Security vulnerabilities and potential threats in the Movie Recommender application.

Introduction

The Movie Recommender application consists of a Python console front end that consumes a series of custom API endpoints served by Flask that accesses a MySQL database and an external API, provided by The Movie Database (TMDB). The user enters their top 5 movies, completes a quiz and is then awarded a list of movie recommendations from the database which align with their quiz answers and the top 5 movies of users who achieved a similar quiz score.

The application was developed using Python (v3.12), Flask (v3.0.3) and MySQL (v8.4.0) and tested on the team's laptops (Windows 10 and 11 and macOS Sonoma). For multi-user access, the application would need to be hosted on a suitably scalable server and the Python console front end replaced by web pages. This report considers common vulnerabilities that exist both in web applications generally and in APIs specifically, drawing on the OWASP Application Security Top 10 (simply the OWASP Top 10) (2021) and the OWASP API Security Top 10 (2023) respectively, and evaluates the Movie Recommender application in this context. This leads into a discussion of secure coding practices for Flask applications, with particular emphasis on HTTPS implementation, robust authentication and authorisation, securing application data, and logging and monitoring.

Understanding Common Threats

We'll start by detailing common threats and vulnerabilities that web applications face and the potential impact on the applications security.

A01:2021 - Broken Access Control

This vulnerability arises when an application fails to properly enforce permissions, allowing users to operate beyond their intended access rights. This can occur through insecure direct object references, missing function-level authorisation, or insufficient restrictions on authenticated users.

Potential Impacts:

- **Unauthorised Data Access**: Attackers may gain access to sensitive information or functionalities that should be restricted, leading to data breaches and exposure of confidential user data.
- **Data Loss or Manipulation**: Attackers could alter or delete critical records, resulting in significant data integrity issues and loss of vital information.
- **Account Takeover**: Exploiting broken access controls may enable attackers to hijack user accounts, leading to further unauthorised actions and potential damage to the user's reputation.

A02:2021 - Cryptographic Failures

Cryptographic failures arise when cryptographic algorithms or implementations are misused or are inherently flawed, leading to weaknesses in the protection of sensitive data. This includes issues such as weak encryption, poor key management or the complete absence of encryption.

Potential Impacts:

- **Sensitive Data Exposure**: Inadequate encryption or flawed cryptographic practices may result in the unauthorised disclosure of sensitive information, such as passwords, credit card details, and personal identification data.
- **Identity Theft and Fraud**: Compromised cryptographic keys or algorithms can enable attackers to impersonate users, leading to identity theft and fraudulent activities.
- **System Compromise**: Weak cryptographic implementations can be exploited to gain unauthorised access to systems, allowing attackers to manipulate or steal data.

A03:2021 - Injection

Injection vulnerabilities occur when an application incorporates untrusted data into a command or query, leading to unintended actions being executed. Common types of injection include SQL injection, command injection and cross-site scripting (XSS).

Potential Impacts:

- **Data Loss or Corruption**: Attackers may exploit injection flaws to manipulate database queries, resulting in unauthorised changes or the deletion of critical data.
- **Bypass Authentication**: Exploiting injection vulnerabilities can enable attackers to bypass authentication mechanisms, giving them unauthorised access to the application.
- **Full System Compromise**: Certain injection attacks can lead to remote code execution, allowing attackers to take full control of the system and its resources.

A04:2021 - Insecure Design

Insecure design refers to security issues that arise from flaws in the architecture of the application. These design flaws often result from a lack of security controls or insufficient threat modelling during the development process.

Potential Impacts:

- **Exploitation of Design Flaws**: An insecure design can introduce vulnerabilities that attackers can exploit, potentially leading to data breaches and unauthorised access.
- **Long-term Security Risks**: Flaws arising from poor design decisions may create persistent vulnerabilities that are difficult to address without a complete redesign.
- **Increased Attack Surface**: Inadequate design considerations can widen the attack surface, making it easier for attackers to identify and exploit weaknesses.

A05:2021 - Security Misconfiguration

Security misconfiguration arises when security settings are incorrectly configured or left at their default settings, leading to vulnerabilities. This can occur at any level of the application stack, from cloud storage settings to server configurations.

Potential Impacts:

- **Data Exposure**: Misconfigured security settings may unintentionally expose sensitive data, resulting in unauthorised access and potential breaches.
- **Service Disruptions**: Security misconfigurations can cause application downtime or service outages, impacting business continuity and eroding user trust.
- **Remote Exploitation**: Attackers may exploit misconfigurations to gain remote access to systems, enabling further attacks on the network.

A06:2021 - Vulnerable and Outdated Components
This category addresses the use of libraries, frameworks, and other software components with known vulnerabilities. When applications rely on outdated or unsupported components, they become vulnerable to exploitation.

Potential Impacts:

- **Exploitation of Known Vulnerabilities**: Outdated components with known vulnerabilities are easily exploited, leading to data breaches and system compromises.
- **Increased Risk of Malware**: Using vulnerable components may expose the application to malware, potentially resulting in data loss and system instability.
- **Compliance Issues**: Organisations may face legal and regulatory consequences for using outdated and vulnerable components, which can damage their reputation and financial stability.

A07:2021 - Identification and Authentication Failures
This category includes flaws related to identification and authentication mechanisms, such as weak password policies, ineffective account lockout mechanisms, and the failure to implement multi-factor authentication.

Potential Impacts:

- **Account Compromise**: Weak identification and authentication mechanisms can lead to unauthorised access to user accounts, resulting in data breaches and a loss of user trust.
- **Unauthorised Access to Sensitive Areas**: Attackers may exploit authentication flaws to gain access to privileged functions, allowing them to perform actions typically reserved for authorised users.
- **Service Abuse**: Flawed authentication processes can be abused by attackers to exploit system resources or carry out malicious actions undetected.

A08:2021 - Software and Data Integrity Failures
This category focuses on the integrity of software updates, critical data, and continuous integration/continuous deployment (CI/CD) pipelines. It highlights the risks of trusting software updates without verifying their authenticity.

Potential Impacts:

- **Compromise of Critical Data**: Without proper verification, malicious software updates or compromised data may lead to integrity issues, causing operational disruptions.
- **Introduction of Vulnerabilities**: Assuming software updates are safe without verification may allow attackers to introduce vulnerabilities into production systems, which could be exploited over time.
- **Loss of Trust**: Compromised integrity can significantly erode trust with users and stakeholders, impacting the organisation's reputation and business relationships.

A09:2021 - Security Logging and Monitoring Failures
This category addresses failures in logging and monitoring, which are essential for detecting and responding to security incidents. Insufficient logging can hinder an organisation’s ability to identify and mitigate attacks.

Potential Impacts:

- **Undetected Security Incidents**: Insufficient logging may result in security breaches going unnoticed, delaying response and remediation efforts.
- **Challenges in Forensics**: A lack of adequate logging hampers forensic investigations, making it difficult to determine the cause and impact of security incidents.
- **Increased Vulnerability to Attacks**: Without proper monitoring, systems remain vulnerable to ongoing attacks, as attackers can operate without being detected.

A10:2021 - Server-Side Request Forgery (SSRF)
SSRF vulnerabilities occur when an attacker manipulates a server into making requests to internal or external resources, potentially gaining access to internal services and restricted data. This type of attack allows crafted requests to bypass protections such as firewalls, VPNs, or network access control lists. The frequency of SSRF incidents is rising due to the increasing complexity of modern web applications and cloud architectures.

Potential Impacts:

- **Exposure of Internal Resources**: Attackers may exploit SSRF vulnerabilities to access internal services, leading to data leakage and further attacks on the network.
- **Remote Code Execution**: SSRF can be exploited to send crafted requests that may result in remote code execution, compromising the entire application.
- **Bypassing Security Controls**: Attackers can use SSRF to bypass security measures, gaining access to sensitive data that should be protected.

API1:2023 - Broken Object Level Authorisation
APIs often expose endpoints that manage object identifiers, presenting opportunities for Object Level Access Control vulnerabilities. It is essential to enforce authorisation checks in every function that interacts with a data source using a user's ID. This ensures users only access their own data.

Potential Impacts:

- **Unauthorised Data Access:** This could result in unauthorised access to other users' objects, leading to potential data disclosure to unintended parties.
- **Data Loss or Manipulation:** Attackers may alter or delete data, causing data loss or integrity issues.
- **Account Takeover:** In some cases, unauthorised access can lead to a full account takeover, significantly compromising the user's security.

API2:2023 - Broken Authentication
Authentication mechanisms can often be flawed, making them vulnerable to exploitation. Attackers could compromise authentication tokens or exploit weaknesses, allowing them to impersonate other users. Ensuring the integrity of an API's authentication system is critical to its overall security.

Potential Impacts:

- **Account Control:** Attackers may assume full control of other users' accounts, accessing sensitive information and performing unauthorised actions.
- **Impersonation Risks:** These weaknesses can allow attackers to carry out actions that appear legitimate, making it difficult to distinguish between authorised and malicious behaviour.

API3:2023 - Broken Object Property Level Authorisation
This vulnerability occurs when APIs fail to enforce proper authorisation checks at the property level. While users may be authenticated, they may still gain access to or modify object properties they should not have permission for.

Potential Impacts:

- **Data Disclosure:** Unauthorised access to sensitive or private object properties may expose information to unauthorised individuals.
- **Data Loss or Corruption:** Attackers could tamper with or delete critical data, resulting in data corruption or loss.
- **Privilege Escalation:** Unauthorised access to object properties could allow attackers to escalate their privileges, compromising the security model of the application.
- **Account Takeover:** In some cases, unauthorised access to object properties can lead to account takeover, undermining user trust and application security.

API4:2023 - Unrestricted Resource Consumption
APIs can be exploited through unrestricted resource consumption when limits on resources like bandwidth, CPU, memory, or storage are not imposed. Integration with third-party services that consume resources, such as emails or SMS, can also be targeted by attackers to overwhelm the system.

Potential Impacts:

- **Denial of Service (DoS):** Exploitation can cause service interruptions by depleting resources, making the application unavailable to legitimate users.
- **Increased Operational Costs:** Excessive resource use may drive up operational costs due to higher CPU usage or expanded storage requirements.

API5:2023 - Broken Function Level Authorisation
Authorisation flaws can stem from complex access control models, making it difficult to distinguish between user-level and administrative-level functions. These weaknesses can allow attackers to gain unauthorised access to resources or perform actions they should not be allowed.

Potential Impacts:

- **Unauthorised Functionality Access:** Attackers may exploit these weaknesses to execute unauthorised actions, including those targeting administrative functionality, which could lead to significant data exposure or loss.
- **Service Disruption:** This could also lead to service disruptions, affecting the application’s overall integrity and availability.

API6:2023 - Unrestricted Access to Sensitive Business Flows
APIs may unintentionally expose critical business processes, such as online transactions, without adequate safeguards against automated abuse. These vulnerabilities typically arise from poor design rather than coding errors.

Potential Impacts:

- **Business Disruption:** While the technical impact may be limited, unauthorised access to business flows can disrupt legitimate operations, such as product purchases.
- **Economic Disruption:** Such vulnerabilities may also lead to economic distortions, such as artificially inflating prices or altering in-game economies.

API7:2023 - Server-Side Request Forgery (SSRF)
SSRF vulnerabilities occur when APIs fetch remote resources without validating user-provided URLs properly. Attackers can trick the API into making requests to unintended destinations, bypassing network protections like firewalls.

Potential Impacts:

- **Information Disclosure:** Successful SSRF attacks can expose internal services or resources (e.g. conducting internal port scans or accessing sensitive data).
- **Bypassing Security Measures:** Attackers may use SSRF to evade security controls, resulting in denial of service (DoS) or using the API as a proxy for malicious activities.

API8:2023 - Security Misconfiguration
Complex API configurations often lead to security misconfigurations when best practices are not followed. These misconfigurations can open up vulnerabilities that attackers can exploit.

Potential Impacts:

- **Data Exposure:** Misconfigurations can expose sensitive data, providing an entry point for various types of attacks.
- **Server Compromise:** Such vulnerabilities could potentially compromise the entire server, leading to full system compromise.

API9:2023 - Improper Inventory Management
APIs expose numerous endpoints, and maintaining proper documentation and an accurate inventory of hosts and API versions is critical. Failure to do so can lead to the exposure of deprecated endpoints or debugging features.

Potential Impacts:

- **Sensitive Data Access:** Attackers may exploit outdated or poorly managed endpoints to gain access to sensitive data.
- **Vulnerability Exploitation:** Deprecated API versions may expose known vulnerabilities or provide access to administrative features.

API10:2023 - Unsafe Consumption of APIs
Developers often place undue trust in third-party APIs, which can lead to the adoption of weaker security measures. Attackers may exploit these trusted third-party APIs to indirectly compromise the target system.

Potential Impacts:

- **Sensitive Information Exposure:** Attackers may use this vulnerability to access sensitive data.
- **Injection Attacks:** Poor handling of data from third-party APIs can lead to injection attacks or denial-of-service (DoS) attacks, depending on how the data is processed.

Vulnerabilities in the Movie Recommender

In this section we explore which vulnerabilities currently exist in our movie recommender application and how they affect the project. A review of the Movie Recommender application against the OWASP Foundation's top 10 vulnerabilities for web applications (2021) reveals the following:

A01:2021 – Broken Access Control

- **No access control mechanisms:** The Movie Recommender application presents features to register as a new user or log in as an existing user. Successful registration or login amounts to authentication, after which users are able to add their top 5 movies, take a quiz and get movie recommendations. The principle of least privilege and deny by default are not applied: all users have the same access privileges and would automatically have access to any new features added.
- **Insecure direct object references:** API endpoints expose object identifiers, such as the user ID, which serve as direct references to records in the database. Endpoint URLs can be modified to gain access to other users' objects as there are no object-level authorisation checks to validate whether a given user has the necessary permissions to perform a given action on a given object (see API1:2023 below).

The lack of access control mechanisms threatens all three components of the CIA triad. An attacker could modify the user ID parameter in GET requests to access the quizzes or movie recommendations for any other user (confidentiality). Similarly, they could modify the user ID parameter in POST requests to edit or delete the top 5 movies or quiz responses of any other user, which would alter (integrity) or remove (availability) their movie recommendations.

A02:2021 – Cryptographic Failures

- **No password hashing:** User account passwords are stored in the database in plaintext.
- **No network encryption:** Data is transmitted over an unencrypted network, using HTTP, rather than HTTPS.

The lack of cryptography threatens all three components of the CIA triad. If an attacker gained access to the data either in transit or at rest, they could view user account passwords and use these to log in as another user and access their quizzes and movie recommendations (confidentiality) or edit or delete their top 5 movies or quiz responses, which would alter (integrity) or remove (availability) their movie recommendations.

A03:2021 – Injection

Limited input validation: Validation of user input is minimal and only implemented on the client side. On initial registration, users are required to enter their first name, last name, email address and password, of which only the latter two are validated: regular expressions are used to ensure that email addresses are entered in a valid format and passwords adhere to certain requirements for length and complexity. On subsequent login, the email address and password alone are required, but neither is validated here. On entering top 5 movies, user input is validated only as far as ensuring that it is a comma-separated list. On entering quiz responses, validation precludes all entries other than integers corresponding to the available response options (e.g. 1, 2, 3 or 4).

The deficiencies in input validation could potentially pose a threat to all three components of the CIA triad as an attacker could potentially insert malicious SQL queries into the improperly validated input fields in order to read (confidentiality), modify (integrity) or delete (availability) data in the database. However, as all queries were parameterised, SQL injection attacks would be unlikely to be effective.

A04:2021 – Insecure Design

- **Improper security architecture:** Application security was considered as an afterthought, rather than at each stage of the software development lifecycle and, as such, there are a number of vulnerabilities inherent in the design of the Movie Recommender application, as is evident from this report. The lack of logging and monitoring mechanisms also means that vulnerabilities are unlikely to be detected prior to malicious exploitation.

The lack of consideration of security principles at the design stage has resulted in a wide attack surface with substantial exploitation potential via a variety of attack vectors. The inherent vulnerabilities in the application pose a threat to all three components of the CIA triad.

A05:2021 – Security Misconfiguration

- **No component segregation:** The security configuration of the Movie Recommender application does not follow the principle of least privilege. Current deployments place Flask and the MySQL database on the same machine; however, as the database need only be accessible to the Flask application and system administrators, the two components should be segregated, such that the database is not exposed to the wider internet unnecessarily.
- **No configuration verification:** There is no mechanism, automated or otherwise, for verifying the effectiveness of security configurations and modifying where necessary.

The lack of segregation widens the attack surface and reduces security restrictions to the lowest common denominator in order that the component requiring the greatest degree of access can operate as intended. Components, such as the database, whose operation requires a lesser degree of access, are more vulnerable to attack as a result and confidentiality, integrity and availability are more readily compromised.

A06:2021 – Vulnerable and Outdated Components

- **No patch management process:** The application was developed using Python v3.12, Flask v3.0.3 and MySQL v8.4.0 – the latest versions at the time of development – however, no mechanism is in place to ensure that these components remain patched and updated as required (see A08:2021 below).
- **No component inventory:** Analysis of Common Vulnerabilities and Exposures (CVEs) for the core components highlights that vulnerabilities can arise from component dependencies (such as Werkzeug for Flask); however, no Software Bill of Materials (SBOM) was created and no monitoring of publicly-disclosed vulnerabilities – either via automatic Software Composition Analysis (SCA) tools or manual review of sources such as the National Vulnerability Database (NVD) – is being carried out.

The lack of vulnerability management mechanisms means that components will become outdated, and vulnerabilities disclosed in the public domain will be unknown to system administrators and therefore remain unpatched, whilst being known to attackers, rendering them ripe for exploitation. Depending on the nature of the vulnerability, one or more components of the CIA triad could be compromised.

A07:2021 – Identification and Authentication Failures

- **Insecure authentication:** User authentication only uses a single verification factor in the form of a password (something you know). Although passwords are validated to ensure they are a minimum of 8 characters in length and contain 1 uppercase character, 1 lowercase character, 1 digit and 1 special character, they are not checked against compromised password lists, making them more vulnerable to credential stuffing, brute-force and other automated attacks, and password rotation is not enforced.
- **No session management:** The concept of a session to track user state is not employed.

Deficiencies in authentication and session management make it easier for an attacker to gain access to another user's account and, having done so, they would then be able to view the user's quizzes and movie recommendations (confidentiality) or edit or delete their top 5 movies or quiz responses, which would alter (integrity) or remove (availability) their movie recommendations.

A08:2021 – Software and Data Integrity Failures

- **No CI/CD automation:** The application is not currently embedded in a Continuous Integration/Continuous Delivery (CI/CD) pipeline with proper segregation, configuration and access control to ensure the integrity of the source code, build or deployment scripts. It lacks a mechanism for checking vulnerabilities of components and dependencies automatically and responding appropriately by patching or breaking the build to prevent deployment.
- **No integrity verification:** User data, movie names, quizzes and quiz responses are transmitted between components as JSON objects and deserialised by the receiver without verifying their integrity by means of digital signatures or other mechanisms.

As with A06:2021, the lack of automated build and deployment mechanisms means that components will become outdated, and vulnerabilities disclosed in the public domain will be unknown to system administrators and therefore remain unpatched, creating an exploit opportunity for attackers that could compromise one or more components of the CIA triad. The lack of integrity verification on serialised objects means that any tampering or relay is likely to go undetected. This threatens data integrity and, whilst it does not pose a direct threat to confidentiality, breaches are less readily identified.

A09:2021 – Security Logging and Monitoring Failures

- **No logging or monitoring:** There are currently no logging, monitoring or event management systems in place, nor is there an incident response plan.

The absence of logging and monitoring mechanisms means that system administrators will not be alerted to suspicious activity in real time and will therefore be unable to respond as promptly or as effectively as they would do if this information were available to them, increasingly the likelihood of one or more components of the CIA triad being compromised. The lack of log data also makes post-incident analysis more difficult, so the impact of an attack may take longer to establish.

A10:2021 – Server-Side Request Forgery

- **Insecure API consumption:** Whilst the Movie Recommender application itself does not currently fetch remote resources via user-supplied URLs, it does retrieve data from the TMDB API which, if compromised by SSRF, could introduce a vulnerability to the Movie Recommender as minimal validation of the third-party data is performed.

If TMDB were subject to an SSRF attack, consuming its API without proper validation of the data returned could result in a malicious payload being relayed by the Movie Recommender, which could compromise one or more components of the CIA triad, depending on the nature of the payload.

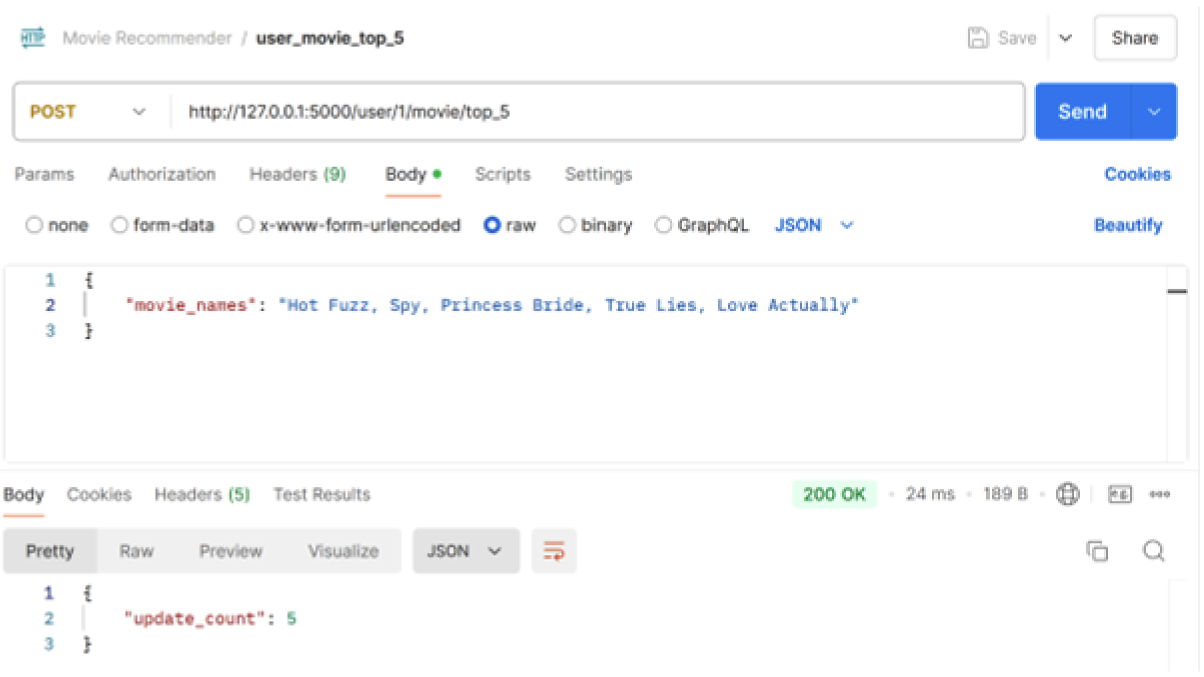
As the Movie Recommender exposes a series of API endpoints (to add a user, to check user credentials, to update a user's top 5 movies, to get a quiz, to update a user's quiz responses and to get personalised movie recommendations for a user), these have been reviewed against the OWASP Foundation's list of top 10 vulnerabilities for APIs (2023), which reveals the following:

API1:2023 – Broken Object-Level Authorisation

- **Insecure direct object references:** API endpoints expose object identifiers, such as the user ID, which serve as direct references to records in the database. Endpoint URLs can be modified to gain access to other users' objects as there are no object-level authorisation checks to validate whether a given user has the necessary permissions to perform a given action on a given object.

The lack of object-level authorisation checks threatens all three components of the CIA triad. An attacker could modify the user ID parameter in GET requests to access the quizzes or movie recommendations for any other user (confidentiality). Similarly, they could modify the user ID parameter in POST requests to edit or delete the top 5 movies or quiz responses of any other user, which would alter (integrity) or remove (availability) their movie recommendations.

For example, having been supplied a valid user ID, the POST /user/<user_id>/movie/top_5 endpoint does not perform an authorisation check to validate whether the requestor has the necessary permissions to update the specified user's top 5 movies.



POST http://127.0.0.1:5000/user/1/movie/top_5

```
{
    "movie_names": "Hot Fuzz, Spy, Princess Bride, True Lies, Love Actually"
}
```

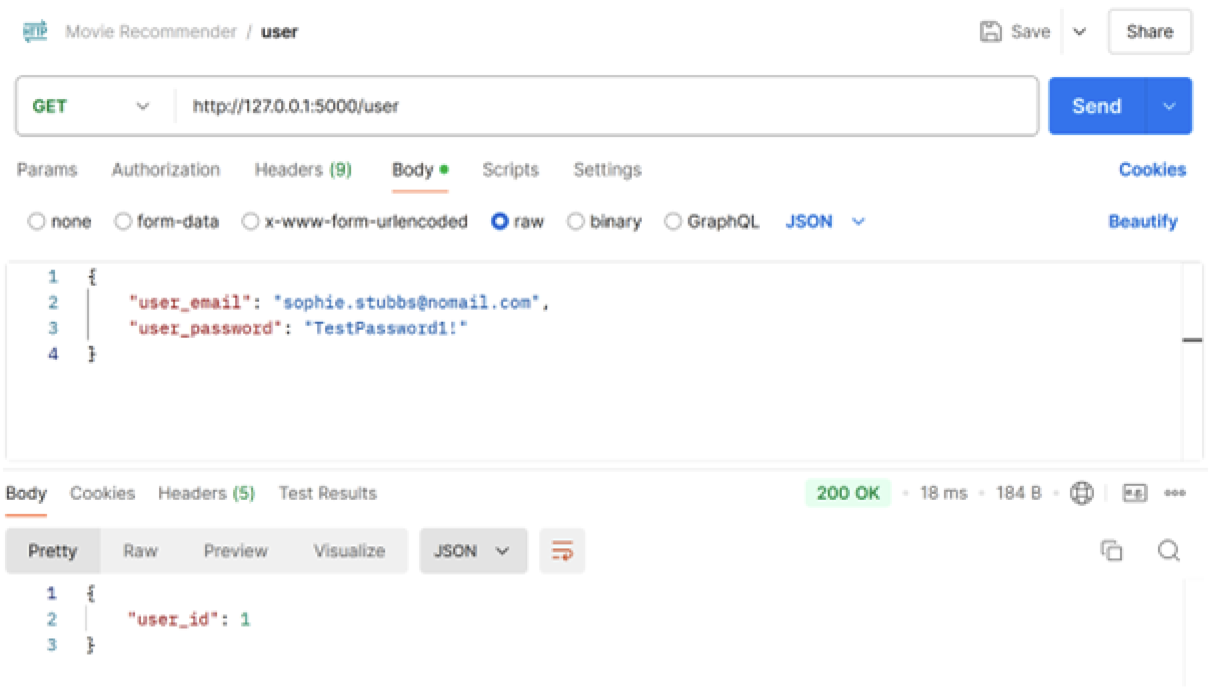
```
{
    "update_count": 5
}
```

API2:2023 – Broken Authentication

- **Insecure authentication:** The Movie Recommender API does not implement mechanisms to guard against automated login attempts against user accounts, does not enforce strong passwords – it relies on validation on the client side – and does not encrypt them in transmission.

The lack of secure authentication threatens all three components of the CIA triad. An attacker mounting a brute-force attack would not be limited in their attempts to log into a given user account and, once a valid email and password combination was identified, they would be able to view the user's quizzes and movie recommendations (confidentiality) or edit or delete their top 5 movies or quiz responses, which would alter (integrity) or remove (availability) their movie recommendations.

For example, the GET /user endpoint could be called repeatedly with email and password combinations until a successful 200 response and valid user_id were returned.



GET http://127.0.0.1:5000/user

```
{
    "user_email": "sophie.stubbs@nomail.com",
    "user_password": "TestPassword1!"
}
```

200 OK · 18 ms · 184 B

```
{
    "user_id": 1
}
```

API3:2023 – Broken Object-Property-Level Authorisation
API endpoints expose properties of objects but do not implement object-property-level authorisation checks to validate whether a given user has the necessary permissions to access the specific object properties. The Movie Recommender API currently only has the concept of a user role which has access to all object properties exposed by the API. It does not have a concept of an admin role that could have enhanced privileges to access additional object properties, though this feature could be added in future.

This vulnerability is present because of broken object-level authorisation (see API1:2023 above), but additional object-property-level vulnerabilities exist because the application only has one type of user.

API4:2023 – Unrestricted Resource Consumption

- **Unrestricted API consumption:** The Movie Recommender API makes multiple calls to the TMDB API to get movie details and recommendations but does not implement any rate limiting or validation of the data retrieved.

The lack of restrictions imposed on calls to the TMDB API could result in the Movie Recommender being blocked by the third party for repeatedly exceeding rate limits. This could simply be due to a high number of calls made to the API in a short period of time by legitimate users, or it could be an attempted denial-of-service attack mounted against TMDB via the Movie Recommender. In either case, if TMDB temporarily or even permanently blocked the Movie Recommender from retrieving its data, users would be unable to access movie recommendations, which would have a critical impact on availability. Improper data validation could introduce additional vulnerabilities that could also compromise one or more elements of the CIA triad.

API5:2023 – Broken Function-Level Authorisation
The Movie Recommender API currently only has the concept of a user role which has access to all functions exposed by the API. It does not have a concept of an admin role that could have enhanced privileges to access additional functions, although these could be added in time via new APIs.

API6:2023 – Unrestricted Access to Sensitive Business Flows
The Movie Recommender API currently only exposes limited business flows that are not sensitive, namely adding or validating a user, adding top 5 movies, taking a quiz, and getting movie recommendations. As the system evolves, additional endpoints could be added for billing customers or adding comments about movies, which could introduce this vulnerability.

API7:2023 – Server-Side Request Forgery

- **Insecure API consumption:** Whilst the Movie Recommender application itself does not currently fetch remote resources via user-supplied URLs, it does retrieve data from the TMDB API which, if compromised by server-side request forgery (SSRF), could introduce a vulnerability to the Movie Recommender as minimal validation of the third-party data is performed.

If TMDB were subject to an SSRF attack, consuming its API without proper validation of the data returned could result in a malicious payload being relayed by the Movie Recommender, which could compromise one or more components of the CIA triad, depending on the nature of the payload.

API8:2023 – Server Misconfiguration
The Movie Recommender API does not currently implement HTTPS and further vulnerabilities could be introduced via the host on which it was eventually deployed.

This vulnerability could affect:

- confidentiality if an attacker accessed user credentials and personal data
- integrity if an attacker used intelligence gained from the misconfiguration to mount further attacks
- availability if an attacker used intelligence gained from the misconfiguration to mount further attacks

API9:2023 – Improper Inventory
Outside project reports and README documentation, the Movie Recommender API does not have specific documentation and versioning, and there are no published plans for adding or removing endpoints over time.

Whilst there are no immediate security concerns arising from the lack of proper inventory, it could pose a risk in the future if updates are not properly managed and outdated API versions remain in use.

API10:2023 – Unsafe Consumption of APIs

- **Insecure API consumption:** The Movie Recommender API consumes two TMDB API endpoints over HTTP and performs limited sanitisation and validation of data received before returning a list of movie recommendations.

If TMDB were subject to an SSRF attack, consuming its API without proper validation of the data returned could result in a malicious payload being relayed by the Movie Recommender, which could compromise one or more components of the CIA triad, depending on the nature of the payload. For instance, confidentiality could be impacted if an attacker injected malicious code that resulted in a Movie Recommender user inadvertently revealing personal data.

Addressing vulnerabilities through secure coding practices

Reviewing the application against the top 10 OWASP vulnerabilities from 2021 demonstrated the value of OWASP guidelines and highlighted specific opportunities to enhance the security of the APIs, network traffic, authentication and authorisation model, application data and logging and monitoring. The 2023 summary of API vulnerabilities offers a framework for developing security-centric guidelines for the Movie Recommender and other Flask applications.

API1:2023 Broken Object-Level Authorisation	Implement authorisation mechanism with user policies and use with every function to check the authenticated user can do what they are attempting. Use random unpredictable GUIDs for object references.
API2:2023 Broken Authentication	Use standard authentication that has solutions to common attacks, including rate limiting and lockout for brute-force attacks, and use multi-factor authentication where possible.
API3:2023 Broken Object-Property-Level Authorisation	Use authorisation mechanism to ensure an authenticated user can access all properties being exposed, be specific when mapping, and return the least possible data.
API4:2023 Unrestricted Resource Consumption	Set and enforce limits on data sizes and call rates by implementing appropriate throttling and validation of parameters that determine size of data returned.
API5:2023 Broken Function-Level Authorisation	Deny by default and require function permissions, particularly admin functions, to be granted to roles defined through the authorisation mechanism.
API6:2023 Unrestricted Access to Sensitive Business Flows	Align strength of access restrictions to the level of risk presented by specific business flows and be wary of automation when humans are expected.
API7:2023 Server-Side Request Forgery	Limit resource-fetching components to expected sources and validate and sanitise data returned before forwarding.
API8:2023 Security Misconfiguration	Ensure all API communication is over TLS and limit HTTP verbs that can be responded to. Expect security headers from web front ends, restrict content types/data formats to minimal required and validate structure of payloads against expected schemas.
API9:2023 Improper Inventory Management	Fully document and version-control APIs and protect all exposed endpoints, not just those designated as production.
API10:2023 Unsafe Consumption of APIs	Assess security posture of API providers, ensure communication is over TLS and validate and sanitise data returned before using or forwarding.

These high-level API security improvement guidelines plus threats raised by the generic OWASP Top 10 translate into specific security-centric guidelines for Flask applications:

1. Use established libraries for user registration, email confirmation, password reset/recovery and email change.
2. Enforce strong passwords.
3. Hash passwords.
4. Implement multi-factor authentication.
5. Implement login tracking.
6. Implement session-based authentication.
7. Implement role-based authorisation.
8. Use established data access/persistence libraries to prevent SQL injection.
9. Use established forms libraries to gather and sanitise user-entered data.
10. Use established rate-limiting libraries to prevent brute-force and DoS attacks.

Rather than building these enhanced security features from first principles, libraries offer an opportunity to build on code that has been tested extensively and for which CVE scrutiny and regular security patches are available.

Use established libraries for user registration, email confirmation, password reset/recovery and email change.
Flask-Security integrates several Flask extensions and libraries to provide this capability, and assumes that a standard library, such as Flask-SQLAlchemy, will be used for data persistence. Setting SECURITY_REGISTERABLE will create a user registration endpoint, configurable through other variable values such as SECURITY_CONFIRMABLE to determine if an endpoint should be created to handle requests to confirm emails. Flask-Security uses the email-validator library to confirm an email supplied is of a valid format. This feature works with the fs_uniquifier, which controls both sessions and authenticated tokens, setting or invalidating tokens depending on the success of registration and account activation.

Enforce strong passwords
Flask-Security has a configurable password validator that checks that minimum length requirements have been satisfied, meets complexity requirements and checks against haveibeenpwned to confirm the password is not on a list of those breached.

Hash passwords
Flask-Security implements password hashing with the passlib library to provide configurable hashing and salting.

Implement multi-factor authentication
Flask-Security offers two-factor authentication, controlled by SECURITY_TWO_FACTOR settings, for use with email, SMS and authenticator apps.

Implement login tracking
Flask-Security can provide login tracking, controlled by the SECURITY_TRACKABLE setting, which maintains basic current and last login date and IP address and a total login count per user.

Implement session-based authentication
Flask-Security uses the Flask-Login extension to provide session-based authentication and associates the Flask-Login token with a fs_uniquifier value for each user. Flask-Login stores the active user's ID in a Flask Session and allows views to be restricted to logged-in users.

Implement role-based authorisation
Flask-Security uses the Flask-Principal extension to enable roles with permissions to be assigned to users, which can be further refined to object level. Flask-Principal is used to define and evaluate relationships between an Identity (e.g. a user), a Need (e.g. an action), a Permission (a set of requirements that must be met), and an Identity Context (the context of a certain identity against a certain permission).

Use established data access/persistence libraries to prevent SQL injection
SQL injection attacks exploit a vulnerability in the way input data is used within SQL code, which can lead to unauthorised access, data breaches, data modification or damage to the database (e.g. inserting a DROP DATABASE command). For instance, a user could exploit the following code to return all users instead of a single user, as intended, because OR '1' = '1' will be appended which evaluates to OR true meaning no restriction is placed on rows returned from the users table:

```
input = " ' or  '1 '= '1"
query = "SELECT * FROM users WHERE username = '%s'" % input
result = connection.execute(query)
```

In a parameterised query, the SQL code to be executed is separated from the user input (rather than being concatenated with it) and passed as a parameter which is treated as data not executable code.

```
input = " ' or  '1 '= '1"
query = "SELECT * FROM users WHERE username = :name"
result = connection.execute(query, name=param)
```

With parameterised queries, the parameters are parsed as data so even if the parameter contains malicious SQL commands, it will be treated as a regular string of characters not as part of the SQL command being executed. This prevents execution of malicious code and established libraries escape parameter input such as single quotes, which further limits potential for SQL injection attacks. Parameterised queries also tend to be easier to read as the data is clearly distinguished from the SQL, which supports maintainability of code.

Flask-Security uses standard persistence libraries to add data access/persistence functionality to Flask. For instance, Flask-SQLAlchemy is used to add support for SQLAlchemy so that developers have access to the power of flexibility of SQL to provide efficient and performant database access. SQLAlchemy provides built-in parameter substitution to prevent SQL injection attacks.

The current application uses parameterised queries and stored procedure calls for all data access from the Flask app, and it will be important to maintain this approach if the application is refactored to use SQLAlchemy to take advantage of other features, such as user registration and session management.

Use established forms libraries to gather and sanitise user-entered data
Flask-Security integrates its session management features with the Flask-WTF library, which provides Cross-Site Request Forgery (CSRF) protection and can be used to validate and sanitise user input before processing.

Use established rate limiting libraries to prevent brute force and DOS attacks
Flask-Limiter can be used to implement rate limiting to prevent excessive requests overwhelming a server. Rate limits can be configured application wide or for specific routes or resources, for instance to protect the /user endpoint as follows:

```
…
from flask_limiter import Limiter
from flask_limiter.util import get_remote_address
 …
limiter = Limiter(app, key_func=get_remote_address)

@app.route('/user', methods=['GET'])
@limiter.limit("20/minute")
…
```

Authentication and Authorisation

Now we understand the common vulnerabilities and how they apply to our application, in this section we discuss the principles of strong user authentication and define a theoretical framework for robust authentication and authorisation.

Authentication

Authentication refers to ensuring a user is who they say they are by verifying information such as passwords and personal information.

Multi-factor authentication - Requiring the user to provide two or more forms of verification such as something they know, something they have and something they are.

Strong passwords - Ensuring users can only create passwords that are at least 12 characters long including a mix of upper and lower case, numbers and special characters and are not reusable will prevent brute force and dictionary attacks.

Password storage - Store passwords in an encrypted manner e.g in a hashed format so that if the database is compromised, they can't read them.

Session management - Providing users with a generated token that expires after a certain time period. Use secure cookies and secure flags to prevent cross-site scripting.

Authorisation

Authorisation refers to allowing someone access to something depending on their authenticated identity, a robust framework for authorisation would be as follows:

Role/Attribute based access control - Users are assigned different control permissions such as admin, editor, viewer depending on their role. Additionally, they are able to access different areas of the database depending on their role, for example, finance can access the payroll folder but Marketing cannot.

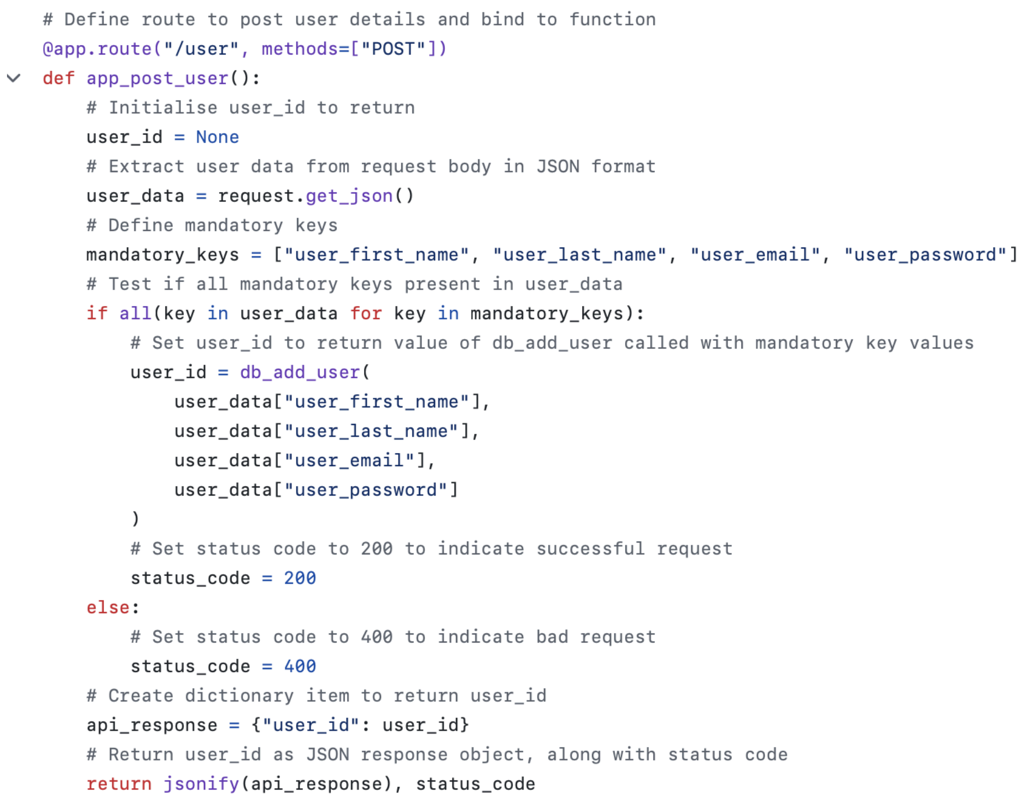
Principle of least privilege - Users should only be given the minimum access they need to complete their role so for example, if someone only needs to read the documents in the HR folder they should not have editing permissions.

Context-based access - Users can be restricted by time of day, location or device type to ensure they can only access sensitive data from their work computer during working hours.

Logging and monitoring - By logging all access attempts and permissions changes, it is easier to track and investigate any suspicious access attempts.

Planning for proper authorisation and access control:
Authentication should be secure and intuitive for users whilst protecting against common security threats. Currently, our plan to authenticate users and control access to the application is as follows:

1. **Create a new user** - Selecting Option 2 prompts a new user to enter first name, last name, email address, and password to register.
    a. The email address must be in a valid format, e.g. example@email.com.
    b. The password must be at least 8 characters, including 1 uppercase, 1 lowercase, 1 digit, and 1 special character.



```
# Define route to post user details and bind to function
@app.route("/user", methods=["POST"])
def app_post_user():
    # Initialise user_id to return
    user_id = None
    # Extract user data from request body in JSON format
    user_data = request.get_json()
    # Define mandatory keys
    mandatory_keys = ["user_first_name", "user_last_name", "user_email", "user_password"]
    # Test if all mandatory keys present in user_data
    if all(key in user_data for key in mandatory_keys):
        # Set user_id to return value of db_add_user called with mandatory key values
        user_id = db_add_user(
            user_data["user_first_name"],
            user_data["user_last_name"],
            user_data["user_email"],
            user_data["user_password"]
        )
        # Set status code to 200 to indicate successful request
        status_code = 200
    else:
        # Set status code to 400 to indicate bad request
        status_code = 400
    # Create dictionary item to return user_id
    api_response = {"user_id": user_id}
    # Return user_id as JSON response object, along with status code
    return jsonify(api_response), status_code
```

2. **Authenticate an existing user -** Selecting Option 1 prompts an existing user to enter their email address and password.

```
# Define route to get user details and bind to function
@app.route("/user", methods=["GET"])
def app_get_user():
    # Initialise user_id to return
    user_id = None
    # Extract user data from request body in JSON format
    user_data = request.get_json()
    # Define mandatory keys
    mandatory_keys = ["user_email", "user_password"]
    # Test if all mandatory keys present in user data
    if all(key in user_data for key in mandatory_keys):
        # Set user_id to return value of db_get_user called with mandatory key values
        user_id = db_get_user(
            user_data["user_email"],
            user_data["user_password"]
        )
        # Set status code to 200 to indicate successful request
        status_code = 200
    else:
        # Set status code to 400 to indicate bad request
        status_code = 400
    # Create dictionary item to return user_id
    api_response = {"user_id": user_id}
    # Return user_id as JSON response object, along with status code
    return jsonify(api_response), status_code
```

The current method ensures:

- Emails and Passwords are inputted in the correct format and that any incorrectly formatted will be rejected.
- Passwords are strong and contain a mix of characters, cases, numbers and special characters.

We can improve authentication and access control by adding the following:

- Ensuring passwords are stored in a **hash format**.
- Perform a check to ensure the entered **email domain exists**.
- After registration, send an email to the user where they have to click on a link to **validate their email**.
- Implement **rate limiting** on registration attempts to prevent bots from flooding the system with fake accounts.
- Include **multi-factor authentication** like asking for a memorable word or answering security questions in addition to providing their email and password.
- **Account lockout** after failed attempts
- A **password reset** option where a link is sent to their registered email.
- Implement **Flask-Login.**

The role of Flask-Login in managing user sessions:
Flask-Login is a Flask extension that simplifies user session management and helps to implement authentication mechanisms in the following way:

Session Management - Once a user has logged in, Flask-Login keeps track of where they are navigating to within the web application until the user logs out or the session expires. Session expiration and automatic log out after a period of inactivity should be implemented.

Log-in/Log-out - Flask-Login has easy to use functions such as login_user() and logout_user().

Protected routes - Flask-Login can require users to log in to be able to access a specific route using the @login_required decorator.

User Loader - Using the user_loader call back, Flask-login is able to retrieve the users data from the database on each log in.

"Remember me" feature - Flask-Login can use this feature to store a cookie that allows the user to continue a session again later even if they close their browser by using login_user(user, remember=True).

HTTPS - Use Flask-Login with HTTPS to secure cookies and session data.

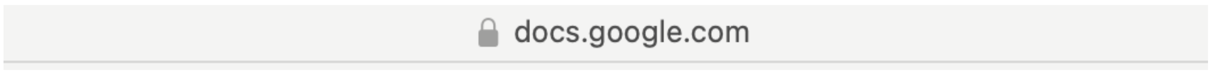
<u>HTTPS Implementation</u>

Now we'll explore the importance of implementing HTTPS in web applications and the role of SSL certificates.

What is HTTPS and why is it important?

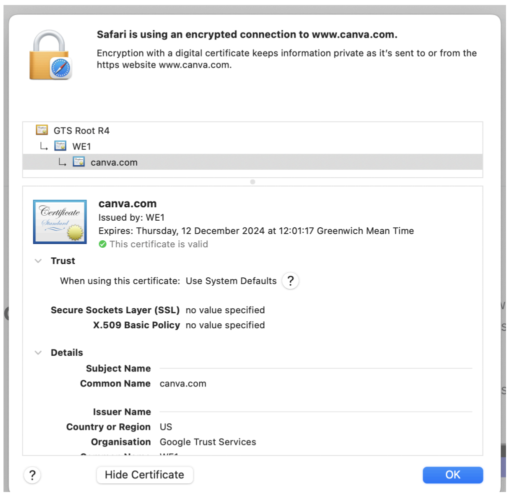
HTTPS is a vital tool for ensuring web applications are safe because:

- It **encrypts data** so that it is not sent in plain text and therefore cannot be read by malicious actors during transmission from the user to the server which prevents things such as man in the middle attacks.
- It **promotes trust** amongst users because they know when they see a padlock icon that it is secure and if HTTPS is not implemented they will be alerted that the website is not safe.



- It helps to **retain the integrity** of data because it ensures it cannot be modified without detection which prevents malicious code being injected into the transmission. It is especially important for websites like banks or email servers that require personal information to log in.

What is the role of SSL certificates in ensuring secure communication?
SSL certificates can be found on any HTTPS website by clicking on the padlock and then clicking “show certificate”. They play a vital role in ensuring secure communication by:


Safari is using an encrypted connection to www.canva.com.
Encryption with a digital certificate keeps information private as it's sent to or from the https website www.canva.com.

GTS Root R4
└ WE1
 └ canva.com

canva.com
Issued by: WE1
Expires: Thursday, 12 December 2024 at 12:01:17 Greenwich Mean Time
This certificate is valid

Trust
When using this certificate: Use System Defaults

Secure Sockets Layer (SSL)	no value specified
X.509 Basic Policy	no value specified

Details

Subject Name	
Common Name	canva.com
Issuer Name	
Country or Region	US
Organisation	Google Trust Services

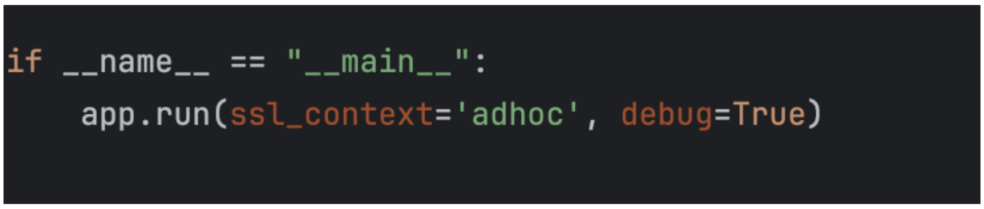
- **Authenticating** websites and ensuring users are able to check that they are interacting with the real one and not a fake, phishing site. Users can see all of the information about the organisation in the certificate's details.

- Establishing an **encrypted connection** between the user and the server using private and public keys to ensure the data being transmitted cannot be read.

- **Protecting the Integrity** of the data because if it detects that the data has been tampered with in any way during transmission it will cut the connection.

How to obtain and install an SSL certificate:
There are a few ways to obtain and install an SSL certificate:

1. **Ad hoc** - You can obtain and install an ad hoc certificate by adding ssl_context='adhoc' to the app.run(). I’ve demonstrated this in our project below:



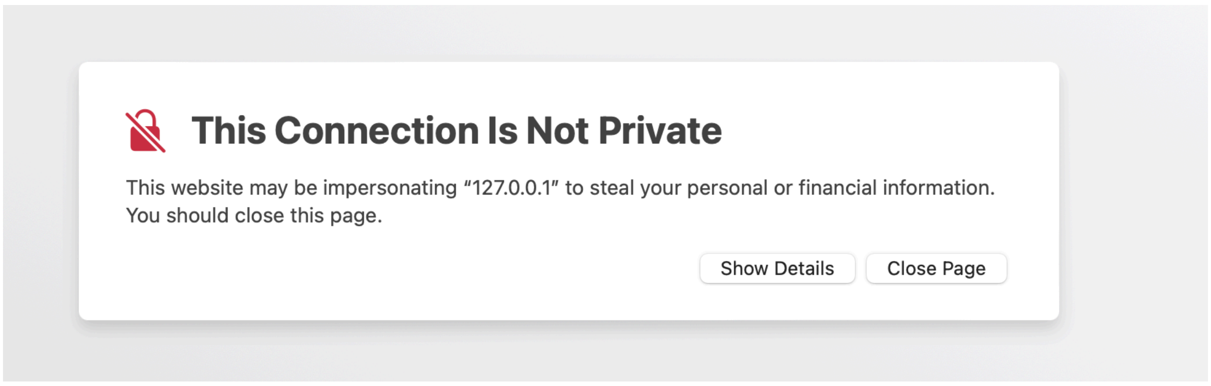
```
if __name__ == "__main__":
    app.run(ssl_context='adhoc', debug=True)
```

- The application will then indicate that it is now running a https server as shown below:

```
ENV_API_BASE_URL = "https://127.0.0.1:5000"
```

- This creates an encrypted connection but it is not a valid certificate. Each time the server runs, a different certificate is generated through pyOpenSSL.

2. **Self-signed certificate -** This can be made by generating a signature using a private key that is associated with that same certificate, however, although unlike the ad hoc method, self signing allows the certificate to be reused, it will not be on the list of trusted CA authorities and therefore validation will still fail and you’ll see an error message similar to below:

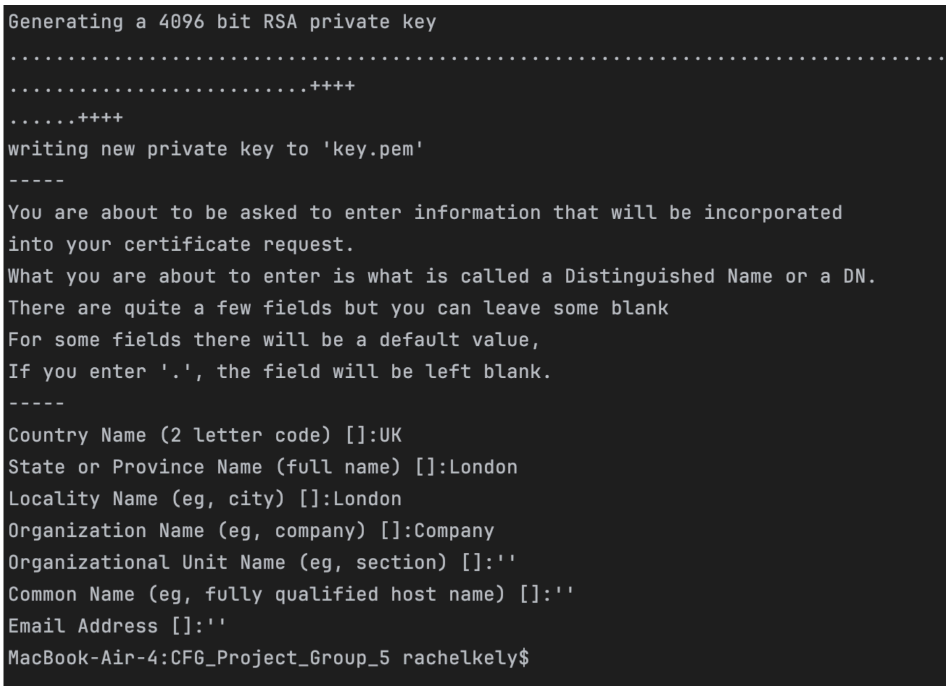

This Connection Is Not Private

This website may be impersonating “127.0.0.1” to steal your personal or financial information. You should close this page.

- You can self certify by typing the following openssl command into your terminal:

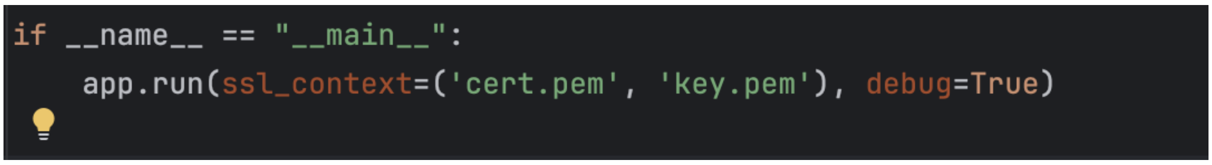
```
rachelkely$ openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days 365
```

- You'll then receive questions that you need to answer for the certificate:



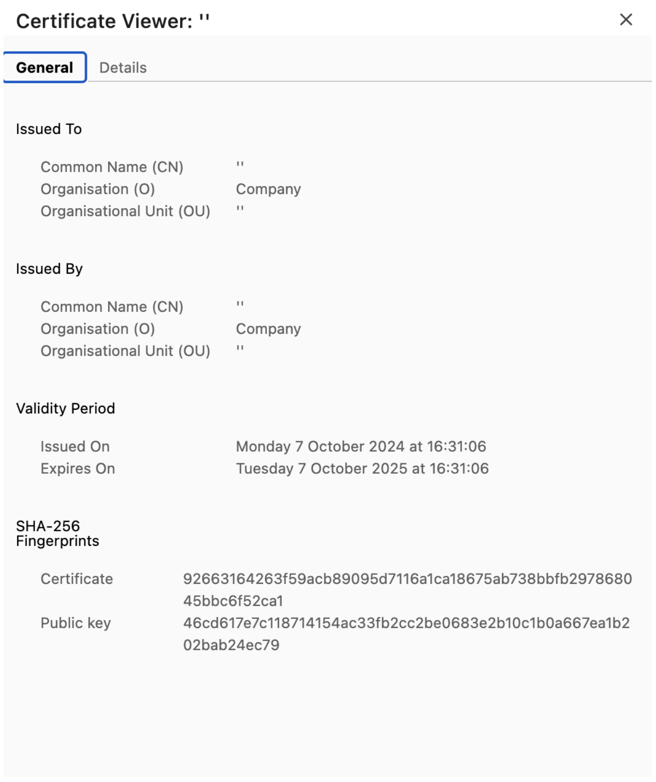
```
Generating a 4096 bit RSA private key
................................................................................
..........................++++
......++++
writing new private key to 'key.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) []:UK
State or Province Name (full name) []:London
Locality Name (eg, city) []:London
Organization Name (eg, company) []:Company
Organizational Unit Name (eg, section) []:''
Common Name (eg, fully qualified host name) []:''
Email Address []:''
MacBook-Air-4:CFG_Project_Group_5 rachelkely$
```

- Add your keys in to the app.run() as follows:



```
if __name__ == "__main__":
    app.run(ssl_context=('cert.pem', 'key.pem'), debug=True)
```

- You'll be able to see your certificate as below, even if it is not secure enough for the server to let you in without warning:



Certificate Viewer: ''

General | Details

Issued To

Common Name (CN)	''
Organisation (O)	Company
Organisational Unit (OU)	''

Issued By

Common Name (CN)	''
Organisation (O)	Company
Organisational Unit (OU)	''

Validity Period

Issued On	Monday 7 October 2024 at 16:31:06
Expires On	Tuesday 7 October 2025 at 16:31:06

SHA-256 Fingerprints

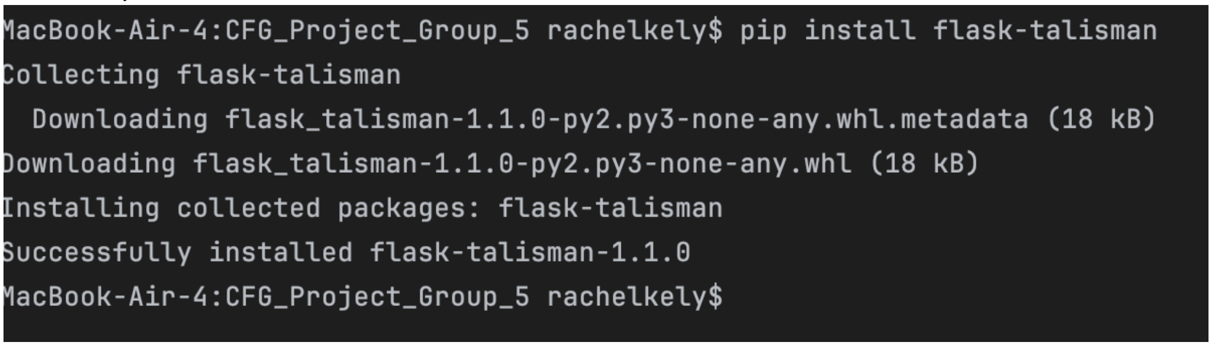
Certificate	92663164263f59acb89095d7116a1ca18675ab738bbfb2978680 45bbc6f52ca1
Public key	46cd617e7c118714154ac33fb2cc2be0683e2b10c1b0a667ea1b2 02bab24ec79

3. **CA certificates -** You can obtain a certificate from a certified authority who will verify that you are in control of your server and domain. Most CA's charge money for this but it is possible to get one for free from Let'sEncrypt. Using a CA, they will have an automated process to create and install your certificate.

How do we enforce HTTPS in our Flask application?

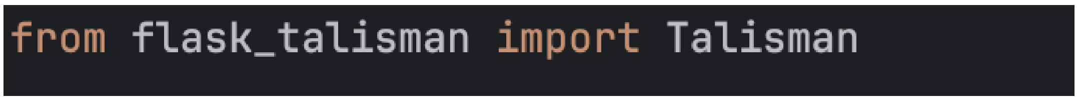
We can enforce HTTPS in our Flask application by using the Flask-Talisman extension.

1. Pip Install flask-talisman:



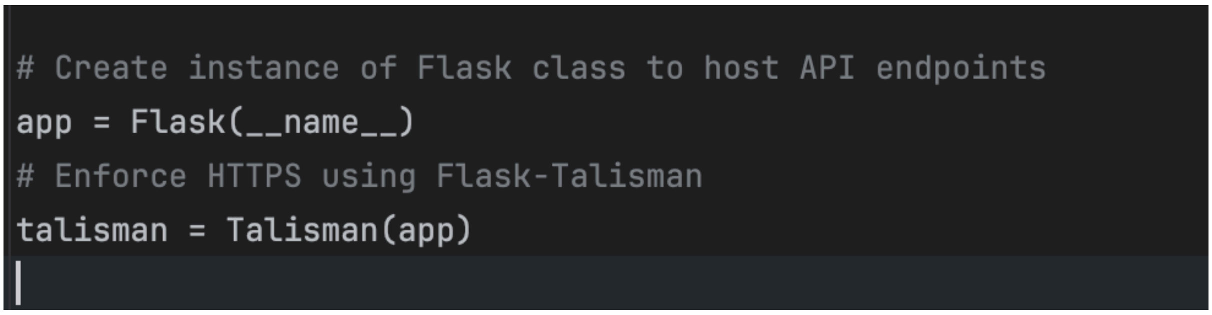
```
MacBook-Air-4:CFG_Project_Group_5 rachelkely$ pip install flask-talisman
Collecting flask-talisman
  Downloading flask_talisman-1.1.0-py2.py3-none-any.whl.metadata (18 kB)
Downloading flask_talisman-1.1.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: flask-talisman
Successfully installed flask-talisman-1.1.0
MacBook-Air-4:CFG_Project_Group_5 rachelkely$
```

2. Import the package on app.py:



```python
from flask_talisman import Talisman
```

3. Declare the Talisman variable:



```python
# Create instance of Flask class to host API endpoints
app = Flask(__name__)
# Enforce HTTPS using Flask-Talisman
talisman = Talisman(app)
```

- Flask-Talisman is now enforcing HTTPS on all routes. Once deployed to a server with an SSL certificate, this ensures that any HTTP requests will be redirected to HTTPS.

Protecting your application data

Once we have developed a strong user authentication and authorisation process, the next step is securing our database which we explore in this section.

Securing an application database is a crucial aspect of ensuring the confidentiality, integrity, and availability of sensitive data. This is known as the CIA triad, a recognised model used in information security.

Confidentiality - Maintaining the appropriate restrictions on access and disclosure of information, with measures in place to safeguard personal privacy and proprietary data.
Integrity - Protecting data from unauthorised alterations, modifications or destruction, and ensuring the authenticity and non-repudiation of information.
Availability - Guaranteeing that information is accessible and usable in a timely and dependable manner when needed.

The key theoretical components involved in protecting our application data would be:
Access control - Role-based access control (RBAC), multi-factor authentication (MFA), principle of least privilege and database firewalls (IP whitelisting).
Data encryption - end to end encryption.
Data integrity and validation - Data validation, data constraints and hashing.
SQL injection prevention - Parameterised queries and object-relational mapping (ORM).
Backup and recovery - Regular backups, testing and backup isolation.
Database auditing and monitoring - Database logs and audit trails.
Security patching and updates - Patch management and updates
Database segmentation and network security - Network segmentation and firewalls.
Compliance and regulatory requirements - Data protection and masking.
Incident response - Response plans.

The best practices to use to secure our user data.
To secure the Movie Recommender application database and user data, a combination of network security, authentication mechanisms, SQL-specific features, and secure coding

practices should be employed. Here is an outline of the best practices from the components above for securing the application:

Network security
IP whitelisting - Restrict access to the MySQL database by allowing only trusted IP addresses, such as localhost or specific ranges. This ensures that only authorised servers or users can connect. To implement this, we could configure the MySQL server to bind to trusted IPs and set up firewall rules to block untrusted addresses.
Firewall rules - Configure firewall rules on the server hosting MySQL to permit connections only from specific IP addresses, blocking unauthorised access.
Encryption - Use encryption to secure communication between the app and the MySQL database, encrypting sensitive user information, login credentials, and the quiz responses during transmission. This could be configured in the application's database connection settings.
Network segmentation - We could have isolated the database server to be on a separate network segment from the application server to reduce vulnerability to attacks.

Multi-factor authentication (MFA)
User authentication - To enhance security in the movie application, implement a multi-factor authentication process that requires users to provide two or more forms of verification during login. Password and one-time code. The users will enter their email address and password and upon successful password verification, they will be prompted to input a one-time code. This code can be sent to their registered email or generated by an authenticator app.
User management - Provide users with the option to manage their MFA settings, including enabling or disabling MFA, providing users with flexibility and control over their account security according to their preferences.
Session management - Implement secure session management using session tokens that require users to re-authenticate after a designated period or when logging in from a new device. This approach could be incorporated into the application's session handling code,to prevent session hijacking by ensuring that users maintain control over their active sessions.

SQL-specific security
SQL injection prevention - Implement parameterised queries in the application's database access logic to treat user input as data, preventing malicious SQL commands. Additionally, could use the object-relational mapping (ORM) for automatic query management, which includes built-in input sanitisation to further reduce SQL injection risks.
Privilege principles - Apply the principle of least privilege by creating specific application roles. This limits application privileges, ensuring, for example, that those responsible for the movie recommendation section of the application cannot modify user accounts. This helps maintain overall security even if an account/role is compromised.
Role-based access control (RBAC) - Define user roles with specific permissions in the MySQL database schema. For instance, establish roles like Admin, User, and Read-Only, determining which actions each can perform.
Stored procedures - Utilise stored procedures for critical data manipulation tasks, such as user registration and fetching movie recommendations. This approach restricts direct access to the underlying database tables, reducing the risk of SQL injection and unauthorised data access.
Database security plugins - Utilise MySQL security plugins to enhance security, such as authentication plugins for additional security layers. In the MySQL server configuration we could install security plugins such as authentication plugins, example: MySQL Enterprise Authentication or LDAP, to strengthen user authentication methods.

Additional security methods
Database auditing and monitoring - Enable MySQL audit logging to track user activities and database changes. This allows monitoring for unauthorised access and ensures accountability. Also integrate monitoring tools for real-time analysis of logs.
Security patching and updates - Regularly apply security patches and updates to the MySQL server and application dependencies. Establish a schedule for reviews and use package managers to keep everything updated, minimising vulnerabilities.
Compliance and regulatory requirements - Implement data protection measures, like data masking to safeguard sensitive user information. Ensuring that compliance with regulations like GDPR and CCPA during user data handling is upheld.
Incident response - Create an incident response plan detailing procedures for handling security breaches. Outline roles and steps for reporting, assessing, and resolving incidents to ensure that if anything happens to the movie recommender it can be fixed in a timely manner.

By integrating a combination of these security measures such as the network security, MFA, and SQL-specific features into the design, deployment, and management of the application database, the Movie Recommender can have a significantly reduced risk of unauthorised access, data breaches, or loss of data.

Regular audits, encryption, and input validation further enhance the overall security, protecting the user data against both internal and external threats. This not only safeguards the applications data but also fosters user trust in the movie application by ensuring robust security practices are consistently upheld.

Logging and Monitoring for enhanced security

The final piece of the puzzle in ensuring we maintain robust security measures for our application, is logging and monitoring access attempts and perceived threats so we can reflect on any major events and plan for possible attacks.

The role of logging and monitoring in enhancing application security and the purpose in recording critical events;
To help us to understand the role of Logging and Monitoring, we first need to understand how they're defined in this context:

Logging is the act of collecting data that we can evaluate at a later date.
Monitoring is the act of observing data in real time.

It is important to be aware of the OWASP guidance on logging and monitoring described below:

Sensitive data in logs must be minimised, encrypted and retained only as long as specified in relevant data retention schedules.

Data recorded in logs must be compliant with local data protection and privacy legislation.

The following events should be logged:
- Successful and failed authentication attempts.
- Failed authorisation attempts.
- Failed deserialisation attempts (for instance, API data from a third party that fails to deserialise as expected could indicate that it had been compromised).
- Failed input validation attempts (for instance, if SQL injection or XSS attempts are detected when validating input data on a form).

Log data should be able to provide incident response teams with a timeline of what happened when. Log data must be protected from unauthorised access and tampering.

Logging critical events enhances security by helping to identify and respond to security threats such as:

- Unauthorised activity and log in attempts.
- Data breaches such as unexpected file changes or data transfers.
- Forensic analyses and timelines of an attack so they can be fully investigated.

Alongside the threat of malicious actors, logging and monitoring can also enhance how the application is working generally, for example:

- They can assist in helping teams troubleshoot any technical issues like network connectivity issues or system failures.
- It is possible to track things like response time and user behaviour which you can analyse and use to improve aspects of the application and an alert can be set for the team in real time.

In the context of Movie Recommender, we would be logging and monitoring attempts to access user data (to maintain the privacy of the data). We would also be logging and monitoring attempts to manipulate the dataset (the integrity of the data), for example by someone creating fake users en masse who are all claiming to like the same movie in order to manipulate the results received by other users.

On a technical level, the critical events in Movie Recommender are: user authentication and authorisation, API calls, errors, data access, and data modification.

Monitoring tools or services that would contribute to this threat detection:
Snort is a widely used monitoring tool. Depending on the size of Movie Recommender's user base, Snort's Packet Logger mode could be used in the first instance, to log data about critical events. However if there is a large user base, or a threat has been detected, its Network Intrusion Detection System (NIDS) can be used so the threats detected by logging and monitoring can more easily be fed into a protection system.

Snort can monitor incoming and outgoing traffic to the API server, allowing for real-time analysis of requests and responses. It helps identify unusual traffic patterns that could indicate malicious activity, such as SQL injection attempts and can be configured to detect common API vulnerabilities, using pre-existing rules collected by the community to identify known patterns that indicate an attack. Movie Recommender has different API endpoints that could be targeted in different ways, and Snort can be configured to detect the different types of attacks associated with different end points and reduce false positives.

Additional tools:
Additional tools that could be used include **Uncomplicated Firewall** to secure the server the application is stored on. This is more focused on stopping attacks, but also contains a log which could be monitored and is generally used for troubleshooting the firewall, rather than monitoring traffic in its own right.

How to handle software updates and patches:
Some updates will be associated with the software the tools rely on, and some updates and patches will be to Movie Recommender's specific use case, i.e. what rules we are using.

Updates and patches would be handled by:

1. Having a plan in place.

2. Understanding if the update is security critical or non-urgent: if it’s critical, it needs to be applied immediately. If it is not related to security then it can be worth waiting a week or so to not risk pushing something before an error has been discovered – for example with crowdstrike.
3. Testing in a staging environment before deployment.
4. Monitoring after the update.

Conclusion

In conclusion, this report of our Movie Recommender application highlights the importance of addressing security vulnerabilities to protect user data and maintain the integrity of the application. Through the identification of potential threats, we have demonstrated how these vulnerabilities can compromise both user data and application functionality.

The proposed enhancements are essential steps in mitigating these risks and each improvement contributes significantly to the overall security of the app, ensuring that user interactions with the application remain secure and trustworthy.

Ultimately, each proposed improvement plays a vital role to a more resilient Movie Recommender application that not only meets user expectations but also safeguards their information against evolving cyber threats, guaranteeing that the Movie Recommender remains a secure and enjoyable application for all users.