**Full Metal Library Security Document**

**SD6503 Testing and Secure Coding Final Project**

Dave Sushames, Sachith Samarasinghe, Shontelle Steeg

Table of Contents

[Security Document 3](#_Toc210769981)

[A07:2021 Identification and Authentication Failures 3](#_Toc210769982)

[What was done to fix it: 3](#_Toc210769983)

[Secure coding practices used: 3](#_Toc210769984)

[AO3:2021: Injection 4](#_Toc210769985)

[What was done to fix it: 4](#_Toc210769986)

[Secure coding practices used: 4](#_Toc210769987)

[A01:2021: Broken Access Control 5](#_Toc210769988)

[What was done to fix it: 5](#_Toc210769989)

[Secure coding practices used: 5](#_Toc210769990)

[Puma Scan 6](#_Toc210769991)

[Figure 1: PasswordHelper on FullMetalLibrary. 3](#_Toc210770036)

[Figure 2: Admin model with validation on FullMetalLibrary. 4](#_Toc210770037)

[Figure 3: Author model with validation on FullMetalLibrary. 4](#_Toc210770038)

[Figure 4: AuthFilter on FullMetalLibrary. 5](#_Toc210770039)

[Figure 5: Puma Scan Results on FullMetalLibrary. 6](#_Toc210770040)

# Security Document

## A07:2021 Identification and Authentication Failures

Identification and Authentication Failures has been identified as a risk to our project, if our database stored plaintext passwords or had weak hashes it leaves it vulnerable to an attack. This can be exploited by an authorised user, it can allow them to gain access through logging in as an admin. The session could be hijacked if it is not managed securely. The apps sessions may not expire or the log out feature might not actually clear the session correctly. This leaves a window for the attacker to exploit the vulnerability and reuse another users session.

### What was done to fix it:

* We used a PasswordHelper class to hash the passwords. We used a hashing algorithm PBKDF2 with a random salt, so if an unknown user gained access to the system, the passwords are not recoverable.
* Sessions have also been protected with a 30-minute timeout and httpOnly cookies. which then means that no JavaScript can read them, or be reused.
* The Logout() method actually clears the session data instantly.

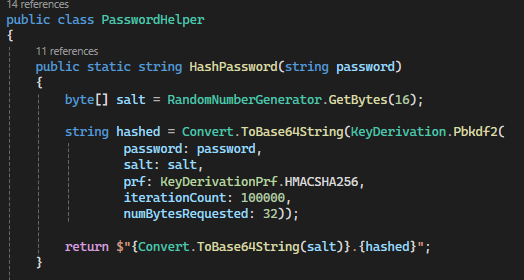


Figure 1: PasswordHelper on FullMetalLibrary.

### Secure coding practices used:

* Strong password hashing to protect the credentials stored.
* Secure session management to prevent any session attacks.
* Logout and timeout enforcement that will limit the time for an attack.

## AO3:2021: Injection

If user input is not validated it could lead to an SQL injection attack or Cross-site scripting when that input is being displayed. A threat actor could use this to read or modify sensitive information from the FullMetalLibrary app.

### What was done to fix it:

* The project was created using Entity Framework Core for the database, which automates parameterises queries.
* All models use validation attributes such as [Required] and [EmailAddress] that rejects unsafe or malformed data.
* The razor views use automatic HTML encoding, which means that user input is not directly rendered to the page.

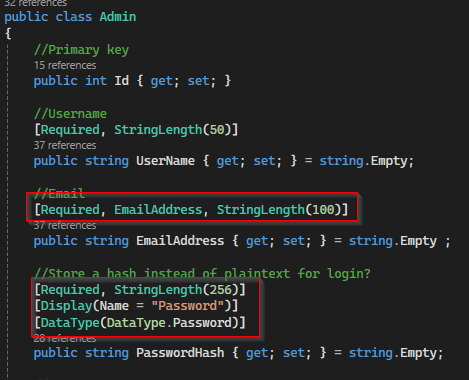


Figure 2: Admin model with validation on FullMetalLibrary.

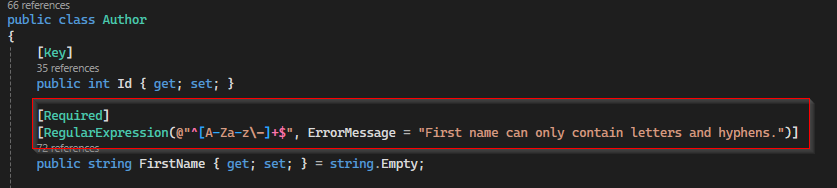


Figure 3: Author model with validation on FullMetalLibrary.

### Secure coding practices used:

* Entity Framework parameterisation to remove the threat of SQL injection.
* Validation attributes to block unsafe data before it can be saved.
* Razor auto HTML encoding to prevent XSS by default.

## A01:2021: Broken Access Control

If the routes or resources from FullMetalLibrary are not properly restricted, an unauthorised user could access the admin pages or perform actions that they should not be able to do.

### What was done to fix it:

To address this risk, the project uses a AuthFilter to protect all restricted routes. Before someone can execute a controller action, the filter checks if the current session has an active admin user. If they do not, it then redirects straight to the login page which then prevents any unauthorised access.

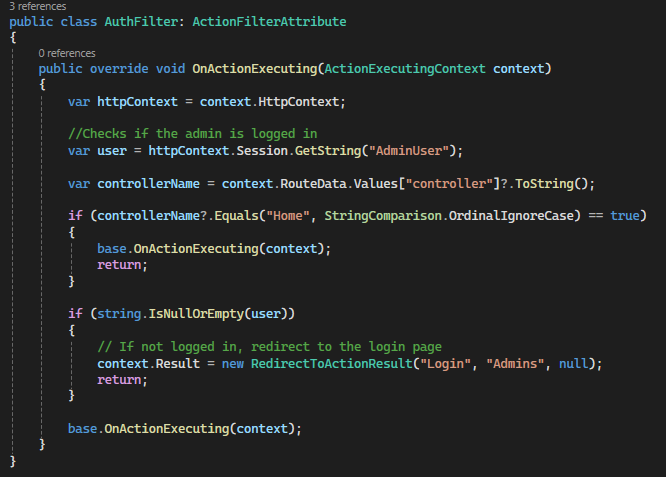


Figure 4: AuthFilter on FullMetalLibrary.

### Secure coding practices used:

* Access control enforced at the controller.
* Session based authentication checks before it loads a sensitive page.
* Default redirect to Login if the user has not been authenticated.

## Static Code Analysis with Puma Scan

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5: Puma Scan Results on FullMetalLibrary.

The purpose of Static Code Analysis is to detect issues with the code security and quality. To run the scan, we used the Puma Scan 2022 extension for Visual Studio. Puma Scan checked the C# files for any syntax issues and rule violations.

The scan returned the following vulnerabilities:

Use collection initializers or expressions (IDE0028):  
Elements are manually added to a collection instead of using an initializer or expression to do it. Using an initializer/expression would improve readability and performance.

Remove unused parameter (IDE0060):   
There was a method that included parameters that weren’t being used in it. This just adds needless complexity and can obfuscate the method’s purpose.

Namespace does not match folder structure (IDE0130):  
The file’s namespace doesn’t match its actual folder structure. This makes the code harder to navigate, and makes organization of code inconsistent.

Use ClassCleanupBehavior.EndOfClass with the [ClassCleanup] (MSTEST0034):   
There is no EndOfClass label at the end of the test class, which can make test cleanup unpredictable.

SYSLIB diagnostics for regex source generation:  
GeneratedRegexAttribute would generate/compile the regular expression implementation at build time rather than run time, improving performance and startup time.

**Interpretation of scan results:**   
None of the issues identified were a security vulnerability, and instead impact code quality and reliability. The most important one to fix in this case is the MSTEST0034 issue, which impacts testing reliability.

**Optimization plan:**Fix the fukn issues?