Project Assignments
PassPass

Team Name: White hats

Team Member List:

- Alexander Jones
- Bryce Mao
- Cameron Sumida
- Kirsten-Elise Rensaa
- Lise Marie Nilsen

Table of Contents

A. Introduction

Our application is called “Passpass” and is a web based password manager. Our plan is to create it with a login site and an overview page over the added accounts. The user can create an account, use credentials to log in, and then store all other passwords and usernames safely. The information is stored in a database and only the authorized user should be able to access this information. It is going to be developed in visual studio code as a web based application with the language C# and JavaScript. The database we will be using is MongoDB.

B. Requirements

1. Security and Privacy Requirements

The personal data processed in our application are username, password, email, and a list of the accounts added to the password manager. Having this in mind, it is necessary to have a couple of security and privacy requirements for the application. The first requirement is that the passwords have to be encrypted with strong cryptography algorithms. This way, if a hacker is able to hack our system, they will only see the encrypted passwords. The second requirement is to have access control for the database. The goal with this requirement is that only authorized users are able to access the system, and hence, not let unauthorized people see all the saved user data. The third requirement is to avoid the direct use of the user's input in the SQL queries which will minimize the risk of an SQL injection. SQL injection is a vulnerability where the user is able to influence the queries towards the database.

To be able to have a secure program, we are going to create a system that keeps track of security flaws. The plan is to develop tests in addition to the program itself to, among other things, check the validity of the data. Validation checks include testing that only certain inputs give the user access to the program. This way, it is possible to easily figure out whether the system allows unauthorized login.

2. Quality Gates (or Bug Bars)

Our program has the following levels of security:

Critical	Elevation of privilege ● Ability to obtain more privilege than granted ● Ability to execute arbitrary code ● SQL injection This is the ability to not only access the users PII but also execute possible harmful changes.

Important	Information disclosure ● Attacker can get information from anywhere in the system. This includes the personal information stored in the database. Data tampering ● Modification of the system or user data (usernames, passwords) that is permanent or persistent (persists after restarting the webpage)
Low	Data tampering ● Modification of data that does not persist after the webpage is restarted. It has a low impact on users.

Our program has the following levels of privacy:

Critical	Poor protection of data ● Personal identifiable information stored in the database (passwords), accessible without proper authentication. The PII is critical to protect and the most important privacy aspect of our program. Poor internal data management ● Personal information stored in the database can be accessed and is not restricted to only those who should have the valid authority. For example managers of the website should not have access to any of the PII of the users.
Important	Poor protection of data ● Storage of non-sensitive information in the database (username/email address) accessible without proper authentication. This is stored in the database and should not be accessible.

Only critical and important levels of privacy are defined here as the program's purpose is to protect the user's information. Any information accessed by unauthorized parties is either critical or important.

3. Risk Assessment Plan for Security and Privacy

Our program will store personally identifiable information (PII); therefore, we have to analyze what kind of threat model we have.

Describe the PII you store or data you transfer: The PII that will be stored is the user's email, and in addition, since it is a password manager, an attacker can use the usernames and passwords to get access to several other services and thus, we can say that the usernames and passwords are indirect PII as well.

Describe your compelling user value proposition and business justification: This will have a value to the user in regard to security and saving time. By using a password manager, the user only has to remember one password. Hopefully, this will make the user have longer passwords, use different ones for different applications, and save time by not having to use the ‘forgot password’ functionality.

Describe how users can control your feature: The users can easily register and create an account, and start to save their passwords in the password manager application. If wanted, they can as quickly delete their account which will lead to the application deleting all the passwords tied to that account from its database.

Describe how you will prevent unauthorized access to PII: The password manager will have access control to both the website and its database. SQL injections will also be prevented by not directly using the user's input to get data from the database.

Considering the PII that the application stores, it is important to have a clear strategy on how to assess the risks. First of all, the database needs to be secure to be able to keep the PII private. As already mentioned, the plan is therefore to have access control and avoid SQL injections by validating the input given by the user. This will minimize the likelihood of a risk. In addition, the data will be encrypted while it is stored in the database. This will not affect the likelihood, but it will decrease the impact of an attack.

C. Design

1. Design Requirements

On a surface level, the program should allow the user to easily create an account and start storing passwords. The user should be given the option to make a secure password and store it in the manager, as well as the option to automatically fill in the information whenever logging in to that site. The user should be able to see which sites they have saved passwords on, and which sites they have opted to not save. The user should also have the option to stop using the program at any time, and remove their data to keep it secure.

On a security level, the program should be able to store user information and retrieve that information whenever its use is required. The information must be able to be saved in a database and encrypted to prevent unauthorized use. The information must not be able to be accessed by anyone other than the owner of that information. In the event of unauthorized access to the database, the unauthorized user should not be able to see any data due to the data being encrypted. The program must include ways for the user to modify the data associated with them. This includes making an account for the manager, adding passwords, deleting passwords, being able to manage their account information, and deleting their account and all associated passwords. The program must also have a way of detecting any unauthorized use, whether through verifying valid input methods or checking for invalid logins.

2. Attack Surface Analysis and Reduction

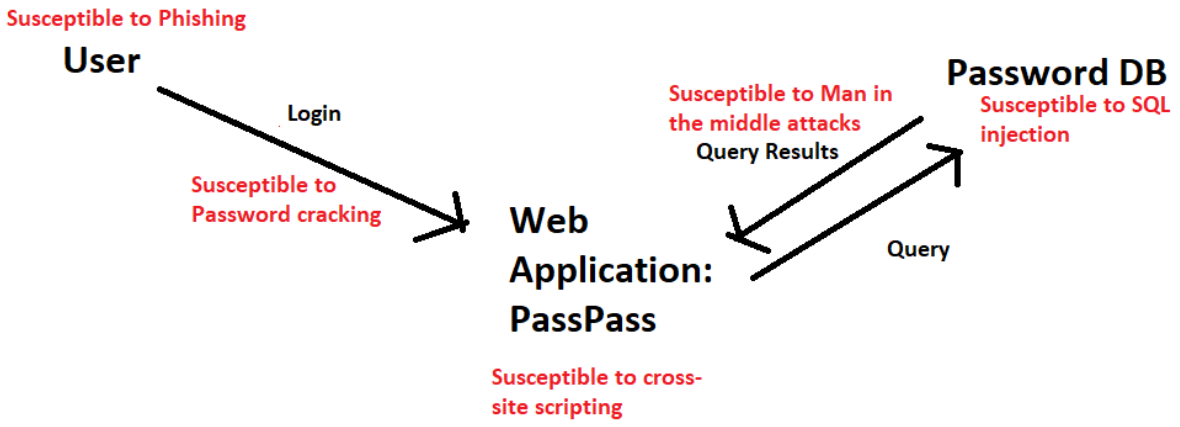
All users should have the same individual privileges as each other. Users will be able to make accounts, add passwords, modify passwords, delete passwords, autofill passwords into sites, and delete their accounts. Users should not be able to view or access any data that does belong to them.

There are a few areas of vulnerability. Storing all the passwords in one place can lead to a risk of losing access to all of your accounts in the event of a data breach. Devices may be stolen or broken into, leading to easy access to passwords without any additional security. Inputting a password into the manager may lead to the risk of keylogging, which may in turn lead to the risk of losing access to the user's accounts. In the event the user loses access to the password manager, regardless of the reason, the user may potentially lose access to any account stored in the password manager. If network traffic is intercepted during login, credentials may be stolen. Sites with malicious code may attempt to steal login credentials or gain access to the database.

3. Threat Modeling

Threat Model:

- Phishing attacks: attackers may attempt to trick users into entering their login credentials on a fake website that mimics the legitimate login site.
- Man-in-the-middle attacks: attackers may intercept network traffic and steal login credentials as they are transmitted.
- Password cracking: attackers may use brute-force or dictionary attacks to guess a user's password.
- SQL injection: attackers may attempt to inject malicious code into the login site's database to steal or manipulate stored data.
- Cross-site scripting: attackers may inject malicious code into the login site or overview page to steal login credentials or perform other malicious actions.
- Session hijacking: attackers may steal or manipulate a user's session to gain unauthorized access to the account overview page



D. Implementation

1. Approved tools

Compiler / Tool	Minimum Required Version and Switches/Options	Comments
C# and JavaScript compiler	Visual Studio Code Version 1.75.0	
.NET Framework	Version 6.0.13	The .NET version is 6.0.13 and the .NET SDK version is 6.0.405.
MongoDB	Version 6.0.4	
Mongosh	Version 1.6.2	
MongoDB Compass	Version 1.36.1	
Roslynator	Version 4.2.0	Collection of 500+ analyzers

2. Deprecated/Unsafe Functions

Below is a list of some unsafe or deprecated functions / APIs that should be avoided in the development of this project;

- eval(), both in JavaScript and MongoDB – very unsafe, since it allows for arbitrary code execution. Since PassPass only performs certain defined functions on the database (retrieving a user's password, storing a new password, deleting the password, creating a new user, etc.), these queries should be pre-created by hand and all of the input to them should be sanitized.
- MongoDB's CR (Challenge Response) authentication mechanism is deprecated – use SCRAM (Salted Challenge Response Authentication Mechanism instead)
- Use MongoDB's WHERE operator – while potentially necessary, be aware if interfacing with it directly from JavaScript, as JavaScript values may not always be initialized and can yield unintuitive results (where studentID == this.studentID, if the latter isn't initialized, may return the first student in the database)

- HTTP protocol for the web app – should be designed to use HTTPS instead, as the web app will be transmitting sensitive user information that could be intercepted
- MongoDB – make sure to create a dedicated user for a web app instead of using the default root one, and potentially have separate users for reading and writing to the database
- Make sure to encrypt passwords stored in MongoDB, potentially using .Net's built-in hashing functions before sending the password to MongoDB

3. Static Analysis

We chose to use Roslynator for this project because it is easily implemented in Visual Studio Code. It is easily downloaded as a packet extension in the IDE. Roselynator includes a set of static analysis tools built on top of the compiler platform. As it contains over 500 analyzers, it is a good fit for the project, as we want to focus on good coding conventions for better security. One of the more advanced features of Roslynator is code analysis for performance and security. It analyzes the source code without compiling it, and helps identify potential problems with code quality, security, and performance. Potential problems can for example be unused variables or redundant conditions. It is useful to detect problems before the code is actually executed. To set it up, we installed the package into Visual Studio Code.

After using it:

The experience using Roslynator has been good. When it detects an error, it immediately appears in the problem tab of Visual Studio Code once you stop typing. Errors are separated by file and contain a short description and the exact line and column of the error, making it easy to find and fix errors. It also gives out warnings for things such as unused variables and unreachable areas of code. Since Roselynator analyzes the code once you stop typing, there may be false positives occasionally, but this shouldn't be a problem as long as you finish typing before looking for errors. One specific success it has, is with duplicated code. It instantly recognizes duplicated code, so that there is no need to look for it. It also suggests solutions on how to refactor or fix it. One downside with Roslynator is that it sometimes can generate a lot of feedback and suggestions, which can be overwhelming if the programmer is not used to it, or used to a different static analysis tool. However, with practice, it is easier to pick out the best suggestions. As the group has focused on getting started and setting up the most important project components for this deliverable, like the database, we have not had the opportunity to test out everything Roslynator has to offer. However, the initial experience is good and we look forward to continuing exploring the tool.

E. Verification

1. Dynamic Analysis

For Passpass, we decided to use the Iroh.js dynamic analyzer. Iroh.js allows one to analyze arbitrary JavaScript code using the concept of stages and listeners; one first initializes the code they want to run into a Stage object, and then adds various listeners based on what they want to observe. For example, if the user wants to examine the value of a variable every time it's modified, they would add the Iroh.ASSIGN listener, which runs some function every time a variable inside of the stage is modified. This code can be as simple as just printing out the value of the variable, but it can be arbitrarily complex, which allows for very powerful monitoring to be done.

However, many issues surfaced as we tried to integrate it into Passpass. At first, it was attempted to integrate it into the render() function, which draws a page using React. However, Iroh needs to have the code be located inside of a Stage object, while React expects to find a standalone function named render(). As such, we were unable to get them both working at the same time. An attempt was made to circumvent the issue by putting the code for the render() function in a separate variable, and then using a stub render() function that just called eval() on this separate variable, but it didn't work, as React didn't preprocess the components located inside of it. This issue can doubtlessly be overcome, but some extra time will be needed.

As such, we then attempted to integrate a simple example into the code, using the first problem from Project Euler as an example, and then using an official example from the documentation as well. Unfortunately, despite extensive modifying, both attempts failed, due to an issue with initializing the Stage object. As it stands, we are evaluating whether the issue is localized to a particular setup, an existing flaw elsewhere in our project, or if the Iroh.js library itself has broken. We may consider transitioning to a different dynamic analysis tool if we are unable to get Iroh.js to work, but we currently believe it will be possible with some additional time and troubleshooting.

2. Attack Surface Review

Vulnerabilities in Tools:

- Visual Studio Code Version 1.75.0 - No new vulnerabilities (last CVE was in August 9, 2022)
- .NET Framework Version 6.0.13 - No new vulnerabilities (last CVE was October 11th, 2022)

- MongoDB Version 5.0.15 - No new vulnerabilities (last CVE was April 21 2022)
- Mongosh Version 1.6.2 - No new vulnerabilities (last CVE was April 21, 2022)
- MongoDB Compass Version 1.36.1 - No new vulnerabilities (last CVE was April 21, 2022)
- Rosylnator Version 4.2.0 - No new vulnerabilities known

Vulnerabilities in Code:

As of now, much of the backend remains to be implemented in code, as a stumbling block was encountered with selecting and setting up the database backend that has delayed implementation. As a result, none of the code vulnerabilities or deprecated functions and APIs were found, except for the use of HTTP instead of the more secure HTTPS (which is known about and will be mitigated before the final release). However, the security advice will continue to be kept in mind and future audits will be performed to verify that the flaws are avoided.

As a modification to the security tips, a dynamic analysis tool has been added to the project. It will be periodically run, alongside the static analysis tool, in order to verify codebase integrity.

F. SDLC: Verification

1. Fuzz Testing

In the course of fuzz testing, three different techniques were tested: SQL (or rather, database manipulation) injection, Cross-site scripting, and a buffer overflow exploit.

For the first test, involving exploiting the MongoDB backend, a line of code that causes MongoDB to print out the contents of the password collection was added into the fields for username, site, and password. If the input to MongoDB isn't sanitized, then this code should get executed, and result in MongoDB spilling out the contents of the entire password database on the command line. However, nothing happened, indicating that the input sanitization is working correctly and that MongoDB isn't executing anything that the user supplies, only storing it.

The next step was to test if JavaScript could be maliciously executed in a similar manner. A console.log statement was used as the input data for the username, site, and password fields, and if either the input wasn't properly sanitized or it wasn't properly sanitized before making it to the user's browser, then a Hello World statement would show up in either the command line (due to nodejs executing it) or in the browser's debug window. Neither case happened - the only thing that occurred was the command showing up in the password view as literal text, indicating that it was being escaped properly.

For the third test, we elected to try a buffer overflow, by injecting a very large string into the site URL field, especially as this could occur in the real world by accident. This was achieved by holding down the "a" button for several seconds, then having the entire contents copied and repeatedly, pasted, copying the new string again and pasting it a few more times, and then repeating that for another time. The end result created a string large enough to cause the browser windows to lag, and upon submitting it and refreshing, it appeared to break the page, causing the list of passwords to move so far to the right that it was no longer visible and making the whole page quite slow. Further testing is required to verify this behavior, but it does appear as though a manual limit on input size will need to be enacted if possible.

2. Static Analysis Review

Our static analysis tool, Eslint, currently only shows a single warning. The warning has to do with a deprecated function inside one of React's dependencies that handles CSS generation, saying that the color-adjust feature should be replaced with print-color-adjust. As this is in a 3rd party library, it is unfortunately not a feature that we can fix, and will need to be handled by the library authors. Other than that, it indicates no further problems.

3. Dynamic Review

For the dynamic analysis section, we unfortunately still do not have a working dynamic analysis tool, though not for lack of trying. Below will document some of the tools we've tried and the reasons we have not been able to successfully incorporate them into our project.

Our first attempt was with Iroh.js, a javascript library dedicated to dynamic analysis. The tool would fit our use case perfectly, but unfortunately does not seem to be maintained anymore, as the last commit was made several years ago. The library, if installed using npm, will crash on an attempted use by saying that the stage() method isn't a valid constructor. This appears to be in conflict with the project's design goals, as it is clearly used as such in the documentation and examples, and even example code copied and pasted directly from the official development repository shows this error. We tried to use the official browser version of Iroh.js, designed to run in a user's web browser directly instead of altering the code on the server's side, but it made no difference.

As a result, we then tried to replace Iroh.js with a dynamic analysis tool called Jalangi2, developed by Samsung. This tool also does not appear to be actively maintained, however; although it has fairly recent commits that appear to add python3 support, the documentation and example code still require python2, a long deprecated version of Python with significant known security vulnerabilities. Additionally, when installing the jalangi2 npm package, npm also warns of 5 critical security vulnerabilities and 7 high concern security vulnerabilities. These two factors together make Jalangi2 a bad fit for our project, as it does not appear to be a secure tool and would greatly complicate our toolchain.

The next attempt was to use a tool called Pex and Moles, a tool made by Microsoft designed for C#. Unfortunately, the tool is actively deprecated; going to Microsoft's official page for it results in a notice that it has been replaced by IntelliTest in Visual Studio 2015.

We then investigated IntelliTest as a suitable replacement. However, IntelliTest requires access to the enterprise edition of Visual Studio, which we do not have access to. Additionally, our project is developed with Visual Studio Code, a different development environment that isn't compatible with Visual Studio.

Therefore, as of now, we do not have a dynamic analysis tool. We will continue to investigate however, and will add one to our project as soon as we can find a dynamic analysis tool that is compatible with our environment and that appears to be actively maintained.

G. SDLC: Release

1. Incident Response Plan

Privacy Escalation Team:
Our privacy escalation core team includes the following roles:

Escalation Manager:
The escalation manager will be in charge of managing the breach from start to finish, They will collaborate with the other team members, assess the situation, and choose the best course of action.

Legal representative:
During the incident response process, the legal representative will make sure that all legal requirements are met. In order to analyze legal risk, identify compliance requirements, and communicate with external stakeholders like regulators, law enforcement, and legal counsel, they will work closely with the escalation manager.

Public Relations Representative:
The PR representative is in charge of handling all outward messages on the breach. To make sure everyone is aware of the issue and that their concerns are handled, they will interact with the general public, the media, and other stakeholders.

Security Engineer:
The security engineer is in charge of figuring out the origin of the breach, estimating the size of the attack, and evaluating the system's security posture. They will collaborate with the escalation manager to put corrective measures in place to stop recurring issues.

The following email address can be used to report any incidents or concerns related to security/privacy breaches: whitehats@passpass.com.
When the first email alerting to the issue is received, escalation should start. The escalation manager is in charge of assessing the escalation's content to decide whether additional information is needed. The following is a set of procedures that should be followed in the case of a breach:

a. Identify the incident: By using monitoring technologies and user complaints, we will swiftly locate and confirm any security or privacy breach.
b. Contain the incident: To stop further harm and to lessen the impact of the breach, we will isolate the impacted system(s).

c. Analyze the Situation: We will evaluate the extent and effects of the breach, ascertain the seriousness of the problem, and pinpoint the incident's primary cause.
d. Team Notification: To coordinate the incident response procedure, we shall communicate with the privacy escalation team and the pertinent stakeholders.
e. Communicate with Users: We will keep users and other interested parties informed of the situation, the steps we are taking to fix it, and any further precautions they should take.
f. Corrective actions and improvements: In order to put corrective measures into place and avert further incidents, we will collaborate with the security engineer. To identify areas in need of improvement, we will perform a post incident review.

2. Final Security Review

Taking our quality gates, threat models, and known bugs into account, it is believed that the rating for the Final Security Review (henceforth referred to as “FSR”) should be either passed with exceptions. As it stands, the application has been tested and is known to be secure against buffer overflows, MITM attacks (due to the successful implementation of HTTPS), and cross-site scripting / SQL injection (as we have verified that input data is properly sanitized). The web service has tentative (though not complete - no permissions are currently applied) support for different user accounts and controls each user’s access to information, and we’ve verified that our application stack doesn’t have any major outstanding security bugs as of the time of this report. We’ve identified a phishing attack as a vulnerability, but unfortunately, we lack the power to do anything about it, other than potentially pursue legal action if the originators of one are located in a jurisdiction that takes phishing attacks seriously.

The main exceptions to our threat modeling have to do with not encrypting user passwords at rest, and that we don’t currently have permissions for different users. Whether this requires escalation or not has to do with the project’s scope - originally, PassPass was envisioned as a remote web service, but currently functions as a local application. In the current context, we consider that reasonably secure (as if a user’s personal computing environment is compromised, an attacker could just install a keylogger or a screen recorder and bypass PassPass completely), but if we were to continue working on PassPass and convert it into a web service, these issues would need to be addressed.

In summary, we believe it passes the FSR in its current state, and in the current context as a local application. If / when PassPass gets converted into a web service, the current exceptions would need to be addressed.

3. Certified Release & Archive Reports

PassPass version 1.0 is a basic password manager that securely stores a user's passwords (or other valuable information) into a convenient and easy to use program. It currently runs on a variety of operating systems, shows a user's passwords in an organized format, allows adding and deleting information, and has tentative multi-user support.

Technical Notes

Installation Guide

- Install MongoDB and .NET on your computer.
- Ensure MongoDB is connected to the correct port by running mongod --dbpath=<directory>, where <directory> is the directory path to an empty folder.
    - Example: mongod --dbpath=user/folder.
- Download a copy of the project and store it somewhere on your computer.
- In a command prompt, cd into the innermost folder labeled PassPass, and run the command dotnet watch run
    - While running, a second command prompt will open.
    - The first time you run it might takea little longer due to some packages being installed.
- Once the command finishes, the app should open by itself in your browser.

Links

Online Repository: https://github.com/Lmnilsen/Passpass

Wiki: https://github.com/Lmnilsen/Passpass/wiki

Initial release: https://github.com/Lmnilsen/Passpass/releases/tag/1.0

Future Development Ideas:

- Implement encryption for passwords in the database
- Implement full multi-user support
- Create an installer, so the user doesn't need to manually install dependencies
- Support the creation of secure notes in addition to passwords
- Allow export of passwords in case a user wants to bulk retrieve them
- Add 2-factor authentication support