Secure Software Development Coursework

SET10113

40169846
HARRIS ASLAM

Table of Contents

Secure Coursework Part 1

1. Fixing the access control implementation

In this first section access control needs to be implemented into the NapierUniPortal Application. Access control allows for the features of a system to be restricted to certain authorised users. In this particular program the restrictions will be placed around viewing and manipulating the Courses, Enrolments and the Program pages of the website.

In order to do so an access control strategy needs to be chosen. To complete this particular problem, I will implement the Role Based Access Control strategy. RBAC allows for there to be specific roles defined to methods, so that only users authorised with that particular role can execute the functionality of those methods. This means that in order to execute a certain feature in the program the user must have permission from their assigned roles. RBAC is very simple and quick to set up and maintain and it is also supported by most frameworks, making it perfect to be implemented in this application. The table provided in the requiremenrts showed how the access control should be restricted and that is exactly what was implemented.

In terms of coding the RBAC, at the beginning of a class the @EnabledGlobalMethodSecurity annotation needs to be set to true as displayed below

```
 */
@RestController
@RequestMapping("/api")
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class EnrollmentResource {
```

After this the @PreAuthorize tag is used to determine who should have access to this particular feature. In this application the Admin is the one who has total control over the system and the user (non-admin) has limited control to make this split the following annotation is used.

```
@PostMapping("/enrollments")
@Timed
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<Enrollment> createEnrollment(@
    log.debug("REST request to save Enrollment : {}"
```

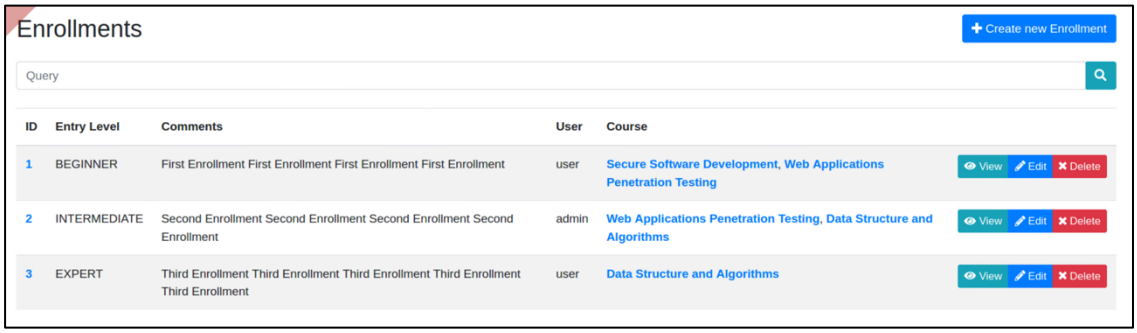
This code @PreAuthorize(“hasRole(‘ADMIN’) is now giving only the Admin the right to the method which allows for new enrolments to be created. This piece of code is added to several methods which only the admin has permission to use.

The @PreAuthorise annotation has been coded into methods within the CourseResource, ProgramResource and EnrollmentResource as it is here that the access can be manipulated. From these Roesources the user is restricted from

performing any actions they shouldn't be able to. These restrictions were implemented in compliance to the table provided above.

Here is another thing that needed to be changed as the enrolment before was finding all the enrolments in the database whereas it needed it to be only showing the ones for whatever user was using it at that time. So, a user should only see their specific enrolments whereas an admin can see all the enrolments.

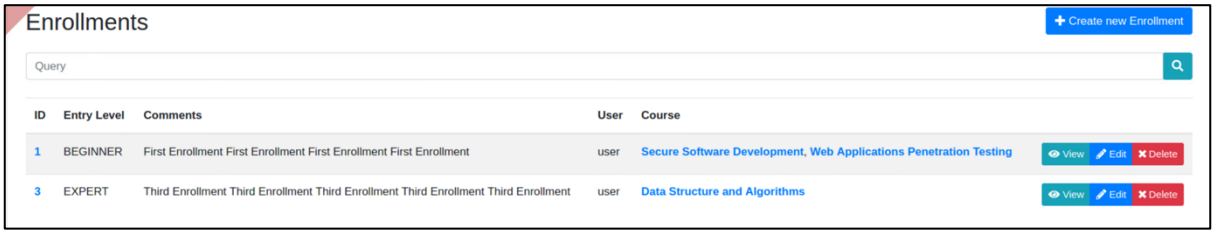
There was one other restriction that was required to be implemented and that was ensuring that the user of the system should only be able to see the enrolments they are currently enrolled in and not any other ones, whilst ensuring that the admin could see all the enrolments, before any change to the code was made the below snapshot shows what the user could see in the enrolments page:



As you can see the user is able to see enrolment 2 which should not be visible to them. So, to restrict the user from seeing this I implemented the following code:

```
@GetMapping("/enrollments")
@Timed
@PostFilter("hasRole('ADMIN') or filterObject.user.login == authentication.name")
public List<Enrollment> getAllEnrollments(@RequestParam(required = false, defaultVa
    log.debug("REST request to get all Enrollments");
```

The postfilter code allowed for the access to be split around who was on the system. By using this line of code, the user was now only able to see their own 2 enrolments.



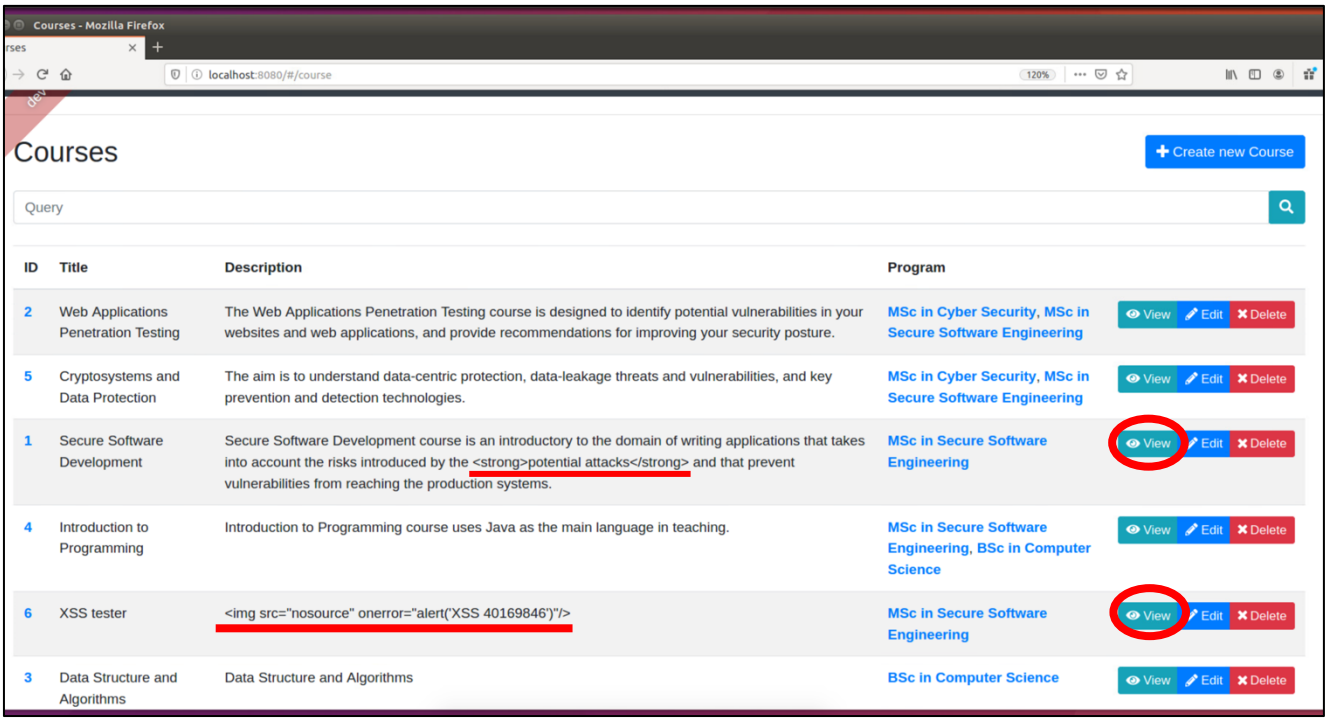
The access control strategy was now implemented and fully functioning.

2. Identifying and fixing XSS vulnerabilities

In this section XSS vulnerabilities had to be found this was carried out through a manual code review. In order to find an attack, it is best to look at the areas where an attack is most penetrable. There are 3 types of XSS attacks Persistent XSS, Reflected XSS and DOM-based XSS.

The way it was found was looking through the sections which were most likely to be attacked. These would be the best way to get access to the database and attack it. There are a few different ways to be able to attack the system with different types of XSS vulnerabilities such as lack of output escaping, lack of filtering.

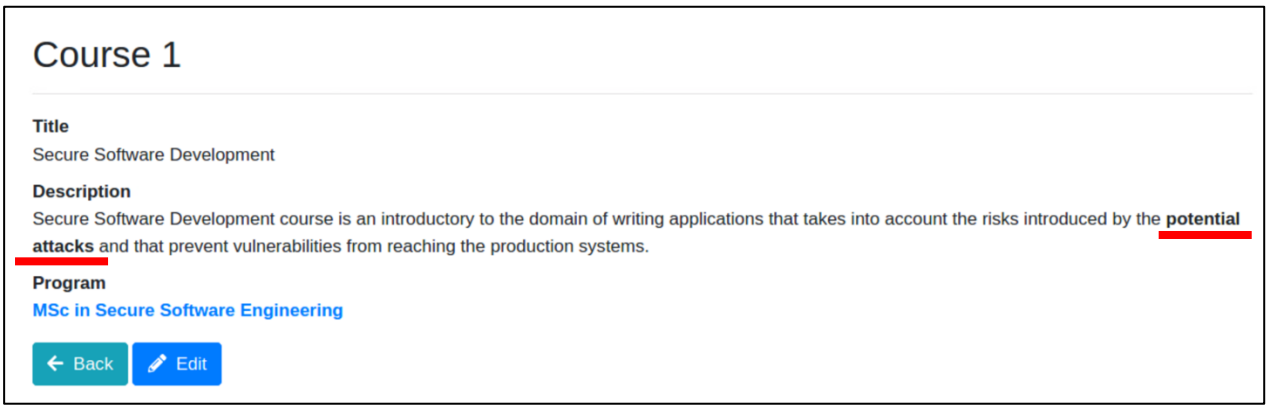
The first vulnerability found was a Persistent Attack. This is when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way. In this particular program to prevent this from happening Output Escaping was required to solve the attack. This is a server-side issue and was solved by implementing the following code within the createCourses and getCourse methods. This vulnerability was given a score of 7. This meant it was of high medium risk as a hacker could implement XSS scripts straight into the program executed by the button to view a particular course. The vulnerability was potentially dangerous and needed to be fixed.



The red lines indicate where XSS vulnerability is. It is executed when the view button (red circle) is pressed.

Persistent Attack Vulnerability 1

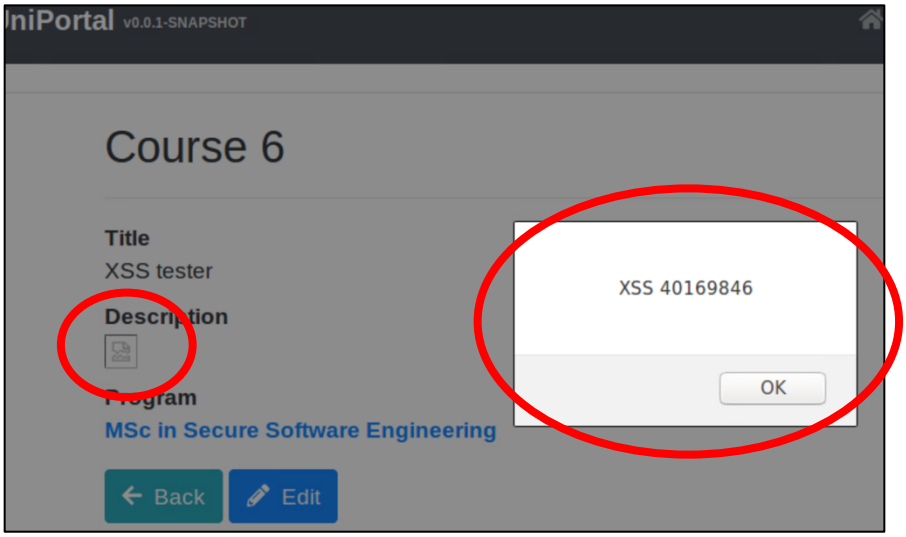
Here I show what it looks like when the view button is pressed, and the vulnerability is executed on the 1st course.



The text “potential program” has been bolded.

Persistent Attack Vulnerability 2

Here I show what it looks like when the view button is pressed, and the vulnerability is executed on the 6th course.



As you can see an XSS attack popup box appears.

To solve this particular vulnerability, I needed to implement a server side fix through Output Escaping into the createCourse method which means when a courses is created this attack would just treat the code as normal text and not malicious code, I then also implemented Output Escaping code in the getCourse method as there may have been methods already created with the attack in place so to ensure that any text was read as plain text and not code ensures the program stays safe and protected against any risks.

Code

createCourse fix

CourseResource.java | EnrollmentResource.java | ProgramResource.java

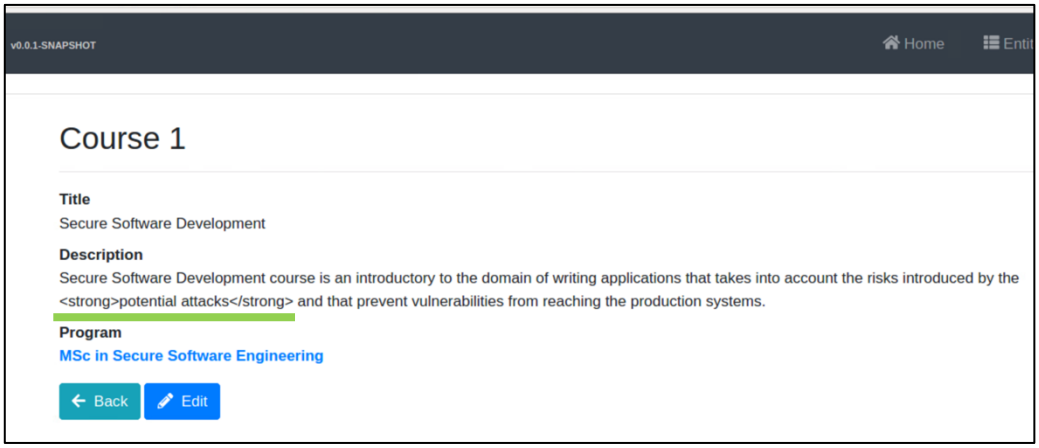
```
@PostMapping("/courses")
@Timed
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<Course> createCourse(@Valid @RequestBody Course course) throws URISyntaxException {
    log.debug("REST request to save Course : {}", course);
    if (course.getId() != null) {
        throw new BadRequestAlertException("A new course cannot already have an ID", ENTITY_NAME, "idexists");
    }

    String encodeTitle = Encode.forHtmlContent(course.getTitle());
    course.setTitle(encodeTitle);

    String encodeDescription = Encode.forHtmlContent(course.getDescription());
    course.setTitle(encodeDescription);

    Course result = courseRepository.save(course);
    return ResponseEntity.created(new URI( str: "/api/courses/" + result.getId()))
        .headers(HeaderUtil.createEntityCreationAlert(ENTITY_NAME, result.getId().toString()))
        .body(result);
}
```

getCourse fix

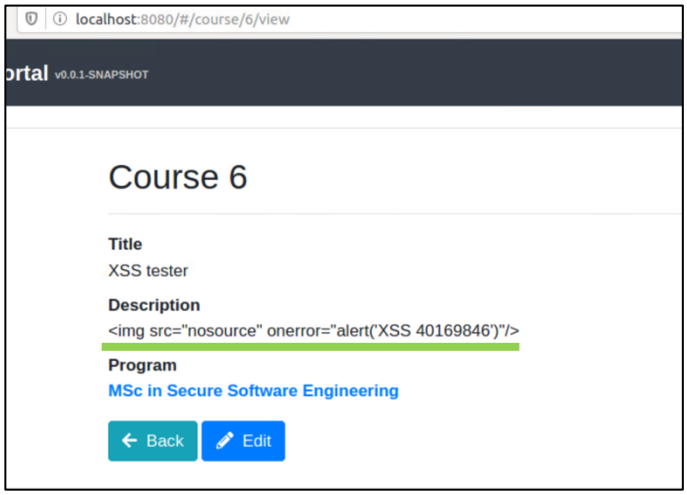
CourseResource.java | EnrollmentResource.java | ProgramResource.java

```
*/
@GetMapping("/courses/{id}")
@Timed
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<Course> getCourse(@PathVariable Long id) {
    log.debug("REST request to get Course : {}", id);
    Optional<Course> course = courseRepository.findOneWithEagerRelationships(id);

    String encoded = Encode.forHtmlContent(course.get().getDescription());
    course.get().setDescription(encoded);

    return ResponseUtil.wrapOrNotFound(course);
}
```

Persistent Attack Vulnerability 1 fixed

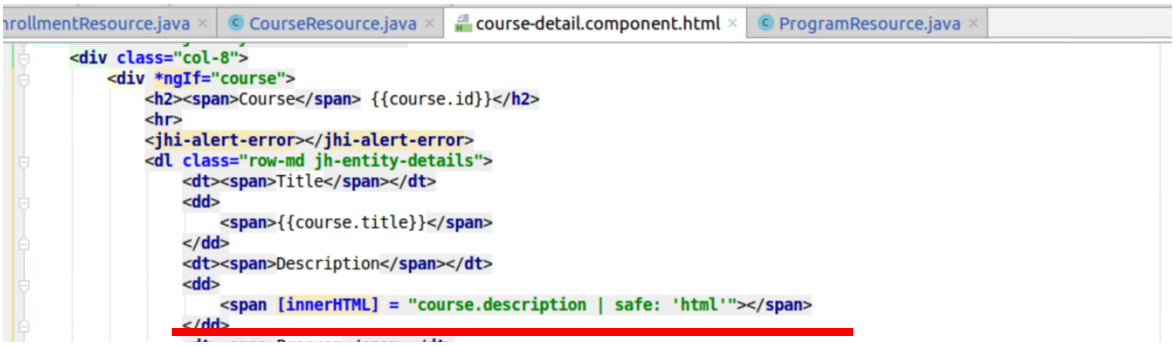


Persistent Attack Vulnerability 2 fixed



The second vulnerability found was another XSS attack that was to be solved through Output Escaping. This vulnerability was a client-side issue it was solved by amending the current code in the course_detail_component.html file. This vulnerability had a score of 5 which makes it a medium risk and it is an HTML_Safe issue.

The current vulnerable code looks like this:

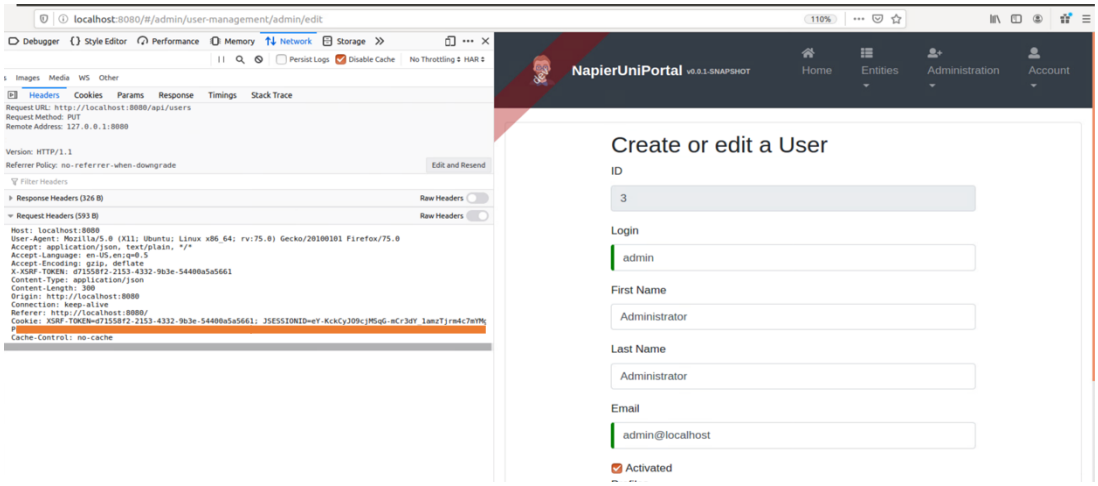


This code works best when it is used in situations where the input from the user is validated and can be trusted by the system. However as it is not, in this program this code allows for any input into the course description section to be treated as if it is safe which can be quite harmful to the system. This means the system won't run all the required checks to ensure the input is actually valid. In doing this the program has been exposed to an XSS vulnerability.

In order to solve this the above code should be replaced with the safer code below which will now no longer just pass the code as safe.

3. Identifying and fixing CSRF vulnerability(ies)

Cross site request forgery attack is a type of attack when a fake request can be sent perceived as if it is coming from the user who is actually on the website. If we take a look at the NapierUniPortal application we can see that there is a CSRF vulnerability. When we access the inspector element and look at the cookies that are being sent, we can clearly see that the CSRF token is within the cookie being sent through a HTTP request meaning it is not secure. A hacker could take advantage of this by using this token to manipulate the system and because the token in valid the system will approve of it as if the user has made the request not knowing the requests are actually malicious. The issue with implementing the prevention mechanism into this page is that this could lead to the problem getting worse. When you implement a HTTP referral header this means that browser information can be revealed. The use of a HTTP referral header could also be susceptible to plugins which modify the header leaving it open to more malicious attacks.



To solve this issue I recommend using the a Double Submit Cookie Pattern. In this method the token is not stored server side. This is when a random value is sent in both a cookie and as a request parameter, with the server verifying if the cookie value and the request value are equal. While sending the session cookie, the server generates another cookie for the CSRF token and sends it to the client. So that when a request is sent to the server, the server will compare the CSRF token in the request body (CSRF token in the hidden field) and the token in the cookie to see whether both of them are matching or not. If both the tokens match then the intended state changing operation will get executed successfully. Since both the CSRF tokens are from the client side this is known as Double Submit Cookie Pattern.

4. Identifying the linkage between vulnerabilities

Now that we have identified there are XSS and CSRF vulnerabilities within the program it is important to understand whether there is a potential link between then both and determine if this can be used to the advantage of the attacker and why.

These two attacks are different when it comes to the problems they can amount to, their popularity and the ease in which they can be performed. However, when XSS and CSRF attacks work together the results can be very damaging to a system. As we are aware there was a XSS vulnerability within the course description input field, and we know that XSS attacks can only exploit the trust of a victims browser when there is a vulnerability within the system. In this system as the tokens are stored in cookies, hackers can use XSS attacks to obtain the tokens which are required to make harmful requests. XSS attacks can record a token and embed this in malicious attacks which allows for CSRF to be carried out.

So if the view button in courses is selected and malicious script and its request would be run. This means that if there is any vulnerability in the program, then the entire system is under threat. Even if there is valid CSRF measures in place they will be useless.

5. Identifying and fixing SQL Injection Vulnerability(ies)

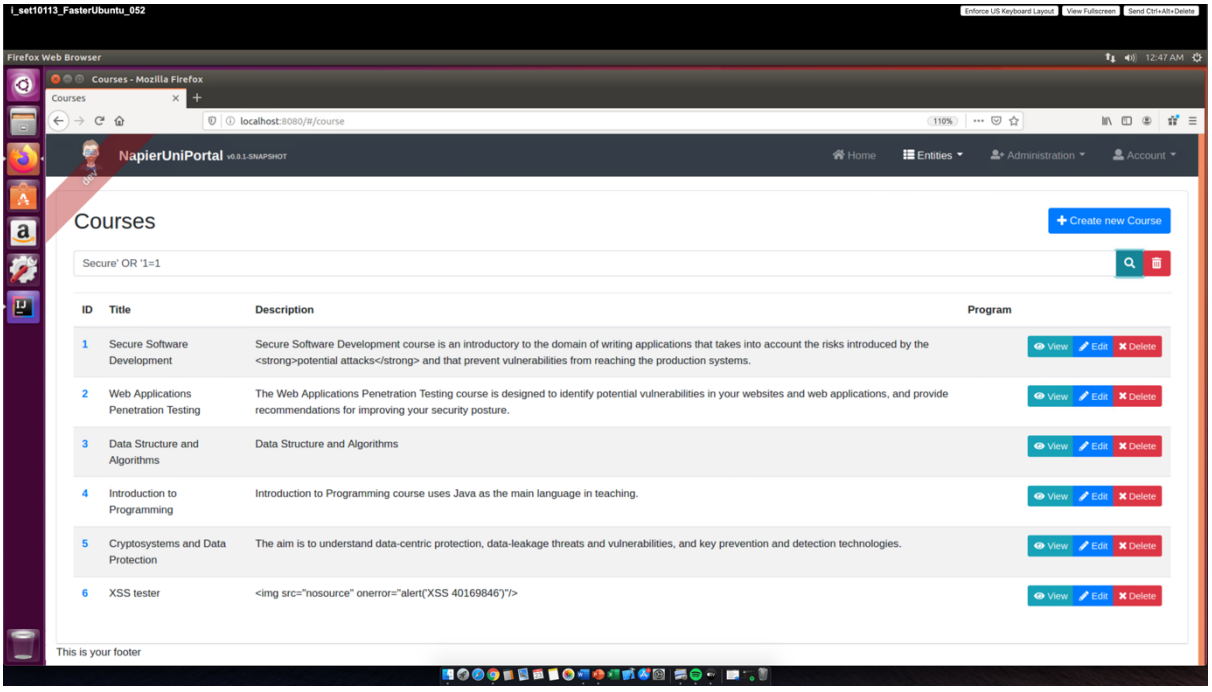
Here I will look to fix SQL Injection Vulnerabilities. In order to find the vulnerability I had to do a manual code review. This was carried out by entering queries into the input boxes where possible in the program. Two areas were found to be vulnerable and this was the course search field and the enrolment search field. I found this by entering the following code into the text fields:

(any relevant word) OR ‘1=1

What this code does is if there is an vulnerability then all the information in that section will be displayed on screen to the person who executed this code. The first word changed per page I typed this code into as I wanted to make it relevant to that page. So for example in the courses page I used the word secure as below:

Secure OR ‘1=1

The SQL injection vulnerabilities were found in both course and enrolment search functions, there were issues found where the code would send back all the information found in the database as can be seen in the screenshot.



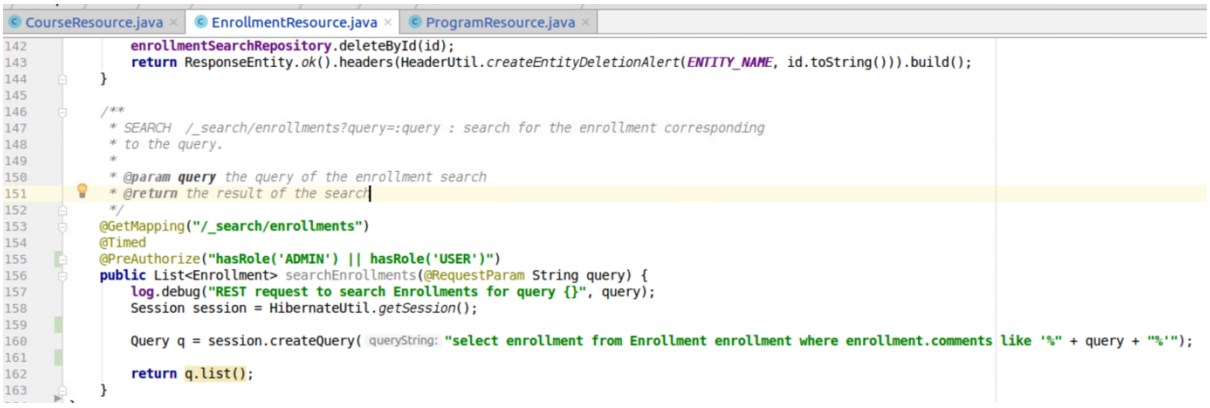
SQL injections are of very high-risk as they allow access to information stored within the database. This kind of weakness needs to solved right away and is can be very dangerous leaking information meaning it gets a risk score of 9. Due to its high score the vulnerability needs to be solved as soon as possible so that the database is more secure.

In order to solve these issues I had to look at the search method within the CourseResource and the EnrollmentResource files.

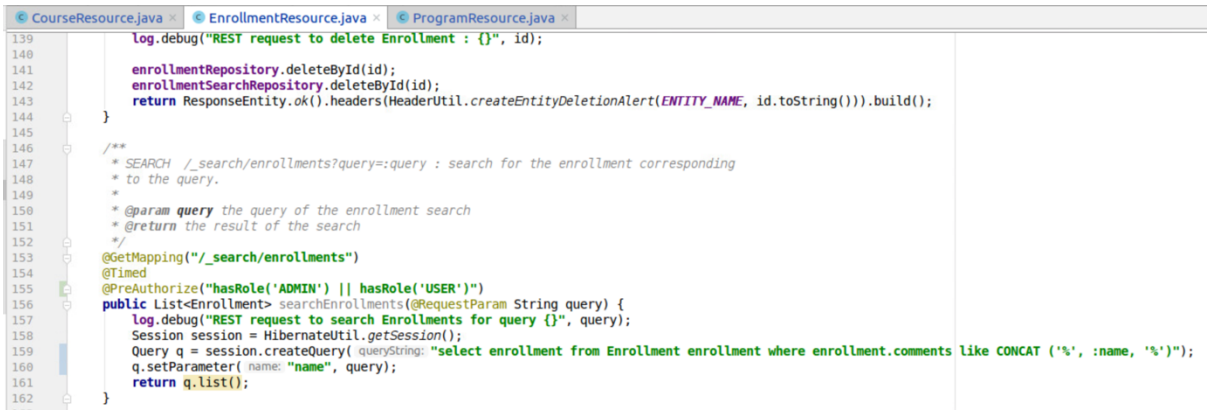
EnrollmentResource

The SQL injection within the Enrolment's page was fixed by using the hibernate method with parameter. This method intercepts the query, once it has done so it will build the parameter making it a dynamic way of fixing any injection. By setting a parameter the SQL statement cannot be modified by an attacker. While HQL is more difficult to exploit than normal SQL injection the risk is still there.

The code before:



The code after:

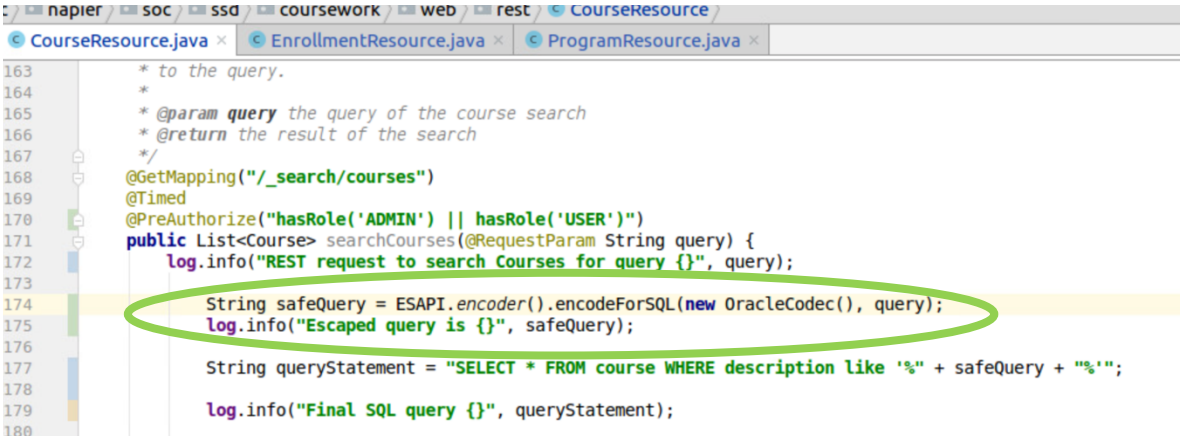


CourseResource

In this method we can see that that the issue was arising due to the query not being encoded properly. To fix this I implemented the following code:

String safeQuery = ESAPI.*encoder*().encodeForSQL(**new** OracleCodec(), query);

This ensures the query is now encoded and passed through as normal text instead of a SQL query. This code will now help prevent SQL injections.

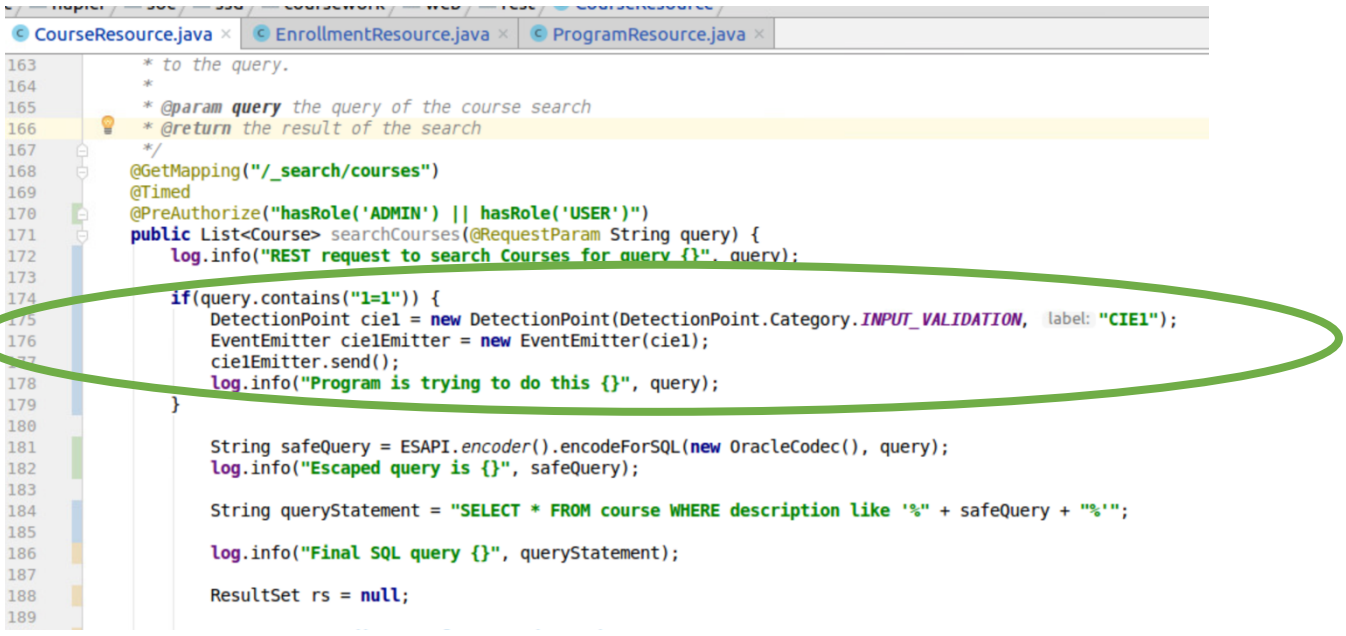


6. Implementing an attack detection mechanism

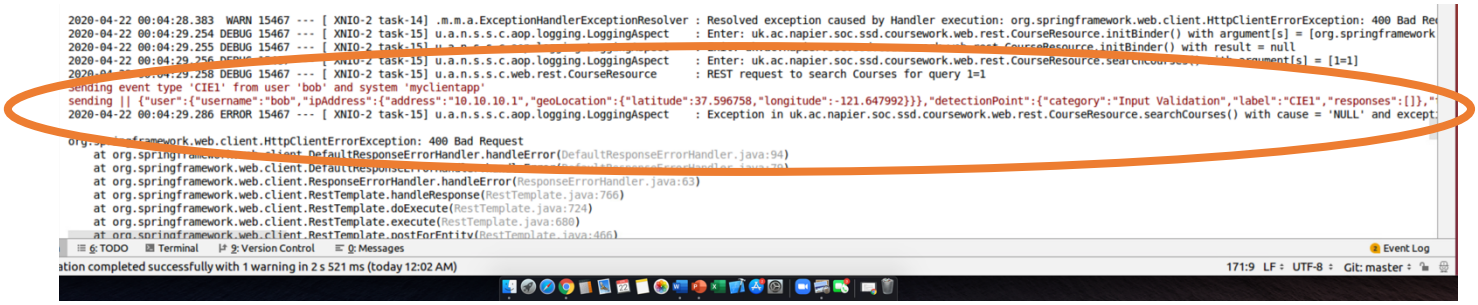
In order to limit the scope, let's focus on detecting the SQL injection attacks. One possible way to implement the detection mechanism is by using the OWASP AppSensor Framework.

In this section we look at implementing an attack detection mechanism so that malicious individuals can be identified before they can identify weaknesses in the system. As per the requirements I have focussed on SQL injection attacks. To implement such a detection mechanism I used the OWASP AppSensor Framework as guided. This framework keep track of a user who is trying to manipulate the system through SQL injections. When a user tries to perform an SQL injection attack the framework will categorise the type of event in accordance to its detection point (CIE1). From here the AppSensor framework can consider a series of these events as an attack, for the example of this program if a user tries to perform an event 5 times within 20 seconds the framework will report this as an attack. It does this through the use of a dashboard to which an the analyst will have access to and any issues will be flagged in the Input Validation category, so the analyst can monitor from this dashboard.

In order to implement the OWASP AppSensor Framework in this program, I needed to run the AppSensor Framework and the AppSensor UI within the terminal so that the user interface (where the analyst can access the dashboard) would be displayed. Within the NapierUniPortal program and more specifically within the *CourseResource.java* page I will search for a particular string that is occurrent in SQL injection queries and that is **("1=1")**. If the user has tried to submit this code, then a detection point will be flagged and then the user information will be sent via the EventEmitter class and HTTP Client to the AppSensor Servers.



After trying very hard to implement the AppSensor Framework I managed to get it to understand that the input (“1=1”) was not valid. As can be seen in the below screenshot, the application recognises that due to the input the CIE1 detection point has been flagged and the information about the user is being sent across the server. However, during the process it is met with a bad request and so the framework does not fully work as intended despite the event correctly being flagged by the system.



7. Providing your thoughts

In terms of discussing what went well I think that the website is fully functioning it does as it is supposed to with little issues. In terms of security there were some obvious flaws within the system but these were not very difficult to find and solve, a few XSS and SQL vulnerabilities in such a program were expected but due to some testing and manual code reviews these were found and fixed. Setting up user access control was also a fairly straight forward process as the user roles were clearly defined so implementing an access control system was easy to do.

In terms of what was wrong with the program there was quite a lot. We start off with the access control, because this was not in place from the start launching an application as it had been could have really affected the product negatively if this had not been picked up on by allowing users admin level control over the system. Also, the fact the program was susceptible to attacks goes to show that even when delivering the minimal viable product, you still need to make sure that the relevant checks have been made as any weaknesses in a system will not only question the integrity of the program but also the company. One thing that went wrong was the implementation of the attack detection mechanism, it proved very difficult to figure out how to solve this issue so not getting this completed despite having implemented this in another program proved frustrating as it would have been hugely beneficial to the security of the system. The good thing being however that it looks unlikely the program should be affected by SQL injection attacks now.

In terms of for the future, it would be beneficial to be given access to some automated testing tools which may be able to look for any vulnerabilities quicker. However, it is important that this goes hand in hand with manual review testing as automated testing doesn’t uncover all issues within a program. These checks should be completed as several stages throughout the development process and this ensures that any issues are found and dealt with much earlier in the development lifecycle. Another major help would be to ensure there is vigorous internal commentary within the program. This will help developers and testers really understand what the code is doing and why it is there.

Secure Coursework Part 2

BSIMM (859 words)

Methodology Description

Software security is about building software to be secure especially when it is under attack. Through many years of reviewing security breaches we have come to the conclusion that when software is built with a priority around security then protecting software becomes a much simpler task. Furthermore, software security being impervious to hacks is not just the process of adding techniques such as encryption or passwords to the program.

BSIMM is short for Building Security in Maturity Model. The BSIMM is a study of real-world software security initiatives (SSIs) organized so that companies or businesses can compare where they stand with their SSIs against other businesses and how they may go about enhancing their security for future developments.

The BSIMM has been built entirely from observations made by studying real software security initiatives. The report is not a one guide fits all, so it does not restrict companies in saying this is what you need to do for secure software. Instead, the BSIMM is a large report which provides insight into what the rest of the world is doing in this area (Building Security in Maturity Model, 2020). The current version BSIMM10 represents the latest evolution of this detailed and sophisticated “measuring stick” for software security initiatives.

Applicability/suitability for the agile development methodology

When we look at the BSIMM and the agile methodology together we would assume they would marry together well as they both look to evolve over time however, considering the difference in speed at how they evolve proves they may not marry all that well. As we know about agile it is about flexibility and fluidity, in the agile development cycle there are increments to which systems are built upon. It can be manipulated and updated without worry for quick change. However, on the other hand we have the BSIMM which is one large report which gets updated once a year and this does not compare when we think of agile standards. Due to this if a situation arises where companies have implemented changes to their security profile the there is little way for other companies to know about this and implement these changes within their own development cycles should they deem it appropriate. Instead they would have to wait until the report at the end of the year comes out to be able to gain insight into this knowledge. This means keeping up with other companies in terms of security could be difficult.

Promises

As mentioned earlier by having security at the forefront of software development it allows for safer products to be created. This in turn will save the product money in the long run whilst also providing their users with confidence in the system. The promises are that whoever chooses to use this model will be able to access really valuable data which has been collected by real world companies. Allowing for there to be clarity and transparency across security standards and their practices. In doing so, we are allowing for companies to gain first-hand knowledge of tried and tested measures to ensure software is secure and less susceptible to attacks from hackers. Acting like a benchmark for companies which in turn can mean better more reliable software.

Cost

There is no cost to buy the BSIMM. It has been released to the general public to use as a framework to security success. This makes it a very viable option for companies who want data that is trusted about their industry and their industry practices. This allows them to stay up to date in current practices whilst not having to worry about the expense of doing so. If the company chooses to use BSIMM fairly and provides the proper references and accreditation then the model can be used however they please. The other benefit of the BSIMM being cost free is that small companies can see and implement security features that top companies are doing. Again allowing for there to be transparency in industry software standards.

Drawbacks

Whilst there are many positives to take from the BSIMM it is clear that the methodology does have some drawbacks. One drawback is that even though the BSIMM report may make companies aware of the security initiatives used by other companies, what it lacks doing is showing whether or not these same measures would be fit and just for your company as it is not a personalised report. Another limitation is that the report is released once annually, this is a draw back as we spoke about earlier that technology and software changes rapidly in this current day and age. To wait a whole year before seeing an updated version of this information is quite a long wait and can often. Lead to their being and mismatch in security standards between companies. Whilst the companies who are able to be proactive and keep their security standards high it could be detrimental to lesser companies who don’t have the power to do so. Another issue is that because the report is free this allows hackers to gain information about what security measures are in place by different companies which may provide them with information on what skills they would require to try and compromise peoples systems.

References

Bsimm.com. 2020. *Building Security In Maturity Model | BSIMM*. [online] Available at: <https://www.bsimm.com/> [Accessed 12 May 2020].

Synopsys.com. 2020. *Building Security Maturity Model (BSIMM) Consulting Services | Synopsys*. [online] Available at: <https://www.synopsys.com/software-integrity/software-security-services/bsimm-maturity-model.html> [Accessed 12 May 2020].