****

Secure System Design

**Flask Web Application**

Anno Accademico 2023/2024

Professoressa

**Prof. Valentina Casola**

Studente

**Marco Cimmino M63001528**

Sommario

[Introduzione e descrizione della Web App 4](#_Toc184837107)

[**Ruoli e Funzionalità** 4](#_Toc184837108)

[**Controllo degli Accessi** 4](#_Toc184837109)

[Asset del Sistema e Tecnologie per Proteggerli 5](#_Toc184837110)

[Diagramma di deployment del sistema: *in fase di sviluppo….* 6](#_Toc184837111)

[Nginx Reverse Proxy 7](#_Toc184837112)

[**Configurazione e Integrazione con la Web App e Keycloak** 7](#_Toc184837113)

[Gestione dei Segreti con Vault 8](#_Toc184837114)

[**Configurazione e Scrittura dei Segreti** 8](#_Toc184837115)

[**Storage dei segreti** 8](#_Toc184837116)

[**Lettura dei Segreti nella Web App** 9](#_Toc184837117)

[**Integrazione con la Web App** 10](#_Toc184837118)

[WebApp con IAM Keycloak + OpenIDconnect 11](#_Toc184837119)

[**Protocollo e configurazione** 11](#_Toc184837120)

[**Autenticazione** 14](#_Toc184837121)

[**Login** 14](#_Toc184837122)

[**Logout** 15](#_Toc184837123)

[**2FA (OTP)** 16](#_Toc184837124)

[**Gestione dei Ruoli e Autorizzazioni** 17](#_Toc184837125)

[Gestione di Sessioni e Cookie 18](#_Toc184837126)

[**Definizioni** 18](#_Toc184837127)

[WebApp con LDAP 21](#_Toc184837128)

[**Configurazione e avvio del Server LDAP** 21](#_Toc184837129)

[**Gestione della Directory LDAP** 23](#_Toc184837130)

[**Associazione tra Utenti e Gruppi** 23](#_Toc184837131)

[**Processo di Login con LDAP** 23](#_Toc184837132)

[**Autenticazione** 24](#_Toc184837133)

[Autorizzazione tramite XACML 25](#_Toc184837134)

[**Limitazioni di XACML con Flask** 26](#_Toc184837135)

[**Py-abac: Simulazione di XACML in Flask** 26](#_Toc184837136)

[**Politica admin\_policy** 27](#_Toc184837137)

[**Politica user\_policy** 28](#_Toc184837138)

[**Differenze principali rispetto a XACML:** 28](#_Toc184837139)

[Valutazione della sicurezza con Microsoft Threat Modeling Tool: KeyCloak 29](#_Toc184837140)

[**Denial of Service (DoS) su Nginx e Flask Web App** 30](#_Toc184837141)

[**Elevation of Privilege: Impersonation attack** 31](#_Toc184837142)

[**Information Disclosure sui File di Configurazione** 32](#_Toc184837143)

[**Repundiation: Potential Data Repudiation by Browser Client** 33](#_Toc184837144)

[**Spoofing: DNS Spoofing e Man-in-the-Middle (MITM)** 34](#_Toc184837145)

[**Tampering su HTTPS: attacco XSS** 35](#_Toc184837146)

[**Tampering sul Database: SQL Injection** 36](#_Toc184837147)

[**Replay Attacks** 37](#_Toc184837148)

[Valutazione della sicurezza con Microsoft Threat Modeling Tool: LDAP 38](#_Toc184837149)

[**Gestione del Login: Web App come Authenticator** 38](#_Toc184837150)

[Catalogo dei controlli di sicurezza dello standard NIST 39](#_Toc184837151)

[**Famiglia: Access Control (AC)** 39](#_Toc184837152)

[**Famiglia: Identification and Authentication (IA)** 39](#_Toc184837153)

[**Famiglia: System and Communications Protection (SC)** 40](#_Toc184837154)

[**Famiglia: System and Information Integrity** (**SI**) 40](#_Toc184837155)

[**Famiglia: Audit and Accountability** (**AU**) 40](#_Toc184837156)

[NIST CSF – KeyCloak + Confronto con LDAP 41](#_Toc184837157)

# **Introduzione e descrizione della Web App**

La web app consente agli utenti registrati di scaricare l'APK di un gioco in fase di sviluppo, permettendo loro di provare una versione preliminare del gioco. La funzionalità di download è disponibile attraverso una pagina dedicata. La web app offre anche una pagina per la visualizzazione di un trailer del gioco ed una per la scrittura di note/suggerimenti. L’admin invece potrà dare upload di nuove .apk del gioco aggiornato e potrà visualizzare i suggerimenti di tutti gli user.

## **Ruoli e Funzionalità**

La web app supporta due ruoli principali: User e Admin, ciascuno con specifiche funzionalità e accessi.

### **1. User (Utente)**

Un user è qualsiasi cliente che si registra e accede al sito tramite la funzionalità di registrazione e login. Gli user hanno accesso alle seguenti pagine:

*Home:* Visualizzazione del trailer e della descrizione del gioco.

*Note:* Possibilità di lasciare suggerimenti e commenti sul gioco.

*Download:* Possibilità di scaricare l'APK della versione attuale del gioco.

Un user non ha accesso alle pagine dedicate agli admin, come la pagina di upload e la pagina di gestione delle recensioni (check\_reviews).

### **2. Admin (Amministratore)**

Un admin è un utente con privilegi speciali che ha accesso a due ulteriori pagine rispetto agli user:

*Upload:* Pagina dedicata all'upload di nuove versioni dell'APK del gioco. Le versioni caricate saranno poi visibili nella pagina di download.

*Vista\_note:* Pagina che consente di visualizzare tutti i commenti e le note lasciate dagli utenti. Gli admin possono accedere a tutte le recensioni, mentre gli user vedono solo le proprie.

## **Controllo degli Accessi**

L'architettura della web app garantisce che gli user possano solo interagire con le funzionalità di base del sito (visualizzare il trailer, scaricare il gioco, lasciare recensioni) e non abbiano accesso alle pagine riservate agli admin (come l'upload di nuove versioni o la visualizzazione delle recensioni degli altri utenti).

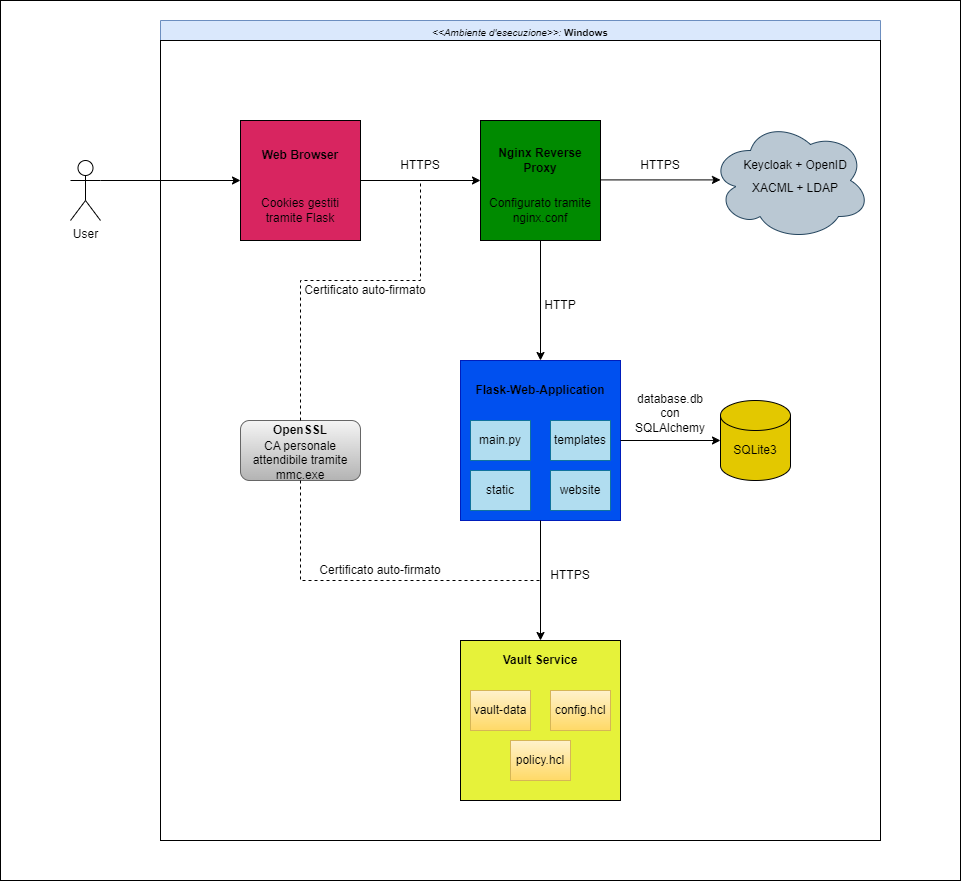
Gli admin, invece, hanno accesso completo alla gestione del contenuto del sito, inclusa la possibilità di caricare nuove versioni del gioco e visualizzare tutte le recensioni.

# **Asset del Sistema e Tecnologie per Proteggerli**

La web app gestisce diversi asset che devono essere protetti da potenziali minacce. Di seguito sono riportati gli asset principali, le minacce a cui sono soggetti e le tecnologie implementate per garantirne la sicurezza:

1. **Web Server che ospita la web app**
   * **Tecnologia di protezione**: Il web server è protetto tramite **reverse proxy Nginx**, che funge da barriera tra il client e il server, migliorando la sicurezza e la gestione del traffico. Nginx aiuta a filtrare il traffico malevolo e distribuisce le richieste in modo sicuro al server web.
2. **Risorse della Web App (percorsi upload e views in fase di sviluppo)**
   * **Tecnologia di protezione**: Le risorse sono protette tramite un **Identity and Access Management System (IAM)**, che garantisce che solo gli utenti con i permessi appropriati possano accedere ai percorsi sensibili come upload e views.
3. **Sessioni e Cookie**
   * **Tecnologia di protezione**: Le sessioni e i cookie sono protetti tramite una configurazione sicura all'interno di **Flask**, che gestisce la creazione, la validità e la protezione contro il furto di sessione. Vengono utilizzate opportune impostazioni come il flag HttpOnly, Secure e SameSite per proteggere i cookie.
4. **Database**
   * **Tecnologia di protezione**: Il database è grazie a Flask che offre protezioni di base contro SQL Injection e XSS tramite query parametrizzate e escaping automatico in Jinja2. I dati come le credenziali degli utenti e altre informazioni sensibili vengono conservati nei rispettivi webserver Keycloak e LDAP, impedendo l'accesso non autorizzato anche in caso di compromissione del database.
5. **Credenziali Scritte in Chiaro nel Codice**
   * **Tecnologia di protezione**: Le credenziali scritte in chiaro nel codice sono protette mediante l'uso di un **secret manager**, come **Vault**, che gestisce in sicurezza le credenziali, evitando che vengano esposte o memorizzate direttamente nel codice sorgente.
6. **Connessioni tra Web App e Vault Server**
   * **Tecnologia di protezione**: Le connessioni tra la web app e il server di Vault sono protette tramite un **protocollo crittografico a livello di trasporto (TLS)**, che garantisce la sicurezza delle comunicazioni attraverso l'uso di HTTPS.

# **Diagramma di deployment del sistema:** *in fase di sviluppo….*

****

# **Nginx Reverse Proxy**

**Nginx** è un server web che può essere utilizzato anche come **reverse proxy**, ossia un intermediario che riceve le richieste HTTP dai client e le inoltra ai server interni, fungendo da barriera protettiva per i server stessi.

Nella web app, **Nginx** è stato implementato come reverse proxy con l'obiettivo di creare un confine di sicurezza tra la web app e l'esterno. In particolare, è stato utilizzato per i seguenti scopi:

* **Isolamento della web app**, proteggendo Vault da accessi esterni, garantendo che solo le richieste legittime raggiungano i server interni.
* **Gestione centralizzata del protocollo TLS/HTTPS**, assicurando che tutte le comunicazioni tra il client e il server siano crittografate e sicure.

## **Configurazione e Integrazione con la Web App e Keycloak**

Nginx viene configurato tramite un apposito file di configurazione, in cui è possibile specificare le impostazioni per ogni server per cui Nginx funge da reverse proxy. In questo caso, la configurazione riguarda per il momento solo la web app.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

La configurazione per la web app include i seguenti elementi:

* **Porta di ascolto**: La porta sulla quale Nginx attende le richieste in ingresso.
* **Abilitazione del protocollo TLS**: Nginx è configurato per gestire il traffico HTTPS, utilizzando certificati digitali.
* **Percorso di base (base path)**: Definito per accedere alla web app e alle sue risorse, indicando l'URL della web app e specificando i parametri che devono essere inoltrati tra client e server. Questi parametri vengono passati come header nelle richieste e risposte HTTP, garantendo che le informazioni necessarie vengano correttamente trasmesse tra il client e il server.

# **Gestione dei Segreti con Vault**

HashiCorp Vault è un sistema di gestione dei segreti e della crittografia basato sull'identità. Nella web app, Vault viene utilizzato per evitare che le chiavi e altre informazioni sensibili siano scritte in chiaro nel codice sorgente.

I segreti protetti da Vault, al momento, includono:

* **La secret key** utilizzata da Flask per cifrare le sessioni.
* **L'URI del database**, utilizzato per connettersi al sistema di gestione dei dati.

## **Configurazione e Scrittura dei Segreti**

Vault è configurato tramite un file di configurazione che definisce i seguenti parametri:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

* Abilitazione del protocollo TLS.
* La porta di ascolto.
* I certificati digitali (questi ultimi generati tramite una CA personale usando OpenSSL e aggiunti ai certificati attendibili tramite mmc.exe).

## **Storage dei segreti**

In questo caso, i segreti sono memorizzati su file (vault-data).

Alla prima esecuzione in modalità produzione, dopo aver eseguito il comando *vault operator init* e configurato opportunamente il file di configurazione, vengono generati i seguenti elementi:

* Cinque chiavi di sblocco (unseal keys), necessarie per sbloccare Vault ad ogni avvio. Tre delle cinque chiavi sono richieste per fare l’unseal di vault. Si può verificare lo stato di vault tramite il comando *vault status*
* Un root token, che fornisce privilegi amministrativi completi su Vault, e viene utilizzato per configurare le politiche di accesso.

Nel contesto della web app, è stata creata una policy ad hoc, in linea con il principio del privilegio minimo, per limitare l'accesso ai segreti:

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Una policy in sola lettura (read-only-policy), che consente alla web app di leggere i segreti senza la possibilità di modificarli.

## **Lettura dei Segreti nella Web App**

Per integrare Vault nella web app, è necessario definire un **AppRole**. L'utilizzo di Vault nella web app implica l'inserimento di credenziali per accedere ai segreti, ma l'accesso deve essere limitato esclusivamente alla web app stessa.

Per questo scopo, Vault consente la creazione di AppRole, che sono ruoli progettati specificamente per le applicazioni. Nella creazione di un AppRole, è necessario configurare i seguenti parametri di sicurezza:

* Tempo limite di utilizzo delle credenziali (ad esempio, 60 minuti).
* Numero massimo di utilizzi delle credenziali (ad esempio, un valore di "1" rende le credenziali usa e getta).
* La policy di accesso a Vault (ad esempio, la policy in sola lettura).

Una volta creato l'AppRole, è possibile ottenere le credenziali necessarie per l'accesso dalla web app: il Role ID e il Secret ID. Il Role ID è fisso, mentre il Secret ID è temporaneo e viene generato in base ai parametri configurati. Il Secret ID funge da password temporanea per il Role ID, e la sua validità è determinata dalla configurazione.

## **Integrazione con la Web App**

L'integrazione di Vault nella web app è stata effettuata utilizzando la libreria hvac, rilasciata da HashiCorp, che facilita l'interazione con l'API di Vault.

Di seguito è riportata la classe VaultConfig che fornisce metodi per inizializzare il client (WebApp), autenticarla e recuperare i segreti tramite il Server vault.

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

1. **Costruttore (\_\_init\_\_)**: Quando viene creata un'istanza della classe, viene stabilita una connessione al server Vault utilizzando l'URL https://127.0.0.1:8200 e un certificato per la verifica del server. Inoltre, vengono definiti un role\_id e un secret\_id, necessari per l'autenticazione tramite AppRole.
2. **Metodo authenticate**: Questo metodo autentica l'applicazione su Vault utilizzando il role\_id e il secret\_id. Dopo l'autenticazione, il token di accesso (client\_token) viene estratto dalla risposta e assegnato al client, consentendo l'accesso alle risorse protette in modalità solo lettura.
3. **Metodo get\_secret**: Questo metodo consente di leggere un segreto da Vault specificando un secret\_path. Una volta recuperato il segreto, il valore viene restituito.

# **WebApp con IAM Keycloak + OpenIDconnect**

La gestione di identificazione, autenticazione e autorizzazione è stata implementata tramite **Keycloak**, un sistema di Identity and Access Management (IAM) utilizzato nella versione fornita all’interno del teams.

Keycloak adotta il concetto di **security realm**, spazi isolati dove vengono definiti utenti, ruoli, risorse e le relative policy di accesso. La gestione del sistema è affidata a un amministratore di sicurezza che accede tramite un **account amministrativo** separato dagli account degli utenti del realm, in conformità al principio della separazione tra amministrazione e accesso. L’interfaccia grafica della **console di amministrazione** consente di configurare e monitorare il sistema in modo facile e veloce.

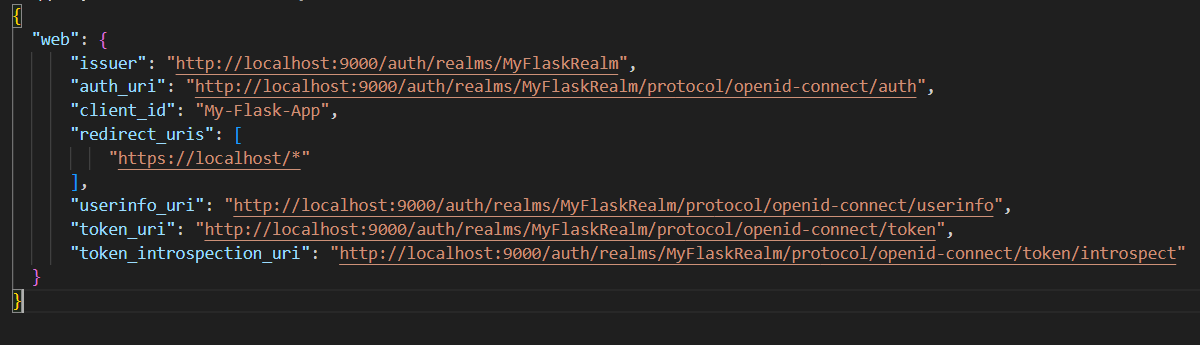
Per la web app è stato creato un **realm dedicato**, denominato *WebAppRealm*, in cui la web app opera come client, identificato come *FlaskWebApp*.

## **Protocollo e configurazione**

Il progetto utilizza il protocollo **OpenID Connect (OIDC)** per la delega di identificazione, autenticazione e autorizzazione. La configurazione del *realm* *WebAppRealm* è stata effettuata tramite la console di amministrazione di Keycloak, includendo:

* **Creazione degli utenti di partenza, dei ruoli e del client** associati al *realm*.
* Configurazione del client *FlaskWebApp*:
  + **Tipo**: *confidential* (capace di mantenere segreti).
  + **Grant type**: *authorization code* (modalità basata su codice di autorizzazione).
  + **Protocollo di delega**: *OpenID Connect*.
  + **Altre informazioni**: specifica dell'URL di redirezione post-login per l'utente.

L’integrazione di Keycloak nel codice della web app è stata implementata utilizzando la libreria **flask\_oidc.** Nella figura seguente sono mostrati i parametri di configurazione di Keycloak salvati in un .json separato necessari per l'inizializzazione della web app.

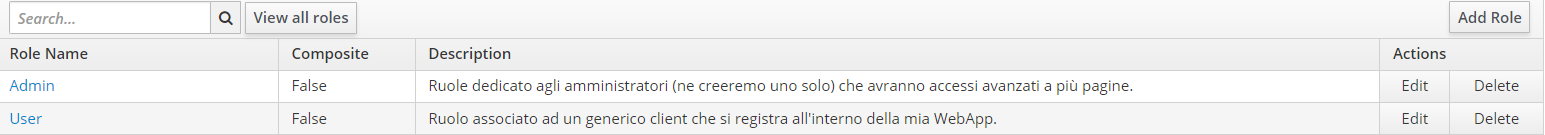


Nell’\_\_init\_\_.py viene richiamato il file .json insieme ad altri parametri di configurazione:  
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Da notare come il client\_secret non era esposto in chiaro nel file ma viene richiamato tramite vault. Keycloak è configurato in http poiché non accetta l’esecuzione con certificati auto-signed e di conseguenza OIDC\_COOKIE\_SECURE è importato a false. Con certificati validi basterebbe settarlo a True e cambiare gli url da http a https per abilitare il protocollo.

Sono poi stati definiti due ruoli: Admin e User



Per poter ricavare il ruolo dagli user\_info dell’utente è stato aggiunto un parametro nella sezione Mappers:

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

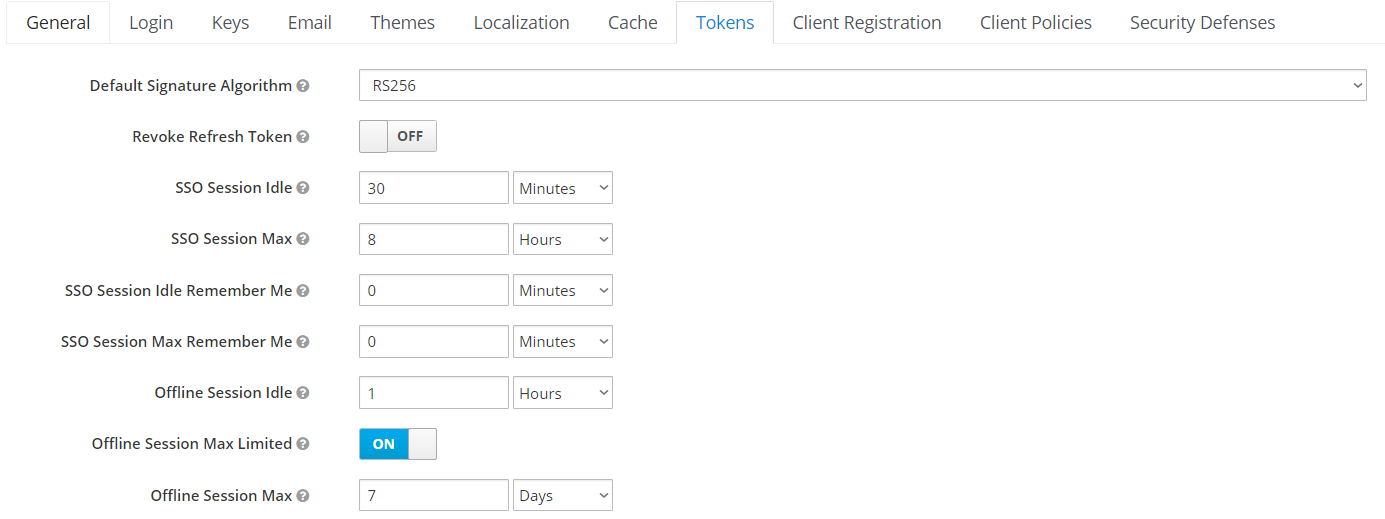
Un utente che si registra avrà come ruolo di default lo User, ed è settabile nella sezione Roles:

Immagine che contiene testo, linea, Carattere, numero

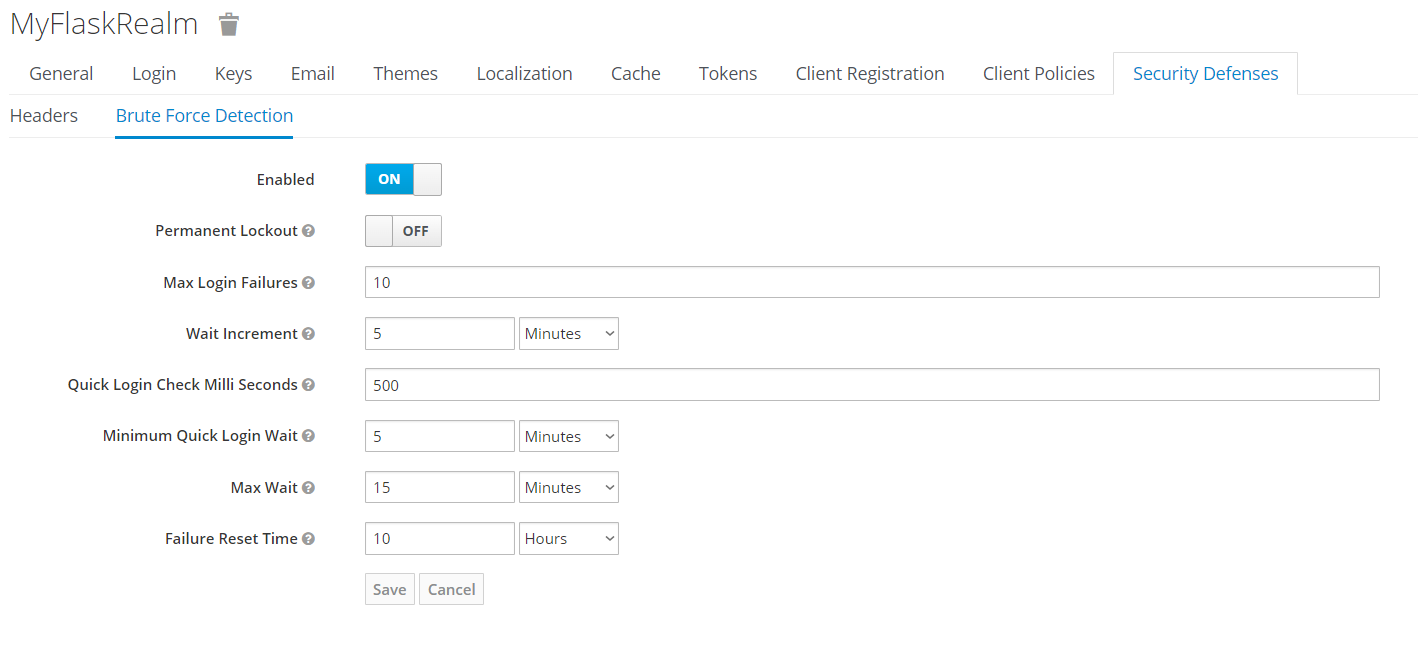
Descrizione generata automaticamente

Di seguito sono riportate le configurazioni per quanto riguarda durata sessioni, login consecutivi e Auditing

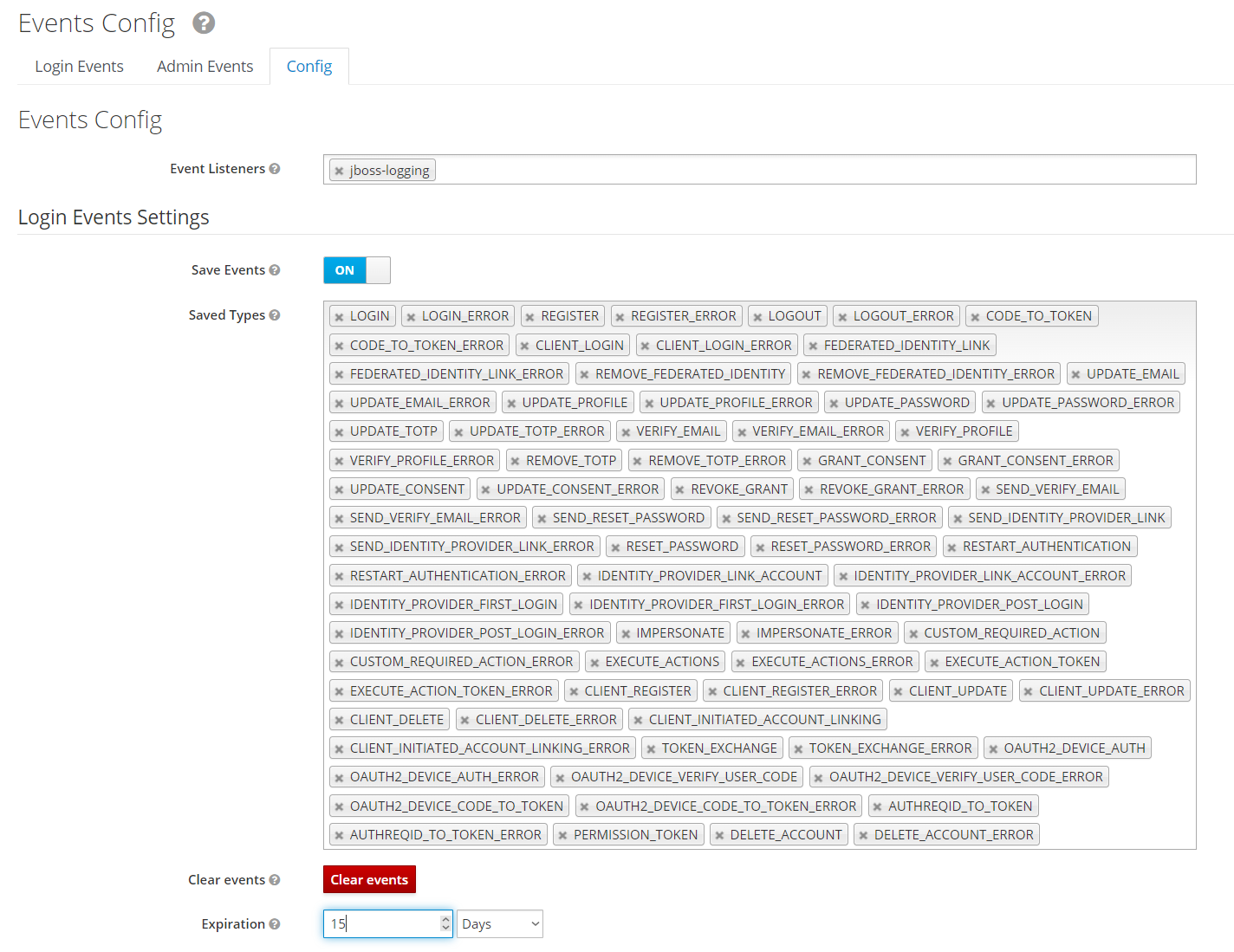
Token sessione: Impostata una durata in idle di 30 minuti, con un limite massimo di 8 ore.



Login multipli: Abilitato il brute force detection per evitare tentativi di brute forcing sul login. Limite di 10 login consecutivi falliti con un wait time di 5 minuti incrementale.



Auditing: Tutti gli eventi sono salvati nei log e visualizzabili nella sezione Admin Events. Hanno una durata di 15 giorni.



## **Autenticazione**

Dopo aver configurato correttamente il webserver Keycloak, si passa alla fase di autenticazione. La web app supporta le funzionalità di Login e Logout offerte da Keycloak, sfruttando il protocollo OpenID Connect (OIDC). Queste operazioni garantiscono una gestione sicura delle sessioni utente, centralizzata e conforme alle best practice di sicurezza.

## **Login**

Il login è stato implementato manualmente seguendo la specifica **OpenID Connect (OIDC)**. Il flusso di autenticazione è rappresentato nel seguente sequence diagram:

1. **Richiesta di login**:
   * L’utente invia una richiesta di login alla web app.
2. **Richiesta di autorizzazione**:
   * La web app redirige l’utente all’**endpoint di autorizzazione** di Keycloak (/auth).
   * Qui, Keycloak autentica l’utente (es. tramite username e password).
3. **Codice di autorizzazione**:
   * Dopo l'autenticazione, Keycloak risponde alla web app con un **authorization code**. Questo codice è temporaneo e viene utilizzato per ottenere i token di autenticazione.
4. **Richiesta di token**:
   * La web app invia una richiesta all’**endpoint di rilascio token** di Keycloak (/token), includendo:
     + Il codice di autorizzazione ricevuto.
     + Le credenziali del client (client ID e client secret).
5. **Rilascio dei token**:
   * Keycloak risponde con i token richiesti:
     + **Access token**: per autorizzare l’accesso a risorse protette.
     + **ID token**: contenente le informazioni dell’utente.
     + **Refresh token**: per ottenere nuovi token senza chiedere nuovamente il login.
6. **Richiesta informazioni utente**:
   * La web app utilizza l’**access token** per inviare una richiesta all’**endpoint delle informazioni utente** di Keycloak (/userinfo).
7. **Risposta con i dettagli utente**:
   * Keycloak restituisce i dettagli utente (es. nome, email), utilizzati per popolare la sessione lato web app.
8. **Accesso completato**:
   * La web app restituisce la homepage (o una pagina specifica) all’utente autenticato.



## **Logout**

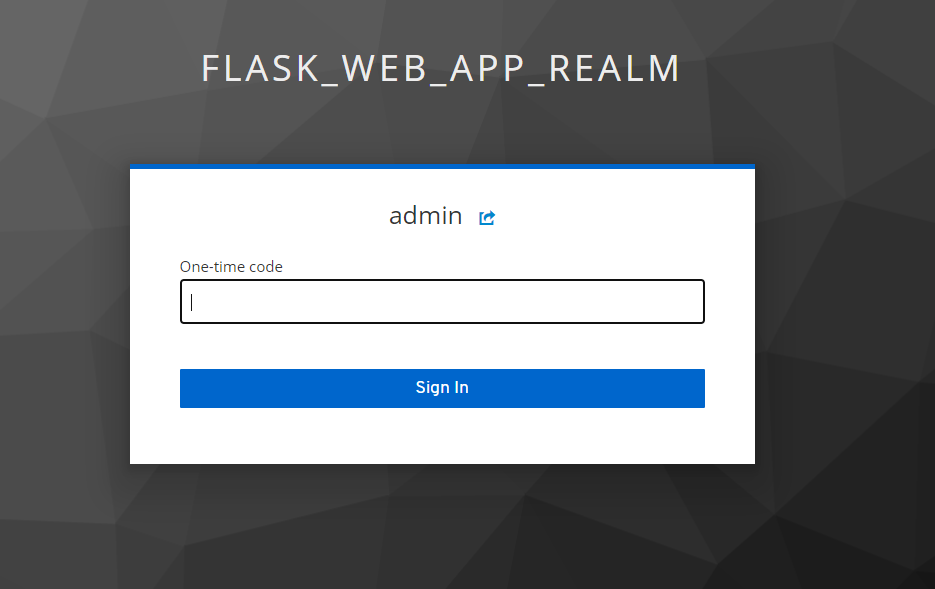
Il logout segue un flusso più semplice:

1. **Richiesta di logout**:
   * La web app invia una richiesta all’**endpoint di logout** di Keycloak (/logout)
2. **Chiusura sessione su Keycloak**:
   * Keycloak elimina la sessione lato server associata all’utente.
3. **Eliminazione sessione lato web app**:
   * La web app effettua un session.clear() per eliminare la sessione anche lato app.

## **2FA (OTP)**

È stata aggiunta anche la 2FA andando ad impostare su required il flusso OTP.

In questo modo si ottiene una maggiore sicurezza ed un ulteriore controllo soddisfatto per il NIST.



## **Gestione dei Ruoli e Autorizzazioni**

Come detto prima, nella web app ogni utente è associato a un ruolo che viene mappato all'interno delle sue informazioni utente (user\_info). Questo approccio permette di gestire in modo dinamico l'accesso alle risorse della web app in base al ruolo specifico di ciascun utente. La gestione dei ruoli è stata implementata come segue:

**1. Ruoli degli Utenti**

* **Ruolo Admin**: L'utente con il ruolo di *admin* è univoco e definito manualmente. L'utente admin ha accesso completo a tutte le risorse della web app e dispone di privilegi amministrativi.
* **Ruolo User**: Gli utenti che si registrano sulla piattaforma vengono automaticamente associati al ruolo *user*, che corrisponde a un accesso limitato alle risorse disponibili. L'accesso è condizionato dai permessi associati a questo ruolo.

**2. Controllo degli Accessi**

Al momento di ogni richiesta di accesso a una pagina, la web app estrae il ruolo dell'utente dalle sue informazioni ed in base al ruolo estratto verifica se l'utente è autorizzato ad accedere alla risorsa richiesta o meno.

Se l'utente ha il ruolo richiesto per quella risorsa, l'accesso viene concesso.

**3. Gestione degli Accessi Non Autorizzati**

Per proteggere le risorse da accessi non autorizzati, sono stati implementati i seguenti meccanismi:

* **Offuscamento delle Pagine Protette**: Le risorse a cui gli utenti non hanno accesso vengono nascoste all'interfaccia utente. Più in generale, un qualsiasi utente potrà visualizzare nella Navbar solo le rotte a cui ha effettivamente accesso in base al suo ruolo. Tuttavia, i percorsi URL corrispondenti a tali risorse rimangono tecnicamente raggiungibili.
* **Pagina di Errore**: È stata creata una pagina di errore che viene visualizzata nel caso in cui un utente tenti di accedere a una risorsa per cui non ha i permessi necessari. Questo meccanismo permette di gestire in modo sicuro i tentativi di accesso forzato.

Immagine che contiene testo, Carattere, schermata, Elementi grafici

Descrizione generata automaticamente

# **Gestione di Sessioni e Cookie**

La gestione delle sessioni è un aspetto fondamentale per implementare le funzionalità di **Login** e **Logout**, in quanto consente al server di mantenere lo stato della comunicazione con il client, superando la natura stateless del protocollo HTTP. In questo contesto, è essenziale comprendere il ruolo di **sessioni** e **cookie**, nonché il loro utilizzo all'interno di Flask.

## **Definizioni**

1. **Cookie**:  
   Un cookie è un piccolo frammento di dati inviato dal server al client, che viene memorizzato nel browser del client. Questo consente di identificare e tracciare il client durante le richieste successive.
2. **Sessione**:  
   Una sessione rappresenta uno stato persistente mantenuto dal server per un determinato client. Le informazioni di sessione possono essere memorizzate lato server o lato client, a seconda del metodo di implementazione.

L'uso combinato di cookie e sessioni permette al server di riconoscere un client specifico e mantenere dati persistenti tra le diverse richieste. Flask supporta due approcci principali per la gestione delle sessioni, entrambi basati sull'utilizzo dei cookie: Client-side e Server-side.

Si è deciso di utilizzare all’interno dell’applicazione sessioni di tipo server-side, in modo che quest’ultima venga memorizzata sul **server**, mentre nel cookie viene memorizzato solo un **ID univoco** della sessione. Questo approccio è più sicuro rispetto a quello client-side, poiché il contenuto della sessione non è esposto nel cookie. Persistono tuttavia attacchi legati al furto del cookie (hijack), Session Fixation, XSS, CSRF ecc…

Per mitigare i rischi legati a sessioni/cookie, sono previste le seguenti misure di sicurezza:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**1. Chiave Segreta (SECRET\_KEY)**  
Definisce una chiave segreta utilizzata da Flask per proteggere i dati sensibili, come i cookie delle sessioni e i token CSRF. Questa chiave deve essere unica e segreta, preferibilmente generata casualmente.

**2. Nome del Cookie della Sessione (SESSION\_COOKIE\_NAME)**  
Specifica il nome del cookie utilizzato per le sessioni Flask. In questo caso, il nome è definito come **prova**. È importante scegliere un nome unico per evitare conflitti con altri cookie.

**3. URI del Database (SQLALCHEMY\_DATABASE\_URI)**  
Imposta il percorso del database SQLAlchemy utilizzato dalla web app. Il valore del parametro database\_uri deve definire il tipo di database e il suo percorso

**4. Cookie HTTPOnly (SESSION\_COOKIE\_HTTPONLY)**  
Abilita il flag **HttpOnly** per i cookie delle sessioni, impedendo l'accesso ai cookie tramite codice JavaScript. Questa impostazione migliora la sicurezza contro attacchi XSS, Hijacking e furto di cookie.

**5. Tipo di Sessione (SESSION\_TYPE)**  
Specifica che le sessioni verranno memorizzate lato server utilizzando il **filesystem**. Questo approccio conserva le sessioni in modo sicuro rispetto al default client-side di Flask.

**6. Persistenza della Sessione (SESSION\_PERMANENT)**  
Configura le sessioni come **non permanenti**. Ciò significa che le sessioni scadono automaticamente quando il browser viene chiuso.

**7. Cookie SameSite (SESSION\_COOKIE\_SAMESITE)**  
Imposta il flag **SameSite** con valore **Strict** per i cookie delle sessioni, impedendo che vengano inviati in richieste provenienti da altri siti. Questa impostazione riduce il rischio di attacchi CSRF e Session Fixation.

**8. Cookie Secure (SESSION\_COOKIE\_SECURE)**  
Abilita il flag **Secure**, che forza l'invio dei cookie delle sessioni solo su connessioni HTTPS. Garantisce che i cookie non siano trasmessi in chiaro su connessioni HTTP.

**9. Directory per le Sessioni (SESSION\_FILE\_DIR)**  
Specifica la directory **flask\_sessions** dove verranno salvati i file delle sessioni lato server. Questa directory è creata dinamicamente nel percorso specificato da app.instance\_path.

**10. Durata della Sessione (PERMANENT\_SESSION\_LIFETIME)**  
Configura una durata massima per le sessioni impostandola a **10 minuti**. Dopo questo intervallo, l'utente dovrà autenticarsi nuovamente.

**11. Schema URL Preferito (PREFERRED\_URL\_SCHEME)**  
Specifica che lo schema preferito per le URL della web app è **HTTPS**, garantendo che tutte le comunicazioni avvengano su connessioni sicure.

# **WebApp con LDAP**

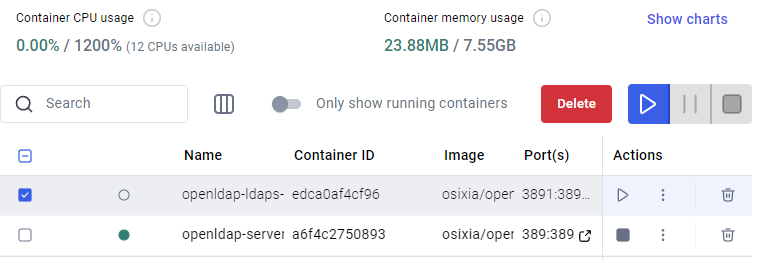
L'integrazione di un **server LDAP (Lightweight Directory Access Protocol)** nella nostra web app consente di gestire in modo centralizzato l'identificazione, l'autenticazione e l'autorizzazione degli utenti. LDAP è un protocollo standard ampiamente utilizzato per accedere e gestire directory di informazioni su utenti, gruppi e altre risorse.

La configurazione del nostro server LDAP avviene tramite tramite Docker, mentre il popolamento della directory avviene attraverso lo strumento **Apache Directory Studio**, nel quale andremo a definire delle unità organizzative (OU) per la gestione di ruoli e utenti nella web application.

## **Configurazione e avvio del Server LDAP**

Il server LDAP è stato avviato utilizzando il seguente comando Docker:

*docker run --name openldap-server -e LDAP\_ORGANISATION="LDAP\_Example" -e LDAP\_DOMAIN="flaskwebapp.com" -e LDAP\_ADMIN\_PASSWORD="admin" -p 389:389 -d osixia/openldap:latest*



La configurazione del server avviene come al solito dentro l’\_\_init\_\_ ed è la seguente:  
Immagine che contiene testo, schermata

Descrizione generata automaticamente

Come per Keycloak, anche qui i certificati auto-signed non sono accettati, e di conseguenza è stato configurato in http, commentando la configurazione in https.

**Connessione al Server LDAP**

* **LDAP\_HOST**: Indica l'host del server LDAP, nel nostro caso localhost.
* **LDAP\_PORT**: Specifica la porta su cui il server LDAP ascolta. Per il protocollo non cifrato (LDAP), il valore predefinito è 389.

**Sicurezza TLS/SSL (LDAPS)**

* **LDAP\_USE\_TLS**: Specifica se utilizzare una connessione sicura TLS.
* **LDAP\_PORT**: La porta predefinita per LDAPS è 636.
* **LDAP\_REQUIRE\_CERT**: Indica se il server deve richiedere certificati.
* **LDAP\_TLS\_CA\_CERTS\_FILE**: Percorso al file del certificato della CA necessario per validare il certificato del server durante una connessione LDAPS.

**Base DN e Autenticazione**

* **LDAP\_BASE\_DN**: Il punto di partenza per le ricerche LDAP, definisce la radice della directory. Nel nostro caso, è dc=flaskwebapp,dc=com.

**Distinguished Names per Utenti e Gruppi**

* **LDAP\_USER\_DN**: Percorso nella directory LDAP che contiene gli utenti (ou=users,dc=flaskwebapp,dc=com).
* **LDAP\_GROUP\_DN**: Percorso nella directory LDAP che contiene i gruppi (ou=groups,dc=flaskwebapp,dc=com).

**Bind LDAP**

* **LDAP\_BIND\_USER**: Nome distinto (DN) dell'utente utilizzato per eseguire il bind al server LDAP (ad esempio, cn=admin).
* **LDAP\_BIND\_PASSWORD**: Password associata all'utente per il bind.

**Attributi per Autenticazione e Gruppi**

* **LDAP\_LOGIN\_ATTR**: Attributo LDAP utilizzato per identificare l'utente durante l'autenticazione (nel nostra caso, uid).
* **LDAP\_GROUP\_ATTR**: Attributo LDAP che specifica i membri di un gruppo (nel nostro caso, member).

**Filtri per Ricerca Utenti e Gruppi**

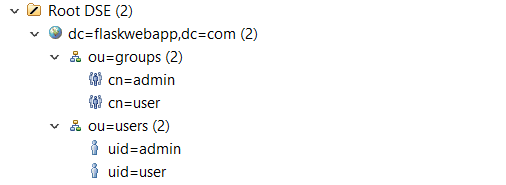
* **LDAP\_USER\_SEARCH\_FILTER**: Filtro LDAP per trovare utenti, basato sulla classe di oggetti person.
* **LDAP\_GROUP\_SEARCH\_FILTER**: Filtro LDAP per individuare i gruppi, basato sulla classe di oggetti groupOfNames.

## **Gestione della Directory LDAP**

La directory LDAP è stata popolata utilizzando **Apache Directory Studio**, un'interfaccia grafica per la gestione e configurazione delle directory LDAP. La struttura è stata organizzata in due **Unità Organizzative (OU)** principali:

1. **ou=groups**: Gestisce i ruoli degli utenti.
2. **ou=users**: Contiene le informazioni sugli utenti.

Graficamente:



## **Associazione tra Utenti e Gruppi**

* **admin** (uid=admin) è stato associato al gruppo **admin** (cn=admin).
* **user** (uid= user) è stato associato al gruppo **user** (cn=user).

Questa associazione è stata effettuata tramite l'attributo **member** nei gruppi, che include i DN (Distinguished Names) degli utenti.

## **Processo di Login con LDAP**

In questa web app, il processo di login degli utenti è gestito tramite il protocollo LDAP, utilizzando la libreria Python ldap3. Questa libreria consente di interagire con il server LDAP per l'autenticazione, la gestione dei ruoli e l'autorizzazione.

Il modulo ldap3 viene utilizzato per gestire le comunicazioni con il server LDAP. La libreria permette di eseguire operazioni di bind (autenticazione), eseguire ricerche nel database LDAP e recuperare informazioni relative agli utenti e ai gruppi.

**Identificazione**

LDAP consente di identificare gli utenti tramite il loro Distinguished Name (DN). Questo è un identificativo univoco che combina diversi attributi, garantendo che ogni utente abbia una rappresentazione unica all'interno della directory.

## **Autenticazione**

L'autenticazione avviene confrontando le credenziali (username e password) fornite dal client con quelle archiviate nella directory LDAP. Questo processo viene eseguito tramite il binding al server LDAP:

* **L'utente fornisce il proprio uid e la password.**
* **Il server verifica la validità delle credenziali.**

Il processo nel dettaglio:

**Connessione al server e bind amministrativo**:  
La web app si connette inizialmente al server LDAP utilizzando un bind con le credenziali amministrative configurate nell'applicazione (ad esempio, LDAP\_BIND\_USER e LDAP\_BIND\_PASSWORD). Questo bind non è eseguito dall'utente ma dall'applicazione stessa. Questo passaggio è necessario per interagire con il server LDAP, recuperare informazioni e fare query sui gruppi.Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

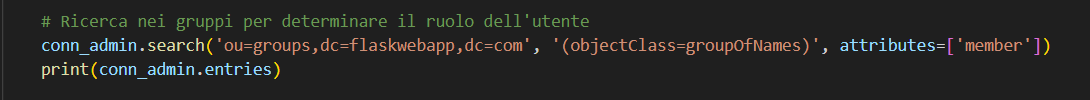
**Bind dell'utente**:  
Una volta ottenuto il Distinguished Name (DN) dell'utente, la web app tenta un nuovo bind utilizzando il DN dell'utente e la password fornita. Questo passaggio verifica se le credenziali dell'utente sono valide e conferma che l'utente può accedere al sistema.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Autorizzazione**

L'autorizzazione è gestita verificando i **gruppi di appartenenza**. Dopo aver verificato le credenziali, la web app esegue una query LDAP per determinare i gruppi a cui l'utente appartiene. La web app associa i gruppi dell'utente a un ruolo.



L'autorizzazione può essere gestita tramite la verifica dell'appartenenza a gruppi specifici. La web app esegue una query per identificare i gruppi di appartenenza dell'utente e mappa i gruppi LDAP ai ruoli dell'applicazione

# **Autorizzazione tramite XACML**

L'autorizzazione come abbiamo visto è un aspetto cruciale nella sicurezza delle applicazioni, in particolare per controllare l'accesso alle risorse in base a ruoli e politiche definite. Uno standard comunemente utilizzato per gestire politiche di accesso è **XACML (eXtensible Access Control Markup Language)**. Questo linguaggio, basato su XML, consente di definire in modo strutturato e flessibile politiche di autorizzazione, separando la logica dell'accesso dall'applicazione.

XACML permette di rappresentare le politiche di accesso attraverso **Policy** e **Rule**, definendo i soggetti (utenti), le risorse (URL, dati, ecc.), le azioni (GET, POST, ecc.) e il contesto (orario, posizione, ecc.). Questi elementi vengono valutati da un **Policy Decision Point (PDP)** per determinare se una richiesta di accesso debba essere consentita o negata.

Se dovessimo scrivere una politica in XML idonea al nostro sistema sarebbe:  
Immagine che contiene testo, schermata

Descrizione generata automaticamente

## **Limitazioni di XACML con Flask**

Tuttavia, **Flask** non supporta nativamente XACML. Integrare un motore di autorizzazione completo basato su XACML come quello fornite al corso non risulta possibile. Per superare questa limitazione, possiamo utilizzare librerie Python come **py-abac**, che simulano i concetti di XACML utilizzando strutture JSON per definire le politiche.

## **Py-abac: Simulazione di XACML in Flask**

**py-abac** è una libreria che consente di definire politiche di autorizzazione in un formato JSON al posto di XML, e permette di valutare le richieste di accesso in modo analogo a XACML. Simula i principi implementando un **PDP** e uno **Storage** (per memorizzare le politiche).

Il seguente codice carica un set di politiche di autorizzazione da un file JSON (policy.json) e le aggiunge a uno **storage in memoria**. Per ogni politica, verifica se esiste già nello storage (utilizzando il suo **UID**); se non esiste, la aggiunge, altrimenti salta l'aggiunta. Viene gestito anche un eventuale errore nel processo di caricamento. Dopo aver caricato le politiche, il codice stampa un elenco delle prime 10 politiche caricate dallo storage. Infine, viene creato un **PDP (Policy Decision Point)** utilizzando lo storage che contiene le politiche caricate.

Immagine che contiene testo, schermata

Descrizione generata automaticamente

La funzione **is\_access\_allowed** esamina i dettagli dell'utente (ruolo, percorso e metodo) dalla sessione e dalla richiesta. Crea un oggetto di richiesta di accesso e lo valuta tramite il **PDP (Policy Decision Point).** Restituisce una decisione che indica se l'accesso è consentito o negato. La funzione esegue anche il debug stampando i dettagli dell'accesso e la decisione presa.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Il file di **policy.Json** è definito nel seguente modo:

Immagine che contiene testo, schermata

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## **Politica admin\_policy**

* **UID**: admin\_policy
* **Descrizione**: Questa politica consente agli utenti con il ruolo di admin di accedere a tutte le pagine.
* **Effetto**: allow – L'accesso è consentito.
* **Regole**:
  + **Soggetto**: L'utente deve avere il ruolo admin per poter essere autorizzato.
  + **Risorsa**: La politica si applica a tutte le risorse il cui percorso corrisponde alla regex ^/.\*, cioè qualsiasi URL che inizi con / (tutte le pagine poiché è l’admin).
  + **Azione**: L'azione deve essere un metodo GET o POST.
* **Target**: La politica si applica a tutte le risorse; quindi, il campo resource\_id è impostato su \*.
* **Priorità**: 1 – Questa è la politica con la priorità più alta.

## **Politica user\_policy**

* **UID**: user\_policy
* **Descrizione**: Questa politica consente agli utenti con il ruolo di user di accedere solo a determinate pagine (specificate nella regex).
* **Effetto**: allow – L'accesso è consentito.
* **Regole**:
  + **Soggetto**: L'utente deve avere il ruolo user per poter essere autorizzato.
  + **Risorsa**: La politica si applica solo a pagine specifiche, identificate dalla regex ^/(home|note|download)$, che corrisponde a /home, /note, e /download.
  + **Azione**: L'azione deve essere un metodo HTTP GET.
* **Target**: La politica si applica solo alle risorse denominate site\_resources, il che significa che riguarda una risorsa specifica del sito web.
* **Priorità**: 2 – Questa politica ha una priorità inferiore rispetto alla admin\_policy.

## **Differenze principali rispetto a XACML:**

1. **Soggetto (role):**
   * In entrambi i casi, la condizione è che l'utente debba avere il ruolo specificato (admin o user).
2. **Risorsa (path):**
   * Nel JSON, le risorse sono selezionate tramite espressioni regolari (regex) per includere o limitare i percorsi.
   * In XACML, le risorse sono elencate esplicitamente come valori (home, download, etc..).
3. **Azione (metodo HTTP):**
   * Nel JSON, sono esplicitamente specificati i metodi consentiti.
   * In XACML, i metodi HTTP non sono specificati direttamente, ma possono essere inclusi come attributi opzionali per le azioni da eseguire.

Entrambi i sistemi esprimono lo stesso concetto di autorizzazione basata sui ruoli, ma in modo diverso. **XACML** tende a essere più esplicito, con definizioni dirette per ogni risorsa e azione. Permette un controllo fine-grained in modo chiaro ma dettagliato. **JSON (ABAC)** è più flessibile e conciso, utilizzando espressioni regolari per le risorse e condizioni di appartenenza per le azioni. È più leggibile per casi d'uso semplici.

# **Valutazione della sicurezza con Microsoft Threat Modeling Tool: KeyCloak**

Il Threat Modeling (TM) è un processo strutturato che consente di identificare, analizzare ed enumerare le minacce potenziali che potrebbero influire su un sistema, oltre a definire le contromisure necessarie per mitigare tali minacce. In particolare, il TM permette di esaminare le vulnerabilità del sistema e pianificare azioni preventive attraverso controlli di sicurezza adeguati.

Il MTMT si basa sulla classificazione STRIDE proposta da Microsoft, che suddivide le minacce in sei categorie principali:

Spoofing: Minacce legate all'intercettazione o alla falsificazione di dati in transito, con l'intento di impersonare un utente o sistema legittimo.

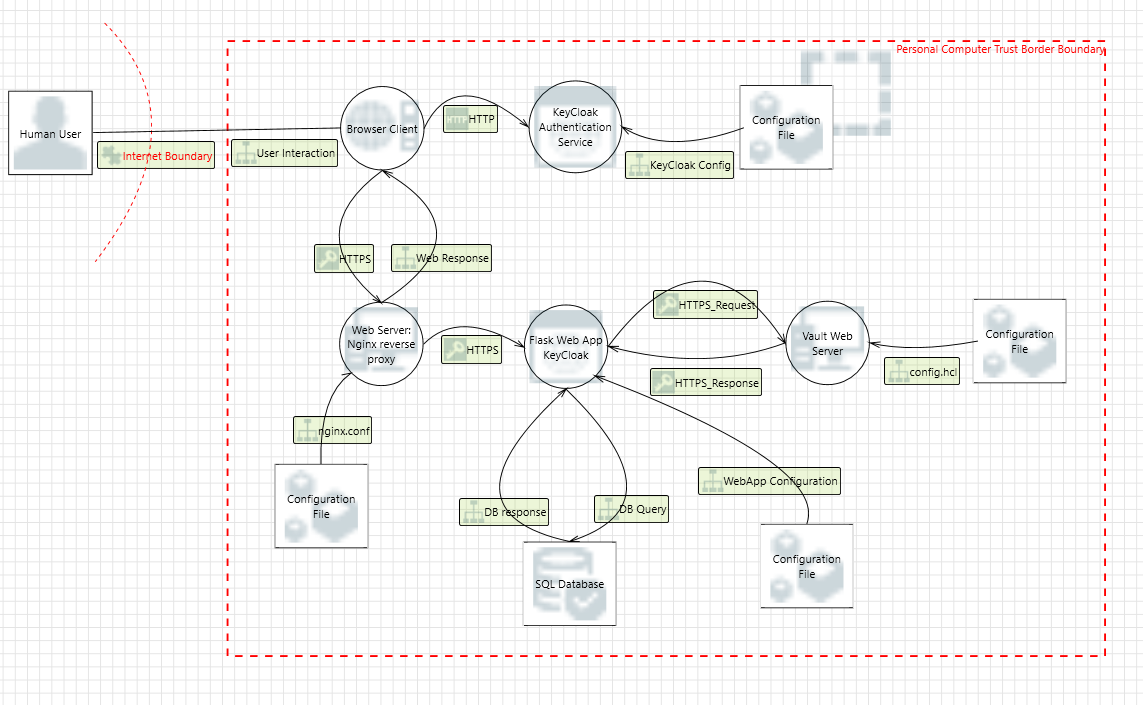
Tampering: Minacce che riguardano la manipolazione o la modifica dei dati, sia in transito che in storage, da parte di attaccanti.

Repudiation: Minacce che possono compromettere la responsabilità e la tracciabilità delle azioni eseguite, mettendo a rischio la reputazione e la fiducia nel sistema.

Information Disclosure: Minacce relative all'accesso non autorizzato a informazioni riservate, come la violazione della privacy o la fuga di dati sensibili.

Denial of Service (DoS): Minacce che mirano a rendere le risorse di sistema inaccessibili, causando un'interruzione dei servizi o una riduzione delle prestazioni.

Elevation of Privilege: Minacce che permettono a un utente di acquisire privilegi superiori a quelli previsti, ottenendo accesso non autorizzato a risorse o funzionalità riservate.



## **Denial of Service (DoS) su Nginx e Flask Web App**

Un attacco Denial of Service (DoS) mira a rendere un servizio o una risorsa non disponibile, tipicamente sovraccaricando un sistema con una quantità eccessiva di traffico. In questo scenario, l'attaccante può tentare di saturare le risorse di rete, server, o applicazione in modo che gli utenti legittimi non possano accedere ai servizi. Questo tipo di attacco può essere particolarmente dannoso per la disponibilità della web app.

### **Interazioni vulnerabili:**

Nginx, configurato come reverse proxy, è il primo punto di contatto con l'applicazione. Un attacco DoS potrebbe consistere nell'invio di una grande quantità di richieste HTTP al server Nginx, sovraccaricando il suo buffer di connessioni o la sua capacità di gestione del traffico, causando un rallentamento o la sospensione dei servizi.

Database SQLite: Sebbene SQLite non sia solitamente utilizzato per carichi di lavoro elevati, in caso di traffico massivo, potrebbe diventare un collo di bottiglia. Le richieste ripetute e non ottimizzate al database potrebbero portare a un rallentamento del sistema, peggiorando la disponibilità complessiva.

Poiché il database è interno e gestito dal server, limitare il numero di richieste che arrivano all'applicazione via Nginx aiuta a prevenire scenari in cui un attaccante potrebbe inviare un elevato volume di richieste per sfruttare le risorse del sistema, compreso il database.

### **Mitigazioni:**

Rate Limiting: Impostare limiti sul numero di richieste accettate da Nginx per unità di tempo (secondi/minuti). Questo aiuta a prevenire che un singolo utente o processo possa sopraffare

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

### **Controlli correlati:**

SI-10 - Information Input Validation: Validazione degli input per prevenire sovraccarichi di sistema.

SI-10(5) - Availability Controls: Controlli specifici per garantire la disponibilità del sistema.

Immagine che contiene testo, schermata, numero, software

Descrizione generata automaticamente

## **Elevation of Privilege: Impersonation attack**

Un attacco di **Elevation of Privileges** si verifica quando un utente riesce ad acquisire privilegi superiori a quelli a lui concessi, come l'accesso a risorse o funzionalità riservate ad amministratori. Nel contesto della nostra web app, tale attacco può verificarsi se il sistema di controllo degli accessi è configurato in modo errato o vulnerabile, permettendo a un utente di alterare i propri privilegi o ottenere accesso non autorizzato.

### **Interazioni vulnerabili:**

Nel nostro scenario, un attacco di **Elevation of Privileges** potrebbe avvenire tramite **Impersonation**. In questo caso, l'attaccante sfrutta la web app per impersonare la vittima, poiché la web app memorizza la sessione dell'utente, consentendo all'attaccante di usarla per accedere a risorse protette. Questo tipo di attacco riguarda le comunicazioni con i server coinvolti, in particolare Keycloak, Vault e Nginx, che gestiscono rispettivamente l'autenticazione, la gestione dei segreti e la sicurezza della connessione.

### **Mitigazioni:**

Per mitigare questa minaccia, è fondamentale implementare una corretta configurazione dei ruoli e dei permessi, assicurandosi che ogni utente possa accedere solo alle risorse per cui è autorizzato. L'uso di **Keycloak** e **OpenID Connect** per gestire i ruoli consente un controllo granulare degli accessi, riducendo significativamente il rischio di elevazione dei privilegi.

Inoltre, è possibile prevenire l'elevazione dei privilegi implementando **sessioni con scadenza**, sia lato web app che lato Keycloak. Ciò consente di limitare la durata della sessione dell'utente e ridurre il rischio che un attaccante utilizzi una sessione compromessa. È importante anche garantire la possibilità di **Logout** per gli utenti, permettendo loro di terminare le sessioni una volta completate le operazioni.

Un ulteriore strumento di mitigazione è l'adozione di un sistema di **audit logging**, che permetta di monitorare e rilevare modifiche sospette nei privilegi degli utenti. Questo sistema può contribuire a identificare tempestivamente eventuali azioni non autorizzate e a prendere provvedimenti correttivi rapidi.

### **Controlli correlati:**

AC-12 - Termination of Session

AC-12(1) - User-initiated Logouts

SC-23 - Session Authenticity

Immagine che contiene testo, schermata, numero, linea

Descrizione generata automaticamente

## **Information Disclosure sui File di Configurazione**

Un attacco di Information Disclosure si verifica quando un attaccante riesce a ottenere informazioni riservate o sensibili, come dettagli di configurazione, credenziali o dati aziendali, che dovrebbero essere protetti. Nel contesto della nostra web app, la divulgazione di informazioni può avvenire a causa di una protezione inadeguata dei file di configurazione o messa in chiaro di chiavi segrete o password nel codice. Se questi file non sono protetti adeguatamente contro l'accesso non autorizzato, un attaccante potrebbe ottenere informazioni sensibili come credenziali di accesso, configurazioni di sistema o altre informazioni critiche.

### **Interazioni vulnerabili:**

La vulnerabilità principale riguarda i file di configurazione che contengono informazioni sensibili, come credenziali di accesso, chiavi di autenticazione per Flask o Keycloak, configurazioni di database o parametri di sistema. Se questi file sono accessibili senza adeguate misure di protezione, un attaccante potrebbe riuscire a leggere i loro contenuti, compromettendo l’intera sicurezza del sistema.

### **Mitigazioni:**

Per prevenire la divulgazione non autorizzata di informazioni sensibili contenute nei file di configurazione, è necessario adottare una serie di misure di protezione:

Cifratura dei file: I file di configurazione contenenti informazioni sensibili, come credenziali e chiavi, devono essere cifrati. Questo impedisce che, anche in caso di accesso non autorizzato, le informazioni possano essere lette facilmente.

Utilizzo di un Secret Manager: È essenziale proteggere le informazioni critiche, come le credenziali di accesso e le chiavi API, utilizzando un Secret Manager, come Vault nel nostro caso. Questo sistema garantisce che solo i servizi autorizzati possano accedere alle informazioni sensibili, riducendo il rischio di esposizione. È inoltre consigliato l'uso di audit logging per monitorare e rilevare eventuali tentativi di accesso non autorizzato ai file sensibili.

### **Controlli correlati:**

AC-2 - Account Management

SI-12 - Information Handling and Protection

Immagine che contiene testo, schermata, numero, Carattere

Descrizione generata automaticamente

## **Repundiation: Potential Data Repudiation by Browser Client**

Un attacco di Repundiation si verifica quando un attaccante può negare l'esecuzione di una determinata azione o attività, come la creazione di una sessione o una modifica a un file, portando a una mancanza di tracciabilità e impossibilità di determinare l'origine di una determinata azione. In un contesto di applicazione web, la Redundation si manifesta quando non vengono registrati adeguatamente log o audit trail, impedendo così la ricostruzione degli eventi in caso di attacco o di violazione della sicurezza.

### **Interazioni vulnerabili:**

La Redundation è un rischio associato all'assenza di un adeguato sistema di registrazione degli eventi o a un sistema di log non sicuro. Se l'applicazione non registra tutte le operazioni critiche o se i log vengono alterati, un attaccante potrebbe negare di aver eseguito determinate azioni, come l'accesso non autorizzato a dati sensibili o l'invio di richieste dannose.

### **Mitigazioni:**

Logging e Auditing: È necessario implementare un sistema di logging robusto che registri tutte le azioni significative, come gli accessi degli utenti, le modifiche ai dati e le modifiche alle configurazioni. I log dovrebbero essere protetti da modifiche non autorizzate e archiviati in modo sicuro per evitare la Redundation.

Immutabilità dei Log: I log devono essere resi immutabili, impedendo la loro alterazione. Questo può essere ottenuto con l'uso di tecnologie come la scrittura dei log su un sistema esterno protetto o l'utilizzo di blockchain per garantire che i log non possano essere manipolati retroattivamente.

### **Controlli correlati:**

AU-3 - Audit Review, Analysis, and Reporting

Immagine che contiene testo, numero, Carattere, schermata

Descrizione generata automaticamente

## **Spoofing: DNS Spoofing e Man-in-the-Middle (MITM)**

Un attacco di Spoofing si verifica quando un attaccante falsifica la propria identità per ingannare la web app o gli utenti, presentandosi come un’entità legittima. Nel contesto della nostra web app, questo tipo di attacco potrebbe avvenire sia a livello di HTTPS, quando un malintenzionato riesce a falsificare l'indirizzo del server, ad esempio attraverso attacchi di DNS Spoofing o Man-in-the-Middle (MITM), intercettando e manipolando le comunicazioni tra il client e il server. In questo caso, l’attaccante potrebbe ottenere l'accesso a informazioni sensibili o risorse protette, come se fosse il server legittimo.

### **Interazioni vulnerabili:**

Le comunicazioni vulnerabili in questo caso potrebbero coinvolgere Nginx, Keycloak e Vault ma anche la stessa user interaction. Infatti, un altro scenario di Spoofing riguarda l'impersonificazione dell’utente, in cui un attaccante, sfruttando credenziali deboli o un processo di autenticazione inadeguato, riesce a impersonare un utente legittimo, accedendo così alla web app con privilegi non autorizzati.

### **Mitigazioni:**

Le mitigazioni per questi attacchi includono l'uso di HTTPS con certificati SSL/TLS correttamente configurati, che cifrano tutte le comunicazioni, prevenendo l’intercettazione dei dati. Inoltre, l’implementazione di un sistema di DNSSEC protegge da attacchi di DNS Spoofing. Per l'impersonificazione dell’utente, la mitigazione principale è l’introduzione di meccanismi di autenticazione sicuri, come l’autenticazione a due fattori (2FA), insieme a limiti sui tentativi di login per prevenire attacchi di forza bruta. È inoltre fondamentale proteggere e gestire correttamente le credenziali degli utenti e monitorare costantemente il processo di autenticazione per evitare falle sfruttabili.

### **Controlli correlati:**

SC-12 - Cryptographic Key Establishment and Management.

SC-13 - Use of Cryptography.

AC-3 - Access Enforcement.

AC-7 - Unsuccessful Logon Attempts.

IA-2 - Identification and Authentication.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

## **Tampering su HTTPS: attacco XSS**

Un attacco Cross-Site Scripting (XSS) si verifica quando un attaccante inserisce del codice malizioso, come Javascript, all'interno di un'applicazione web, con l'obiettivo di manipolare i dati dell'utente o rubare informazioni sensibili come cookie o credenziali. Nel contesto della nostra web app, questo tipo di attacco può avvenire attraverso l'iniezione di script maligni via URL, input non sanitizzati o altre forme di input da parte dell'utente. L'attaccante potrebbe iniettare codice nel sistema per eseguire operazioni non autorizzate a nome dell'utente vittima, come il furto di sessioni o la redirezione dell'utente verso siti pericolosi.

### **Interazioni vulnerabili:**

Questa minaccia fa riferimento specificamente all'attacco XSS e in particolare all'importanza di sanitizzare gli input. In un attacco XSS, l'URL modificato contiene del codice Javascript che può essere utilizzato per reindirizzare l'utente ad altre pagine o per rubare i cookie dell'utente e utilizzarli per eseguire operazioni sulla web app.

### **Mitigazioni:**

La minaccia può essere mitigata in parte attraverso una corretta gestione delle sessioni e dei cookie server side, e in parte mediante l'uso di HTML escaping. Quest'ultimo è un meccanismo che converte simboli, parole chiave o istruzioni presenti negli input del client in caratteri o stringhe, impedendo così l'esecuzione di codice maligno. Flask implementa già di default l'HTML escaping per la gestione dei template tramite il suo motore di templating Jinja Engine, che protegge automaticamente contro gli attacchi XSS. Un ulteriore approccio di mitigazione è l'adozione di politiche di Content Security Policy (CSP), che permettono di limitare l'esecuzione di script non autorizzati provenienti da origini non verificate.

### **Controlli correlati:**

SI-10 - Information Input Validation

SI-10(6) - Injection Prevention

Immagine che contiene testo, schermata, numero, linea

Descrizione generata automaticamente

## **Tampering sul Database: SQL Injection**

Il tampering sul database si verifica quando un attaccante è in grado di modificare direttamente o alterare i dati memorizzati nel database in modo non autorizzato. Questo tipo di attacco può avvenire attraverso vari vettori, come vulnerabilità nell'applicazione web, errori di configurazione, o attacchi come SQL Injection. L'attaccante può manipolare i dati per rubare informazioni, alterare registri, o compromettere l'integrità del sistema, ad esempio cambiando i privilegi degli utenti o iniettando dati dannosi.

### **Interazioni vulnerabili:**

Nel contesto della nostra web app, il database SQLite è un elemento critico che potrebbe essere soggetto a tampering, in particolare se non vengono implementate adeguate misure di sicurezza. Ad esempio, un attacco SQL Injection potrebbe consentire a un attaccante di eseguire comandi arbitrari sul database, alterando o cancellando dati vitali.

### **Mitigazioni:**

Per prevenire il tampering sul database, sono necessarie diverse contromisure. Una di queste è la sanitizzazione e la validazione degli Input, che è fondamentale per prevenire attacchi di SQL Injection. Flask utilizza query parametrizzate per prevenire l'inserimento di codice maligno nei comandi SQL (prepared statment). È inoltre essenziale validare ogni input utente in modo rigoroso per garantire che non contenga codice eseguibile. Anche l'accesso al database deve essere limitato solo agli utenti e ai processi che necessitano effettivamente di interagirvi (nel nostro caso gli admin). Utilizzare tecniche come il Least Privilege per garantire che gli utenti possano accedere solo alle informazioni necessarie e non possano alterare dati critici o eseguire comandi pericolosi.

### **Controlli correlati:**

SI-10 - Information Input Validation

SI-10(6) - Injection Prevention

AC-3 - Access Enforcement

SC-28 - Protection of Information at Rest

Immagine che contiene testo, Carattere, numero, linea

Descrizione generata automaticamente

## **Replay Attacks**

Un replay attack è un tipo di attacco in cui un attaccante intercetta e poi ripete un messaggio legittimo inviato da un utente o sistema per ottenere un'azione non autorizzata. Ad esempio, un attaccante può catturare una richiesta HTTP valida e riprodurla successivamente per impersonare l'utente o eseguire un'operazione non desiderata.

### **Interazioni vulnerabili:**

Nel contesto della nostra web app, un replay attack potrebbe avvenire se un attaccante riuscisse a intercettare una sessione di autenticazione, per poi riprodurre la stessa richiesta e aggirare i controlli di sicurezza.

### **Mitigazioni:**

Timestamp e Nonce: Una tecnica comune per prevenire i replay attacks è l'uso di timestamp o nonce (numeri casuali usati una sola volta), che rendono un messaggio valido solo se viene inviato entro un intervallo di tempo limitato o se il nonce è unico.

Crittografia: Utilizzare la crittografia per proteggere i dati in transito (ad esempio tramite HTTPS) garantisce che i messaggi non possano essere facilmente intercettati o alterati dal mittente.

Autenticazione e Sessioni Sicure: Assicurarsi che ogni richiesta venga associata a un token univoco e valido, che scade dopo un certo periodo di tempo. L'uso di sessioni sicure e la validità limitata delle credenziali possono prevenire il riutilizzo di messaggi legittimi.

Controlli dei Duplicati: Monitorare i messaggi ripetuti e verificare che ogni azione importante, come una transazione o una modifica delle credenziali, venga eseguita solo una volta.

### **Controlli correlati:**

SC-12 - Cryptographic Key Establishment and Management

IA-5 - Authenticator Management

SC-13 - Use of Cryptography

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

# **Valutazione della sicurezza con Microsoft Threat Modeling Tool: LDAP**

Nel caso in cui si utilizzi un server LDAP al posto di Keycloak per gestire l'autenticazione, il processo presenta alcune differenze fondamentali, pur mantenendo una struttura simile. Di seguito sono riportate le principali considerazioni tecniche e di sicurezza, oltre al TM aggiornato:

Immagine che contiene testo, diagramma, Piano, linea

Descrizione generata automaticamente

## **Gestione del Login: Web App come Authenticator**

A differenza di Keycloak, in questo caso la web app gestisce direttamente il processo di login, inviando le credenziali dell'utente al server LDAP per l'autenticazione (come mostrato nel paragrafo dedicato a LDAP).Non vi è un contatto diretto tra il browser del client e il server LDAP. Tutte le richieste di autenticazione vengono mediate dalla web app.

### **Interazioni vulnerabili:**

Analogamente a Keycloak, le interazioni vulnerabili sono una possibile esposizione o modifica non autorizzata dei file di configurazione oppure attacchi sulla comunicazione tra la web app e il server LDAP, specialmente se non protetta.

### **Mitigazione:**

Come al solito, proteggere i file di configurazione con accessi controllati e utilizzare Vault per gestire segreti come password e certificati. Per quanto riguarda possibili Replay e Collision Attacks, l’abilitazione di HTTPS è sufficiente.

**Similarità con Keycloak:** Le principali minacce, come l'information disclosure e lo spoofing dei file di configurazione, sono condivise e richiedono strategie di mitigazione analoghe.

**Differenze con Keycloak:** La comunicazione diretta tra client e authentication server è eliminata, semplificando l'architettura, ma spostando una maggiore responsabilità di sicurezza sulla web app.

# **Catalogo dei controlli di sicurezza dello standard NIST**

L'ultimo stadio di questo progetto prevede l'applicazione del catalogo dei controlli di sicurezza stabilito dallo standard NIST, con l'obiettivo di verificare se il sistema sviluppato soddisfi i requisiti relativi ai tre livelli di impatto sulla sicurezza definiti dallo standard:

* **Low-Impact**: Questo livello si applica quando una compromissione della sicurezza del sistema comporta conseguenze limitate o di bassa entità.
* **Intermediate-Impact**: Questo livello si applica quando una compromissione della sicurezza del sistema potrebbe avere effetti significativi o severi.
* **High-Impact**: Questo livello si applica quando una compromissione della sicurezza del sistema può causare danni gravi o catastrofici.

Il documento di riferimento per l'analisi è il NIST SP 800-53.

## **Famiglia: Access Control (AC)**

In questa famiglia sono stati implementati 10 controlli e 6 enchantment di livello moderate.

Nel dettaglio abbiamo:

* 6 controlli di livello low/moderate: AC-1, AC-2, AC-3, AC-7, AC-8, AC-14
* 4 controlli di livello moderate: AC-4, AC-5, AC-6, AC-12

### **Livello raggiunto: Low**

Livello Low: Sì, sono stati implementati tutti i controlli richiesti per questo livello.

Livello Moderate: No, poiché mancano controlli aggiuntivi come AC-10, AC-11, AC-17, AC-18, AC-19, AC-20.

## **Famiglia: Identification and Authentication (IA)**

In questa famiglia sono stati implementati 6 controlli e nessun enchantment.

Nel dettaglio abbiamo:

* 6 controlli di livello low/moderate: IA-1, IA-2, IA-5, IA-6, IA-7, IA-8

### **Livello raggiunto: Nessuno**

Livello Low: No, poiché mancano IA-3 (Device Identification and Authentication) e IA-4 (Identifier Management).

Livello Moderate: No, poiché mancano controlli chiave come IA-9 (Service Identification and Authentication) e altri richiesti per questo livello.

## **Famiglia: System and Communications Protection (SC)**

In questa famiglia sono stati implementati 13 controlli e 2 enchantment di livello moderate.

Nel dettaglio abbiamo:

* 7 controlli di livello low/moderate: SC-1, SC-5, SC-7, SC-12, SC-13, SC-28, SC-39
* 5 controlli di livello moderate: SC-2, SC-8, SC-10, SC-17, SC-23
* 1 controllo di livello high: SC-3

### **Livello raggiunto: Nessuno**

Livello Low: No, poiché mancano controlli fondamentali come SC-4, SC-6, SC-9 e altri che sono richiesti per il livello Low.

Livello Moderate: No, poiché mancano diversi controlli come SC-4, SC-6, SC-9, SC-11, SC-14, SC-15.

Livello High: Oltre a SC-3 mancano tutti gli altri controlli richiesti per il livello High.

## **Famiglia: System and Information Integrity** (**SI**)

In questa famiglia sono stati implementati 8 controlli e 2 enchantment di livello moderate.

Nel dettaglio abbiamo:

* 6 controlli di livello low/moderate: SI-1, SI-2, SI-3, SI-4, SI-10, SI-12
* 1 controlli di livello moderate: SI-11
* 1 controllo di livello high: SI-6

### **Livello raggiuto: Low**

Livello Low: Sì, poiché sono stati implementati i controlli fondamentali richiesti per il livello Low.

Livello Moderate: No, manca l'implementazione di alcuni controlli necessari come SI-5, SI-7, SI-8, SI-9, SI-13, SI-14, SI-15, SI-16, SI-17.

Livello High: No, oltre a SI-6, manca l'implementazione di altri controlli fondamentali come SI-7, SI-8, SI-9, SI-10, SI-11, SI-12, SI-13, SI-14, SI-15, SI-16, SI-17.

## **Famiglia: Audit and Accountability** (**AU**)

In questa famiglia sono stati implementati 2 controlli e nessun enchantment.

Nel dettaglio abbiamo:

* 2 controlli di livello low/moderate: AU-3, AU-9

### **Livello raggiunto: Nessuno**

Livello Low: No, poiché mancano controlli essenziali come AU-1, AU-2, AU-4, AU-5, AU-6, AU-7, AU-8.

# **NIST CSF – KeyCloak + Confronto con LDAP**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Control** | **Low** | **Moderate** | **Justification** | **Confronto con LDAP: Controlli mancanti/raggiunti** |
| [AC-1](https://nvd.nist.gov/800-53/Rev4/control/AC-1) | ACCESS CONTROL POLICY AND PROCEDURES | AC-1 | AC-1 | Politiche di accesso definite e implementate, procedura di controllo accessi impostata. |  |
| [AC-2](https://nvd.nist.gov/800-53/Rev4/control/AC-2) | ACCOUNT MANAGEMENT | AC-2 | AC-2 (1) (2) (3) (4) | Identificazione e gestione dei ruoli tramite Keycloak. Non ci sono account di gruppo, solo account personali per garantire la responsabilità individuale. I ruoli sono predefiniti (User e Admin) e possono essere modificati solo dall’amministratore di sicurezza. L’amministratore è anche l’unico autorizzato a creare account di tipo Admin, mentre gli account User possono registrarsi liberamente previa verifica del sistema. Gli accessi al sistema sono regolati in base ai ruoli, secondo quanto stabilito da AC-3, con mapping di permessi e scopes definiti in Keycloak. L’auditing è abilitato sia per gli utenti che per gli amministratori in Keycloak e sul database (connessioni, query e tabelle). La revisione degli account è manuale e periodica per monitorare attività sospette o inattività prolungata. Tuttavia, il livello moderato non è ancora raggiunto, poiché mancano automazione avanzata, rimozione automatizzata degli account non necessari e audit più sofisticati. | Utilizzo di Apache directory studio per popolare il server e gestire gli utenti. |
| [AC-3](https://nvd.nist.gov/800-53/Rev4/control/AC-3) | ACCESS ENFORCEMENT | AC-3 | AC-3 | La protezione delle risorse sensibili è implementata tramite autenticazione OIDC e controllo dei ruoli basato su Keycloak. Rotte sensibili come /upload e /vista\_note sono accessibili solo agli utenti con ruolo Admin, con enforcement centralizzato utilizzando metodi che verificano il ruolo tramite current\_app.oidc.user\_getinfo. Qualsiasi tentativo di accesso non autorizzato è gestito con un errore HTTP 403 e viene registrato nei log del sistema per scopi di auditing. | E' necessario mappare manualmente ed esplicitamente ruoli, permessi e utenti. |
| [AC-4](https://nvd.nist.gov/800-53/Rev4/control/AC-4) | INFORMATION FLOW ENFORCEMENT |  | AC-4 | Gli utenti con ruolo "Admin" sono autorizzati a caricare file tramite la pagina di upload, mentre gli utenti con ruolo "User" possono solo scaricare i file caricati dalla pagina download. Questo flusso informativo è segregato in base ai privilegi dei ruoli definiti in Keycloak e gestito tramite controlli implementati a livello di applicazione. Per proteggere il flusso delle informazioni, tutte le comunicazioni interne alla web app avvengono tramite HTTPS, garantendo la cifratura end-to-end. I file caricati sono archiviati localmente con accesso regolato tramite autorizzazioni basate sul ruolo e protezione del filesystem. Anche per le "note" si applica una segregazione: solo gli utenti admin possono visualizzare e interagire con le note di altri utenti, garantendo l'isolamento tra utenti standard. L'accesso al database è limitato ai server autorizzati, con l'URI del database protetto tramite Vault. Le informazioni sensibili, come credenziali, sono gestite e archiviate in Vault per prevenire esposizioni accidentali. Inoltre, i dati sensibili degli utenti sono salvati nel webserver di Keycloak, separati dal database principale, per mitigare i rischi di attacchi SQL injection. Le query al database sono gestite tramite SQLAlchemy, che garantisce una separazione sicura tra dati e logica applicativa. Tutte le operazioni sui database e sui file sono tracciate per scopi di auditing. |  |
| [AC-5](https://nvd.nist.gov/800-53/Rev4/control/AC-5) | SEPARATION OF DUTIES |  | AC-5 | Le operazioni critiche, come l'upload di file o la visualizzazione delle note di tutti gli utenti, sono separate tra i ruoli di Admin e User grazie a Keycloak, con gli Admin autorizzati a caricare file e guardare le note, mentre gli User possono solo visualizzare le proprie note e scaricare i file. l'Admin tuttavia non ha poteri totali, non può modificare parametri al di fuori di quelli definiti dal suo ruolo. Il ruolo di amministratore di sicurezza e Admin non conincidono. |  |
| [AC-6](https://nvd.nist.gov/800-53/Rev4/control/AC-6) | LEAST PRIVILEGE |  | AC-6 (1) (2) (5) (9) (10) | Sono stati impostati correttamente i permessi per le risorse sensibili in tutti i contesti. L'accesso alle risorse della WebApp è limitato in base ai ruoli degli utenti, permettendo loro di eseguire solo le azioni strettamente necessarie. Inoltre, l'applicazione assicura che l'accesso al webserver Vault sia di sola lettura, utilizzando AppRole con una policy di "read-only" per limitare i privilegi ai soli processi necessari. |  |
| [AC-7](https://nvd.nist.gov/800-53/Rev4/control/AC-7) | UNSUCCESSFUL LOGON ATTEMPTS | AC-7 | AC-7 | Il controllo per i tentativi di login falliti è delegato a Keycloak, che gestisce questi limiti. | I tentativi di login falliti non sono monitorati e limitati automaticamente. |
| [AC-8](https://nvd.nist.gov/800-53/Rev4/control/AC-8) | SYSTEM USE NOTIFICATION | AC-8 | AC-8 | Dopo il login il sistema avvisa l'utente riguardo il suo stato nella homepage. | Non notifica gli utenti in modo nativo |
| [AC-12](https://nvd.nist.gov/800-53/Rev4/control/AC-12) | SESSION TERMINATION |  | AC-12 (1) | Logout gestito correttamente, sia lato WebApp sia lato server Keycloak. In seguito al logout viene correttamente eliminata la sessione da entrambi gli endpoint. In ogni caso le sessioni hanno durata limitata. | La sessione viene gestita interamente lato server |
| [AC-14](https://nvd.nist.gov/800-53/Rev4/control/AC-14) | PERMITTED ACTIONS WITHOUT IDENTIFICATION OR AUTHENTICATION | AC-14 | AC-14 | Nessun utente può accedere a risorse protette prima di aver effettuato correttamente il login o la registrazione. Se un utente non autenticato prova ad accedere alla home page o a qualsiasi altra pagina protetta, verrà automaticamente reindirizzato alla schermata di login di Keycloak. Una volta loggato, se un utente tenta di accedere a risorse protette che non sono autorizzate per il suo ruolo, verrà visualizzato un messaggio di errore che indica che l'autorizzazione non è sufficiente. |  |
| [IA-1](https://nvd.nist.gov/800-53/Rev4/control/IA-1) | IDENTIFICATION AND AUTHENTICATION POLICY AND PROCEDURES | IA-1 | IA-1 | Nella documentazione sono presenti tutte le policy di identificazione e autenticazione per i vari ruoli. Le politiche sono state implementate tramite OIDC, utilizzando Keycloak come provider di identità. L'autenticazione è obbligatoria per tutte le risorse protette e non. |  |
| [IA-2](https://nvd.nist.gov/800-53/Rev4/control/IA-2) | IDENTIFICATION AND AUTHENTICATION (ORGANIZATIONAL USERS) | IA-2 | IA-2 (1) (2) (3) (8) (11) (12) | Per gli utenti organizzativi, l'autenticazione avviene tramite Keycloak che fornisce l'accesso a risorse protette. Gli utenti sono identificati tramite i loro credenziali di accesso, e l'autenticazione è gestita in modo centralizzato, garantendo un controllo sicuro e una gestione coerente degli accessi. Grazie alla 2FA si raggiunge il livello moderate. |  |
| IA-5 | AUTHENTICATOR MANAGEMENT | IA-5 | IA-5 | La gestione degli authenticator (password, token, chiavi, ecc.) è delegata a Keycloak, che offre funzionalità avanzate | OpenLDAP non offre nativamente funzionalità avanzate come invece fa Keycloak. Di conseguenza non soddisa il controllo |
| [IA-6](https://nvd.nist.gov/800-53/Rev4/control/IA-6) | AUTHENTICATOR FEEDBACK | IA-6 | IA-6 | Gestito da Keycloak. Durante il login, Keycloak non specifica se è sbagliata la password o l'username. |  |
| [IA-7](https://nvd.nist.gov/800-53/Rev4/control/IA-7) | CRYPTOGRAPHIC MODULE AUTHENTICATION | IA-7 | IA-7 | La web app implementa un modulo di autenticazione crittografico tramite Vault per la protezione delle credenziali sensibili come il client secret per l'OIDC |  |
| [IA-8](https://nvd.nist.gov/800-53/Rev4/control/IA-8) | IDENTIFICATION AND AUTHENTICATION (NON-ORGANIZATIONAL USERS) | IA-8 | IA-8 (1) (2) (3) (4) | L'applicazione è in grado di gestire anche utenti che si registrano in autonomia, e che non vengono caricati in modo diretto manualmente. Gli utenti appena registrati avranno privilegio minimo di accesso | Il server LDAP è stato popolato manualmente tramite Apache Directory Studio e non offre la possibilità di registrare nuovi utenti, ma solo di effettuare il login |
| [SC-1](https://nvd.nist.gov/800-53/Rev4/control/SC-1) | SYSTEM AND COMMUNICATIONS PROTECTION POLICY AND PROCEDURES | SC-1 | SC-1 | E' stata definita formalmente una policy documentata per la protezione di sistemi e comunicazioni e procedure operative specifiche. |  |
| [SC-2](https://nvd.nist.gov/800-53/Rev4/control/SC-2) | APPLICATION PARTITIONING |  | SC-2 | I vari moduli sono separati (Webapp, vault, keycloak, nginx, etc..). In particolare, sia Keycloak che Vault forniscono un'interfaccia separata per l'amministratore di sicurezza. |  |
| [SC-3](https://nvd.nist.gov/800-53/Rev4/control/SC-3) | SECURITY FUNCTION ISOLATION |  |  | Le funzioni di sicurezza come l'autenticazione e l'autorizzazione sono gestite da Keycloak, un sistema separato e dedicato. Questo rispetta il principio di isolamento delle funzioni di sicurezza, riducendo il rischio che un compromesso nell'app principale impatti direttamente l'autenticazione. | E' presente minore isolamento, poiché per arrivare ad eseguire l'autenticazione bisogno passare per l'applicazione stessa, che fa uso di una pagina di login personalizzata e non fornita nativamente da LDAP. |
| [SC-5](https://nvd.nist.gov/800-53/Rev4/control/SC-5) | DENIAL OF SERVICE PROTECTION | SC-5 | SC-5 | L'utilizzo di Nginx come reverse proxy configurato con proxyfix offre una protezione base contro richieste malformate o eccessive |  |
| [SC-7](https://nvd.nist.gov/800-53/Rev4/control/SC-7) | BOUNDARY PROTECTION | SC-7 (1) (2) | SC-7 (3) (4) (5) (7) | L'uso di un reverse proxy Nginx per tutte le comunicazioni tra i client e la web app implementa una protezione di confine. Nginx gestisce HTTPS, proteggendo i dati in transito, e limita l'accesso diretto al server backend. | L'uso di docker per il server LDAP garantisce una rete maggiormente isolata e sicura rispetto a Keycloak. |
| [SC-8](https://nvd.nist.gov/800-53/Rev4/control/SC-8) | TRANSMISSION CONFIDENTIALITY AND INTEGRITY |  | [SC-8 (1)](https://nvd.nist.gov/800-53/Rev4/control/SC-8?baseline=moderate#enhancement-1) | La confidenzialità e l'integrità dei dati trasmessi sono protette attraverso l'uso di HTTPS configurato su Nginx. Le comunicazioni tra client e server sono crittografate, garantendo protezione contro l'intercettazione o la modifica dei dati. Purtroppo per via dei certificati autosigned non è stato possibile implementarlo anche per Keycloak e LDAPS. |  |
| [SC-10](https://nvd.nist.gov/800-53/Rev4/control/SC-10) | NETWORK DISCONNECT |  | SC-10 | Le sessioni hanno una durata limitata in seguito alla chiusura del browser |  |
| [SC-12](https://nvd.nist.gov/800-53/Rev4/control/SC-12) | CRYPTOGRAPHIC KEY ESTABLISHMENT AND MANAGEMENT | SC-12 | SC-12 | La WebApp e Nginx utilizzano chiavi crittografiche per HTTPS. Tuttavia, non sono processi specifici per la gestione del ciclo di vita delle chiavi, come la rotazione periodica o la revoca in caso di compromesso. Il tutto viene fatto manualmente, ma soddisfa comunque il requisito |  |
| [SC-13](https://nvd.nist.gov/800-53/Rev4/control/SC-13) | CRYPTOGRAPHIC PROTECTION | SC-13 | SC-13 | La crittografia è utilizzata per proteggere i dati in transito (HTTPS) e per l'autenticazione tramite protocolli sicuri (OIDC). Tutti i segreti sono al sicuro tramite Vault. Non è presente per i dati a riposo poiché non presentano contenuti sensibili. |  |
| [SC-17](https://nvd.nist.gov/800-53/Rev4/control/SC-17) | PUBLIC KEY INFRASTRUCTURE CERTIFICATES |  | SC-17 | Vengono effettivamente utilizzati certificati generati da una CA personale, aggiunti alla lista dei certificati validi di windows. |  |
| [SC-23](https://nvd.nist.gov/800-53/Rev4/control/SC-23) | SESSION AUTHENTICITY |  | SC-23 | Keycloak garantisce l'autenticità delle sessioni tramite l'uso di token firmati digitalmente (JWT), che includono meccanismi per evitare la modifica o l'uso non autorizzato. Le sessioni vengono anche gestite server-side, che aumenta la sicurezza. | Non viene effettuato tutto il protocollo di autenticazione con i token JWT |
| SC-28 | PROTECTION OF INFORMATION AT REST | SC-28 | SC-28 | Utilizzo di Vault per la protezione di credenziali, chiavi e dati sensibili. Il database delle note è in chiaro. | Docker salvaguarda tutti i dati utilizzati in volumi cifrati, che unito a vault raggiunge la totalità dei dati sensibili |
| [SC-39](https://nvd.nist.gov/800-53/Rev4/control/SC-39) | PROCESS ISOLATION | SC-39 | SC-39 |  | L'uso di container state-less evita che uno stato compromesso persista, poiché possono essere distrutti e ricreati facilmente, minimizzando i danni |
| [SI-1](https://nvd.nist.gov/800-53/Rev4/control/SI-1) | SYSTEM AND INFORMATION INTEGRITY POLICY AND PROCEDURES | SI-1 | SI-1 | Flask, Keycloak e il database operano come processi separati. La WebApp fa uso di ambiente virtuale .venv. | Con l'aggiunta di docker per il server LDAP sono tutti isolati |
| [SI-2](https://nvd.nist.gov/800-53/Rev4/control/SI-2) | FLAW REMEDIATION | SI-2 | [SI-2 (2)](https://nvd.nist.gov/800-53/Rev4/control/SI-2?baseline=moderate#enhancement-2) | E' stata formalizzata una policy documentata e procedure riguardanti l'integrità dei sistemi e delle informazioni. |  |
| [SI-3](https://nvd.nist.gov/800-53/Rev4/control/SI-3) | MALICIOUS CODE PROTECTION | SI-3 | SI-3 (1) (2) | La manutenzione dei difetti è gestita manualmente tramite patching del codice e aggiornamenti delle librerie. |  |
| [SI-4](https://nvd.nist.gov/800-53/Rev4/control/SI-4) | INFORMATION SYSTEM MONITORING | SI-4 | SI-4 (2) (4) (5) | Flask + SQLAlchemy offrono protezioni di base contro SQL Injection e XSS tramite query parametrizzate ed escaping automatico in Jinja2 |  |
| [SI-6](https://nvd.nist.gov/800-53/Rev4/control/SI-6) | SECURITY FUNCTION VERIFICATION |  |  | L'app include funzionalità di monitoraggio e logging continuo per rilevare comportamenti sospetti o violazioni della sicurezza. |  |
| [SI-10](https://nvd.nist.gov/800-53/Rev4/control/SI-10) | INFORMATION INPUT VALIDATION | SI-10 | SI-10(5) (6) | Sono presenti diversi Debug e funzioni di logging che verificano il corretto funzionamento di autenticazione, crittografia, etc… |  |
| [SI-11](https://nvd.nist.gov/800-53/Rev4/control/SI-11) | ERROR HANDLING |  | SI-11 | Flask offre protezioni di base contro SQL Injection e XSS tramite query parametrizzate e escaping automatico in Jinja2 |  |
| SI-12 | INFORMATION HANDLING AND PROTECTION | SI-12 | SI-12 | L'app gestisce correttamente gli errori tramite l'uso di pagine di errore personalizzate e logging delle eccezioni. |  |
| [AU-3](https://nvd.nist.gov/800-53/Rev4/control/SI-12) | CONTENT OF AUDIT RECORDS | AU-3 | AU-3 | Le informazioni sensibili come il client\_secret di Keycloak sono gestite tramite Vault. |  |
| [AU-9](https://nvd.nist.gov/800-53/Rev4/control/SI-16) | PROTECTION OF AUDIT INFORMATION | AU-9 | AU-9 | La web app registra eventi di accesso tramite Keycloak, che tiene traccia di log dettagliati come login, logout e modifiche degli utenti oltre ad un logging interno alla webapp stessa |  |