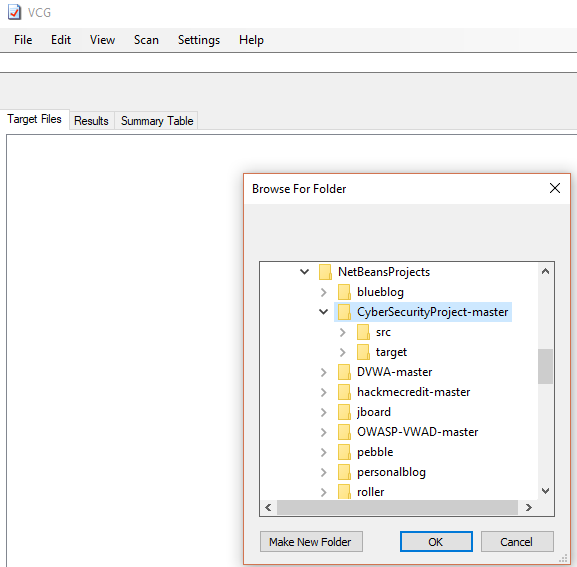
## Visual code Grepper (VCP) V2.1.0

First, we select the appropriate language for our program. VCP is useful in that it offers a reasonably wide selection of language scanning options including: C, C++, Java, PL, SQL, C#, VB and PHP. For a free shareware program, this is impressive functionality.



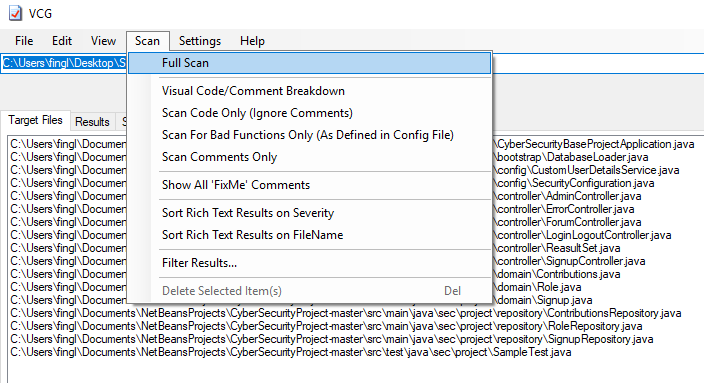
*First, we set our language scan option to Java.*

Now, we select our working directory in NetBeans. This allows to apply our fixes once and scan the same directory with both programs once the fixes have been applied.

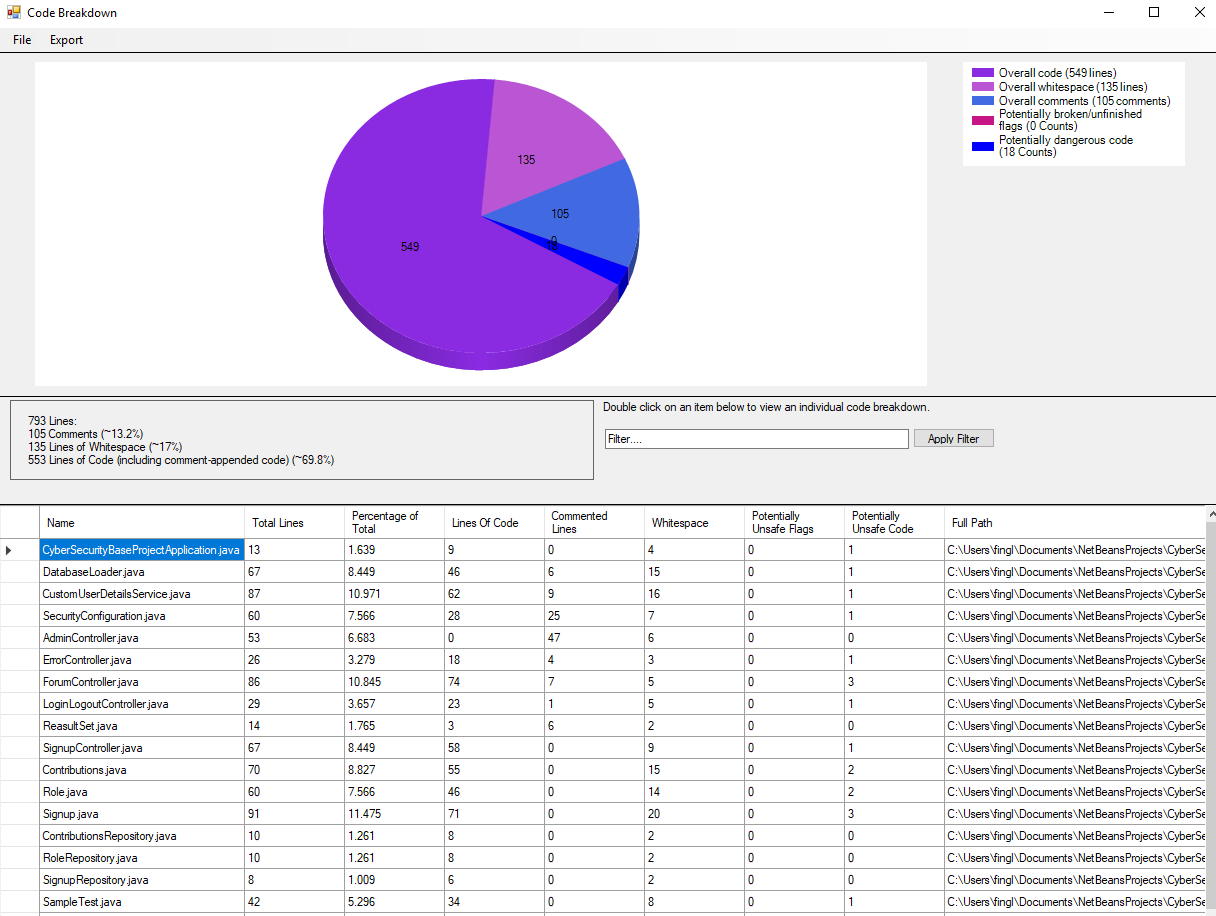
**

*Here we selected our working directory to scan.*

Now we begin our full scan of our target working directory.

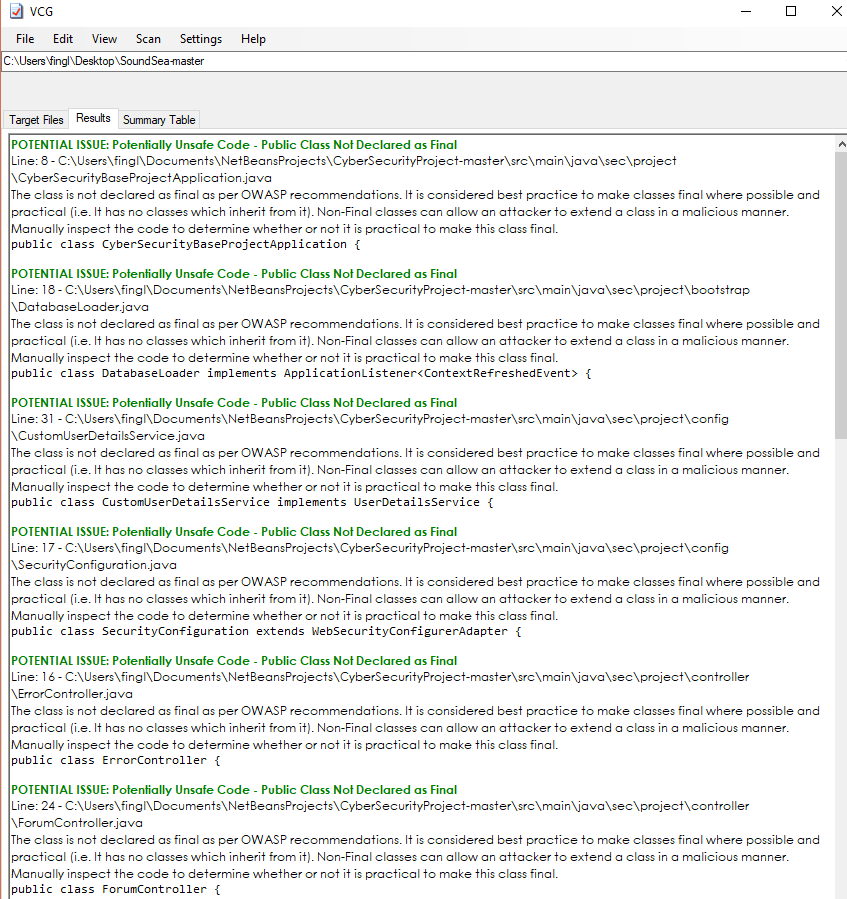


*Here we begin our full scan.*

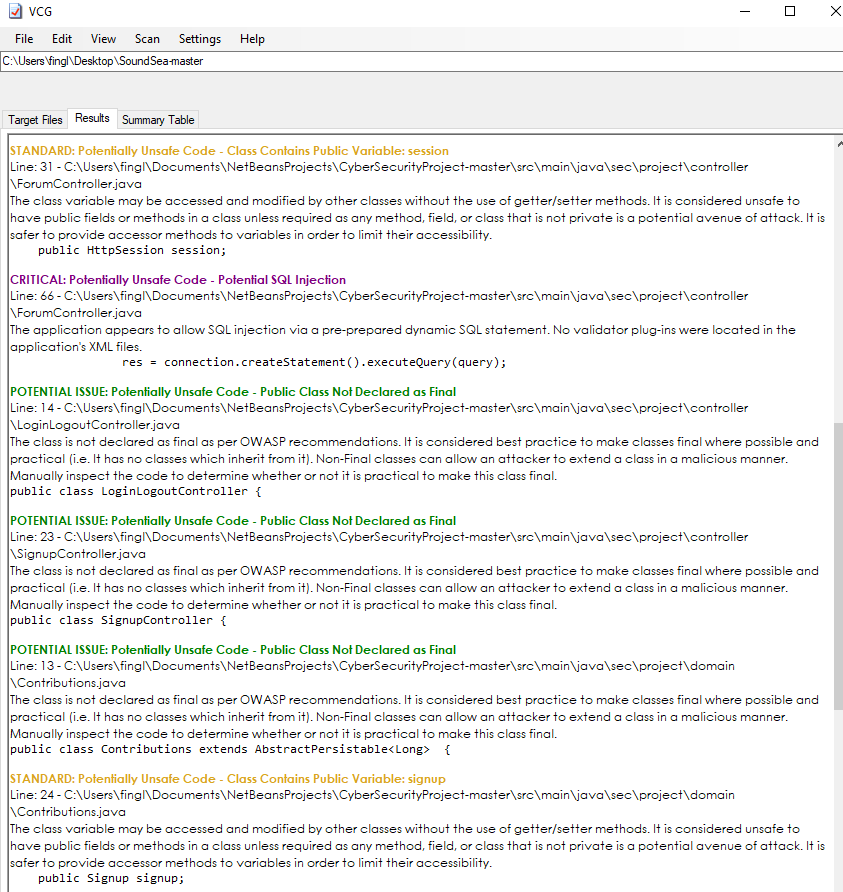


*Here we see a chart breakdown of our scan results.*

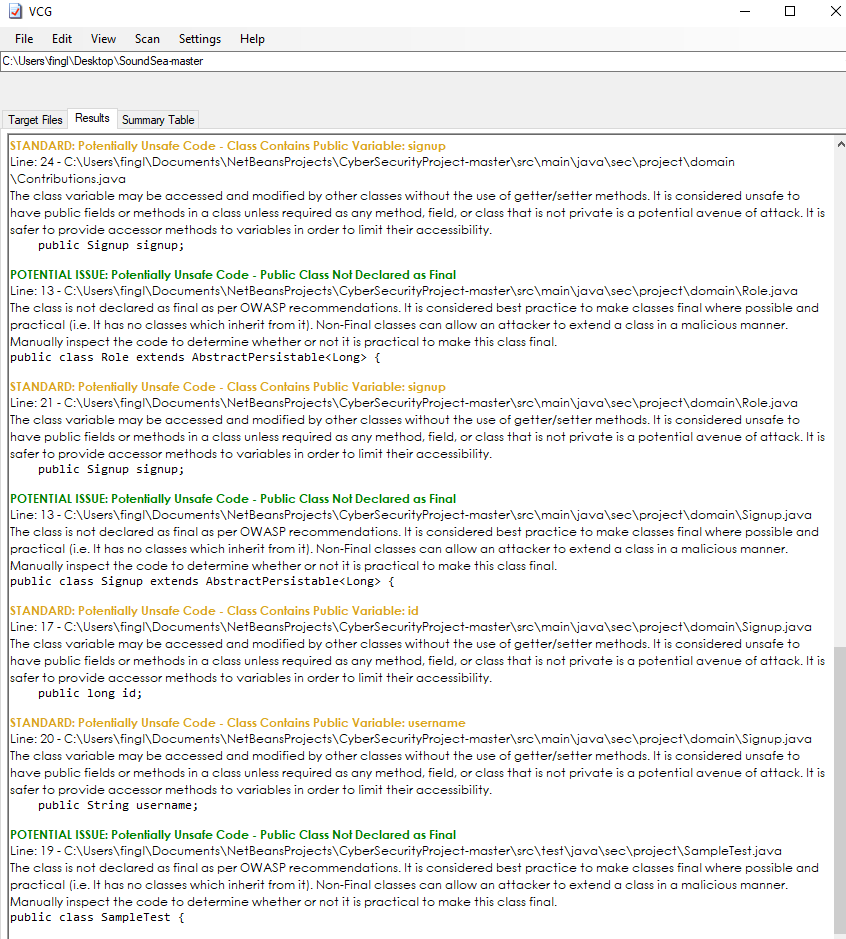
The first thing we are presented with is a chart breakdown of our scan results. What this tells us is that in the dark blue section highlighted above, is that 18 instances of dangerous code have been detected. It is interesting to note that while we have yet to examine these issues in depth we can already see that VCG has potentially located double the amount of issues that FindBugs detected.



*Here we see minor potential security flaws detected.*



*Here we see our SQL Injection vulnerability detected again. Followed by more minor potential vulnerabilities.*

**

*Here we see the final set of potential vulnerabilities.*

As we see from the above vulnerabilities, yet again only our SQL Injection vulnerability has been detected by our second scanner. I will briefly cover the bugs detected by VCG however, despite them being very minor compared to our previously displayed bugs. I will list them in groups sequentially as follows:

Bugs 1-6: classes not declared as final. This might allow for malicious code to alter the state of variables. This is minor security issue.

Bug 7: Public variable detected: Similar in nature to bugs 1-6, again allowing for potential alteration. However, this being a session variable a session ID could potentially be spoofed to make the program think somebody who is not an admin is logged in as admin. This is a very low risk vulnerability requiring a high level of skill to achieve.

Bug 8: Our SQL injection attack as demonstrated previously.

Bugs 9-11: more classes not declared as final. Minor vulnerabilities as mentioned above.

Bug 12: another non-final variable.

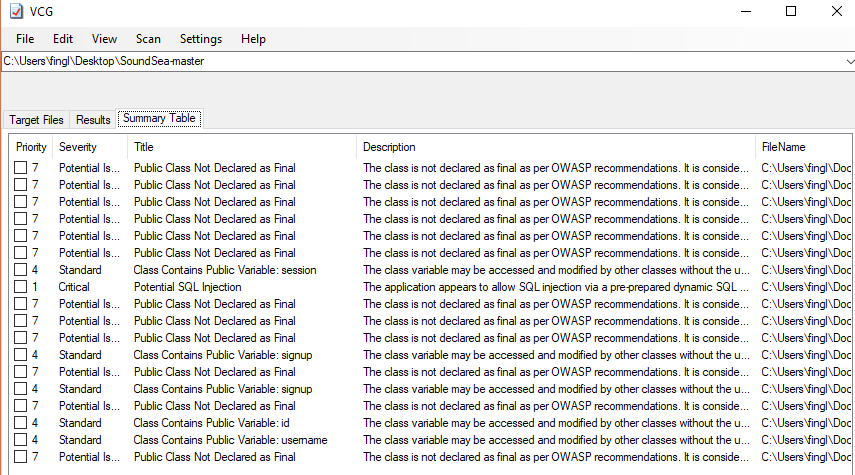
Bug 13: another non-final class detected.

Bug 14: another non-final variable.

Bug 15: another non-final class detected.

Bugs 16-17: more classes not declared as final. Minor vulnerabilities as mentioned above

Bug 18: another non-final class detected.



*Here we see a summary table provided by VCG.*

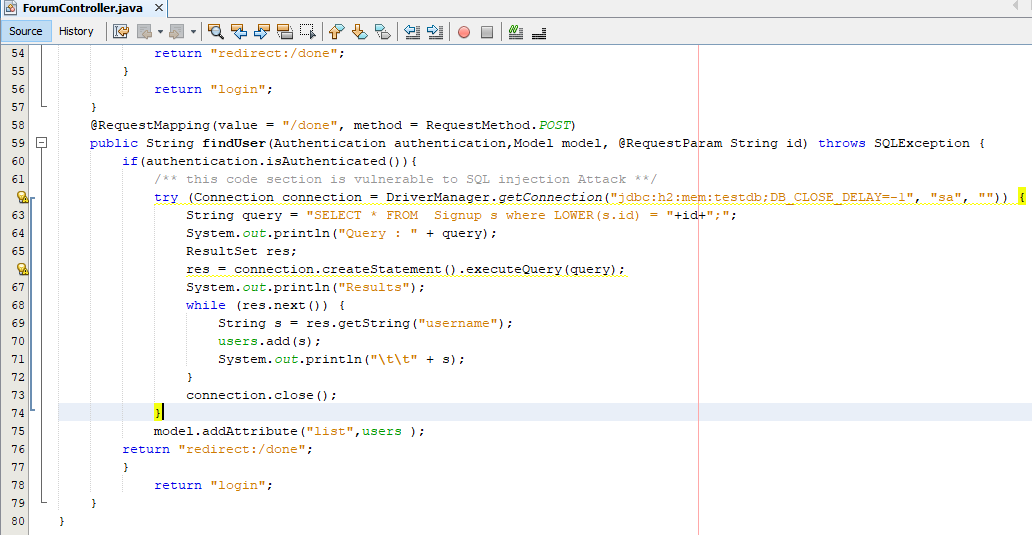
This summary table provides us with some useful option by right clicking such as: Open in associated IDE, open in noted pad, order by columns, filters, Export results, alter colour of severity, change severity and finally delete a result from a scan.

Yet again our scanner has only achieved a 1 out of 4 return of detections of our previously demonstrated security vulnerabilities., which is again a disappointing level of effectiveness.

# Fixes Implemented & Scans Revisited

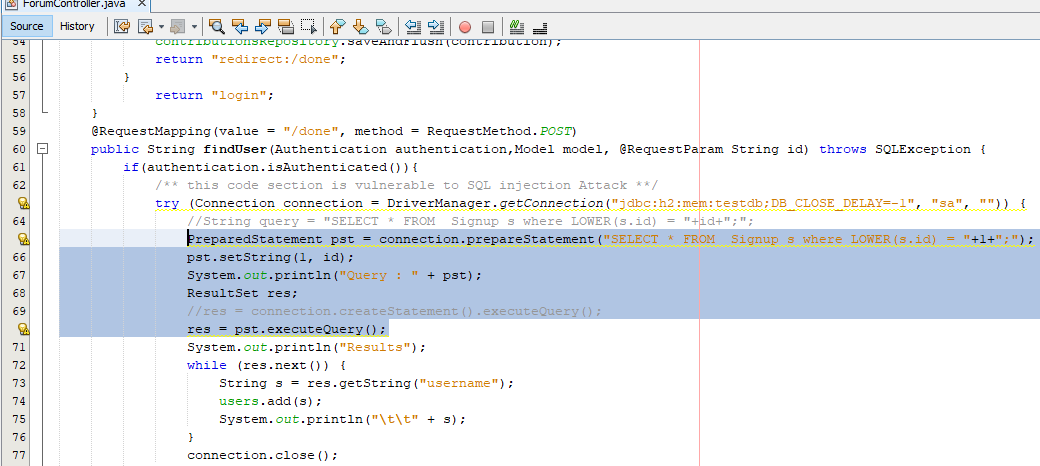
## SQL Injection Fixed

SQL injection is achieved in this case due to a total lack of verification of user data input. A basic fix for this would be to have all data verified via a prepared statement. However, this would not incorporate validation or authentication.

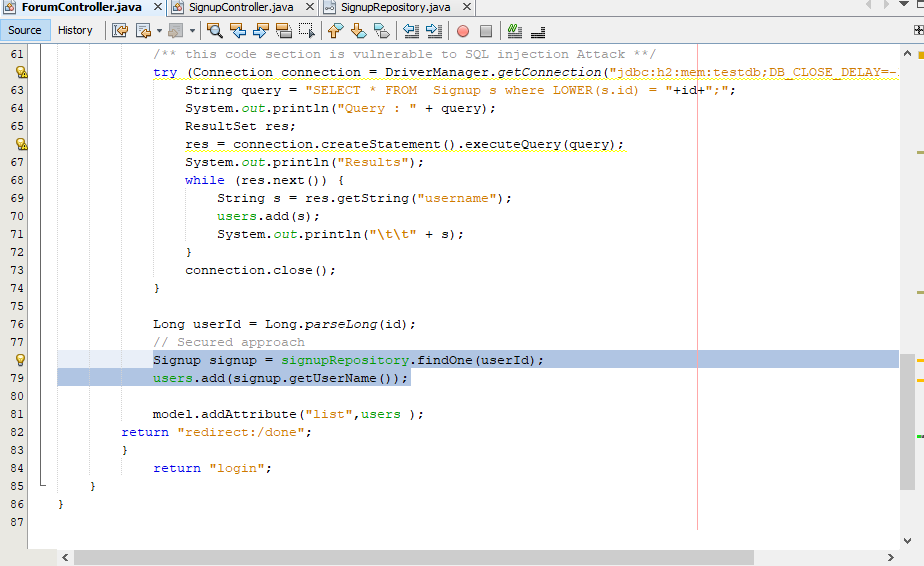


*As we see above, no prepared statement or verification is carried out to validate the user input.*

So, the fix here was to create a method to verify the user input.

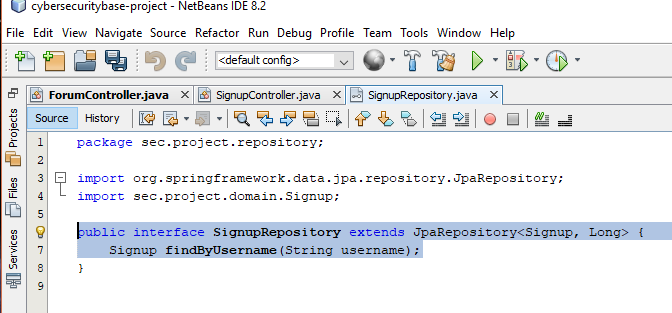


Here we see the addition of our prepared statement.



*Here we see the redirect for inter class validation*

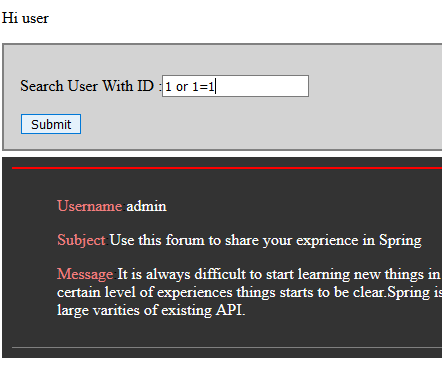
Above we see a validation statement which then sends the data to the signupRepositiry class to be further verified.



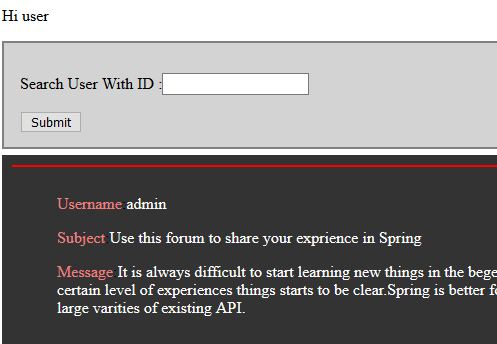
*Here we see the JPA class being implemented to verify data between classes.*

The final step in securing against this SQL attack is to implement the java JPA class. The JPA class is a persistence management class for managing relational data classes.

This has now prevented our SQL Injection attempts from succeeding.

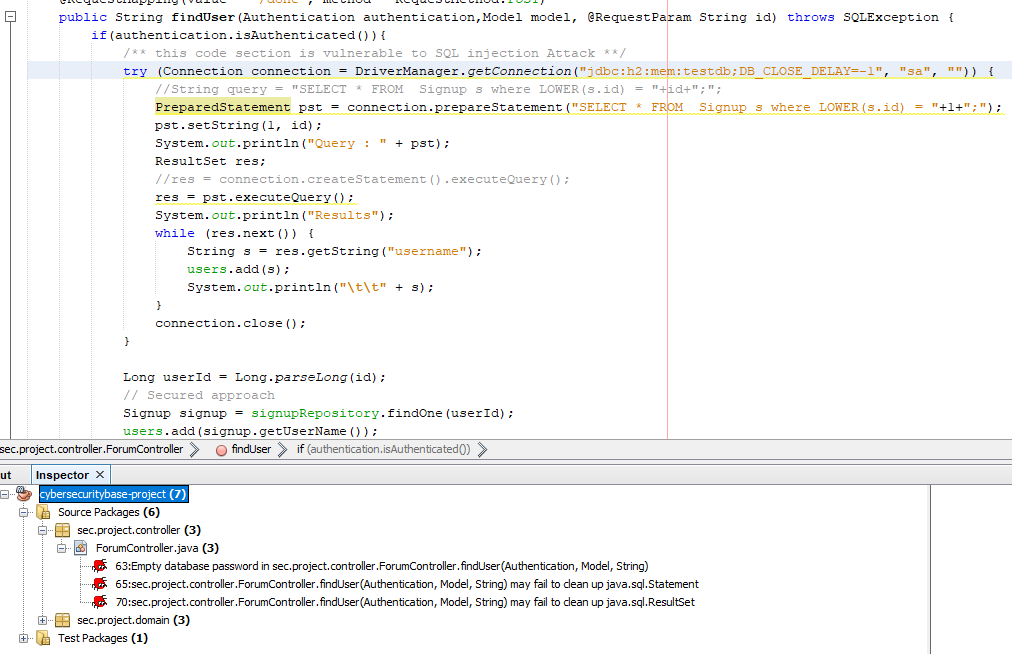


*Once again, we attempt to execute our SQL Injection attack.*



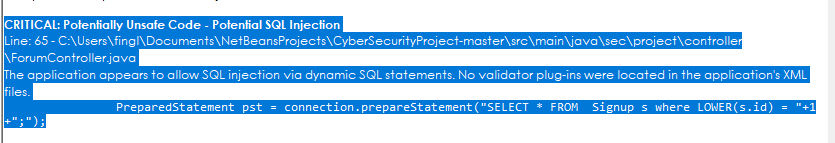
No results displayed from our attempt.

As we see above our attempt has result in no information being returned from the database. This attempt has further highlight that no error handling is implemented. While in this case that is not a security concern, and fixing this issue is beyond the scope of the brief, it is worth noting that errors should never be left unhandled.



*As we can see, a rescan with FindBugs no longer detects our SQL Injection Vulnerability.*

Rescanning now with FindBugs reveals that we have removed this vulnerability.

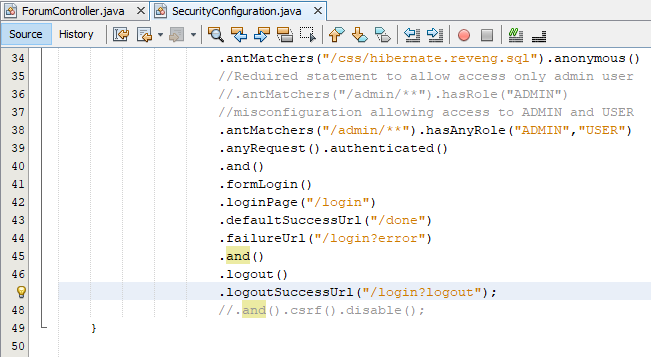


*Curiously, a rescan with VCG still detects a potential SQL Injection Vulnerability.*

A rescan with VCG still detects a potential SQL Vulnerability despite the confirmation that it no longer works. This leads me to wonder as to the reliability to the programs results.  
  
From here on out, scanning will no longer detect our fixes as the issues themselves went undetected, however the fixes were of course implemented none the less.

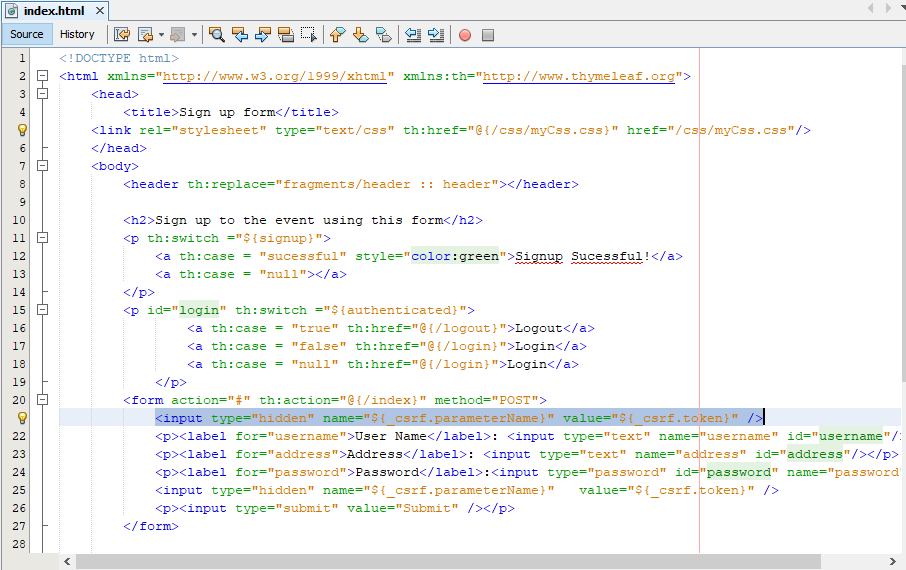
## Cross Site Scripting Fixed

The first step is securing against XXS attacks is to enable the csrf() function in the security config file.

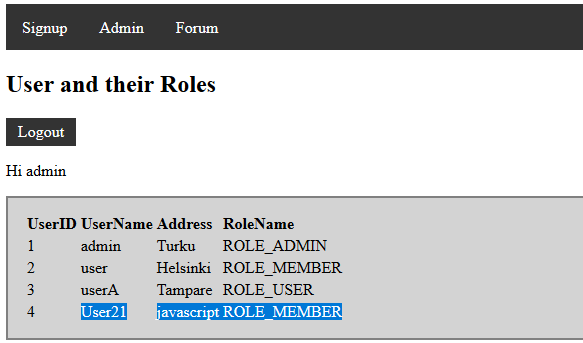


*Here we see the line disabling the use of the csrf function commented out.*

Next, we modify the html page on which the input is taken to pass all inputs through the csrf90 function, which will disallow XSS attacks.



*Here we see the addition of csrf() XSS protections to the index.html page.*

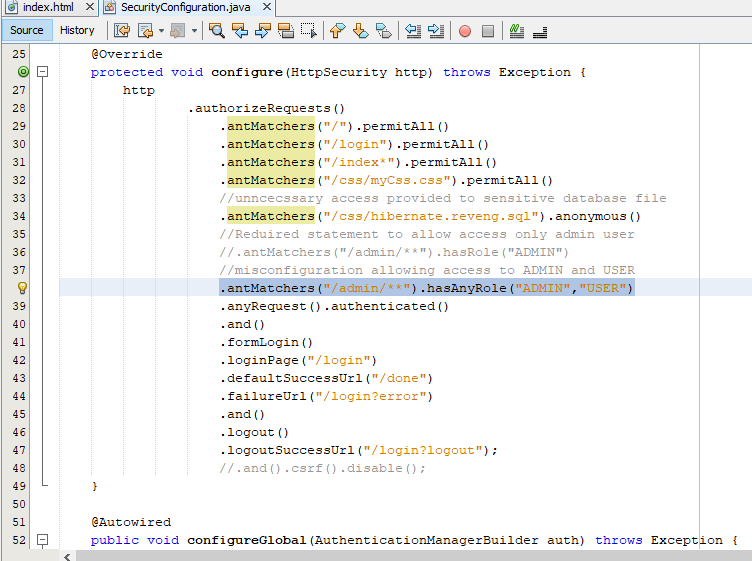
**

*Here we see we have blocked the use of XXS attacks.*

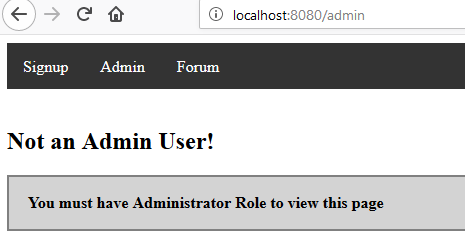
Above, we can see that while we have re-entered the same script, it has only been accepted as plain text and the script has not executed as it has previously. We have successfully blocked XSS scripting.

## Access Control Fixed

The fix here is very simple. The security configuration file has been misconfigured to allow for both admins and user to be allowed to access this page. The fix is simply to remove USER from this line denying access to all but verified admins.



*Here we see USER has been allowed access to this page.*

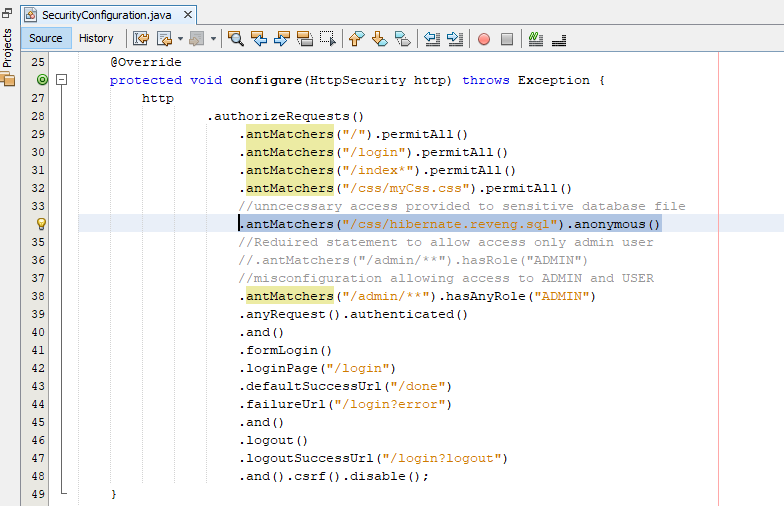
**

*Access is now denied to non-admins.*

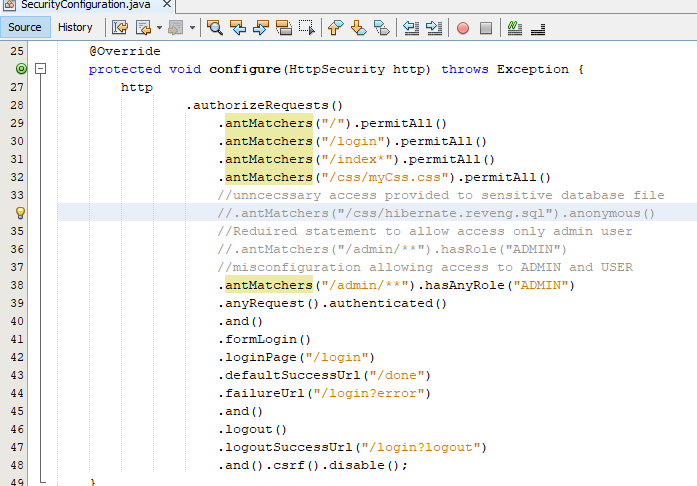
As we see above, we have now blocked access to non-admins.

## Insecure Direct Object Reference (IDOR) Fixed

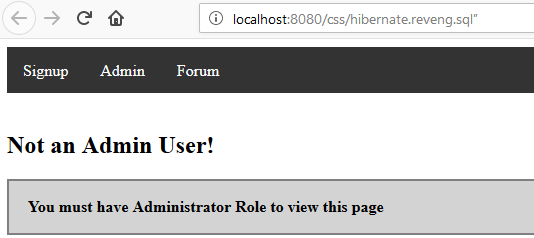
This is yet again a simple issue to fix in this instance, once again resulting from a simple mis configuration in the security configuration file. Access to this file should be restricted to admins.



*Here we see that any user, even those not logged in at all, is permitted access to this file.*

**

*Here we have commented out this code to prevent access to this file to all but admins.*

**

*Success, only admins may now access this file.*

As we can see, we have now successfully removed this bug and restricted access to this file to admins only.

Conclusions

The failure of both programs to detect all but one of the security flaws I detected manually was quite unexpected. It does however teach us a valuable lesson. While automated tools can serve to cut down on time taken to discover and fix many minor bugs, both security related and non-security related, we should never assume that simply because a bug is not detected by a tool it does not exist.  
  
Time should always be given over to the seeking of serious high-risk bugs though manually attempts to attack potentially vulnerable areas of the program or web application in development. Furthermore, no1 tool should be relied upon. As we saw, both programs detected both differing amounts of potential bug, as well different types. We should also strive to keep our software as up to date as possible.  
  
Interestingly, and defiantly worth a mention; is the fact that Visual Code Grepper has not been maintained for several years with the last update being on May 3rd, 2016. Yet this program seemed the more thorough of the two.  
  
The integration of FindBugs into NetBeans is naturally a valuable time saving detail in its favour, however I found it to be highly lacking in its ability to detect vulnerabilities.  
  
I did find that these two tools complemented each other however. Where one lacked, the other complimented and working hand in hand the two served their intended quite well, despite failing to detect 3 of the four major vulnerabilities.  
  
Of the two, despite it not being maintained I found the VCG won me over. However, with it not being adequately maintained I would not be able to rely on it going forward. There for FindBugs wins out.  
  
The process of investigating both the bugs contained in the program, and the static code analysers proved both engaging and highly informative. I learned valuable lessons in relation to the whole process of bug testing and detection and terms of the importance of making the right selection of tools based on the testing of tools.

# References

1. Dinesh2043: https://github.com/dinesh2043/CyberSecurityProject.