A Platform for Managing Security Evaluations

Project Progress Report

SYSC 4907 Winter 2022

Team Members
(Last name in alphabetical order)
Zijun Hu 101037102
Tiantian Lin 101095243
Jiawei Ma 101034173
Ruixuan Ni 101092506

Supervisor
Jason Jaskolka, Ph.D., P.Eng.

Date: January 21st, 2022

Table of Contents

Table of Figures

Table of Tables

Summary

When developers design and develop large-scale systems, the storage and traceability of security evidence usually become a significant problem due to the complexity and vast amount of evidence sources. It leads the follow-up security assessment process to become very difficult. Therefore, a management system is needed to help security analysts collect evaluation data and help developers manage evidence.

This project aims to design and develop a platform for developers to store and manage the security evidence generated during the entire system development lifecycle for future security assessments.

So far, the development of the project is going well. Users can now interact with the system through HTML pages to register, log in, upload and search for evidence and criteria, which achieved the storage and management of the security evidence.

Currently, the project already includes most of the required features. For the rest of the development process, the team will focus on CSS development and polish existing features to improve product usability.

Project Introduction

Background

People are becoming more and more dependent on information technology nowadays. To satisfy people's needs and protect their privacy and assets, corporations pay more attention to system security. There are already multiple well-implemented security mechanisms applied in the working environment, and to evaluate them, multiple security evaluation criteria are made for computer security certification.

There would be a large amount of security evidence produced in the development lifecycle. This evidence can come from different sources for complex or large projects. An evidence management system is needed to manage evidence for evaluation corresponding to criteria.

Tools used for security testing and vulnerability management are already widely used in the system development cycle, but these tools usually pay less attention to evidence of security evaluation. An evidence management tool could help developers store and manage evidence and be convenient for security evaluators and regulatory authorities to verify and evaluate the security of the system.

Objective

The goal of this project is to design an evidence management platform. The platform should allow users to upload and manage evidence and criteria. It should also fit the production environment and keep data safe.

Responsibility

Zijun Hu

I am mainly responsible for designing and developing the project structure, authentication section, and criteria management section during the development cycle.

Jiawei Ma

I am responsible for database-related implementation, including designing, creating, and structuring the database. I am also responsible for implementing the methods for backend logic to store or retrieve the data in the database by using Flask-SQLAlchemy and SQLite.

Tiantian Lin

In the early stage, I designed the user case diagram, access control matrix, and the flow charts for each function, like upload, search, and view. During the development process, I was responsible for implementing the search function and designing the HTML page for the search service.

Ruixuan Ni

I participated in the discussion of the structural design of this project. After the decision was made, I took responsibility for developing some front-end and router logic. I implemented the function of upload evidence and the HTML page for the upload service.

Phases

In the first phase of this project, our group decided to use the Flask framework. Since only one member of our team had experience in web development, we spent some time digging into the document of Flask. We implemented the basic structure of this project with Flask in November. For the server-side, we only have simple features related to registering and login in this version with functions that exist in Flask.

In the database structure and creation phase, we used SQLite3 as the language of our database, which is supported by Flask SQLAlchemy. The database applies object-relational mapping (ORM) to store data and create Flask-related objects.

During designing the database, we created the ER diagram, reduced it to the relation schema, normalized the database, and drew the schema diagram to reduce the redundancy and normalize the database. Finally, our database formed by four tables: user, role, evidence, and criteria.

In the following phases, we implemented login and signup functions based on the new database built to ensure the database could work as expected. Zijun Hu implemented a base HTML file that other HTML pages could reference to ensure all web pages have similar structures. We then implemented router logic for other functions like search and upload.

Report Overview

This report provides an overview of our team's progress in this project from the last term to the beginning of this winter term. This report will provide an overview of our project's background and objectives. We would then provide a detailed summary of the progress of our project, which includes a discussion of alternatives, completed and uncompleted features, and challenges we have met. We would also evaluate our progress compared with our original plan written in the proposal.

Progress Summary

Team's Activities

Until now, the existing project feature can be divided into three main sections:

- Router Logic, which includes the server logic on response to users' requests and router between different HTML pages;
- HTML Templates, which display needed information to users;
- Database, which stores all related data and provides it to the server when needed.

Router Logic

The router logic can be split into three main sections:

- Evidence, which includes the server logic on evidence upload, search and view;
- Authentication, which maintains the users' session, supports login/signup, and provides users' profile management functionality;
- Criteria, which provides criteria management-related features.

Evidence

The following sections are functions directly related to evidence management, including evidence uploading and searching. The evidence view feature is still in development, more detailed information please review the section: *View Evidence*.

Upload

In terms of evidence management, we have implemented an upload form that allows developers and administrators to input the evidence name, description of evidence (optional), and the project name related to this evidence. Uploaders can choose criteria from the criteria list as a tag of this evidence and upload the evidence file. Currently, only text files are allowed to be uploaded as evidence.

Every time an uploader tries to upload a file, the system requests a list of criteria from the database to ensure that the version of the criteria list is up to date. Since we have not deployed the web to the cloud yet, the upload function only saves uploaded files to the local directory and passes the local directory to the database. If a file is not successfully uploaded, the submit button will redirect the uploader to the upload page with a flash of the file not successfully uploaded. If evidence is successfully uploaded, the uploader would be redirected to the view page, which displays the detailed information of this evidence. The server will record the user id of the uploader and pass this information to the database. The database then will record the time of evidence upload and the last edit time of the evidence. Each evidence uploaded to the database would have a specific evidence id which is automatically generated.

Search

In terms of searching for security evidence, we have added several filters to make it easier for

users to search for the evidence they need. In the current version of the management platform, under the search page, we provide users the following search conditions: ID, criteria, project name, uploader, creation time, and last modification time. After users enter those searching conditions, the server will record those conditions and pass those filters to the database to search. Then, the database will return the search result to the server. The HTML page then reads the search results for the server page and displays the results as a list with the evidence name and the corresponding URL link. When we click the link, then we will be redirected to a view page to see the details of the chosen evidence.

Authentication

The authentication section covers multiple features, including signup/login, user management, and profile management.

Login & Signup:

The login and signup feature imports the flask login library and provides user session management for the Flask app. Users can sign up by providing their name, email, and any password. The email is required as the unique. The first registered user will be assigned as the Admin user to make sure the system has at least one Admin user. The password check mechanism now allows any type of password which is not ideal, and some password rules will be developed and added soon. More detail will be provided later in the coming section: *Secure Password*.

The login mechanism will ask the current user to provide their email address and password. If the email exists and the password matches, the system will log the user in and maintain the login session for the current user.

Different User Roles and the Access Control Matrix:

Each user is assigned a role in the system to differentiate user privileges and support access control matrix (ACM). All router logic checks the user's role before redirecting to ensure the page is only displayed to users with the right access privileges. The same check also happens before interacting with the database. *Modify: View + Change

Table 1 Access Control Matrix listed the detailed privileges for each type of role,

Roles\Source	Criteria	Evidence	User Management
Developer	View	Modify*/Create/Delete	N/A
Admin	View/Create/Delete	Modify*/Create/Delete	Modify/Delete
Quality Analyst	View	View	N/A
User	N/A	N/A	N/A

**Modify: View + Change*

Table 1 Access Control Matrix

User management mechanism:

The user management mechanism provides user management services for the Admin users. Admin users can change other users' names and roles through the admin page. They can also delete users' accounts or reset other users' passwords via the page. Please notice that the admin users can only manage users other than themselves. This restriction ensures at least one admin user is in the system.

Profile Management

The profile management feature allows users to change their name and password. Using the profile section in the Welcome Page, users can interact with this feature.

Criteria

Criteria are necessary for evidence management. In the current system, only Admin Users can upload criteria including the name and description. The system would record the uploader's user id to ensure the activities are traceable. The database would also generate a criteria id for each criteria uploaded. Criteria uploaded would be saved within the database and used as tags to evidence.

HTML Templates

In Flask, templates are files containing both static and dynamic data. The template is rendered with specific data and presents the result [1]. In this project, the template is the base.html. All the other pages are developed based on the base.html. In the current base.html, we added the Carleton Logo, copyright, and menu bar, which could direct users to other pages.

There are four kinds of pages in the current management security platform: initial pages, evidence-related pages, authentication -related pages, and criteria manage pages.

Initial Pages

Initial pages include base.html and welcomepage.html.

base.html

The base.html is the initial page that every other page extends from it. In base.html, we check users' roles and show different available options based on their roles' permission. In general, there are four options, as Figure 1 base.html - Menu bar shows, which are Admin, Upload, Search, and Criteria.

Figure 1 base.html - Menu bar

welcomepage.html

The welcomepage.html welcomes and directs users to the login page. After the user login, this page will also display the user's profile. Figure 2 welcomePage.html - Before Login and Figure 3 welcomepage.html - After Login shows the welcome page before and after login.

Before login, users will be directed to the login page, either using the menu bar or the link after the welcome message.

Figure 2 welcomePage.html - Before Login

After login, the welcome page will show the user's profile with their name, email address and the corresponding role. This page also provides a service for users to change their password.

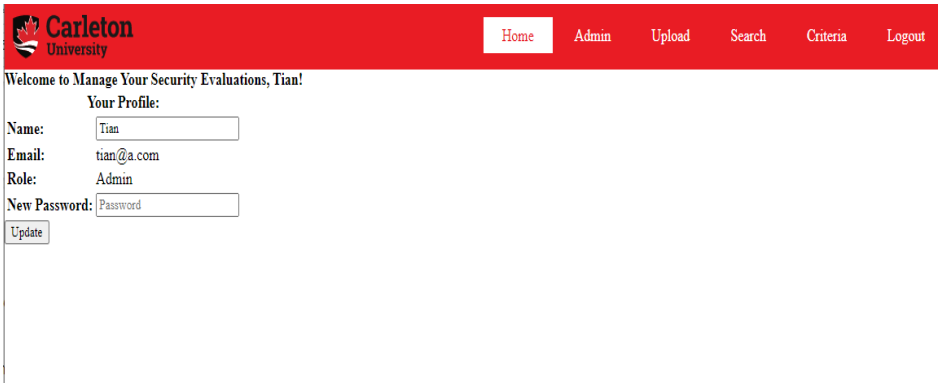

Figure 3 welcomepage.html - After Login

Evidence Related Pages

Evidence-related pages include search.html, upload.html, and view.html.

search.html

The search.html is used for users to search the security evidence. This page contains the dropdown boxes and the input boxes for users to apply filters to evidence tags when searching. After they click the search button, the search result will then display below with its name and the corresponding URL link to the view evidence page, view.html.

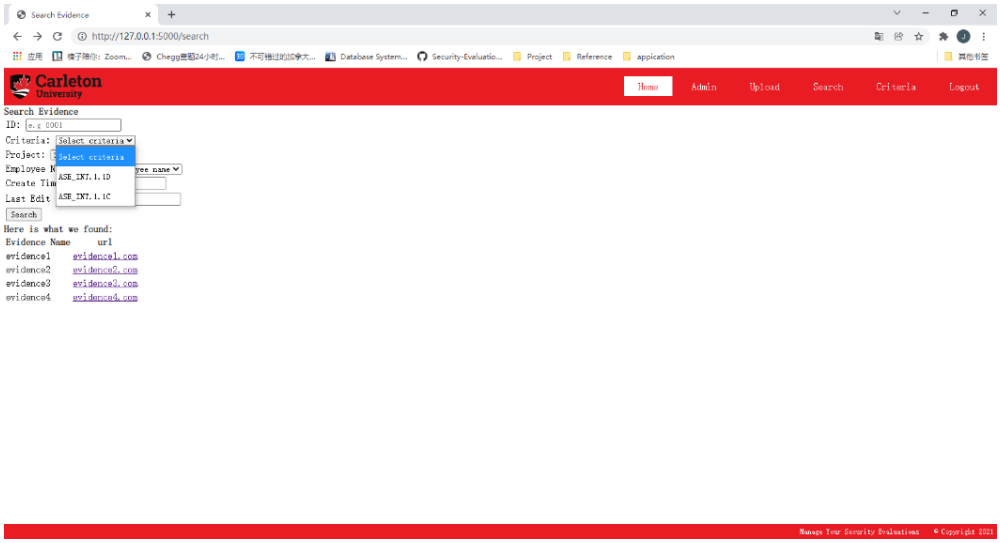

Figure 4 search.html

upload.html

The upload.html is for users to upload the security evidence. By entering the related information such as evidence name, project name, description, and criteria, users could create new evidence and save the new evidence to the database. After the evidence has been uploaded successfully, the page will show the message.

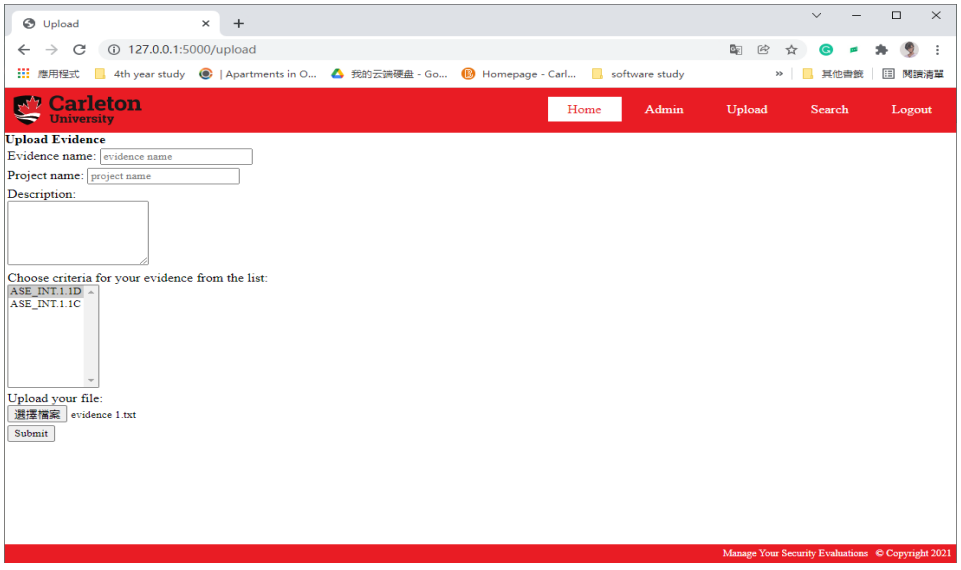

Figure 5 upload.html

view.html

The view.html is for the user to view the detail of the specific evidence. So far, the view.html is still in development. For the detailed introduction, please see the section View Evidence.

Authentication Related Pages

Authentication-related pages include login.html, signup.html, admin.html.

login.html

The login.html allows users to log in to the system by entering the correct username and password. Each email address is unique. If a user enters an email address that does not exist in the database, the user will be redirected to the signup page.

signup.html

The signup.html is for users to register to the system. By entering the username, email and password, a user will be registered.

admin.html

The admin.html is for Admin users to manage other users. In the current version, the first user has the administrator role. The Admin user could assign different roles to later registered users.

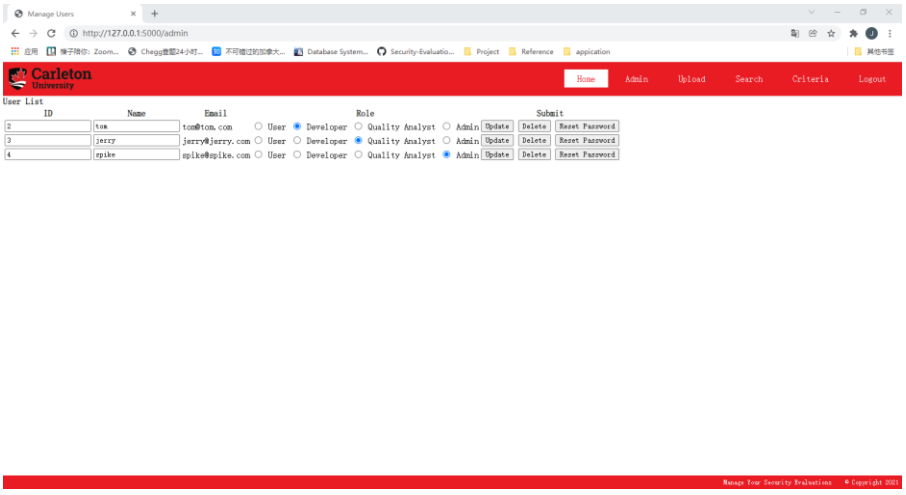

Figure 6 admin.html

Criteria Management Page - criteria.html

The criteria.html is for users to create new criteria. The new criteria could be created and stored in the database by entering the name and description. The created criteria will then be shown on the page with its name, creator, unique id, and description. Admin users could also delete the criteria by clicking the delete button.

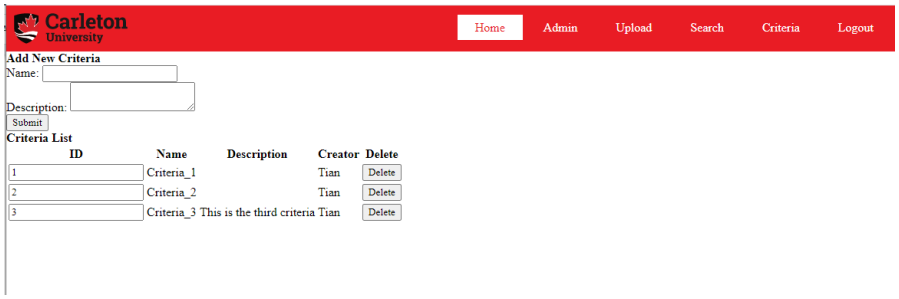

Figure 7 criteria.html

Database

The database is designed and implemented in object-relational mapping (ORM). We have implemented four object-oriented classes: user, role, evidence, and criteria.

- The user table stores the user-related data such as the user's id, name, password, email, and role type.
- The role table stores the role's id and name. The roles' access rights are coded in a python file rather than in the database. When a user is redirected to another web page, the program will repeatedly retrieve the access control matrix to find the correct data. Thus, this design would increase the speed when searching a user's access right.
- The evidence table stores the evidence's id, name, the creation date and time, the latest modified date and time, description, URL, project name. It also stores the developer's user id who uploads the evidence and the criteria's id which the evidence belongs to.
- The criteria table stores the criteria's id, name, description, and the admin's user id who create the criteria.

Those classes will automatically create their corresponding tables in SQLite if they do not exist in the database.

The relationships are one-to-many relationships between role and user, between user and evidence, between user and criteria, and between criteria and evidence. Therefore, the foreign key and back referenced relationship are used when implementing the database in Python.

Several methods support the backend logic to store and retrieve data in the database, such as storing, updating, querying, and deleting the information in four tables. When querying the related data, we applied an interface SQLAlchemy to query the desired data using raw SQL.

Alternative plans during development

ORM vs Relational database

The object-relational mapping (ORM) supports webpage development. The ORM can store lightweight data, and these lightweight data can be stored and retrieved quickly [2]. It does not use raw SQL to interact with the database. Instead, ORM provides a way to interact with the

database using object-oriented language to facilitate database development. However, the ORM syntax would become complicated when dealing with multi-table join queries and queries with multiple and complex conditions.

A relational database is used to manage the heavyweight database. Relational databases such as SQLite and PostgreSQL use raw SQL statements to create tables, store, and retrieve data in the tables. The developers can fully control the relational database by using raw SQL statements.

Although the relational database is well-used in a heavyweight database and we can have complete control of all data in the database, we chose ORM. We use ORM to structure and implement our database because Flask-SQLAlchemy has APIs to support the ORM directly. There is also no solution to adapt the relational database with the Flask-SQLAlchemy.

Flask SQLAlchemy

When designing and implementing the relational database, we tried to find a solution to adapt the relational database with the Flask-SQLAlchemy or bypass the Flask-SQLAlchemy to support the object-oriented data that Flask needs.

We tried various methods, including global variables, to solve the issue, but none worked. Finally, we realized that we could not adopt the relational database to the Flask-SQLAlchemy unless we developed a new Flask database expansion to replace Flask-SQLAlchemy. However, that solution would take much time, not the project's primary goal. Therefore, we used Flask-SQLAlchemy to re-build, re-structure, and re-implement our database.

Raw SQL statements in ORM

As discussed above in ORM vs. Relational database, the ORM disadvantage is dealing with multi-table join queries and multiple and complex conditions. However, we do need many complex queries. For example, we need to query all evidence that a specific developer uploads. Three tables may be used to complete the example query: user, role, and evidence. Therefore, using raw SQL statements is essential to query data with complex conditions.

Moreover, we found a way in SQLAlchemy to execute the query in raw SQL statements. At present, we applied the way into most queries such as query all criteria id and name, query the evidence info by the given filters from backend with the information in the tables of criteria, evidence, and user. By overcoming the disadvantage of ORM, we can quickly implement the query methods and support the backend logic better.

Uncompleted work

Secure Password

There are still functions or features we have not completed yet. The first is a feature for registers. Currently, the register function has no requirement in the password set by users, which is a threat

to security. In the subsequent development, we would add requirements for passwords entered. In our plan, the password set by users should at least have eight characters. Insecure passwords like username and locations should be prohibited with a file, including insecure passwords. The password should also contain both numbers and letters.

File Parse

The current upload function saves files locally instead of a cloud server, passing a local directory to the database. Our team is considering saving the contents within the text file directly into the database to optimize. The upload function should read from uploaded files and pass the contents to the database. In this case, users could view the content of evidence on the view page and even directly edit the content of the evidence.

We now have a function that allows users to enter criteria one by one with their name and description. However, criteria usually contain many standards, and it would take too much time to upload them manually. Our group plans to implement a function that allows users to upload a CSV file with criteria name and description and then read and upload these criteria automatically.

View Evidence

Currently, the system does not support viewing evidence. The team will add a new page and router logic in the upcoming development to support evidence download and online preview.

CSS

The current version of CSS doesn't have much customization. We will conduct more research on human-computer interaction in later development to design a user-friendly and easy-to-use UI interface.

Deploy the System to Cloud

Currently, the database and program are stored locally. However, this implementation does not meet the company's development needs, as all data and files in the database should be synchronized between users. After implementing all functional features, we will deploy the local database and the system to the cloud.

Challenges

Database implementation

When designing the database, the relational database was considered the first choice because there would be many criteria and evidence data stored and retrieved in the database. However, we found that the relational database is not well adapted to Flask-SQLAlchemy in implementing databases in applications. For example, Flask-SQLAlchemy provides APIs with a user object to implement functions such as login and redirection to other pages.

We searched many resources about adapting the relational database to Flask. We had implemented a way to connect the relational database to Flask-SQLAlchemy. Moreover, several methods in Python were implemented to store and retrieve data in the relational database. However, there is no solution to adapt the relational database to the Flask APIs. Therefore, we re-built the database, implemented the methods using ORM, and abandoned the original implementation using the relational database.

Timeline change

As the deadlines for some of the deliverables on the project website have changed, so has our timeline for those deliverables. The revised timeline is shown below:

Schedule \ Task Name	**Progress Report**	**Oral Presentation**
Original Schedule	December 5th, 2021	January 22nd, 2022
New Deadline	January 20th, 2022	March 20nd, 2022
Original Deadline	December 10th, 2021	January 24th, 2022
New Deadline	January 21st, 2022	March 25th, 2022

Table 2 Timeline Comparison Table

The scheduled complete dates and deadlines for other deliverables and tasks are aligned with the proposal's timetable. Only the progress report and the oral presentation schedules have been changed. We follow the schedule mentioned in the proposal, except for these two tasks.

Conclusion

In conclusion, we have achieved continuous and some effective development. So far, the progress meets the proposal timeline except for the deadline of the progress report and the oral presentation. This timeline is delayed due to the department's revisions. Therefore, we did the corresponding edition on our timeline. The project's general framework and essential functions have been realized in the main.

As well as that, the discussion on further integration and UI design will also play an essential role in the future development process. The Managing Security Evaluations Platform now has login and signup mechanisms for the project itself. Also, it can now offer upload, search and create criteria services for users to manage security evidence. This platform is a tool for developers, security evaluators and regulatory authorities to evaluate and review the system's level of security. It will help developers store and collect evidence and ensure traceability to security evaluation criteria used in the security evaluation process.

References

[1] Palletsprojects, "Templates," 2010. [Online]. Available: https://flask.palletsprojects.com/en/2.0.x/tutorial/templates/#:~:text=Templates%20are%20files%20that%20contain,display%20in%20the%20user's%20browser.

[2] R. Gori, "Stack Abuse," Stack Abuse, 29 January 2020. [Online]. Available: https://stackabuse.com/using-sqlalchemy-with-flask-and-postgresql/. [Accessed 21 January 2022].