Technical Report 1

Web App Security

Team ThisIsUnSAFe:

Richard How, Quang Nghiep Ly, Giahuy Truong, Sriram Viswanathan, Zexian Wu

Due date:

23rd May 2020

TABLE OF CONTENTS

1. LIST OF ACRONYMS

Acronym	Term
ART	Agile Release Train
CORS	Cross-Origin Resource Sharing
GDPR	General Data Protection Regulation
HSTS	HTTP Strict Transport Security
JWT	JSON Web Token
MFA	Multi-Factor Authentication
ORM	Object-Oriented Relational Modelling
OWASP	Open Web Application Security Project
PI	Private Information
PII	Personally Identified Information
SPMD	Student Project Management Dashboard
SQLi	SQL Injection
XSS	Cross-Site Scripting

2. INTRODUCTION

The ThisIsUnSAFe team is tasked with investigating security for web apps. This technical report is a deep dive into 6 major web apps security threats, specifically the vulnerabilities the SPMD, are susceptible too. The report aims to provide a clearer and more transparent threat landscape in which the software will be operating in addition to putting forth recommendations to minimize risk exposures. Vulnerabilities are chosen based on their prevalence, criticality, and relevance to the current project, and overall web apps. Each vulnerability is thoroughly examined in their nature, exploiting pathways, potential impacts, and mitigation strategies. In addition, a mock scenario is incorporated into each security risk to elucidate and further reinforce their reason for presence.

3. SCOPE

This report provides an extensive evaluation of each security vulnerability documenting:

- The nature of the vulnerability,
- Their relevance to the project,
- Its impacts from a successful attack, and
- Common exploits performed to leverage the vulnerability.

For the above security vulnerabilities, specific mitigation solutions were researched. These threat-specific solutions are then aggregated based on categories of software that can be developed individually to provide a more compact view on the suggested security measures.

Each category is assigned a score based on the number of security threats that it mitigates. The highest-ranking categories serve as the basis of ThisIsUnSAFe team’s response and recommendations to the System Architect cross-team concerning security challenges SPMD would likely face in the future.

4. SECURITY THREATS

4.1. Injection

What is Injection:

An injection is a web security vulnerability where unsanitized data is sent to the database in a command or query. Threat actors can leverage this loophole to transmit malicious code embedded in the query to trick the database interpreter into performing unintended unauthorized/illegal operations.

Why this is important for our system:

SPMD is a web app and SQLi is one of the most common exploits leveraged by threat actors to perform malicious activities to web apps. In addition, SPMD uses a NoSQL database which is a major attack vector of SQLi.
Moreover, SPMD will ultimately be operating at a university scale, handling and accessing outstanding amounts of data, including PI and PII. This further reinforces SQLi importance to our project development given that successful exploits can result in significant data complications. Direct consequences involve loss of data, data corruption/data quality issues, and disclosure of data to unknown third parties. Indirect consequences can range from regulatory fines, to loss of reputation or even a potential shutdown of the project.
These implications are especially of concern to Monash given the tertiary institution established a reputation globally. Moreover, such impacts can generate major disruptions to the organisation's operations and shutdown of the current project is likely due to its small and trivial nature.
In addition to organisation-scale repercussions, the vulnerability can cause students to lose their project work, rendering a huge disruption to a student's unit work. An increase in plagiarism is novelly possible if SPMD has no protection against SQLi and infiltrating the database is straightforward.
Statistics-wise, the vulnerability has always been part of the top 10 web security risks according to OWASP [1]. As reported by Akamai, SQLi alone accounted for 51% of web application attacks in 2017 [2], more than ⅔ (77%) in 2019 [3], and overall 72% between December 2017 - November 2019 [4]. This illustrates that SQLi is extremely prevalent and is becoming increasingly common, indicated by the sharp increase of 51% within the 24 months.
Therefore, SQLi is of great importance to our project due to our app nature, toolchain, and the environment in addition to the threat's impact and prevalence.

Example: NoSQL injection for MongoDB
A threat actor generates and executes this URL:

http://studentprojectmanagementdashboard.com/login?user=admin&password[%24ne]=

The database being attacked is assumed to store usernames and passwords of SPMD users.
Additionally, we assume that there is a user named "admin".
The database interpreter looks at the query "user=admin&password

%24ne]=", parses it and deducts the following:

- "user": "admin"
- "password": {"&ne": ""}
    - "%24ne" is converted in the database to "&ne".
    - "ne" in MongoDB language is "not equal"
    - There is no input after the equal sign at the end of the query so the interpreter sees "" as the inputted password.

MongoDB will check if the password in the database is different from the inputted password which is an empty string. This attack allows the actor to bypass the login and gain access to the system as an admin.

4.2. Broken Authentication

What is Broken Authentication:
Broken Authentication is listed as one of the top 10 vulnerabilities by the OWASP.

So, what exactly is Broken Authentication? Flaws in the implementation of a web application's authentication system or failing to correctly implement authentication and/or management of sessions could often lead to nefarious attackers gaining access to a user's account. [5]

Threat agents for this exploit usually are:

· External malicious attackers
· Insiders who impersonate others

Why this is important for our system:
Bad implementations of session management and authentication systems have caused this vulnerability to be common. In fact, according to OWASP, Broken Authentication has ranked 2nd in the list of top 10 security vulnerabilities for web applications from 2013 to 2017, which goes to show how hackers find it easy to gain access to a user's account. [6]

The SPMD system, in the initial release, will be made available to only Monash Students and IT faculty members, which will require students/Staff to log in to the system using their Monash email address. If an attacker is successful in breaking into a user's account, they will have access to all their project information, and all the other applications that are integrated with the account. Additionally, this could mean that the attacker could access all the resources linked to the student/staff's Monash account, and also access other sensitive information related to the user. It's even worse in the case of admin accounts. Admin accounts usually have access to a wider range of resources and have a higher level of authority, and hence, the attackers will only need access to the admin account to compromise the whole system, which will have a high impact.

Example: Broken Authentication with bad session ID management
Let's suppose that Anna, a Software Engineering student at Monash University, wants to monitor the progress of a high priority project that's due this week. She wants to discuss the project's progress to her teammate, Emily. Emily and Anna are teammates for 3 other units this semester. Anna copies and sends Emily the URL from her web browser because Emily had forgotten the URL for the SPMD.

URL sent by Anna to Emily:- http://studentprojectmanagementdashboard.com/FIT9999-TeamKitKat?sessionid=123

As you can see, the URL has the session ID in it, which is a bad way of managing the session because:
1. When Emily uses the same URL, she will have logged in as Anna, and any changes she makes will be registered as changes made by Anna in the dashboard.
2. An attacker/or another competing team could randomly try different session IDs and could stumble onto an active session ID, giving them access to another team's project details.

4.3. Sensitive Data Exposure

What is Sensitive Data Exposure:
Sensitive data exposure is the un-authorised accessibility and interpretability of records, credentials, personal data, and other important data which often require protection as defined by laws or regulations such as the EU GDPR or local privacy laws. [7]

This occurs as the result of inadequate defence for data stores, such as databases and caches and also insufficient protection for data in transit to and from the application on an insecure network. Sensitive data that do not receive extra protection in these cases can be exposed by attackers and exploited for malicious purposes. [8]

Why this is important for our system:
Sensitive Data Exposure in our application may lead to student emails, passwords, marks, or other sensitive information being leaked. Since we are using REST APIs to access Okta authentication mechanisms, this can be problematic since the SPMD may link the accounts to Monash using Okta OAuth2. An insecure connection with Okta may expose Monash staff and/or student credentials to others in the same network. [9]

If sensitive data is exposed the attacker will have information that can be sold or used for malicious purposes.

Consequences can cause significant impacts including fines from regulatory obligations, loss of reputation to the organisation, the potential shutdown of the project. The ramifications of this exploit can go beyond the life and scope of the project itself. [10]

This is one of the most useful exploits for attackers over the past few years due to weak or lack of encryption. Which goes to show how effortless hackers find it to gain access into a user's account. Sensitive Data Exposure is capable of being detected by automated scanners but usually requires manual patching to protect software against the vulnerability. [11]

There are many tools that can assist an attacker in retrieving sensitive data. Some tools techniques that can be used to detect Sensitive Data Exposure are: [12]

- Packet sniffing
- Auto-complete
- Browser memory leaks

A web application may have vulnerabilities if the following well-known weaknesses are not addressed while developing it: [10]

- Can any data be transmitted from the server/database in cleartext or with bad encryption. (eg. Lack of HTTPS on authenticated pages,)
- Is any data stored unencrypted or as clear text in the database or server. (eg. Hashed passwords with lack of salt, making the password easily cracked)
- Can data be accessed from users who should not have access to it. (eg. Tokens disclosed in public source code (see the second well-known event))

Example: Sensitive Data Exposure with lack of HSTS:
Since our application will require user accounts, our browser will need to communicate with the applications server to negotiate the creation of a session. When data is in transit, it is capable of being intercepted which will require encryption to protect it from being interpreted. [13]
Let us say for example we have a site
http://StudentProjectManagementDashboard.com/login

Without the HTTP header, HSTS

Strict-Transport-Security: max-age=<expire-time>

Our browser may be able to access the site using HTTP instead of HTTPS. This is problematic for a login page which will transmit usernames/emails and passwords. HTTP will transmit data unencrypted, which may allow attackers to potentially get the usernames/emails and passwords. HSTS is one of many HTTP security headers which can help protect the client from exploitation. HTTP security headers are often overlooked or misconfigured. [14]

4.4. Broken Access Control

What is broken access control:

Access control, sometimes called authorization, is how web applications control what content and functions should be accessible to different users[15]. Broken Access control occurs when users can act outside of their intended permissions. This would lead to allowing attackers to steal information from other users, modify data and perform actions as other users.

Two common categories are horizontal privilege escalation and vertical privilege escalation [16]. Horizontal Privilege Escalation occurs when a user can perform an action or access data of another user with the same level of permissions. Vertical Privilege Escalation occurs when a user can perform an action or access data that requires a level of access beyond their role.

Why this is important for our system:

Broken Access Control greatly depends on what kind of information or features the attacker can gain access to. This can be anything from seemingly useless information to a full system takeover. It can leave applications at a high-risk for compromise, typically resulting in the impact of confidentiality and integrity of data[16]. It is bad for our system because of the following reason.

Easy to detect and take advantage of the vulnerability:
If an attacker attempts a specific action that should require authentication and the request succeeds, the page is considered vulnerable.

High prevalence:
Broken Access Control is often a problem. It usually happens when

- application size gradually grows.
- no design at the beginning and add schemes in the middle
- lack of automated detection
- lack of effective functional testing by application developers.
- access control not centralised, which results in a very complex scheme and leads to mistakes and vulnerabilities

Which satisfy almost all of our situation

Lose fairness:
In the student project dashboard. If this happens, attackers may get the assignment downloaded and copy or sell it without permission. They may change the data and history storing in the application which will affect the mark.
Lose confidence:
Attackers may delete all the projects in the system. This may require students to redo the work and cause the system and the university to lose trust from its students and teaching staff.

Example:
<u>Horizontal Permission Issues</u>:
Imagine when a student logs in to the SPMD using their account details. When the student views the application, the browser requests the web-server for project detail.
As IT students, they probably know how to get the request simply by opening the developer tool in the browser. The student will observe the following request made by the application when loading the student dashboard.

https://studentprojectmanagementdashboard.com/dashboard?studentId=abcd123

Students will be able to view other students' dashboards by changing the studentId to another person.

<u>Vertical Permission Issues</u>:
SPMD will also have an admin role that allows the admin to fix things that are broken. It may have the ability to search a database of all users and get users and projects' information, this feature should not be available to students and some of the teaching staff. The attacker discovers that this feature exists through javascript that looks similar to below.

```
<script>
        var isAdmin = false;
        if (isAdmin) {
          ...
         var adminPanelTag = document.createElement('a');
        adminPanelTag.setAttribute('https://studentprojectmanagementdashboard.com/admin');
          ...
        }
</script>
```

The attacker may go to this link and edit the system as an admin. Moreover, this admin path is easy to guess. Attackers may easily get the admin path by guessing. Without permission checking, it will be very insecure.

4.5. Security Misconfiguration

What is Security Misconfiguration:
Security misconfiguration in web app security is the incorrect implementation of security features in the web application. This can give a false sense of safety within the program and leave exposed areas within the application that can be exploited.

There are both manual and automated methods to find and exploit security misconfigurations. Security configuration mistakes may happen on any level of an application and it is not difficult to find. While companies can integrate security practises to defend their software, the human error in development has begun to be more prominent in allowing exploits to occur.
[17]

Possible reasons for security misconfiguration include: [18]

- Development constraints (time and/or cost)
- Inexperienced development team in terms of secure coding
- Bad patches, causing the app to downgrade over time

Why this is important for our system:
The SPMD is undertaking a microservice approach to development. This provides individual teams with the ability to pick their languages and libraries for their parts in the application, allowing more opportunities for security misconfigurations. Due to AWS implementation, the cloud service must be properly configured to protect the application. [19]

If an FTP, RDP or Telnet port is implemented in the SPMD, it can give unauthorised access to resources in the development or production server.

Consequences may vary from assignment or project work being leaked to complete control over the SPMD server.
Typically a successful exploit using security misconfigurations are unlikely to provide the attacker with complete control over the application. Nevertheless, the consequences of security misconfigurations are unauthorised access to parts of the application and sometimes complete system control.

Since the impacts of a security misconfiguration being exploited in our application are varied. It is difficult to state how important security misconfigurations can be for the program. Possible impacts may include plagiarism due to students or staff gaining unauthorised access to other profiles and projects, deletion or alteration of work for users or projects, or addition of work that did not happen. [20]

Example: Security Misconfiguration with insecure ports
Our application will require the transferral of information from our applications server to the users' browser. This means that our server has to have an open port to transfer the data the user requires. Some ports are well known, with predetermined functions associated with them. These ports can be quite secure or vulnerable based on how they are configured.
Let us say for example we have a site
http://StudentProjectManagementDashboard.com/

However, there is an open port on port number 21 which allows FTP access to the server. If the FTP server allowed anonymous login then files from the applications server may be accessed remotely. [21]

There are a few security misconfigurations in this example: [22]

- Port number 21 should not be open to the public, it should be placed behind a firewall or limit IP addresses using a whitelist (preferably) or a blacklist.
- FTP access to the server should not allow anonymous login.
- Since the application may be communicating using an insecure network, it would be better to use SFTP (port 22).

4.6. Cross-Site Scripting (XSS)

What is it:
Cross-Site Scripting is a malicious code injection attack that is performed on the client-side. This attack typically involves the execution of a malicious script on a victim's browser when loading an otherwise legitimate web page or web application. [23]

Why this is important for our system:
Vulnerability points for XSS appear whenever an application uses data provided by a user without first sanitizing or validating the input. These vulnerability points can, therefore, appear wherever user input is implemented. XSS would likely be a prevalent issue for the SPMD as there are multiple vectors where this can occur. From user inputs such as comments, entering a file name, or even the user's login details, if improperly handled, these can all become avenues for XSS to occur.

While there are multiple applications of XSS, if a successful attack was to occur, the consequences are typically the same with the main difference being where the payload is attacking. The common consequence of XSS is session cookie theft. Through this session cookie, the attacker can perform further actions such as session hi-jacking and potentially account seizing. Other than this, XSS attacks can also be used to retrieve the victim's sensitive data or forward the victim's login credentials to the attacker. [24]

An XSS attack on our application may result in an attacker gaining access to a student's or staff member's session by stealing their session cookie. This would be problematic for the SPMD as the system intends to have high traceability of actions. Under the guise of a victim's session, an attacker could perform various malicious activities that would be traced back towards the victim. This would make it possible to not only damage the integrity of a student's project but also direct the blame towards the victim.

Through XSS, an attacker could also steal the login credentials of a legitimate user. Using this information, it may become possible to access other Monash services since the SPMD may link its accounts to Monash. This could lead to a further compromise of sensitive data.

Example:
There are three applications of XSS: Stored, Reflected, and DOM-based.

Stored:
Stored XSS otherwise known as Persistent XSS or Type 1 revolves around a vulnerable site storing malicious scripts which are later executed when the victim's web browser requests this 'safe' data.

An example of this would be an attacker inputting the following malicious payload into a comment section.

```
<script type="text/javascript">alert("You were hacked by ThisIsUnSAFe")</script>
```

This payload would be stored in the system's database doing nothing for now. If this inputted data is not rendered for safe usage, a victim can attempt to access this data thinking it was a 'safe' comment. The malicious payload would then be retrieved and executed on the victim's web browser resulting in the alert message "You were hacked by ThisIsUnSAFe" popping-up.

Reflected:
Reflected XSS is otherwise known as Non-Persistent or Type 2 revolves around the malicious script being reflected to the victim as part of a page from a vulnerable site. This is usually performed by having the victim click on a malicious link which would send the malicious payload to the web-server. This web-server would then reflect the attack back towards the victim's web browser where it is then executed. [25]

An example of this would be using the following malicious link for a legitimate but vulnerable web application.

```
http://StudentProjectManagementDashboard.com/get.php?origin=<script type="text/
javascript">alert("You were hacked by ThisIsUnSAFe")</script>
```

If a victim was to click on this malicious link, the payload would be reflected from the web-server. The victim's web browser would then execute the script and an alert message "You were hacked by ThisIsUnSAFe" would pop-up.

DOM-based:
DOM-based XSS otherwise known as Type 0 revolves around the malicious script being directly executed on the client-side using the web application's client-side processing. For the previous two XSS approaches, the malicious payload would be implemented onto the response page produced by the web-server. DOM-based XSS, however, instead alters the DOM environment itself rather than the web page resulting in a different page execution. [26]

An example of this would be if the airline web application, ThisIsUnSafeAirlines, could change the displayed language to Italian through the URL:

http://StudentProjectManagementDashboard.com/page.html?language=Italian

This feature could be useful for a web application that was used on a global scale as it could format the web page into the desired language. This could be done without a request to the web-server which would reduce the server load. If, however, this web application was vulnerable, the following malicious link could instead be used.

http://StudentProjectManagementDashboard.com/page.html?default=<script>*alert(“You were hacked by ThisIsUnSAFe”)</script>*

If a victim was to click on this link, the web browser would create a DOM object with the document location object containing the malicious payload. This would result in an altered DOM environment that would execute the payload during the runtime of this flawed script.

5. SOLUTIONS

5.1 Solution Categories

Category A: Access Control

Solution A1:

Use of an access control matrix to define the access control rules. Documenting the security policy with types of users and what functions and content each of these types of users should be allowed to access

Security Threats Addressed: Broken Access Control

Solution A2:

Verify every single request with this central application component in order to decide whether the request from the user is permitted to access the resources. Information should be classified appropriately based on their use and level of confidentiality and sensitivity.

Security Threats Addressed: Broken Access Control, Sensitive Data Exposure

Solution A3:

Access permissions should by default always be denied, so that developers will reduce the probability of accidentally giving user permissions which shouldn't be given to the users when coding.

Security Threats Addressed: Broken Access Control

Category B: Authentication Layer

Solution B1:

We need an authentication layer for each service as a middleware that receives all the requests with Okta JWT token. The authentication layer will verify the token to get user information from okta and get permission for the user which is stored in the student project dashboard database. Then it sends a request to the backend and sends a response back to the frontend. This backend only trusts this authentication layer by using the security group mentioned above.

Security Threats Addressed: Broken Access Control, Broken Authentication

Category C: Automated Testing

Solution C1:

Make sure software is up to date using automated update checking. Outdated software is likely to have vulnerabilities that require the patches to resolve, with the exploits that can be found from online sources.

Security Threats Addressed: Security Misconfiguration

Solution C2:

Implement automated and/or manual checks for software misconfiguration. Security misconfigurations are part of the application itself, automated CI testing is necessary to ensure that changes to the code do not cause values in the application to become exploitable. The same automated scripts to find these misconfigurations for exploitation can also be used by developers to find them in their own applications. For example:

- Nikto
- Nessus
- Nmap

Security Threats Addressed: Security Misconfiguration

Solution C3:

Currently available online are multiple security tools that can help detect XSS vulnerabilities in a web application. This can be extremely helpful for individuals who are not experienced with security vulnerabilities.

Security Threats Addressed: XSS

Category D: AWS Security Groups

Solution D1:

By setting the security group in AWS, we can limit the access to API, so that only the frontend can call the API and people outside cannot use the API even though they find it. Similar to API, we can set the security group to the database so that the database is only accessible by some of the API. Other people don't have the permission to connect to the API.
For the frontend, we may need to set the administration page only accessible when under Monash VPN / network.
In considering the cost on AWS, each team will use their own AWS account to use the free tier. It will be hard to add a security group between API and frontend as other teams may need to use the API as well. If we verify the JWT token in the frontend and call the API, users will be able to see and use the API.

Security Threats Addressed: Broken Access Control

Category E: Secure Database and Storage Design

Solution E1:

A LIMIT clause in addition to other SQL controls can be embedded within queries to prevent mass record disclosure in case of a successful SQLi [27]. A LIMIT clause is used to set an upper limit on the number of results to be returned by SQL. Enforcing a mechanism to limit exposure can help reduce the severity of the impact.

Security Threats Addressed: Injection

Solution E2:

Do not use cache for storing user information. If users permission is updated and is not updated in the cache, it will cause the broken access control issue. In addition, client browsers sometimes automatically caching the website. If students login to our system using the public computers, there will be a risk that the next person can access the other's information.

Security Threats Addressed: Broken Access Control, Sensitive Data Exposure

Solution E3:

The database can be encrypted using a strong hashing protocol (e.g. SHA512). This allows exposed data, in case of a successful attack, to be unreadable/indecipherable and serves as a final defence line to the system's data.

Security Threats Addressed**:** Sensitive Data Exposure

Solution E4:

Store non-duplicate data on different databases so that if one database is breached, not all data is lost and also data exposure is low. This follows the principle of "Don't put all your eggs in one basket".

Security Threats Addressed**:** Sensitive Data Exposure

Solution E5:

If confidential info is not needed, it should be archived. This reduces the likelihood of confidential data being compromised via regular database transactions, and overall minimizes its exposure to the outside.

Security Threats Addressed**:** Sensitive Data Exposure

Categories F: Input Sanitisation

Solution F1:

All user input should be sanitized or validated before being stored or used. If there is an expected type of user input, a whitelist could be used to ensure only acceptable values are being parsed to the system. While a blacklist can be used, they often tend to be fragile to a malicious attack as it is difficult to escape all dangerous characters.

Security Threats Addressed: Injection, XSS

Solution F2:

Parameterized statements are a means of separating the query structure and the input query. This method allows a SQL statement to be pre-compiled and the input/query needed are just parameters/"variables". Hence, the issue of string concatenation which is the main culprit of SQLi is greatly mitigated as there is only a need to supply the "parameters" and not the complete query.

Security Threats Addressed: Injection

Categories G: Limited and Secure Communication

Solution G1:

Use the Object-Oriented Relational Mapping technique. The technique allows query and manipulation of data using an object-oriented paradigm where entities are treated as classes. ORM reduces the need for constructing SQL queries and overall poor SQL queries. Using a library supporting this technique allows communicating with the database to be quick, simple and sanitizing at the same time as database commands are written in a style of functions similar to the use of parameterized statements.

Security Threats Addressed: Injection

Solution G2:

Administrator access restriction with IP address e.g. only under Monash network or Monash VPN to protect users outside the world.

Security Threats Addressed: Broken Access Control

Solution G3:

Rate and set a limit on API calls per time period and controller access to minimize the harm from attackers guessing the token or password using automated attack tooling. This mitigates attacks such as brute force, DDoS, etc.

Security Threats Addressed: Broken Access Control, Sensitive Data Exposure, Broken Authentication

Solution G4:

Remove unnecessary communication with the database. This reduces the chance of database transactions being exploited. This solution prevents security threats from the root by limiting opportunities for attack.

Security Threats Addressed: Injection, Sensitive Data Exposure

Solution G5:

Use HTTPS. HTTPS creates a secure connection where transferred data is encrypted. Thus, even if an attacker gets their hands on the data, they will not be able to read nor modify it. This protocol provides a safeguard for data privacy and integrity.

Security Threats Addressed: Sensitive Data Exposure

Solution G6:

When transmitting data over insecure channels, protect the traffic using encryption (VPN). VPN creates a virtual encrypted channel from the server to a remote server operated by a VPN service. All internet traffic is routed through this tunnel, protecting our public internet connection and our data from the prying eyes. This solution sets an additional encryption layer on top of HTTPS and ensures that our internet connection used for transmission is a secure one.

Security Threats Addressed: Sensitive Data Exposure

Categories H: Manual Testing

Solution H1:

Use the access control matrix to document the type of users and their permission. This can be used as a reference during developing and manual testing.

Security Threats Addressed: Broken Access Control

Solution H2:

Code reviews and manual inspections can greatly reduce the likelihood of deploying poorly structured / vulnerable code.

Security Threats Addressed: Injection

Solution H3:

Ensuring penetration testing is performed in the final stages of development to identify issues not identified during development.

Security Threats Addressed: Injection, Broken Access Control

Categories I: MFA

Solution I1:

Implement the increasing popular MFA system for all logins to the application. The MFA provides an additional layer of security to user accounts. Some of the examples of MFA include soft tokens, fingerprints, facial recognition, voice recognition, etc. [6]

Security Threats Addressed: Broken Authentication

Categories J: Password Policies

Solution J1:

Change default credentials after the user's first time logging in. Default credentials are often publicly available, which allows attackers to find the default username and/or password without needing to enumerate the login interface.

Security Threats Addressed: Security Misconfiguration

Categories K: Secure Frameworks and Libraries Usage

Solution K1:

Frameworks and libraries should be kept up to date. As new possible attack vectors always appear, it is integral that frameworks and libraries being used are updated to have the latest security features.

Security Threats Addressed: XSS, Sensitive Data Exposure

Solution K2:
Use frameworks that have in-built XSS protection measures such as automatic XSS character escape templates. Frameworks that currently provide this include Ruby on Rails and ReactJS. [28]

Security Threats Addressed: XSS

Solution K3:
Avoid calling dangerous API methods and props that bypass in-built XSS protections [29]. While most frameworks provide a work-around for secure methods, these dangerous methods should be avoided otherwise manual sanitization of all data is required.

Security Threats Addressed: XSS

Solution K4:
Okta service and API can help us manage user's information and token so that the student project dashboard does not need to worry about token/cookie management.
Okta has an authentication API which facilitates user login, MFA for our web application. Okta uses a session cookie in order to grant access to the web application. These session cookies are short-lived, and their expiration intervals can be configured by the developers/ admins meaning that the cookies are only valid till:

- they expire
- the user logs out of their account
- the user closes the browser

Security Threats Addressed: Broken Access Control, Broken Authentication

Categories L: Session Management

Solution L1:
Generate random Session IDs, so that if a particular session ID has been compromised, the next generated session ID would not be the same.

Security Threats Addressed: Broken Authentication

Solution L2:
The application could log the user out automatically after a certain interval of time or have the session IDs expire after a reasonable interval of time so that it keeps the user account safe if the previous session ID was compromised.

Security Threats Addressed: Broken Authentication
[6]

Solution L3:
Access will need to be checked and renewed after a period of time by timing out the cookies, access tokens or session IDs. Enhanced screening means earlier detection and better prevention of prolonged unauthorized access.

Security Threats Addressed: Broken Authentication, Sensitive Data Exposure

Solution L4:

Invalidate authentication tokens upon logging out because our system is not for anonymous users. Invaliding authentication tokens avoids attackers use the token to do stuff that is not permitted

Security Threats Addressed: Broken Access Control, Broken Authentication

Solution L5:

Confidential data in a transaction should be available for a set amount of time and should be deleted/unavailable and should be authorized again for access. This prevents prolonged unauthorized data access as well as reducing the chance of data being exposed. After all, the best way to hide data is to not have it at all.

Security Threats Addressed: Sensitive Data Exposure

Categories M: Set Appropriate Security Headers, Flags, Value

Solution M1:

Make sure all appropriate headers and security values are correctly implemented. Incorrectly set security values will create weaknesses within the application that can then be used to exploit the program.

Security Threats Addressed: Security Misconfiguration

Solution M2:

Setting the HTTP Only cookie flag. This flag prevents any client-side scripts such as Javascript's Document.cookie API from accessing the cookie. The cookie is, instead, only accessible by the server [30]. Any XSS attacks would, therefore, not have access to the victim's session cookie even if the malicious attack was to succeed.

Security Threats Addressed: XSS

5.2. Category Score Table

Category	Unique Security Threats	Score
A	Broken Access Control, Sensitive Data Exposure	2
B	Broken Access Control, Broken Authentication	2
C	Security Misconfiguration, XSS	2
D	Broken Access Control	1
E	Broken Access Control, Injection, Sensitive Data Exposure	3
F	Injection, XSS	2
G	Broken Access Control, Injection, Sensitive Data, Exposure	3
H	Broken Access Control, Injection	2
I	Broken Authentication	1
J	Broken Authentication, Security Misconfiguration	2
K	Broken Access Control, Broken Authentication, Sensitive Data Exposure, XSS	4
L	Broken Access Control, Broken Authentication, Sensitive Data Exposure	3
M	Security Misconfiguration, XSS	2

A score is assigned to each category indicating the number of unique security threats the high-level category addresses (i.e. Category M has a score of 2 because it addresses 2 unique vulnerabilities).

6. LIMITATIONS

In the industry, security checks are usually done by hiring a team of experienced software testers. Since this is a student project, the ART doesn't have a budget, which means that this will most likely have to be done by the ART. However, as students, the ART lacks experience and skills in web application security. Therefore, some of the solutions may not be able to be implemented.

Most of the solutions provided for the security threats are the common good practices that the industry follows. So, we can either choose to implement these solutions and reduce the vulnerabilities in the application or choose not to implement them and accept the risk. This is because implementing these solutions would not only be time-consuming but also expensive and laborious.

For each security threat, multiple solutions are required to completely mitigate the risks associated. Furthermore, some of these solutions provide specific responses to a particular security threat. It would, therefore, be impracticable to compare the solutions with each other as they target different vulnerability points.

Since the system is currently still under development it is difficult to determine which specific libraries all teams will be using for the system. it is, therefore, hard for us to analyse all the security issues for the particular libraries at this current point in time.

7. RECOMMENDATIONS

Ideally, all of the proposed solutions above should be implemented into the system's design. With the vast range of security threats prevalent, all of these security vulnerabilities should be considered throughout all development stages of the SPMD. However, this is unlikely to occur because of the team's lack of experience and time. These solutions have, therefore, been categorised to establish the common security vulnerabilities that are inhibited. Using this information, a score has been associated with each category. Based on these results, we have decided upon four security recommendations. While the implementation of all four categories is ideal, a rationale for each security category has been provided below to allow for prioritisation. The recommended security categories are:

Secure Database and Storage Design

- With the proper database design the risks associated with Broken Access Control, Injection, and Sensitive Data Exposure are reduced.
- Has only an initial implementation cost. Once these secure designs are implemented into the system, little security maintenance for this category is required.
- A server-side encryption and decryption implementation is required during development if this category is applied. Currently, most secure storage encryption features are freely available online.
- However, our system implements Okta so it is not necessary to store sensitive information in our local servers.

Limited and Secure Communication

- With proper communication design the risks associated with Broken Access Control, Injection, and Sensitive Data Exposure are reduced.
- Has only an initial implementation cost. Once these secure designs are implemented into the system, little security maintenance for this category is required.
- Implementation of this feature would also reduce usage costs as the number of calls to the system would be minimized.

Secure Frameworks and Libraries Usage

- With the proper framework and library design selections the risks associated with Broken Access Control, Broken Authentication, Sensitive Data Exposure, and XSS are reduced.
- This security category should ideally be incorporated into the system's design early. If this security category is considered at a late development stage, it may become costly to implement.
- Implementation of this security category will reduce manual security measures. As secure Frameworks and Libraries usually account for security vulnerabilities themselves, this would reduce the security responsibilities of the developers. Furthermore, the features and methods provided by these options would likely be more secure compared to developing these security measures ourselves.

Session Management

- With proper session management design the risks associated with Broken Access Control, Broken Authentication, and Sensitive Data Exposure are reduced.
- Session management is not only useful for security against unwanted intrusions but also allows personalised interfaces.
- Making sessions secure is important as cookies are highly sought after targets for attackers, given the number of exploits to gain unauthorised access to a session it can be difficult to implement good session management. Okta shifts part of session control from the developers which will make it easier to develop sessions that are secure.
- Since our system will have personal assignments and projects, making sure that sessions are properly isolated is important so information cannot be unintentionally exposed to students and staff.

8. REFERENCES

[1] "OWASP Top Ten Web Application Security Risks | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed: 18- May- 2020].

[2] Akamai.com, 2020. [Online]. Available: https://www.akamai.com/de/de/multimedia/documents/state-of-the-internet/q2-2017-state-of-the-internet-security-report.pdf. [Accessed: 18- May- 2020].

[3] Akamai.com, 2020. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-a-year-in-review-report-2019.pdf. [Accessed: 17- May- 2020].

[4] Akamai.com, 2020. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/soti-security-financial-services-hostile-takeover-attempts-report-2020.pdf. [Accessed: 17- May- 2020].

[5] "What is and how to prevent Broken Authentication and Session Management | OWASP Top 10 (A2)", Hdivsecurity.com, 2020. [Online]. Available: https://hdivsecurity.com/owasp-broken-authentication-and-session-management. [Accessed: 15 May- 2020].

[6] "A2:2017-Broken Authentication | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication. [Accessed: 16- May- 2020].

[7] "A3:2017-Sensitive Data Exposure | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A3-Sensitive_Data_Exposure. [Accessed: 18- May- 2020].

[8] "Sensitive data exposure: What is it and how it's different from a data breach", Us.norton.com, 2020. [Online]. Available: https://us.norton.com/internetsecurity-privacy-sensitive-data-exposure-how-its-different-from-data-breach.html. [Accessed: 17- May- 2020].

[9] "Securing REST APIs", Okta Developer, 2020. [Online]. Available: https://developer.okta.com/blog/2019/09/04/securing-rest-apis. [Accessed: 17- May- 2020].

[10] "OWASP TOP 10: Sensitive Data Exposure | Detectify Blog", Detectify Blog, 2020. [Online]. Available: https://blog.detectify.com/2016/07/01/owasp-top-10-sensitive-data-exposure-6/. [Accessed: 15- May- 2020].

[11] "Slack bot token leakage exposing business critical information", Detectify Labs, 2020. [Online]. Available: https://labs.detectify.com/2016/04/28/slack-bot-token-leakage-exposing-business-critical-information/. [Accessed: 15 May- 2020]

[12] "Major Ways of Stealing Sensitive Data", Infosec Resources, 2020. [Online]. Available: https://resources.infosecinstitute.com/major-ways-stealing-sensitive-data/#gref. [Accessed: 15- May- 2020].

[13] "OWASP #6 Preventing Sensitive Data Exposure - Part 3 - Lock Me Down", Lock Me Down, 2020. [Online]. Available: https://lockmedown.com/owasp-6-preventing-sensitive-data-exposure-part-3/. [Accessed: 15- May- 2020].

[14] "How important are HTTP security headers ?", Medium, 2020. [Online]. Available: https://medium.com/@SundownDEV/how-important-are-http-security-headers-ad511848eb95. [Accessed: 16- May- 2020].

[15] "Broken Access Control for Software Security | OWASP Foundation", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-community/Broken_Access_Control. [Accessed: 15- May- 2020].

[16] 2020. [Online]. Available: https://www.packetlabs.net/broken-access-control/. [Accessed: 17- May- 2020].

[17] "A6:2017-Security Misconfiguration | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A6-Security_Misconfiguration. [Accessed: 15- May- 2020].

[18] "A Guide to Preventing Common Security Misconfigurations", Infosec Resources, 2020. [Online]. Available: https://resources.infosecinstitute.com/guide-preventing-common-security-misconfigurations/#gref. [Accessed: 19- May- 2020].

[19] Cs.tufts.edu, 2020. [Online]. Available: http://www.cs.tufts.edu/comp/116/archive/fall2019/aepstein.pdf. [Accessed: 19- May- 2020].

[20] "The Impact of Security Misconfiguration and Its Mitigation", Cypressdatadefense.com, 2020. [Online]. Available: https://www.cypressdatadefense.com/blog/impact-of-security-misconfiguration/. [Accessed: 16- May- 2020].

[21] D. Geer, "Securing risky network ports", Network World, 2020. [Online]. Available: https://www.networkworld.com/article/3191513/securing-risky-network-ports.html. [Accessed: 18- May- 2020].

[22] "Is Port 21 Secure? Fully Explained | ExaVault Blog", ExaVault Blog, 2020. [Online]. Available: https://www.exavault.com/blog/port-21-secure-fully-explained/. [Accessed: 21- May- 2020].

[23] "What is Cross-site Scripting and How Can You Fix it?," Acunetix, 16-Mar-2020. [Online]. Available: https://www.acunetix.com/websitesecurity/cross-site-scripting/. [Accessed: 17- May-2020].

[24] "The Real Impact of Cross-Site Scripting - Dionach", Dionach, 2020. [Online]. Available: https://www.dionach.com/blog/the-real-impact-of-cross-site-scripting/. [Accessed: 18- May- 2020].

[25] "Cross Site Scripting (XSS) Software Attack | OWASP Foundation", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-community/attacks/xss/#reflected-xss-attacks. [Accessed: 20- May- 2020].

[26] "DOM Based XSS Software Attack | OWASP Foundation", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-community/attacks/DOM_Based_XSS. [Accessed: 20- May- 2020].

[27] "A1:2017-Injection | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection. [Accessed: 20- May- 2020].

[28] "A7:2017-Cross-Site Scripting (XSS) | OWASP", Owasp.org, 2020. [Online]. Available: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS). [Accessed: 15- May- 2020].

[29] "Cross Site Scripting Prevention · OWASP Cheat Sheet Series", Cheatsheetseries.owasp.org, 2020. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html. [Accessed: 15- May- 2020]

[30] "HTTP cookies", MDN Web Docs, 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. [Accessed: 14- May- 2020]